FAULT TOLERATING PROTOCOL IN 2-D MESH NETWORK

Project Report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology.

in

Computer Science & Engineering

under the Supervision of

Dr. Hemraj Saini

By

Abhinish Popli

To

Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

Certificate

This is to certify that project report entitled "**FAULT TOLERATING PROTOCOL IN 2-D MESH NETWORK**", submitted by **ABHINISH POPLI** in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science & Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

Date: **Supervisor's Name: Dr. Hemraj Saini**

ACKNOWLEDGEMENT

I am using this opportunity to express my gratitude to everyone who supported me throughout the course of this B.Tech project. I am thankful for their aspiring guidance, invaluably constructive criticism and friendly advice during the project work. I am sincerely grateful to them for sharing their truthful and inspiring views on a number of issues related to the project.

A special gratitude I give to my final year project guide, Dr. Hemraj Saini, whose contribution in stimulating suggestions and encouragement, helped me to coordinate my project.

Date: **Abhinish Popli**

(111293)

TABLE OF CONTENT

LIST OF FIGURES

Figure **Page No.**

ABSTRACT

During their lifetime, multi hop wireless mesh networks (WMNs) experience frequent link failures caused by channel interference, dynamic obstacles, and/or applications' bandwidth demands. These failures cause severe performance degradation in WMNs or require expensive manual network management for their real-time recovery. This paper presents an *autonomous network reconfiguration system* (ARS) that enables a multi radio WMN to autonomously recover from local link failures to preserve network performance. By using channel and radio diversities in WMNs, ARS generates necessary changes in local radio and channel assignments in order to recover from failures. Next, based on the thus-generated configuration changes, the system cooperatively reconfigures network settings among *local* mesh routers.

CHAPTER 1

1. INTRODUCTION

WIRELESS mesh networks (WMNs) are being developed actively and deployed widely for a variety of applications, like environmental monitoring, and Wireless Internet for metro cities etc. Wireless Internet service has also been evolving in various forms (e.g., using multi radio/channel systems) to meet the increasing capacity demands by the above-mentioned and other emerging applications. Nevertheless, because of heterogeneous and disturbing wireless links conditions, preserving the efficient performance of such WMNs are still a tricky problem. Likewise, some links of a WMN experiences considerable channel interference from other coexisting wireless networks. Sometime, parts of wireless network topology might not be able to meet increasing bandwidth demands from new mobile users and applications. Links in a certain area (e.g., a hospital) might not be able to use some frequency channels because of spectrum etiquette or regulation. Several solution have been projected for WMNs to recover from wireless link failures, but they still have many drawbacks as mentioned below:

1) Resource-allocation algorithms: It can provide various methods for initial network resource management. Nevertheless, even though this approach provides a optimal network configuration plan, but they often require "global" configuration changes, which are not efficient in case of frequent local link failures.

2) Greedy channel-assignment algorithm: It can reduce the requirement of network changes by changing settings of only the faulty link(s). Nevertheless, this change might not capable to realize full improvements, which can be fulfilled by considering configurations of neighboring mesh routers in addition to the faulty links.

3) Fault-tolerant routing protocols: Protocols such as local re-routing can be implemented, to use network level path diversity to avoid the faulty links. Nonetheless, they rely redundant transmissions, that require more network resources, having less efficiency than link-level network reconfiguration.

To overcome the given limitations, I propose an Autonomous Network Reconfiguration System (ARS) that allows a mr-WMN to autonomously reconfigure its local network

environment i.e. channel, radio, and route allocation for real-time recovery from link failures. Specifically, ARS is equipped with a reconfiguration planning algorithm that identifies local configuration changes instead of global changes to recover the fault, and lessen changes in important network settings. In short, Autonomous Network Reconfiguration System first searches for feasible local configuration changes available around a faulty area, based on channel available and radio configuration and then, by considering given network specifications as constraints, it implemented the suitable reconfiguration plans that require the minimum number of changes for the healthy network settings.

Next, ARS also includes a monitoring protocol that enables a WMN to perform real-time failure recovery in conjunction with the planning algorithm. The appropriate link-quality information from the above mentioned protocol is used to identify network changes that satisfy applications' new QoS demands or that avoid propagation of QoS failures to neighboring links (or "ripple effects"). Running in every mesh node, the monitoring protocol periodically measures wireless link conditions via a hybrid link-quality measurement technique. Based on the above measured information, ARS detects link failures and generates QoS-aware network reconfiguration plans upon detection of a link failure.

ARS has been implemented and evaluated via experimentation on mr-WMN test cases as well as via Network Simulator 2 (NS2), evaluation results show that ARS is far efficient than existing failure-recovery methods.

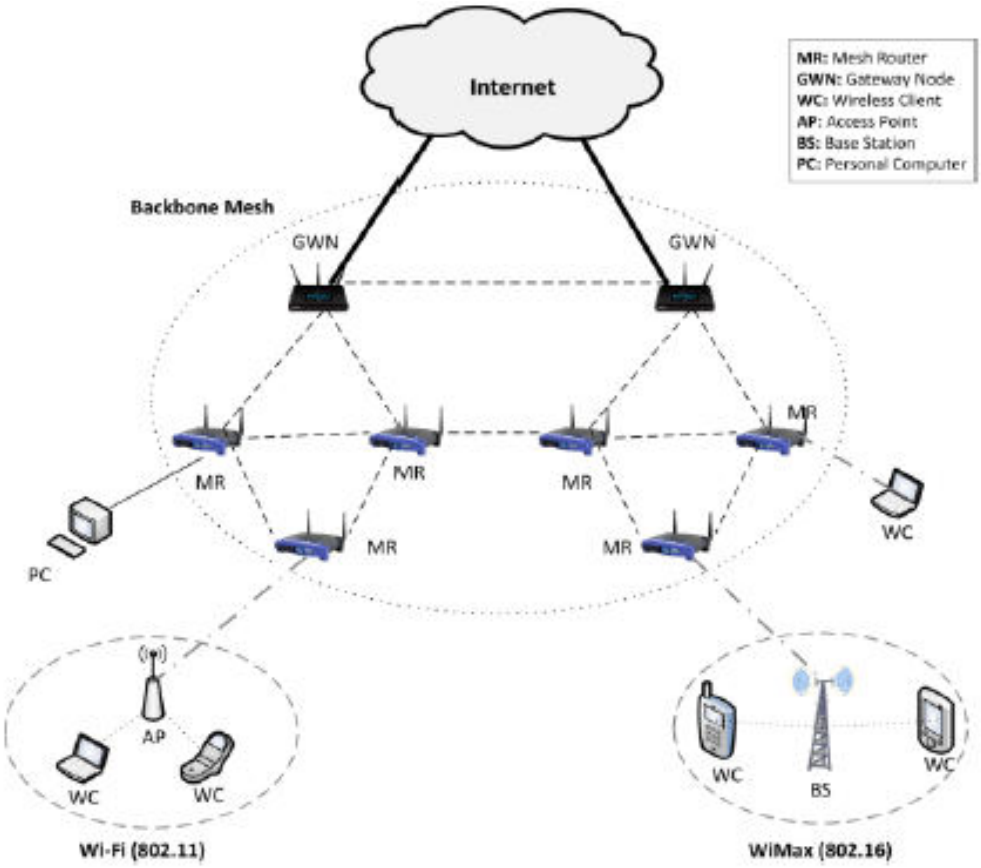


Fig.1.1 Wireless Mesh Network

CHAPTER 2

1. PRELIMINARIES

1.1. Fault-Tolerance in Real System

A system is something that provides some service. A system can be standalone or can be a part of a bigger system. Services provided by the systems may be used by another system to provide another service. For example, the memory of a computer system provides service to the user of the computer system. A fault-tolerant system is one that continues to perform its specified service in the presence of hardware and/or software faults. In designing fault-tolerant systems, mechanisms must be provided to ensure the correctness of the expected service even in the presence of faults. Due to the real-time nature of many fault-tolerant systems, it is important that the fault-tolerance mechanisms provided in such systems do not compromise the timing constraints of the real-time applications.

Hard real time systems require assurance about completion of each task within their deadline even in the presence of faults whereas required QoS must be achieved for weakly hard real time systems. A fault can be defined as some physical defect that can cause a component to malfunction. For hard real time system missing of a deadline would be dangerous whereas deterioration in QoS is observed in case of weakly hard real time systems. The Quality of Service of weakly hard real-time network systems, deteriorates more rapidly as number of faults increases. Thus, weakly hard real systems should gracefully degrade the QoS as the number of faults increases the system and must not suddenly collapse but contains to operate with reduced QoS limiting to some value. Thus, fault-tolerance is the property that enables a system to continue operating properly even in the event of faults in some of its components. The key concept used to achieve tolerance is the use of redundancy. The requirement of redundancy depends upon types of tolerance it to tolerate and level of tolerance. In next subsection, I explain the types of faults followed by the detection techniques used.

1.2. Failure, Error and Faults

Failure

A system failure occurs when the service provided by the system deviates from the specified service. For example, when a user cannot read his stored file from computer memory, then the expected service is not provided by the system.

Error

An Error is a perturbation of internal state of the system that may lead to failure. A failure occurs when the erroneous state causes an incorrect service to be delivered, for example, when certain portion of the computer memory is corrupted or broken and stored files therefore cannot be read by the user.

Faults

The cause of the error is called a fault. An active fault leads to an error; otherwise the fault is dormant. For example, impurities in the semiconductor devices may cause computer memory in the long run to behave unpredictably.

If a fault remains dormant during system operation, then there is no error. If the fault leads to an error, then the fault must be tolerated so that the error does not lead to system failure. To tolerate faults, errors must be detected in any fault-tolerant system. Identifying the characteristics of the faults that are manifested as errors is an important issue to design effective fault-tolerant system.

A failure is said to occur in the system when the system's environment observes an output from the system that does not conform its specifications. An error is the part of the system, e.g. one of the system components, which is liable to lead to a failure. A fault is the adjusted cause of an error and may itself be the result of a failure. Hence, a fault causes an error that produces a failure, which subsequently may result to a fault, and so on.

Let us consider the following example: A software bug in an application is a fault that leads to an error, when the application execution reaches the point affected by the bug. This

causes the application crash, which is a failure. By crashing, the application leaves blocked the socket ports it used, which is a fault. The computer on which the application crashed has socket ports that are not used by any process but still not accessible to running applications, which is an error. This is turn leads to a failure when another application requests these ports. Faults may occur either in the state of a component or in the state of a communication channel. The resulting errors can be arbitrary modification of state data, loss of state data, and loss or delays of the events in the communication channels. The consequences of these errors can be a variety of failures ranging from malicious deviation from system specifications to receive-omission failures and crash failures

1.3. Fault Types

Based on temporal effect of fault, a fault can be classified into permanent and transient faults.

Permanent Fault

Permanent Faults are the faults that makes one or several components of the system stop running forever, i.e. affect of faults does not die away with time. They are caused due conditions such as blowing off an integrated circuit (IC) chip. A permanent or hard fault is hardware is an erroneous state that is continuous and stable. Permanent faults in hardware are caused by the failure of the computing unit.

Transient Fault

Transient Faults are faults that place one or several components of the system stop running for a moment and then ready to be operation. The time gap between faulty state of component and become ready after faulty state is infinitely small. They are caused due to some environmental effects. The modern network systems are more susceptible to this type of faults than permanent ones.

CHAPTER 3

1. SIGNIFICANCE

In today's world, the computations are getting much complex. So the problem of reliability is getting severe day by day.

i.e. the physical systems that compose a network are subjected to a wide range of problems, ranging from signal distortion to component failures. Similarly, the software that supports the high-level interface often contains unknown bugs that make the system unreliable or inefficient.

So, the system operator could have taken correct measure that would have prevented or reduce the effect of failure. For this, we must require an efficient Fault Tolerating routing protocol that enables the system to continue with its operation, rather than completely failing, when some part of the network fails. This would help the operator to recover from the fault occurred in the network.

Fault tolerance is achieved in a routing algorithms either from redundant message copies in the network or redundant message route. Some of them are:

1. Gossip: flood selective message copies based on information from neighbors)

2. Stochastic communication.

These routing algorithms generally deliver shortest path and good tolerance against failures, but on the other side, they are extensive in terms of power consumption and show low flexibility for congestion.

Some algorithms, like dimension order routing or x-y routing, utilize presence of redundant routes in the network .These algorithms do not require global knowledge of the network state, and they guaranteed shortest path for packet to be destined.

CHAPTER 4

1. HARDWARE AND SOFTWARE SPECIFICATION

1.1. Hardware specification

Main processor : Intel(R) Core(TM) i5-2410M CPU @ 2.30GHz

Hard disk capacity : 500 GB

Cache memory : 512 MB

1.2. Software specification

Operating system : Ubuntu 11.04 (linux)

Scripting language : Network Simulator 2.35

Protocol developed : C++

Scripting : Tool Command Language

CHAPTER 5

1. ROUTING PROTOCOLS IN AD HOC NETWORK

1.1. Table-Driven (or Proactive)

The nodes maintain a table of routes to every destination in the network, for this reason they periodically exchange messages. At all times the routes to all destinations are ready to use and as a consequence initial delays before sending data are small. Keeping routes to all destinations up-to-date, even if they are not used, is a disadvantage with regard to the usage of bandwidth and of network resources.

1.1.1. PROACTIVE
a) DSDV (Destination-Sequence Distance Vector)

DSDV has one routing table, each entry in the table contains: destination address, number of hops toward destination, next hop address. Routing table contains all the destinations that one node can communicate. When a source A communicates with a destination B, it looks up routing table for the entry which contains *destination* address as B. Next hop address C was taken from that entry. A then sends its packets to C and asks C to forward to B. C and other intermediate nodes will work in a similar way until the packets reach B. DSDV marks each entry by sequence number to distinguish between old and new route for preventing loop.

DSDV use two types of packet to transfer routing information: full dump and incremental packet. The first time two DSDV nodes meet, they exchange all of their available routing information in full dump packet. From that time, they only use incremental packets to notice about change in the routing table to reduce the packet size. Every node in DSDV has to send update routing information periodically. When two routes are discovered, route with larger sequence number will be chosen. If two routes

have the same sequence number, route with smaller hop count to destination will be chosen.

DSDV has advantages of simple routing table format, simple routing operation and guarantee loop-freedom. The disadvantages are (i) a large overhead caused by periodical update (ii) waste resource for finding all possible routes between each pair, but only one route is used.

1.2. On-Demand (or Reactive)

These protocols were designed to overcome the wasted effort in maintaining unused routes. Routing information is acquired only when there is a need for it. The needed routes are calculated on demand. This saves the overhead of maintaining unused routes at each node, but on the other hand the latency for sending data packets will considerably increase.

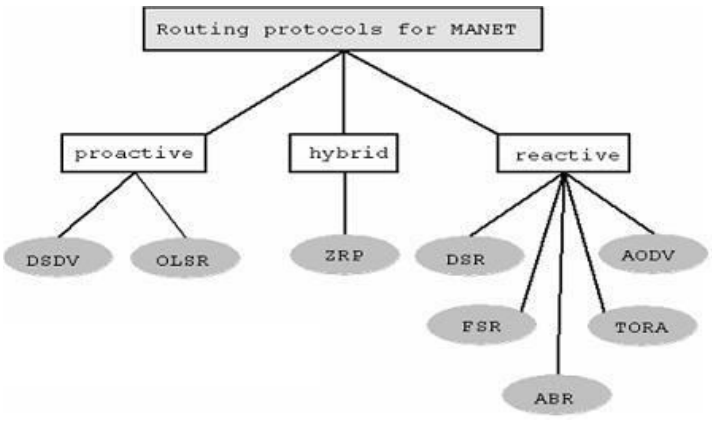


Fig.5.1 Routing Protocols

1.2.1. REACTIVE

a) On-demand Routing Protocols

In on-demand trend, routing information is only created to requested destination. Link is also monitored by periodical Hello messages. If a link in the path is broken, the source needs to rediscovery the path. On-demand strategy causes less overhead and easier to scalability. However, there is more delay because the path is not always ready. The following part will present AODV, DSR, TORA and ABR as characteristic protocols of on-demand trend.

b) AODV Routing

Ad hoc on demand distance vector routing (AODV) is the combination of DSDV and DSR. In AODV, each node maintains one routing table. Each routing table entry contains:

- Active neighbor list: a list of neighbor nodes that are actively using this route entry. Once the link in the entry is broken, neighbor nodes in this list will be informed.
- Destination address
- Next-hop address toward that destination
- Number of hops to destination
- Sequence number: for choosing route and prevent loop
- Lifetime: time when that entry expires

Routing in AODV consists of two phases: Route Discovery and Route Maintenance. When a node wants to communicate with a destination, it looks up in the routing table. If the destination is found, node transmits data in the same way as in DSDV. If not, it start Route Discovery mechanism: Source node broadcast the Route Request packet to its neighbor nodes, which in turns rebroadcast this request to their neighbor nodes until finding possible way to the destination. When intermediate node receives a RREQ, it

updates the route to previous node and checks whether it satisfies the two conditions: (i) there is an available entry which has the same destination with RREQ (ii) its sequence number is greater or equal to sequence number of RREQ. If no, it rebroadcast RREQ. If yes, it generates a RREP message to the source node. When RREP is routed back, node in the reverse path updates their routing table with the added next hop information. If a node receives a RREQ that it has seen before (checked by the sequence number), it discards the RREQ for preventing loop. If source node receives more than one RREP, the one with greater sequence number will be chosen. For two RREPs with the same sequence number, the one will less number of hops to destination will be chosen. When a route is found, it is maintained by Route Maintenance mechanism: Each node periodically send Hello packet to its neighbors for proving its availability. When Hello packet is not received from a node in a time, link to that node is considered to be broken. The node which does not receive Hello message will invalidate all of its related routes to the failed node and inform other neighbor using this node by Route Error packet. The source if still want to transmit data to the destination should restart Route Discovery to get a new path. AODV has advantages of decreasing the overhead control messages, low processing, quick adapt to net work topology change, more scalable up to 10000 mobile nodes . However, the disadvantages are that AODV only accepts bi-directional link and has much delay when it initiates a route and repairs the broken link.

c) DYNAMIC SOURCE ROUTING PROTOCOL

DSR is a reactive routing protocol which is able to manage a MANET without using periodic table-update messages like table-driven routing protocols do. DSR was specifically designed for use in multi-hop wireless ad hoc networks. Ad-hoc protocol allows the network to be completely self-organizing and self-configuring which means that there is no need for an existing network infrastructure or administration.

For restricting the bandwidth, the process to find a path is only executed when a path is required by a node (On-Demand-Routing). In DSR the sender (source, initiator) determines the whole path from the source to the destination node (Source-Routing) and deposits the addresses of the intermediate nodes of the route in the packets.

Compared to other reactive routing protocols like ABR or SSA, DSR is beacon-less which means that there are no hello-messages used between the nodes to notify their neighbors about her presence.

DSR was developed for MANETs with a small diameter between 5 and 10 hops and the nodes should only move around at a moderate speed. DSR is based on the Link-State-Algorithms which mean that each node is capable to save the best way to a destination. Also if a change appears in the network topology, then the whole network will get this information by flooding.

DSR contains 2 phases

- Route Discovery(find a path)
- Route Maintenance (maintain a path)

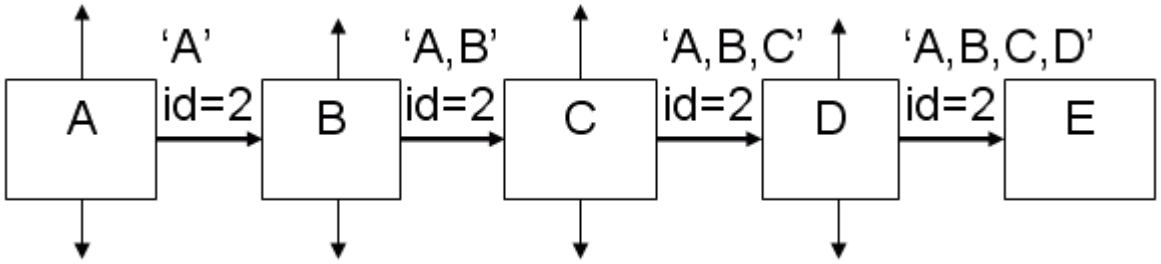


Fig 5.2 Route Discovery

If node A has in his Route Cache a route to the destination E, this route is immediately used. If not, the Route Discovery protocol is started:

1. Node A (initiator) sends a RouteRequest packet by flooding the network
2. If node B has recently seen another RouteRequest from the same target or if the address of node B is already listed in the Route Record, Then node B discards the request!

3. If node B is the target of the Route Discovery, it returns a RouteReply to the initiator. The RouteReply contains a list of the "best" path from the initiator to the target. When the initiator receives this RouteReply, it caches this route in its Route Cache for use in sending subsequent packets to this destination.
4. Otherwise node B isn't the target and it forwards the Route Request to his neighbors (except to the initiator).

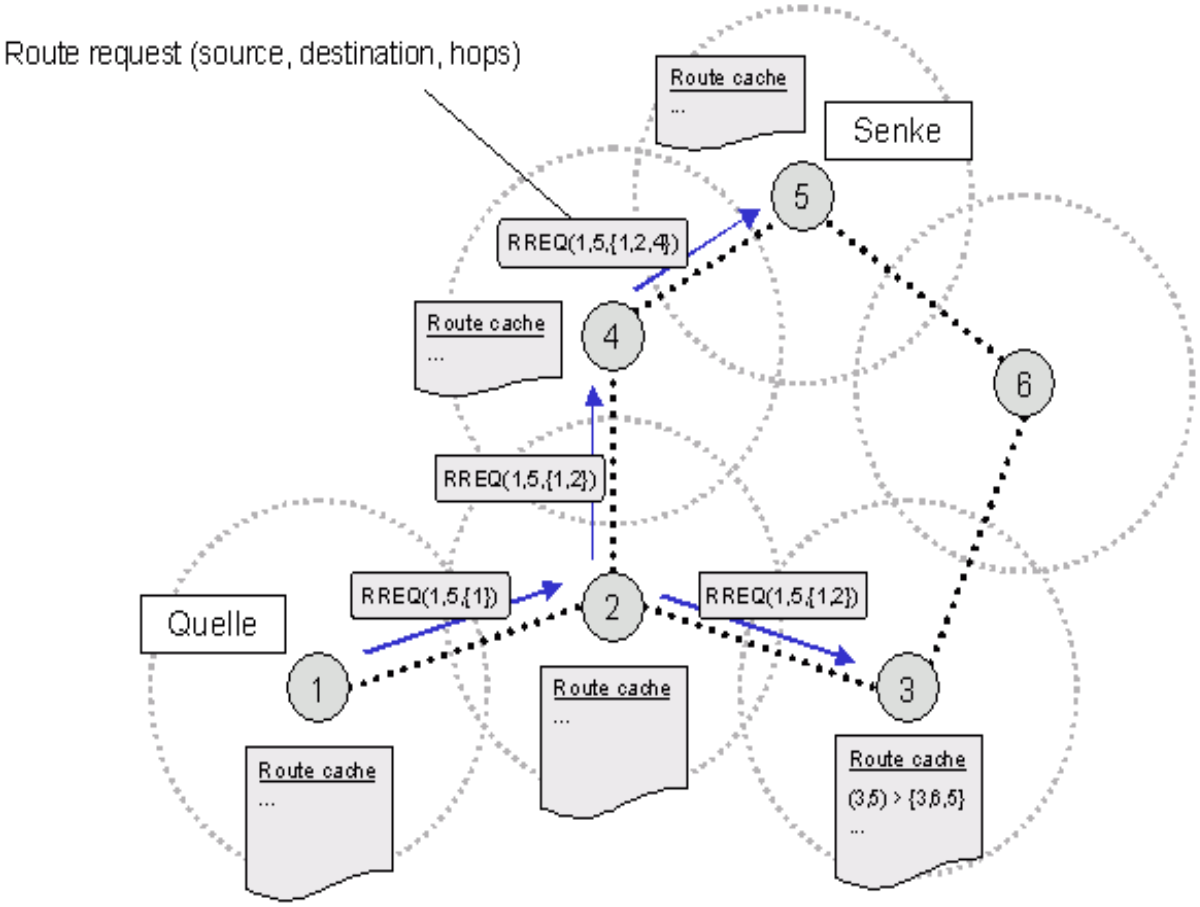


Fig 5.3. Path-finding-process: Route Request

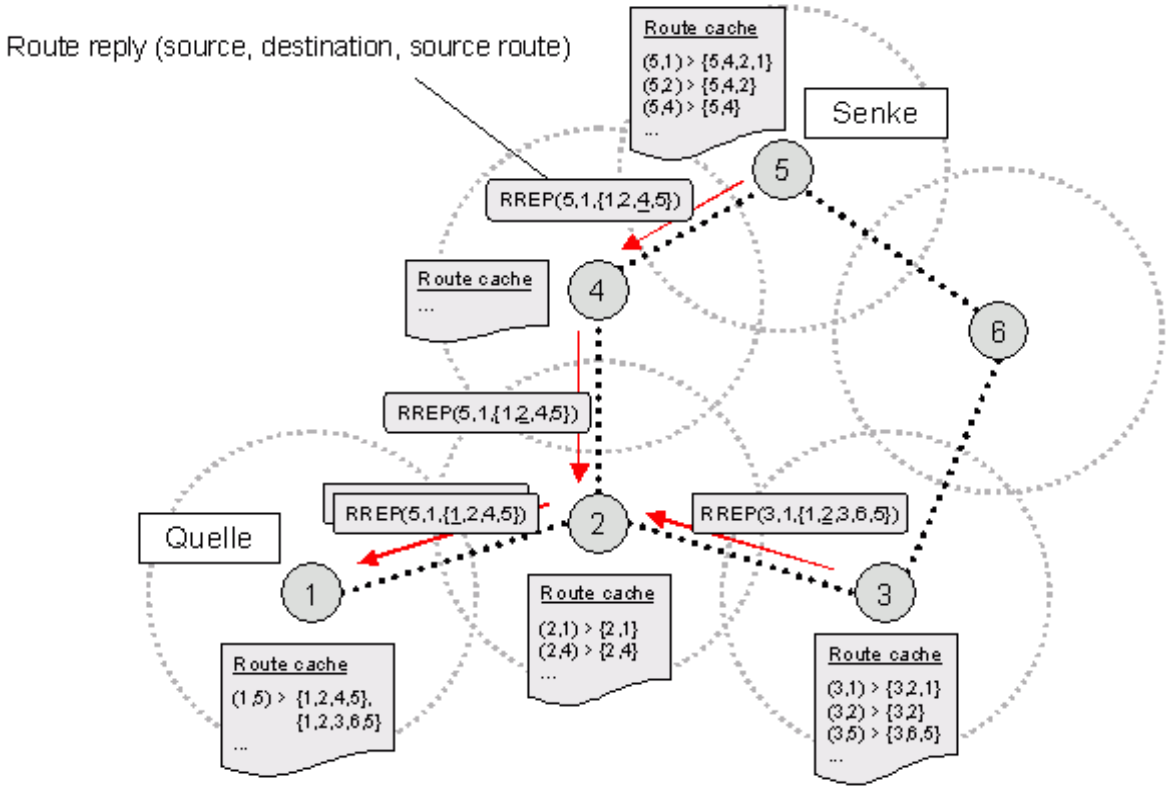


Fig 5.4. Path-finding-process: Route Reply

Route Maintenance

In DSR every node is responsible for confirming that the next hop in the Source Route receives the packet. Also each packet is only forwarded once by a node (hop-by-hop routing). If a packet can't be received by a node, it is retransmitted up to some maximum number of times until a confirmation is received from the next hop.

Only if retransmission results then in a failure, a RouteError message is sent to the initiator that can remove that Source Route from its Route Cache. So the initiator can check his Route Cache for another route to the target. If there is no route in the cache, a RouteRequest packet is broadcasted.

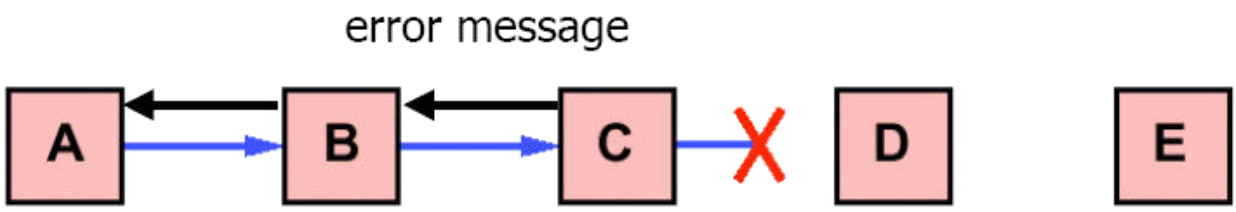


Fig 5.5 Error occurred between node C and D

1. If node C does not receive an acknowledgement from node D after some number of requests, it returns a RouteError to the initiator A.
2. As soon as node receives the RouteError message, it deletes the broken-link-route from its cache. If A has another route to E, it sends the packet immediately using this new route.
3. Otherwise the initiator A is starting the Route Discovery process again.

Advantages

Reactive routing protocols have no need to periodically flood the network for updating the routing tables like table-driven routing protocols do. Intermediate nodes are able to utilize the Route Cache information efficiently to reduce the control overhead. The initiator only tries to find a route (path) if actually no route is known (in cache). Current and bandwidth saving because there are no hello messages needed (beacon-less).

Disadvantages

The Route Maintenance protocol does not locally repair a broken link. The broken link is only communicated to the initiator. The DSR protocol is only efficient in MANETs with less then 200 nodes. Problems appear by fast moving of more hosts, so that the nodes can only move around in this case with a moderate speed. Flooding the network can cause collusions between the packets. Also there is always a small time

delay at the begin of a new connection because the initiator must first find the route to the target.

d) TORA (Temporary Ordered Routing Algorithm)

TORA is based on link reversal algorithm. Each node in TORA maintains a table with the distance and status of all the available links. Detail information can be seen at [38]. TORA has three mechanisms for routing:

- **Route Creation**: TORA uses the "height" concept for discovering multiple routes to a destination. Communication in TORA network is downstream, from higher to lower node. When source node does not have a route to destination, it starts Route Creation by broadcasting the Query messages (QRY). QRY is continuing broadcasted until reaching the destination or intermediate node that have the route to the destination. The reached node then broadcast Update (UPD) message which includes its height. Nodes receive this UPD set a larger height for itself than the height in UPD, append this height in its own UPD and broadcast. This mechanism is called reversal algorithm and is claimed to create number of direct links from the originator to the destination.
- **Route Maintenance**: Once a broken link is discovered, nodes make a new reference height and broadcast to their neighbors. All nodes in the link will change their reference height and Route Creation is done to reflect the change.
- **Route Erasure:** Erases the invalid routes by flooding the "clear packet" through the network. The advantages of TORA are: having multiple paths to destination decreases the route creation in link broken case therefore decrease overhead and delay to the network. TORA is also claimed to be effective on large and mildly congested network [9]. The drawbacks are requiring node synchronization due to "height" metric and potential for oscillation. Besides that, TORA may not guarantee to find all the routes for reserving in some cases.

CHAPTER 6

1. RELATED WORKS

Existing channel assignment and scheduling algorithms provide guidelines for channel assignment during a network deployment stage. But the given existing algorithms do not consider the changes from previous network settings, this type of algorithms are suitable for static or periodic network management, but they may cause network service interruptions, and thus are unsuitable for dynamic network reconfiguration that has to deal with frequent local link failures. The greedy algorithm may replace a faulty channel with a new channel but these faulty neighboring links, whose channel has been recovered, may fail to meet QoS demands if the links in the new channel experience interference from other coexisting networks that operate in the same channel.

CHAPTER 7

1. PROPOSAL

Here, I have proposed an Autonomous Network Reconfiguration System (ARS) that allows a multiradio WMN (mr -WMN) to autonomously reconfigure its local network settings for real time recovery from link failures.

ARS is equipped with a reconfiguration planning algorithm that identifies local configuration changes for the recovery. Briefly, ARS first searches for feasible local configuration changes available around a faulty vicinity, based on given available channel and radio associations. Then, by daunting current network specifications as constraints, ARS establish reconfiguration plans that require the minimum number of changes for the healthy network. Autonomous Network Reconfiguration System (ARS) also implements a monitoring protocol that enables a WMN to perform real-time failure recovery in conjunction with the planning algorithm.

2. Modules

2.1. 1st Module: Deployment Planning and Monitoring period

In this Deployment Planning stage, planning of resources is done to provide the shape to the network topology with n number of nodes (say 25), adjacent nodes are separated by certain distance (say 180m),each and every node is equipped with different number of radios, depending on its propinquity to a gateway. In monitoring period, ARS in every mesh node monitors the quality of its outgoing wireless links and reports the results to a gateway.

2.2. 2nd Module : Transmission and failure

There are settings to emulate real-time network activities. First, to generate users' traffic from source node to destination node, i.e. multiple User Datagram Packets flows between a gateway and randomly chosen mesh node (destination node) is introduced with a packet size of 1000 bytes. Secondly, it will create link failures, faults are injected and lasts for a specific time (failure period).

2.3. 3rd Module: Failure detection and Reconfiguration

Once it detects a link failure, ARS in the detector node(s) triggers the formation of a group among local mesh routers that use a faulty channel, and one of the group members is appointed as a leader via gateway using the well-known bully algorithm for coordinating the reconfiguration. The leader node sends a planning-request message to a gateway. Then, the gateway generates a reconfiguration plan for the request from the faulty node in the mesh. After that, gateway sends a reconfiguration plan to the leader node and the group members. Lastly, every node of the group executes the corresponding configuration changes, if any, and re-establishes the connection and restart the process (starts re-sending the UDP packets).

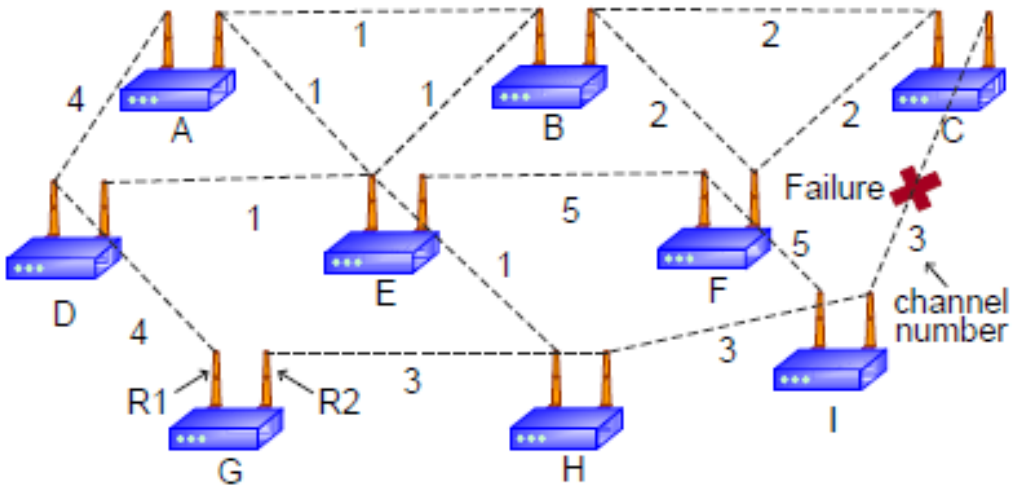


Fig. 7.1. Multiradio WMN.A WMN has an initial assignment of frequency channels as shown. Network experiences wireless link failure.

ARS effectively identifies QoS satisfiable reconfiguration plans by :

- Estimating the QoS satisfiability.
- Deriving their expected benefits in channel utilization.

3. Pseudo Code

(1) Monitoring period(tm)

for every link j **do**
measure link-quality (lq) using passive monitoring;
end for
send monitoring results to a gateway g;

(2) Failure detection and group formation period (tx)
if link l violates link requirements r **then**
request a group formation on channel c of link l ;
end if
participate in a leader election if a request is received;

(3) Planning period (M ,tp)
if node i is elected as a leader **then**
send a planning request message (c ,M) to a gateway;
else if node is a gateway **then**
synchronize requests from reconfiguration groups Mn
generate a reconfiguration plan (p) for Mi ;
send a reconfiguration plan to a leader of Mi;
end if

(4) Reconfiguration period (p ,tx)
if p includes changes of node i **then**
apply the changes to links at t;
end if
relay to neighboring members, if any

CHAPTER 8

1. SIMULATION

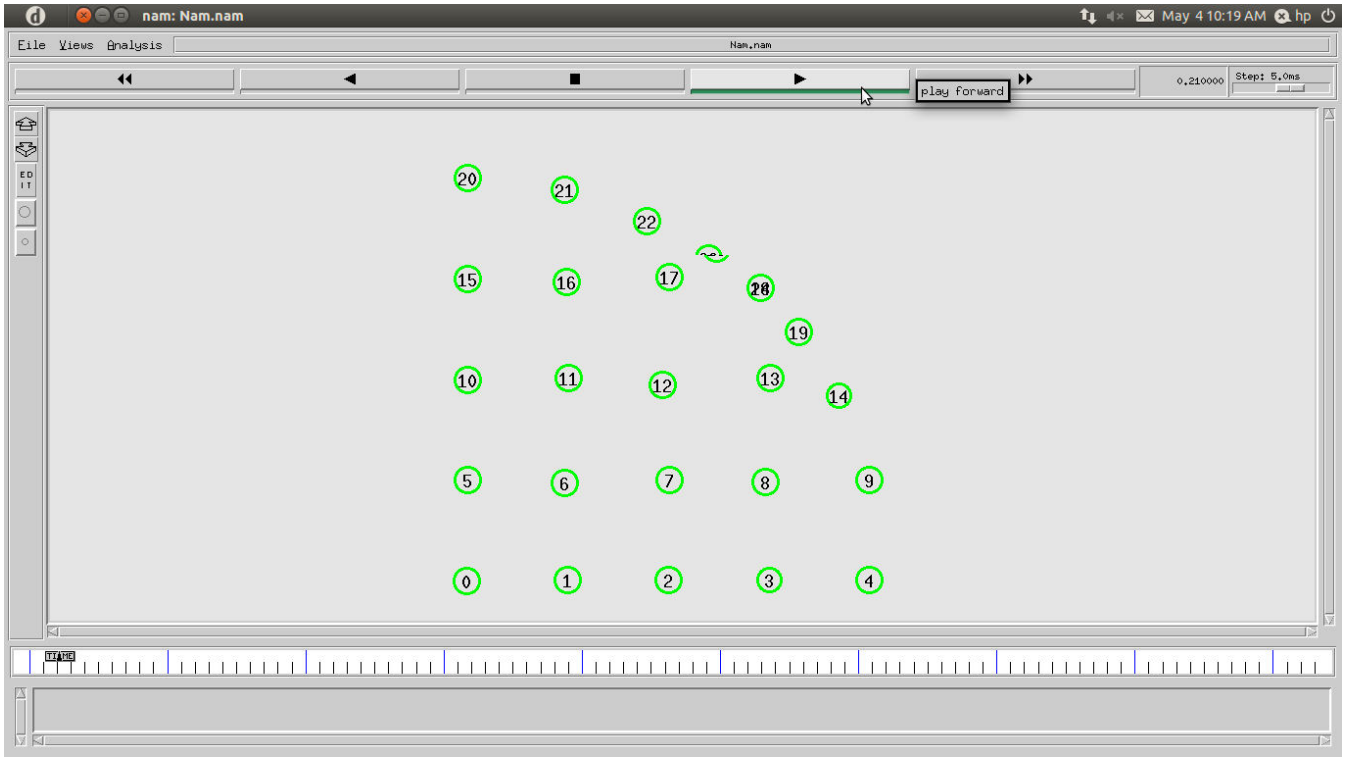


Fig 8.1. Deployment Phase

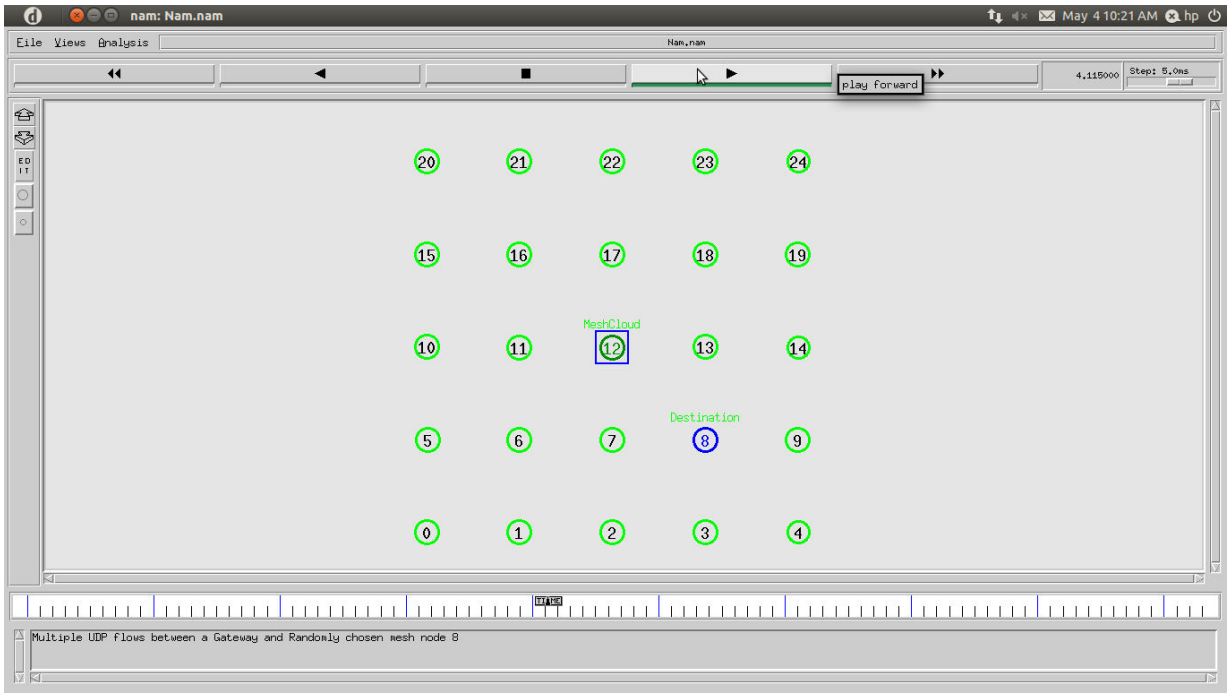


Fig 8.2. Marking the nodes

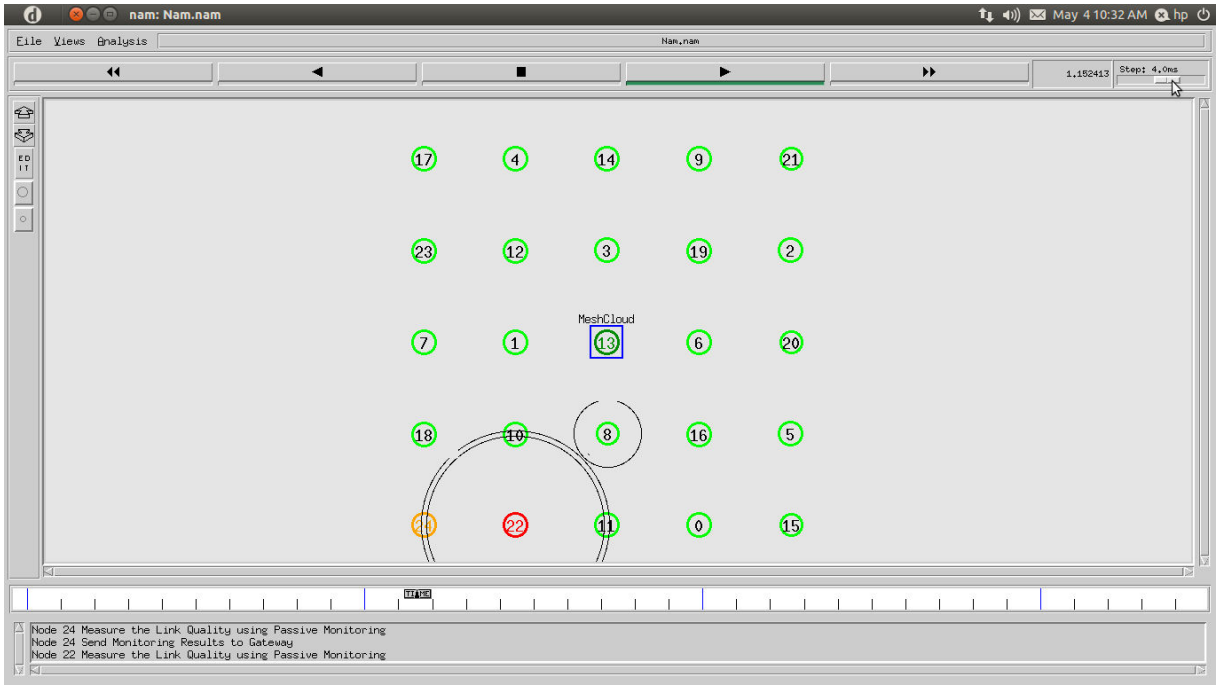


Fig 8.3(a). Check Connectivity

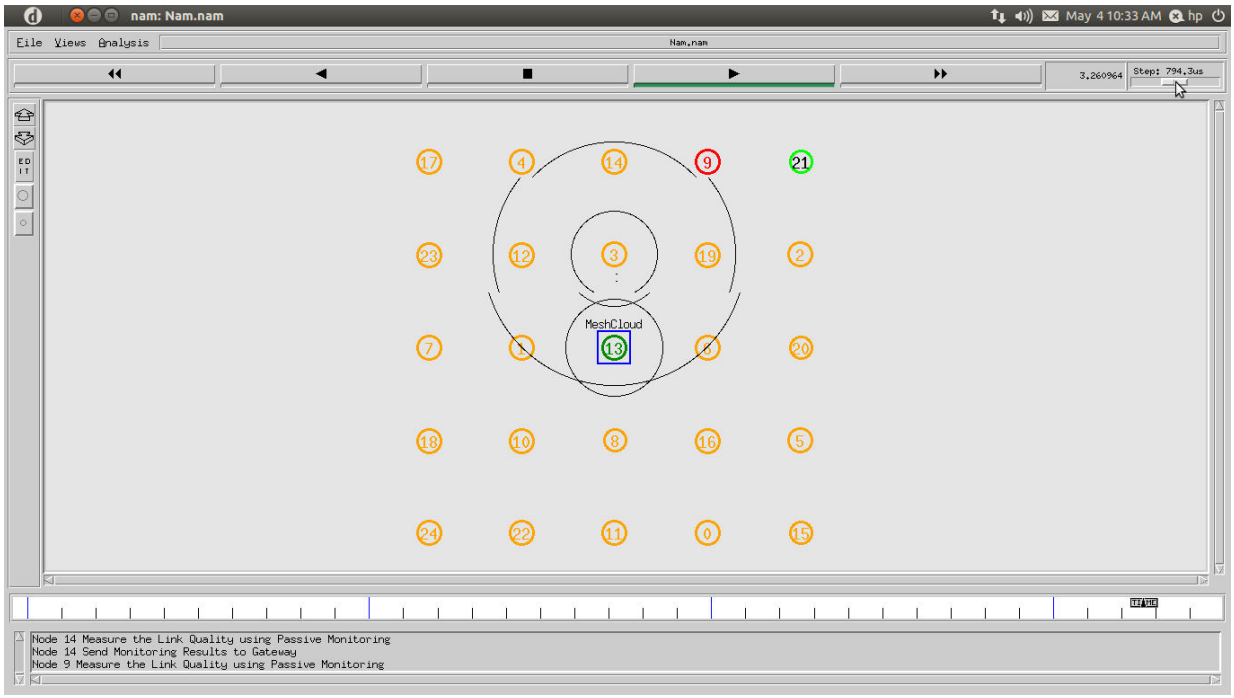


Fig 8.3(b). Check Connectivity

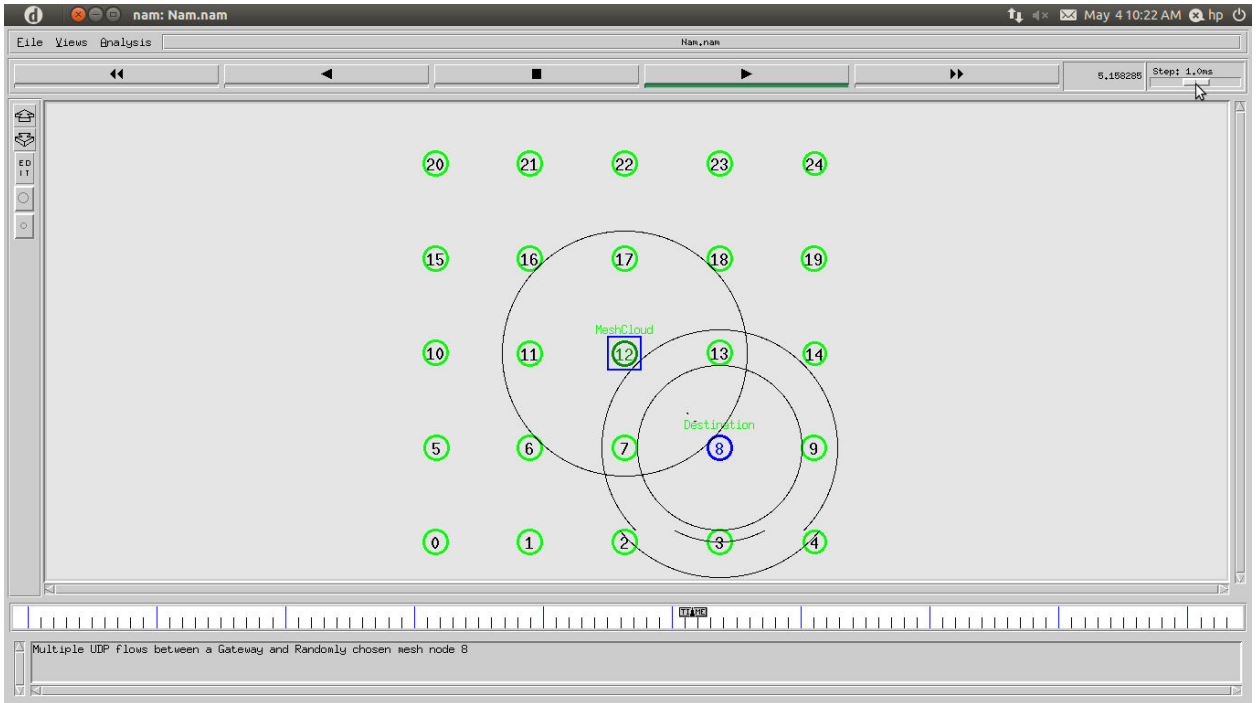


Fig 8.4(a). Sending UDP Packets

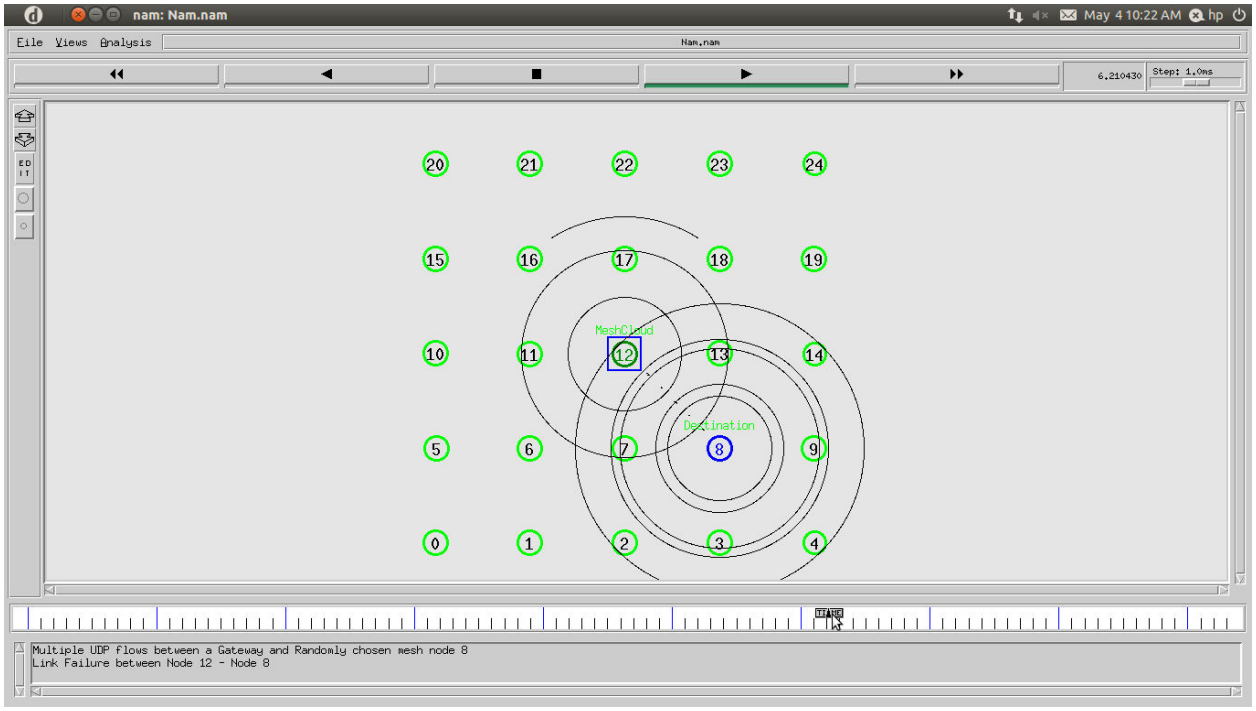


Fig 8.4(b). Sending UDP Packets

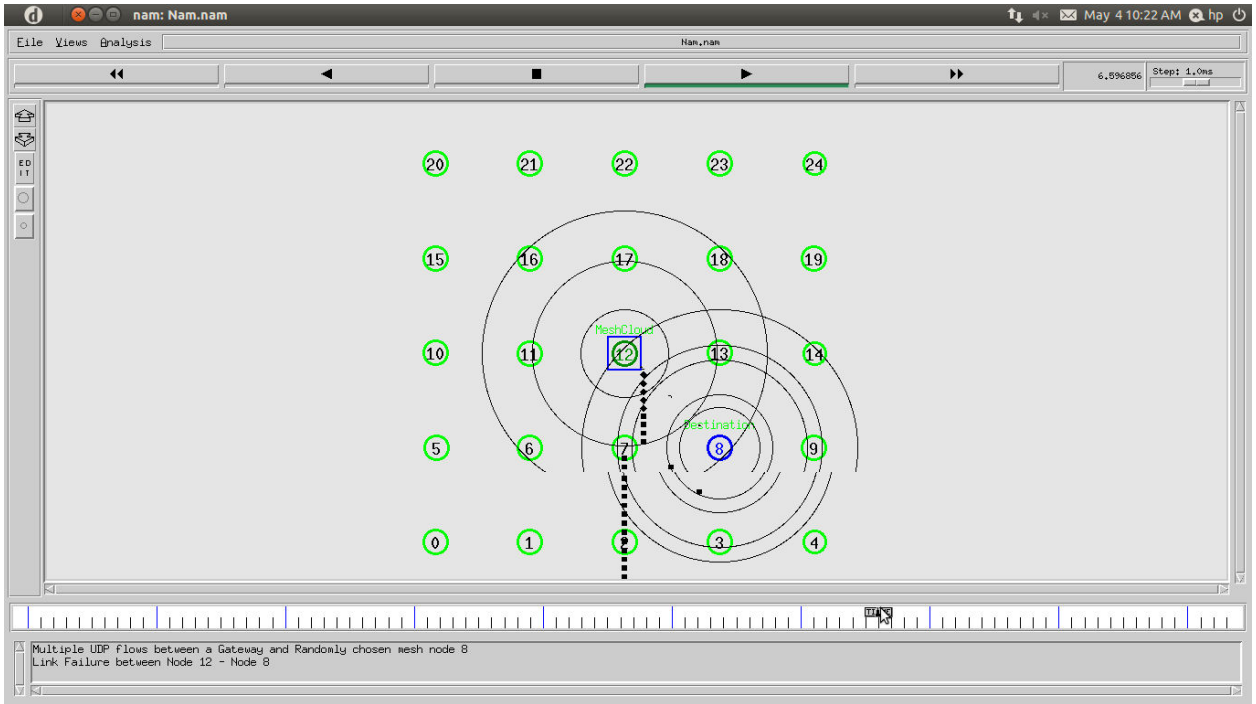


Fig 8.5. Packet starts Dropping

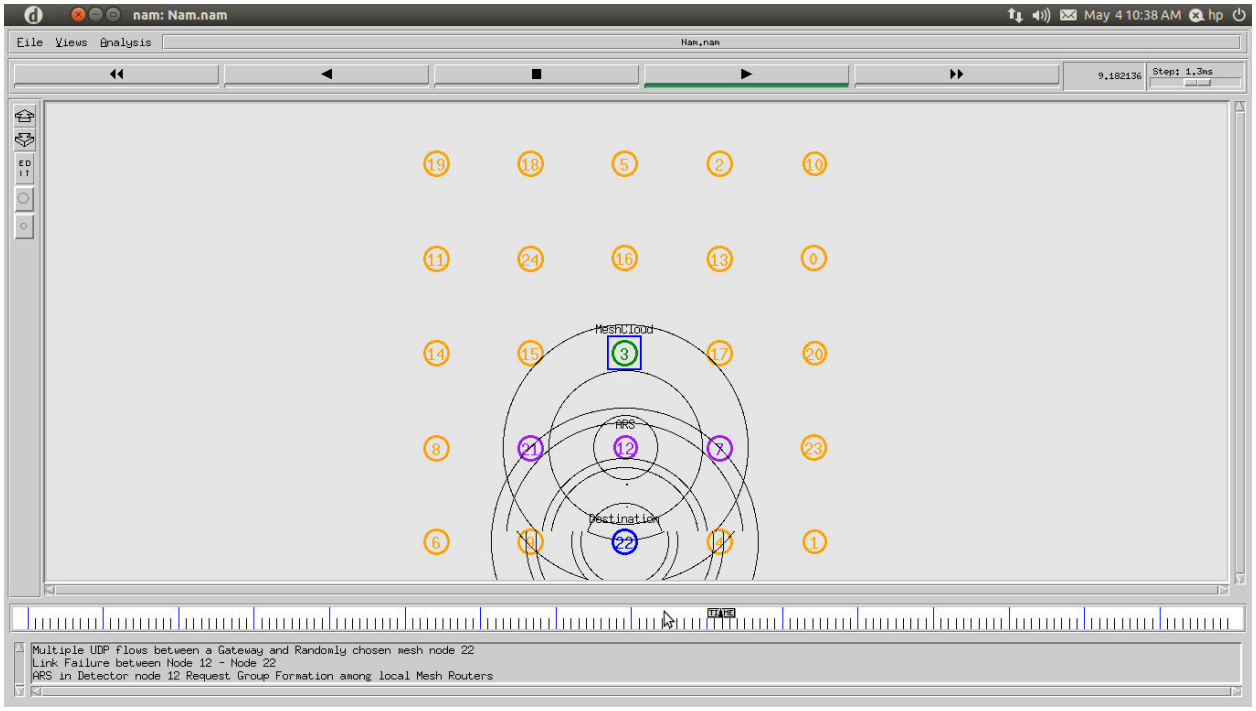


Fig 8.6(a). Reconfiguration started

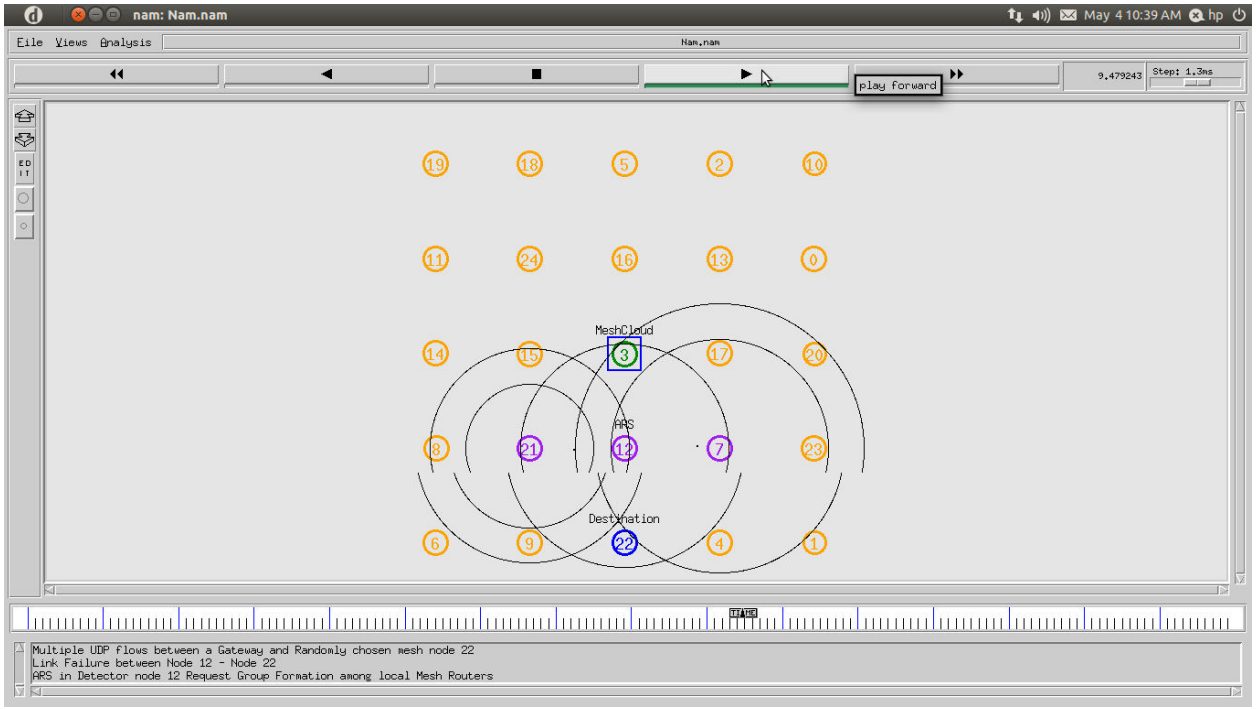


Fig 8.6(b). Reconfiguration started

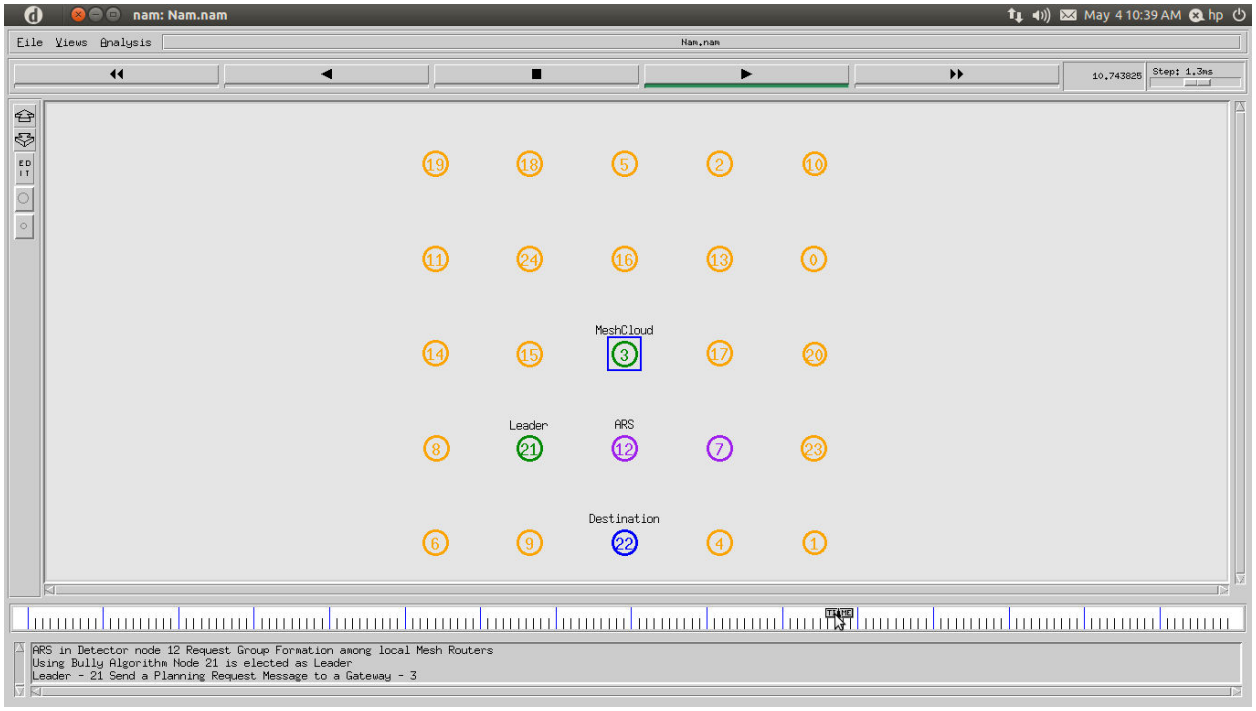


Fig 8.7(a). Leader Election

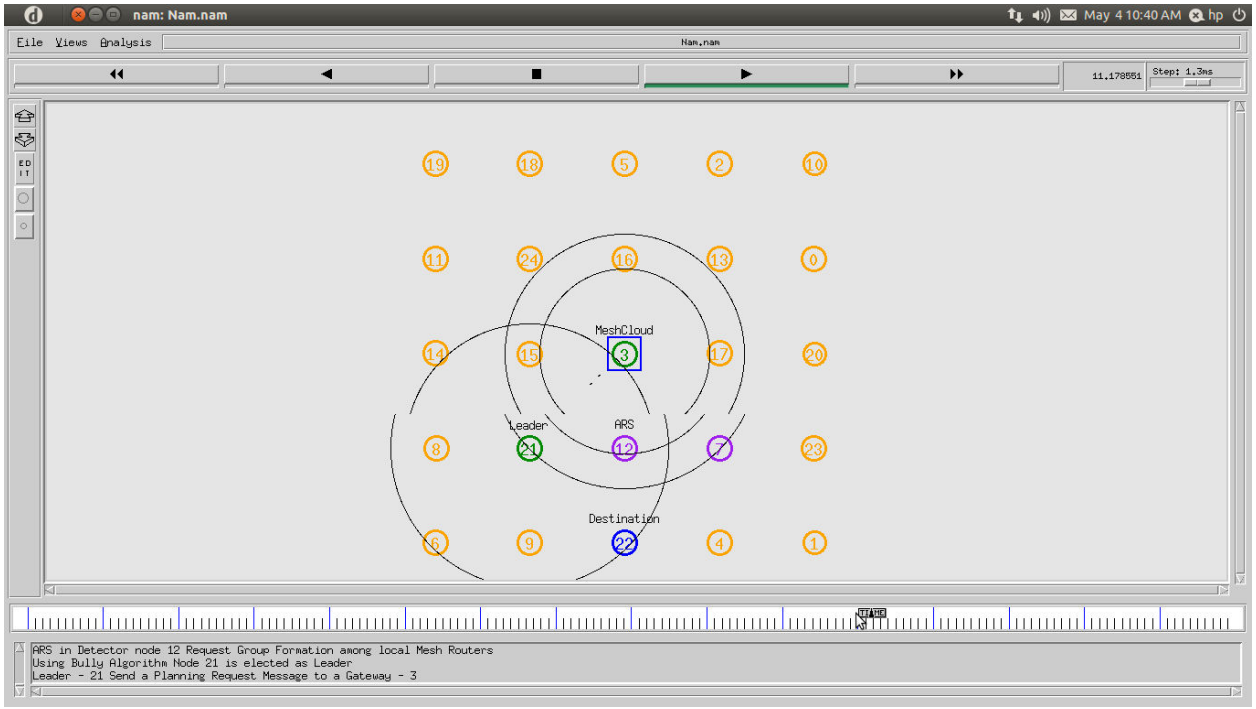


Fig 8.7(b). Leader Election

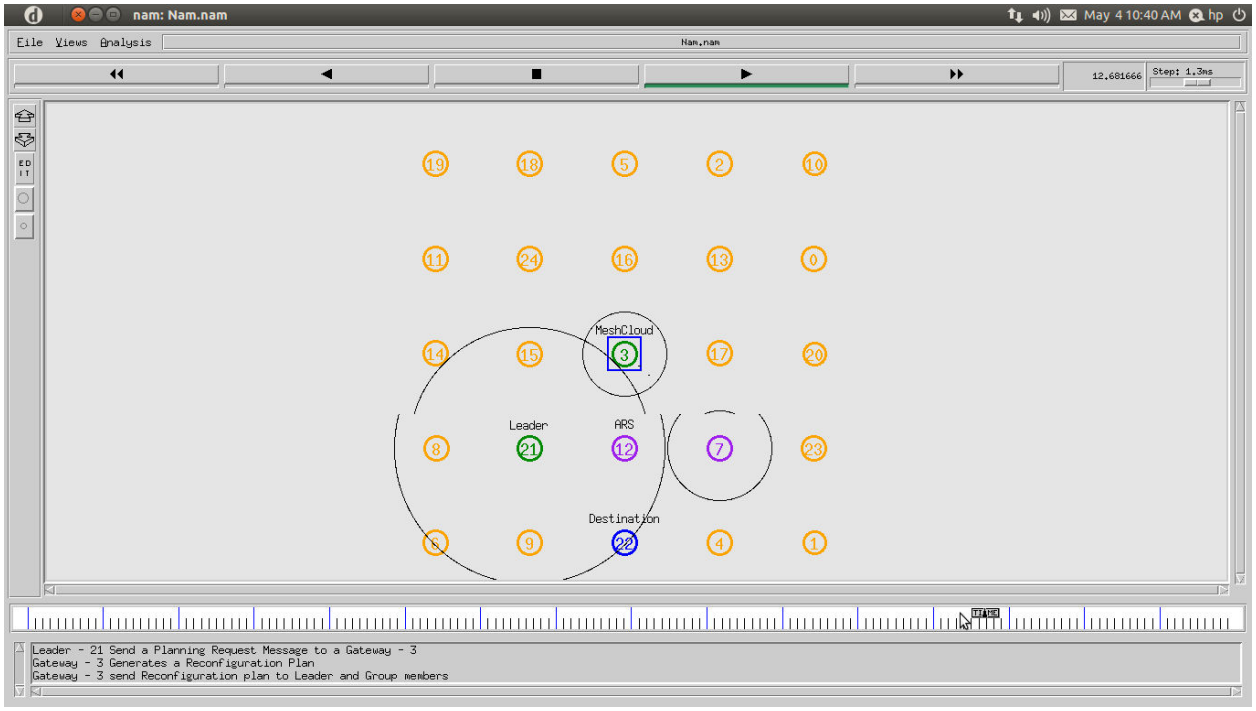


Fig 8.8. Broadcasting

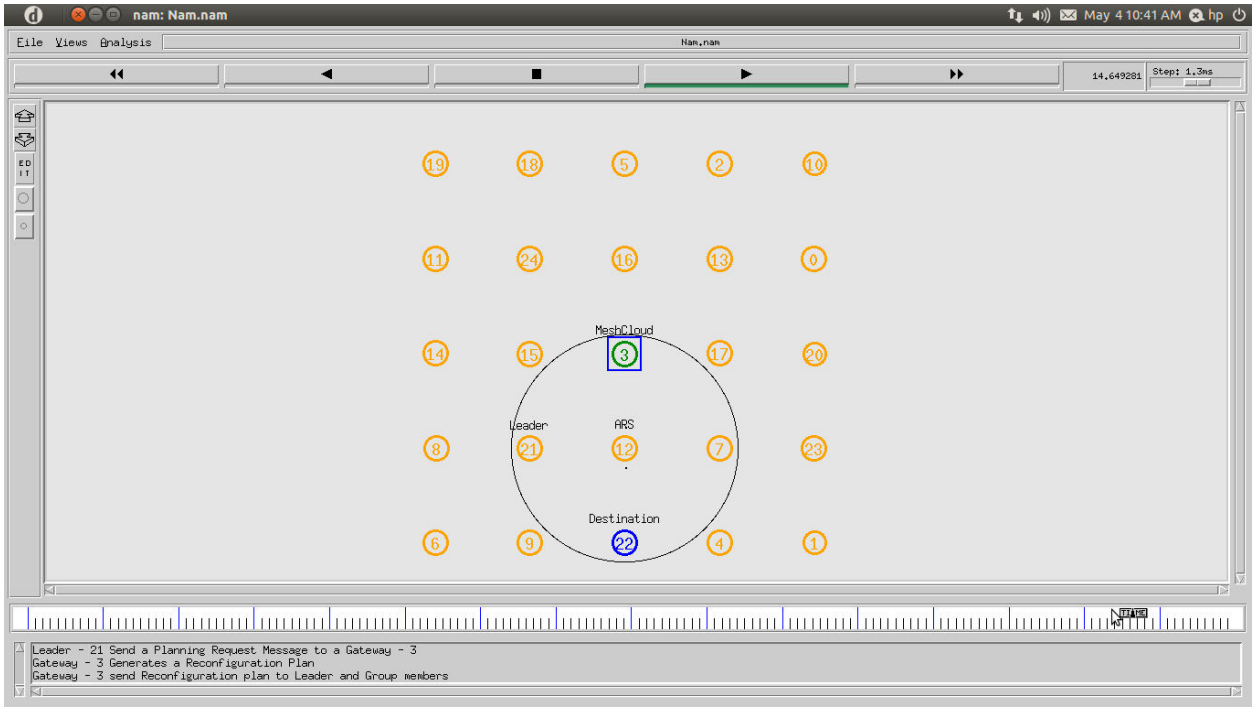


Fig8.9(a). Re-establishment of Connection

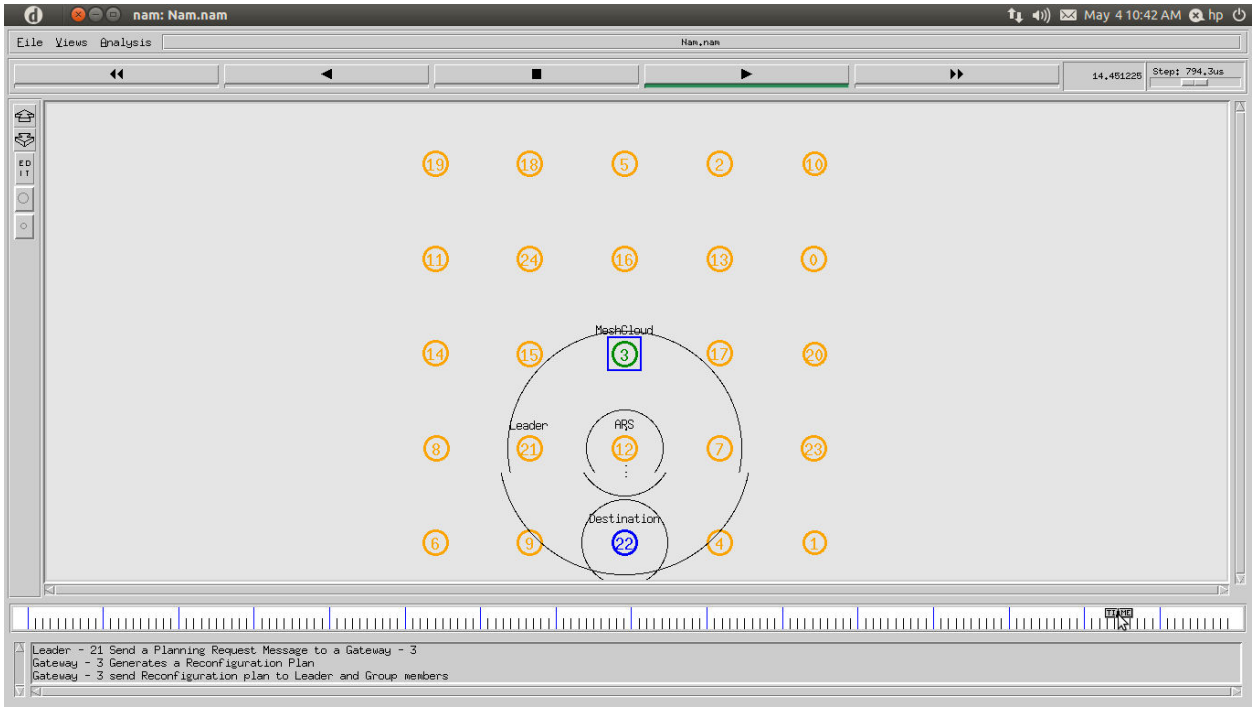


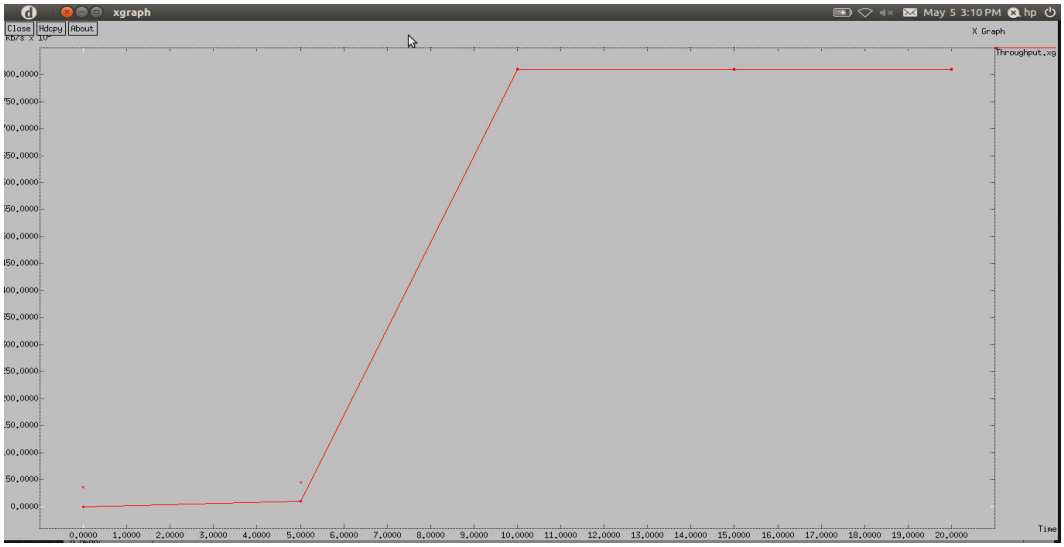
Fig8.9(b). Re-establishment of Connection

CHAPTER 9

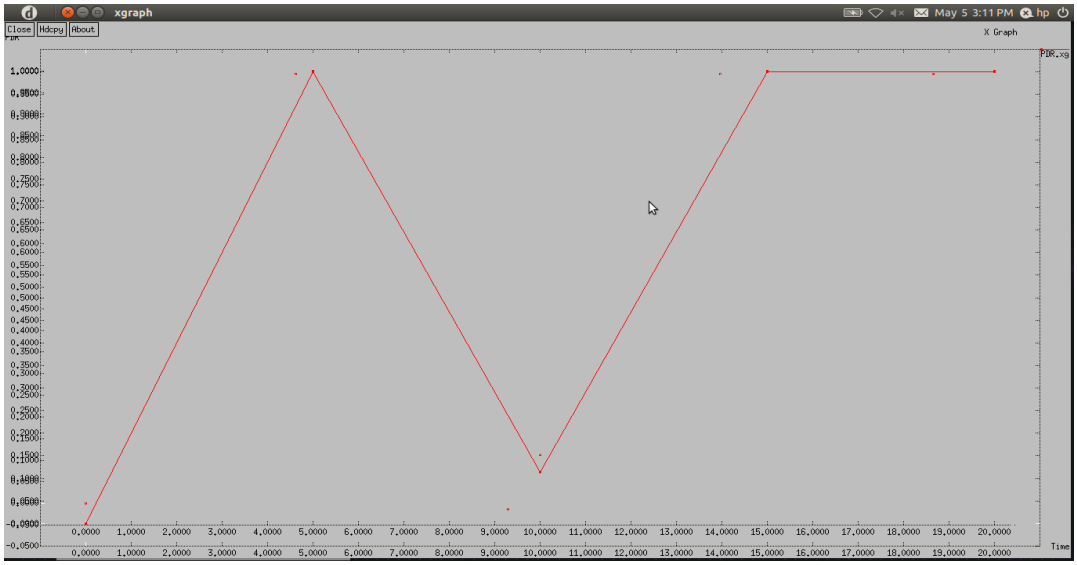
1. PERFORMANCE EVALUATION

1.1. EXISTING METHODOLOGY

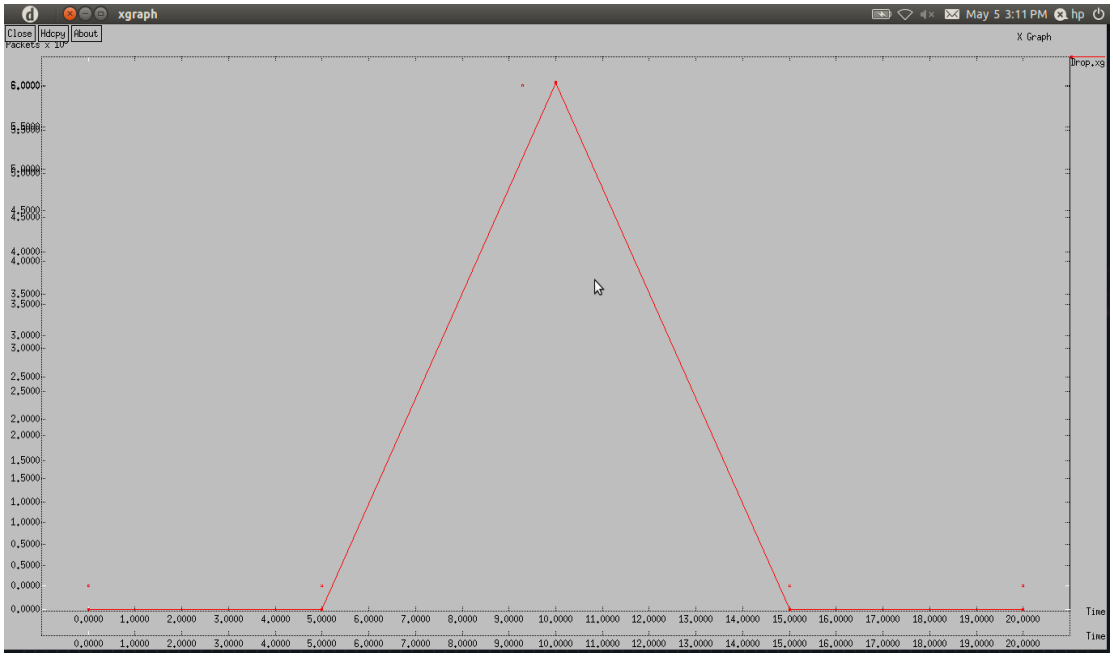
1.1.1. Throughput



1.1.2. Packet Ratio

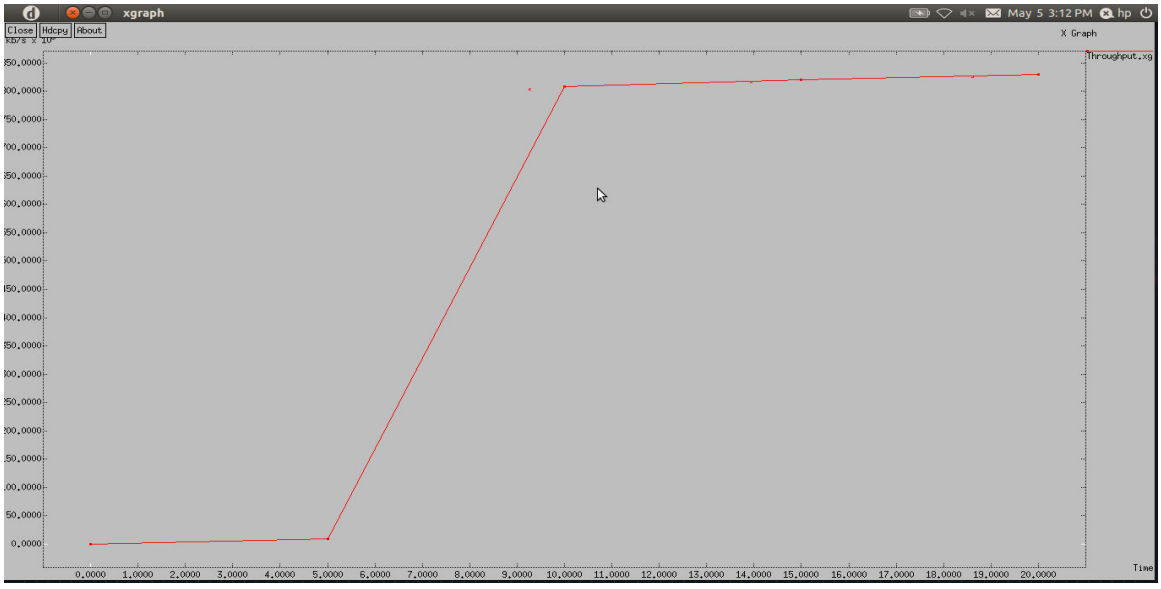


1.1.3. Packet Drop Ratio

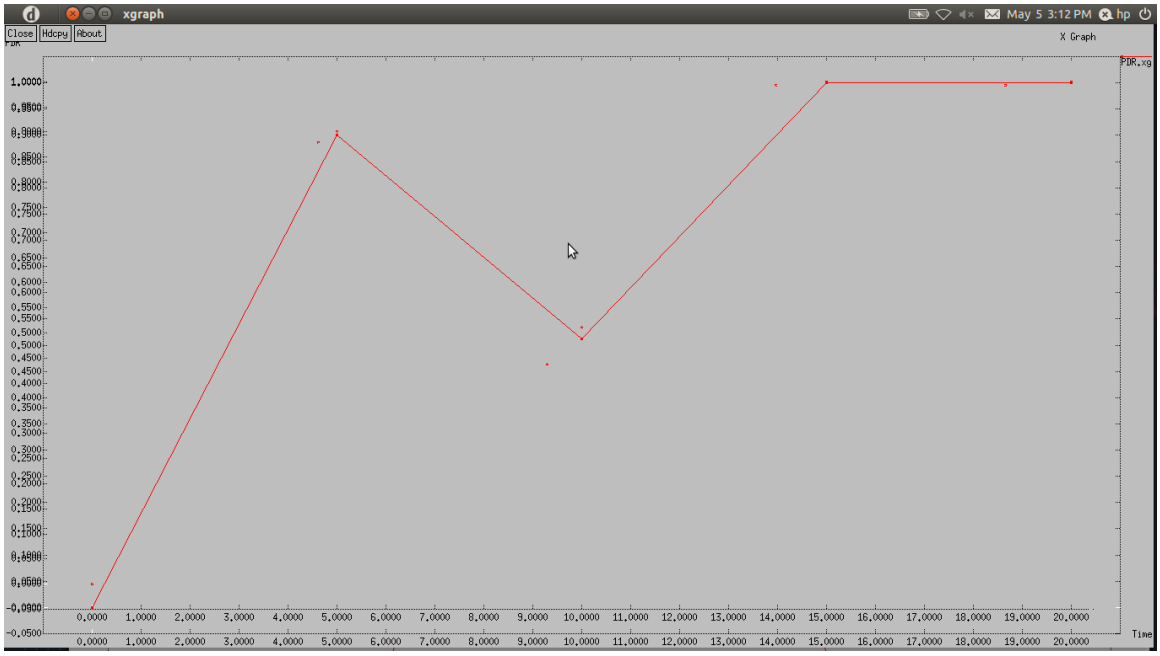


1.2. PROPOSED METHODOLOGY

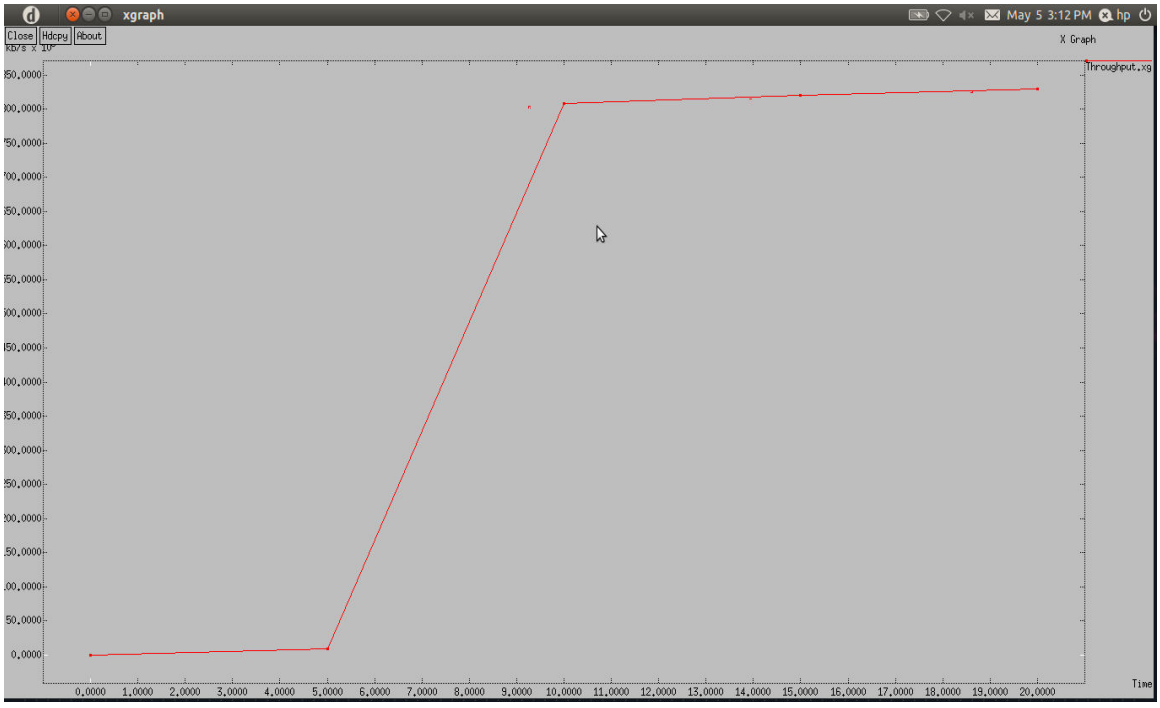
1.2.1. Throughput



1.2.3. Packet Ratio



1.2.4. Drop Ratio



CHAPTER 10

1. CONCLUSION

In this report, I presented an ARS that a WMN to recover from wireless link failures. It generates an effective reconfiguration plan which requires only local network changes by using radio and path diversity.

ARS effectively identifies reconfiguration plans that satisfy QoS constraints, admitting up to two times more flows than static assignment, through QoS aware planning. Next, ARS allows for real-time failure detection and network reconfiguration, thus improving channel efficiency. This experimental evaluation on a Linux-based implementation and ns2-based simulation has demonstrated the effectiveness of ARS in recovering from local link-failures and in satisfying applications.

2. FUTURE WORK

i) Flow Assignment along with Routing : This (Autonomous Reconfiguration Network System) protocol disassociate network reconfiguration from flow assignment and routing. This might be more efficient if these two issues are considered together.

ii) This algorithm works only for 2-D meshes. This can also be implemented for n-Dimensional meshes.

CHAPTER 11

1. REFERENCES

[1]. F. Kaabi,S. Ghannay, and F. Filali "Channel Allocation and Routing in Wireless Mesh Networks: A survey and qualitative comparison between schemes" International Journal of Wireless and Mobile Networks(IJWMN),Vol.2,No.1,February 2010.

[2]. Kyu-Han Kim, *Member, IEEE*, and Kang G. Shin, *Fellow, IEEE, ACM "*Self-Reconfigurable Wireless Mesh Networks*"* IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 19, NO. 2, APRIL 2011.

[3]. Xiaoqin Chen, Haley M. Jones, and Dhammika Jayalath, Senior Member, IEEE, "Channel-Aware Routing in MANETs with Route Handoff" IEEE TRANSACTIONS ON MOBILE COMPUTING, VOL. 10, NO. 1, JANUARY 2011.

[4]. Majid Ghaderi, Member, IEEE, Ashwin Sridharan, Member, IEEE, Hui Zang, Senior Member, IEEE,Don Towsley, Fellow, IEEE, and Rene Cruz, Fellow, IEEE "TCP-Aware Channel Allocation in CDMA Networks" IEEE TRANSACTIONS ON MOBILE COMPUTING, VOL. 8, NO. 1, JANUARY 2009.

[5]. I. Akyildiz, X. Wang, and W. Wang, "Wireless mesh networks: A survey," *Comput. Netw.*, vol. 47, no. 4, pp. 445–487, Mar. 2005.

[6]. P. Kyasanur and N. Vaidya, "Capacity of multi-channel wireless networks: Impact of number of channels and interfaces," in *Proc. ACM MobiCom*, Cologne, Germany, Aug. 2005, pp. 43–57.

[7]. A. Brzezinski, G. Zussman, and E. Modiano, "Enabling distributed throughput maximization in wireless mesh networks: A partitioning approach," in *Proc. ACM MobiCom*, Los Angeles, CA, Sep. 2006, pp.26–37.

[8]. A. Raniwala and T. Chiueh, "Architecture and algorithms for an IEEE 802.11-based multi-channel wireless mesh network," in *Proc. IEEE INFOCOM*, Miami, FL, Mar. 2005, vol. 3.

[9]. A. P. Subramanian, H. Gupta, S. R. Das, and J. Cao, "Minimum interference channel assignment in multiradio wireless mesh networks," *IEEE Trans. Mobile Comput.*, vol. 7, no. 12, pp. 1459–1473, Dec. 2008.

APPENDICES

1. CONNECTION ESTABLISHMENT

1.1. TCL SCRIPT

```
#..... Environmental Settings .....

set Nn          25              ;# Number of Mobilenodes
set val(x)      710             ;# X Co-ordinate
set val(y)      710             ;# Y Co-ordinate
set chan        [new Channel/WirelessChannel] ;# Chaneel Type

#..... Simulator Object Creation .....

set nso [new Simulator]

#..... Trace File to record all the Events .....

set fpt [open Trace.tr w]
$nso trace-all $fpt
$nso use-newtrace

#..... NAM Window creation .....

set fpn [open Nam.nam w]
$nso namtrace-all-wireless $fpn $val(x) $val(y)

#..... Topology Creation .....

set tpy [new Topography]
$tpy load_flatgrid $val(x) $val(y)
```

```
#..... General Operational Director .....

create-god $Nn

#..... Node Configuration .....

$nso node-config -adhocRouting DSR \
                 -llType LL \
                 -macType Mac/802_11 \
                 -ifqType CMUPriQueue \
                 -ifqLen 50 \
                 -antType Antenna/OmniAntenna \
                 -propType Propagation/TwoRayGround \
                 -phyType Phy/WirelessPhy \
                 -topoInstance $tpy \
                 -channel $chan \
                 -agentTrace ON \
                 -routerTrace ON \
                 -macTrace ON \
                 -movementTrace ON \
                 -idlePower 1.2 \
                 -rxPower 1.0 \
                 -txPower 1.5 \
                 -sleepPower 0.000015 \
                 -initialEnergy 200 \
                 -energyModel EnergyModel

#..... Node Creation .....

for { set i 0 } { $i < $Nn } { incr i } {
```

```
                    set node($i) [$nso node]
                    $nso initial_node_pos $node($i) 40
                    $node($i) color black
                }

#..................................... Agent Creation .....................................

proc Connecting-Agent { src sink pkt itv } {
                        global nso
                        #--- Source Agent ---
                        set udp [new Agent/UDP]
                        $nso attach-agent $src $udp
                        #- Create a Traffic Apllication -
                        set cbr [new Application/Traffic/CBR]
                        $cbr attach-agent $udp
                        $cbr set packetSize_ $pkt    ;# Set Packet size
                        $cbr set interval_ $itv      ;# Set Intervel

                        #-- Attach CBR source to sink --
                        $nso connect $udp $sink
                        return $cbr
                    }


#..................................... Destination Agent .....................................

for { set i 0 } { $i<$Nn } { incr i  }  {
                    set sink($i) [new Agent/LossMonitor]
                    $nso attach-agent $node($i) $sink($i)
                   }
```

```
#..... Neighbors Distance Calculation .....

set GN -1
proc Distance {  } {
                    global node nso Nn GN
                    set fpd [open "Distance.Cal" w]
                    puts $fpd "------------------------------------------------------------------------"
                    puts $fpd "Node\t\tNeighbors\tx-cor\ty-cor \tDistance"
                    puts $fpd "------------------------------------------------------------------------"
                for { set i 0 } { $i <$Nn } {  incr i } {
                                                    set tn 0
                                                    set x1 [$node($i) set X_]
                                                    set y1 [$node($i) set Y_]
                                                    if { [expr int($x1)]== 322 && [expr
int($y1)]== 322 } { set GN $i }
                        for { set j 0 } { $j <$Nn } { incr j } {
                                                    set x2 [$node($j) set X_]
                                                    set y2 [$node($j) set Y_]
                                                    set dis [expr int(sqrt(pow([expr $x2 -
$x1],2)+pow([expr $y2 - $y1],2)))]
                                        if { $dis <240 && $i!= $j } { incr tn
                                                    puts $fpd "$i\t\t $j\t\t[expr int($x1)]\t[expr
int($y1)]\t$dis"  }  }
                                                    puts $fpd "Total_Neighbors $i = $tn"
                                                   }
                    close $fpd
                }

#..... Node Deployment .....
```

```
set dyn(0) -1

proc Deploy { tm ntm upt } {
                        global nso node Nn dyn
                      if { $ntm==0 } {
                        #------------- Random Value Generate -------------
                        if { $upt==1} {   #------ Execute number of time --------
                                  for { set i 0 } { $i<$Nn } { incr i } {  set flg 1
                              #----- Repeat execuation-----
                         while { $flg } {
                                                set chk 1
                                                set val [expr int(rand()*$Nn)] ; #--- Random
value Generate ----
                                                #----- Check Duplicate Value --------
                                                for { set j 0 } { $j<$i } { incr j } { if {
$val==$dyn($j) } { set chk 0 } }
                                                #----- Store In Array ---------------
                                                if { $chk==1 } {  set dyn($i) $val  ;  set flg 0  }
                                              } }   }

                        $nso at $tm  "$node($dyn(0))  setdest 001.23 001.80 4000"
                        $nso at $tm  "$node($dyn(1))  setdest 161.21 001.31 4000"
                        $nso at $tm  "$node($dyn(2))  setdest 322.29 001.41 4000"
                        $nso at $tm  "$node($dyn(3))  setdest 483.97 001.73 4000"
                        $nso at $tm  "$node($dyn(4))  setdest 644.86 001.31 4000"
                        $nso at $tm  "$node($dyn(5))  setdest 001.75 161.17 4000"
                        $nso at $tm  "$node($dyn(6))  setdest 161.01 161.22 4000"
                        $nso at $tm  "$node($dyn(7))  setdest 322.56 161.74 4000"
                        $nso at $tm  "$node($dyn(8))  setdest 483.71 161.06 4000"
                        $nso at $tm  "$node($dyn(9))  setdest 644.81 161.85 4000"
                        $nso at $tm  "$node($dyn(10)) setdest 001.23 322.80 4000"
```

```
                $nso at $tm  "$node($dyn(11)) setdest 161.21 322.31 4000"
                $nso at $tm  "$node($dyn(12)) setdest 322.29 322.41 4000"
                $nso at $tm  "$node($dyn(13)) setdest 483.97 322.73 4000"
                $nso at $tm  "$node($dyn(14)) setdest 644.86 322.31 4000"
                $nso at $tm  "$node($dyn(15)) setdest 001.07 483.17 4000"
                $nso at $tm  "$node($dyn(16)) setdest 161.07 483.22 4000"
                $nso at $tm  "$node($dyn(17)) setdest 322.16 483.74 4000"
                $nso at $tm  "$node($dyn(18)) setdest 483.81 483.06 4000"
                $nso at $tm  "$node($dyn(19)) setdest 644.42 483.85 4000"
                $nso at $tm  "$node($dyn(20)) setdest 001.23 644.80 4000"
                $nso at $tm  "$node($dyn(21)) setdest 161.21 644.31 4000"
                $nso at $tm  "$node($dyn(22)) setdest 322.29 644.41 4000"
                $nso at $tm  "$node($dyn(23)) setdest 483.97 644.73 4000"
                $nso at $tm  "$node($dyn(24)) setdest 644.86 644.31 4000"
        }
          }

$nso at 0.05 "Deploy 0.05 0 1"
$nso at 0.50 "Deploy 0.60 0 0"

$nso at 0.7 "Distance"

for { set i 0 } { $i < $Nn } { incr i } { $nso at 0.0 "$node($i) label-color black" }

#..................................... Gateway Communication .....................

proc Attribute { tm } {
                global nso node GN Nn dyn sink
                $nso at $tm  "$node($GN) color green4"
                $nso at $tm  "$node($GN) label MeshCloud"
                $nso at $tm  "$node($GN) add-mark c1 blue square"
```

```
                for { set i 0 } { $i < $Nn } { incr i } {
                        if { $dyn($i)!= $GN } {
                                                set tmp [ open btemp w]
                                                puts $tmp "$dyn($i) $GN $tm 0.1 0"
                                                close $tmp
                                                exec awk -f Hop.awk btemp Distance.Cal
                                                source Hop_count.tcl
                                                set tm [expr $tm+0.1]
                                               }  }
        }

$nso at 1.0  "Attribute 1.0"

proc Link_Failure { tm } {
                        global GN node nso sink
                        set flg 1
                        while { $flg==1 } { set dest [expr int(rand()*25)]
                                        if { $dest!=$GN } { incr flg } }
                        set tmp [ open btemp w]
                        puts $tmp "$GN $dest $tm 8.0 1"
                        close $tmp
                        exec awk -f Hop.awk btemp Distance.Cal
                        source Hop_count.tcl
                       }

$nso at 3.8 "Link_Failure 4.0 "

#..................................... Execution NAM Window .............................

proc Stop { } {
                global nso fpn node GN
```

```
        $nso flush-trace
        close $fpn
        exec nam -r 5m Nam.nam &
        exit 0
    }


$nso at 25.0 "Stop"
$nso run
```

2. EXISTING METHODOLOGY

2.1. TCL SCRIPT

```
 #..... Environmental Settings .....

 set Nn          25                  ;# Number of Mobilenodes
 set val(x)      710                     ;# X Co-ordinate
 set val(y)      710                     ;# Y Co-ordinate
 set chan          [new Channel/WirelessChannel] ;# Chaneel Type

#..... Simulator Object Creation .....

set nso [new Simulator]

#..... Trace File to record all the Events .....

set fpt [open Trace.tr w]
$nso trace-all $fpt
$nso use-newtrace

#..... NAM Window creation .....
```

```
set fpn [open Nam.nam w]
$nso namtrace-all-wireless $fpn $val(x) $val(y)

#..................................... Topology Creation .....................................
set tpy [new Topography]
$tpy load_flatgrid $val(x) $val(y)

#.............................. General Operational Director ..............................

create-god $Nn

#..................................... Node Configuration ..............................

$nso node-config -adhocRouting DSR \
                 -llType LL \
                 -macType Mac/802_11 \
                 -ifqType CMUPriQueue \
                 -ifqLen 50 \
                 -antType Antenna/OmniAntenna \
                 -propType Propagation/TwoRayGround \
                 -phyType Phy/WirelessPhy \
                 -topoInstance $tpy \
                 -channel $chan \
                 -agentTrace ON \
                 -routerTrace ON \
                 -macTrace ON \
                 -movementTrace ON \
                 -idlePower 1.2 \
                 -rxPower 1.0 \
                 -txPower 1.5 \
```

```
                -sleepPower 0.000015 \
                -initialEnergy 200 \
                -energyModel EnergyModel

#..................................... Node Creation .....................
for { set i 0 } { $i < $Nn } { incr i } {
                                        set node($i) [$nso node]
                                        $nso initial_node_pos $node($i) 40
                                        $node($i) color black
                                    }

#................................. Agent Creation ...................................

proc Connecting-Agent { src sink pkt itv } {
                                                global nso
                                                #--- Source Agent ---
                                                set udp [new Agent/UDP]
                                                $nso attach-agent $src $udp
                                                #- Create a Traffic Apllication -
                                                set cbr [new Application/Traffic/CBR]
                                                $cbr attach-agent $udp
                                                $cbr set packetSize_ $pkt    ;# Set Packet size
                                                $cbr set interval_ $itv      ;# Set Intervel

                                                #-- Attach CBR source to sink --
                                                $nso connect $udp $sink
                                                return $cbr
                                            }

#................................... Destination Agent ..............................
```

```
for { set i 0 } { $i<$Nn } { incr i  }  {
                                        set sink($i) [new Agent/LossMonitor]
                                        $nso attach-agent $node($i) $sink($i)
                                       }

#..................................... Neighbors Distance Calculation .....................

set GN -1
proc Distance {   } {
                        global node nso Nn GN
                        set fpd [open "Distance.Cal" w]
                        puts $fpd "-------------------------------------------------------------------------"
                        puts $fpd "Node\t\tNeighbors\tx-cor\ty-cor \tDistance"
                        puts $fpd "-------------------------------------------------------------------------"
                    for { set i 0 } { $i <$Nn } {  incr i } {
                                                        set tn 0
                                                        set x1 [$node($i) set X_]
                                                        set y1 [$node($i) set Y_]
                                                        if { [expr int($x1)]== 322 && [expr
int($y1)]== 322 } { set GN $i }
                                for { set j 0 } { $j <$Nn } { incr j } {
                                                        set x2 [$node($j) set X_]
                                                        set y2 [$node($j) set Y_]
                                                        set dis [expr int(sqrt(pow([expr $x2 -
$x1],2)+pow([expr $y2 - $y1],2)))]
                                        if { $dis <240 && $i!= $j } { incr tn
                                                        puts $fpd "$i\t\t $j\t\t[expr int($x1)]\t[expr
int($y1)]\t$dis"   }  }
                                                        puts $fpd "Total_Neighbors $i = $tn"
                                                       }
                        close $fpd
```

```
}

#..................................... Node Deployment .....................................

set dyn(0) -1

proc Deploy { tm } {
                                global nso node Nn dyn
                                $nso at $tm  "$node(0)  setdest 001.23 001.80 4000"
                                $nso at $tm  "$node(1)  setdest 161.21 001.31 4000"
                                $nso at $tm  "$node(2)  setdest 322.29 001.41 4000"
                                $nso at $tm  "$node(3)  setdest 483.97 001.73 4000"
                                $nso at $tm  "$node(4)  setdest 644.86 001.31 4000"
                                $nso at $tm  "$node(5)  setdest 001.75 161.17 4000"
                                $nso at $tm  "$node(6)  setdest 161.01 161.22 4000"
                                $nso at $tm  "$node(7)  setdest 322.56 161.74 4000"
                                $nso at $tm  "$node(8)  setdest 483.71 161.06 4000"
                                $nso at $tm  "$node(9)  setdest 644.81 161.85 4000"
                                $nso at $tm  "$node(10) setdest 001.23 322.80 4000"
                                $nso at $tm  "$node(11) setdest 161.21 322.31 4000"
                                $nso at $tm  "$node(12) setdest 322.29 322.41 4000"
                                $nso at $tm  "$node(13) setdest 483.97 322.73 4000"
                                $nso at $tm  "$node(14) setdest 644.86 322.31 4000"
                                $nso at $tm  "$node(15) setdest 001.07 483.17 4000"
                                $nso at $tm  "$node(16) setdest 161.07 483.22 4000"
                                $nso at $tm  "$node(17) setdest 322.16 483.74 4000"
                                $nso at $tm  "$node(18) setdest 483.81 483.06 4000"
                                $nso at $tm  "$node(19) setdest 644.42 483.85 4000"
                                $nso at $tm  "$node(20) setdest 001.23 644.80 4000"
                                $nso at $tm  "$node(21) setdest 161.21 644.31 4000"
                                $nso at $tm  "$node(22) setdest 322.29 644.41 4000"
```

```
                $nso at $tm  "$node(23) setdest 483.97 644.73 4000"
                $nso at $tm  "$node(24) setdest 644.86 644.31 4000"
        }

$nso at 0.05 "Deploy 0.05"
$nso at 0.50 "Deploy 0.60"

$nso at 0.7 "Distance"

$nso at 1.0  "$node(12) color green4"
$nso at 1.0  "$node(12) label MeshCloud"
$nso at 1.0  "$node(12) add-mark c1 blue square"



proc Link_Failure { tm } {
                global GN node nso sink
                set flg 1
                while { $flg==1 } { set dest [expr int(rand()*25)]
                                if { $dest!=$GN } { incr flg } }
                set tmp [ open btemp w]
                puts $tmp "$GN $dest $tm 5.0 1"
                close $tmp
                set tmp [ open Dest w]
                puts $tmp "$dest"
                close $tmp
                exec awk -f Hop.awk btemp Distance.Cal
                source Hop_count.tcl
               }

$nso at 3.8 "Link_Failure 4.0 "
```

```
#..................................... For Graph .. .....................................

set pr [open PDR.xg w]
puts $pr "LabelFont: Monospace"
puts $pr "TitleFont: Monospace"
puts $pr "Geometry: 1280x800"
puts $pr "Background: grey"
puts $pr "BoundBox: true"
puts $pr "NoLines: false"
puts $pr "Markers: true"
puts $pr "Ticks: true"
puts $pr "0 0"

set tp [open Throughput.xg w]
puts $tp "LabelFont: Monospace"
puts $tp "TitleFont: Monospace"
puts $tp "Geometry: 1280x800"
puts $tp "Background: grey"
puts $tp "BoundBox: true"
puts $tp "NoLines: false"
puts $tp "Markers: true"
puts $tp "Ticks: true"
puts $tp "0 0"

set dr [open Drop.xg w]
puts $dr "LabelFont: Monospace"
puts $dr "TitleFont: Monospace"
puts $dr "Geometry: 1280x800"
puts $dr "Background: grey"
puts $dr "BoundBox: true"
```

```
puts $dr "NoLines: false"
puts $dr "Markers: true"
puts $dr "Ticks: true"
puts $dr "0 0"

proc Graph { snk } {
                global nso pr tp dr sink
                set nw [$nso now]
                set tmp [open Dest r]
                if { $nw<6 } {
                set snk [gets $tmp]
                close $tmp }
                set rec [$sink($snk) set npkts_]
                set los [$sink($snk) set nlost_]
                set byt [$sink($snk) set bytes_]

                set kb [expr double(($byt*8.0)/(2))]
                set pd [expr $rec/($rec+$los+0.0)]

                $sink($snk) set nlost_ 0
                puts $pr "$nw $pd"
                puts $tp "$nw $kb"
                puts $dr "$nw $los"
                $nso at [expr $nw+5.0] "Graph $snk"
                 }

$nso at 5.0 "Graph 0"

#..................................... Execution NAM Window .............................

proc Stop { } {
```

```
        global nso fpn node GN
        global pr tp dr
        close $tp ; close $pr ; close $dr
        $nso flush-trace
        close $fpn
        exec nam -r 5m Nam.nam &
        exec xgraph Throughput.xg -x "Time" -y  "kb/s" &
        exec xgraph PDR.xg -x "Time" -y  "PDR" &
        exec xgraph Drop.xg -x "Time" -y  "Packets" &
        exit 0
    }


$nso at 25.0 "Stop"
$nso run
```

2.2. AWK SCRIPT

```
BEGIN {
        i=0
        pks=256
        itv=0.1
    }

    {
         if(FILENAME=="btemp")
         {
          src=$1
          des=$2
          tm=$3
          itval=$4
          flg=$5
```

```
        }

        if(FILENAME=="Distance.Cal")
          if(i==$1)
          {
           s[i,1]=$1
           s[i,2]=$3
           s[i++,3]=$4
          }
    }

END   {
        GN=src
        min=2500
        k=0
        desx=s[des,2]
        desy=s[des,3]

        a[0]=src
        for(hct=0;a[hct]!=des;hct++)
        {
          src=a[hct] # Alternative src node
          k=k+1
          srcx=s[src,2]
          srcy=s[src,3]
          for(j=0;j<i;j++) # find all node Distances
          {
            if(src!=j)
        {
                x=s[j,2]
                y=s[j,3]
```

```
                srcd=int(sqrt(((x-srcx)^2)+((y-srcy)^2)))
                desd=int(sqrt(((x-desx)^2)+((y-desy)^2)))

                if(srcd<=250 && desd<min) # Check high dist from src and low dis from des
                {
                  min=desd
                  a[k]=j
                }
              }
             }
            }

 #..... Display HOP nodes from src to des .....

 tm=tm+0.05
 print "set inf0 [Connecting-Agent $node("a[0]") $sink("a[q+1]") "pks" "itv"]" > "Hop_count.tcl"
 print "$nso at "tm" \"$inf0 start\"" > "Hop_count.tcl"
 print "$nso at "tm+itval" \"$inf0 stop\"" > "Hop_count.tcl"

 if(flg==0)
 {
  print "$nso at "tm-0.05" \"$node("a[0]") color red\"" > "Hop_count.tcl"
  print "$nso at "tm-0.05" \"$nso trace-annotate \\\"Node "a[0]" Measure the Link Quality using
Passive Monitoring\\\"\"" > "Hop_count.tcl"
  print "$nso at "tm+itval-0.05" \"$node("a[0]") color orange\"" > "Hop_count.tcl"
  print "$nso at "tm+itval-0.05" \"$nso trace-annotate \\\"Node "a[0]" Send Monitoring Results
to Gateway \\\"\"" > "Hop_count.tcl"
 }

 if(flg==1)
 {
```

```
    print "$nso at "tm" \"$node("a[hct]") color blue\"" > "Hop_count.tcl"
    print "$nso at "tm" \"$node("a[hct]") label Destination\"" > "Hop_count.tcl"
    print "$nso at "tm" \"$nso trace-annotate \\\"Multiple UDP flows between a Gateway and
Randomly chosen mesh node "des"\\\"\"" > "Hop_count.tcl"
  }

 tm=tm+0.01

 for(q=1;q<hct;q++)
 {
  print "set inf"q" [Connecting-Agent $node("a[q]") $sink("a[q+1]") "pks" "itv"]" >
"Hop_count.tcl"
  print "$nso at "tm" \"$inf"q" start\"" > "Hop_count.tcl"
  print "$nso at "tm+itval" \"$inf"q" stop\"" > "Hop_count.tcl"
  tm=tm+0.01
 }

 if(flg==1)
 {
  print "set inff"q" [Connecting-Agent $node("a[hct-1]") $sink("a[hct]") "pks" 0.002]" >
"Hop_count.tcl"
  print "$nso at "tm+int(tm/2) " \"$inff"q" start\"" > "Hop_count.tcl"
  print "$nso at "tm+itval" \"$inff"q" stop\"" > "Hop_count.tcl"
  print "$nso at "tm+int(tm/2)" \"$nso trace-annotate \\\"Link Failure between Node "a[hct-1]" -
Node "a[hct]" \\\"\"" > "Hop_count.tcl"
 }
}
```

3. PROPOSED METHODOLOGY

3.1. TCL SCRIPT

```
#..... Environmental Settings .....

set Nn          25                  ;# Number of Mobilenodes
set val(x)      710                 ;# X Co-ordinate
set val(y)      710                 ;# Y Co-ordinate
set chan        [new Channel/WirelessChannel] ;# Chaneel Type

#..... Simulator Object Creation .....

set nso [new Simulator]

#..... Trace File to record all the Events .....

set fpt [open Trace.tr w]
$nso trace-all $fpt
$nso use-newtrace

#..... NAM Window creation .....

set fpn [open Nam.nam w]
$nso namtrace-all-wireless $fpn $val(x) $val(y)

#..... Topology Creation .....

set tpy [new Topography]
$tpy load_flatgrid $val(x) $val(y)

#..... General Operational Director .....

create-god $Nn
```

```
#..................................... Node Configuration .....................

$nso node-config -adhocRouting DSR \
                 -llType LL \
                 -macType Mac/802_11 \
                 -ifqType CMUPriQueue \
                 -ifqLen 50 \
                 -antType Antenna/OmniAntenna \
                 -propType Propagation/TwoRayGround \
                 -phyType Phy/WirelessPhy \
                 -topoInstance $tpy \
                 -channel $chan \
                 -agentTrace ON \
                 -routerTrace ON \
                 -macTrace ON \
                 -movementTrace ON \
                 -idlePower 1.2 \
                 -rxPower 1.0 \
                 -txPower 1.5 \
                 -sleepPower 0.000015 \
                 -initialEnergy 200 \
                 -energyModel EnergyModel

#..................................... Node Creation .....................

for { set i 0 } { $i < $Nn } { incr i } {
                                        set node($i) [$nso node]
                                        $nso initial_node_pos $node($i) 40
                                        $node($i) color black
                                    }
```

```
#..................................... Agent Creation .....................................

proc Connecting-Agent { src sink pkt itv } {
                                                global nso
                                                #--- Source Agent ---
                                                set udp [new Agent/UDP]
                                                $nso attach-agent $src $udp
                                                #- Create a Traffic Apllication -
                                                set cbr [new Application/Traffic/CBR]
                                                $cbr attach-agent $udp
                                                $cbr set packetSize_ $pkt    ;# Set Packet size
                                                $cbr set interval_ $itv      ;# Set Intervel

                                                #-- Attach CBR source to sink --
                                                $nso connect $udp $sink
                                                return $cbr
                                          }

#..................................... Destination Agent .............................

for { set i 0 } { $i<$Nn } { incr i  }  {
                                              set sink($i) [new Agent/LossMonitor]
                                              $nso attach-agent $node($i) $sink($i)
                                            }

#................................ Neighbors Distance Calculation .....................

set GN -1
proc Distance {  } {
                        global node nso Nn GN
                        set fpd [open "Distance.Cal" w]
```

```
                puts $fpd "-----------------------------------------------------------------------"
                puts $fpd "Node\t\tNeighbors\tx-cor\ty-cor \tDistance"
                puts $fpd "-----------------------------------------------------------------------"
            for { set i 0 } { $i <$Nn } {  incr i } {
                                                set tn 0
                                                set x1 [$node($i) set X_]
                                                set y1 [$node($i) set Y_]
                                                if { [expr int($x1)]== 322 && [expr
int($y1)]== 322 } { set GN $i }
                        for { set j 0 } { $j <$Nn } { incr j } {
                                                set x2 [$node($j) set X_]
                                                set y2 [$node($j) set Y_]
                                                set dis [expr int(sqrt(pow([expr $x2 -
$x1],2)+pow([expr $y2 - $y1],2)))]
                                        if { $dis <240 && $i!= $j } { incr tn
                                                puts $fpd "$i\t\t $j\t\t[expr int($x1)]\t[expr
int($y1)]\t$dis"  }  }
                                                puts $fpd "Total_Neighbors $i = $tn"
                                                }
                close $fpd
            }

#..................................... Node Deployment .....................................

set dyn(0) -1

proc Deploy { tm ntm upt } {
                        global nso node Nn dyn
                    if { $ntm==0 } {
                        #------------- Random Value Generate -------------
                        if { $upt==1} {   #------ Execute number of time --------
```

```
                    for { set i 0 } { $i<$Nn } { incr i } {  set flg 1
                #----- Repeat execuation-----
            while { $flg } {
                                set chk 1
                                set val [expr int(rand()*$Nn)] ; #--- Random
value Generate ----
                                #----- Check Duplicate Value --------
                                for { set j 0 } { $j<$i } { incr j } { if {
$val==$dyn($j) } { set chk 0 } }
                                #----- Store In Array ---------------
                                if { $chk==1 } {  set dyn($i) $val  ;  set flg 0  }
                              } }   }

        $nso at $tm  "$node($dyn(0))  setdest 001.23 001.80 4000"
        $nso at $tm  "$node($dyn(1))  setdest 161.21 001.31 4000"
        $nso at $tm  "$node($dyn(2))  setdest 322.29 001.41 4000"
        $nso at $tm  "$node($dyn(3))  setdest 483.97 001.73 4000"
        $nso at $tm  "$node($dyn(4))  setdest 644.86 001.31 4000"
        $nso at $tm  "$node($dyn(5))  setdest 001.75 161.17 4000"
        $nso at $tm  "$node($dyn(6))  setdest 161.01 161.22 4000"
        $nso at $tm  "$node($dyn(7))  setdest 322.56 161.74 4000"
        $nso at $tm  "$node($dyn(8))  setdest 483.71 161.06 4000"
        $nso at $tm  "$node($dyn(9))  setdest 644.81 161.85 4000"
        $nso at $tm  "$node($dyn(10)) setdest 001.23 322.80 4000"
        $nso at $tm  "$node($dyn(11)) setdest 161.21 322.31 4000"
        $nso at $tm  "$node($dyn(12)) setdest 322.29 322.41 4000"
        $nso at $tm  "$node($dyn(13)) setdest 483.97 322.73 4000"
        $nso at $tm  "$node($dyn(14)) setdest 644.86 322.31 4000"
        $nso at $tm  "$node($dyn(15)) setdest 001.07 483.17 4000"
        $nso at $tm  "$node($dyn(16)) setdest 161.07 483.22 4000"
        $nso at $tm  "$node($dyn(17)) setdest 322.16 483.74 4000"
```

```
                $nso at $tm  "$node($dyn(18)) setdest 483.81 483.06 4000"
                $nso at $tm  "$node($dyn(19)) setdest 644.42 483.85 4000"
                $nso at $tm  "$node($dyn(20)) setdest 001.23 644.80 4000"
                $nso at $tm  "$node($dyn(21)) setdest 161.21 644.31 4000"
                $nso at $tm  "$node($dyn(22)) setdest 322.29 644.41 4000"
                $nso at $tm  "$node($dyn(23)) setdest 483.97 644.73 4000"
                $nso at $tm  "$node($dyn(24)) setdest 644.86 644.31 4000"
        }
          }

$nso at 0.05 "Deploy 0.05 0 1"
$nso at 0.50 "Deploy 0.60 0 0"

$nso at 0.7 "Distance"

for { set i 0 } { $i < $Nn } { incr i } { $nso at 0.0 "$node($i) label-color black" }

#..................................... Gateway Communication .....................

proc Attribute { tm } {
                global nso node GN Nn dyn sink
                $nso at $tm  "$node($GN) color green4"
                $nso at $tm  "$node($GN) label MeshCloud"
                $nso at $tm  "$node($GN) add-mark c1 blue square"
                for { set i 0 } { $i < $Nn } { incr i } {
                        if { $dyn($i)!= $GN } {
                                        set tmp [ open btemp w]
                                        puts $tmp "$dyn($i) $GN $tm 0.1 0"
                                        close $tmp
                                        exec awk -f Hop.awk btemp Distance.Cal
                                        source Hop_count.tcl
```

```
                                                    set tm [expr $tm+0.1]
                                                   }  }
                        }

$nso at 1.0  "Attribute 1.0"

proc Link_Failure { tm } {
                                global GN node nso sink
                                set flg 1
                                while { $flg==1 } { set dest [expr int(rand()*25)]
                                                    if { $dest!=$GN } { incr flg } }
                                set tmp [ open btemp w]
                                puts $tmp "$GN $dest $tm 5.0 1"
                                close $tmp
                                set tmp [ open Dest w]
                                puts $tmp "$dest"
                                close $tmp
                                exec awk -f Hop.awk btemp Distance.Cal
                                source Hop_count.tcl
                                exec awk -f ARS.awk ARS_tmp Distance.Cal
                                source ARS.tcl
                               }

$nso at 3.8 "Link_Failure 4.0 "

#..................................... For Graph .. .....................................

set pr [open PDR.xg w]
puts $pr "LabelFont: Monospace"
puts $pr "TitleFont: Monospace"
puts $pr "Geometry: 1280x800"
```

```
puts $pr "Background: grey"
puts $pr "BoundBox: true"
puts $pr "NoLines: false"
puts $pr "Markers: true"
puts $pr "Ticks: true"
puts $pr "0 0"

set tp [open Throughput.xg w]
puts $tp "LabelFont: Monospace"
puts $tp "TitleFont: Monospace"
puts $tp "Geometry: 1280x800"
puts $tp "Background: grey"
puts $tp "BoundBox: true"
puts $tp "NoLines: false"
puts $tp "Markers: true"
puts $tp "Ticks: true"
puts $tp "0 0"

set dr [open Drop.xg w]
puts $dr "LabelFont: Monospace"
puts $dr "TitleFont: Monospace"
puts $dr "Geometry: 1280x800"
puts $dr "Background: grey"
puts $dr "BoundBox: true"
puts $dr "NoLines: false"
puts $dr "Markers: true"
puts $dr "Ticks: true"
puts $dr "0 0"

proc Graph { snk } {
        global nso pr tp dr sink
```

```
                set nw [$nso now]
                set tmp [open Dest r]
                if { $nw<6 } {
                set snk [gets $tmp]
                close $tmp }
                set rec [$sink($snk) set npkts_]
                set los [$sink($snk) set nlost_]
                set byt [$sink($snk) set bytes_]

                set kb [expr double(($byt*8.0)/(2))]
                set pd [expr $rec/($rec+$los+0.0)]

                $sink($snk) set nlost_ 0
                puts $pr "$nw $pd"
                puts $tp "$nw $kb"
                puts $dr "$nw $los"
                $nso at [expr $nw+5.0] "Graph $snk"
                 }

$nso at 5.0 "Graph 0"

#..................................... Execution NAM Window .............................

proc Stop { } {
                global nso fpn node GN
                global pr tp dr
                close $tp ; close $pr ; close $dr
                $nso flush-trace
                close $fpn
                exec nam -r 5m Nam.nam &
                exec xgraph Throughput.xg -x "Time" -y  "kb/s" &
```

```
            exec xgraph PDR.xg -x "Time" -y  "PDR" &
            exec xgraph Drop.xg -x "Time" -y  "Packets" &
            exit 0
        }


$nso at 25.0 "Stop"
$nso run
```

3.2. AWK SCRIPT

```
BEGIN {
        i=0
        pks=256
        itv=0.1
    }

    {
         if(FILENAME=="btemp")
         {
          src=$1
          des=$2
          tm=$3
          itval=$4
          flg=$5
         }

         if(FILENAME=="Distance.Cal")
          if(i==$1)
          {
           s[i,1]=$1
           s[i,2]=$3
```

```
            s[i++,3]=$4
            }
    }

END   {
          GN=src
          min=2500
          k=0
          desx=s[des,2]
          desy=s[des,3]

          a[0]=src
          for(hct=0;a[hct]!=des;hct++)
          {
           src=a[hct] # Alternative src node
           k=k+1
           srcx=s[src,2]
           srcy=s[src,3]
           for(j=0;j<i;j++) # find all node Distances
           {
            if(src!=j)
          {
                  x=s[j,2]
                  y=s[j,3]
                  srcd=int(sqrt(((x-srcx)^2)+((y-srcy)^2)))
                  desd=int(sqrt(((x-desx)^2)+((y-desy)^2)))

                  if(srcd<=250 && desd<min) # Check high dist from src and low dis from des
                  {
                    min=desd
                    a[k]=j
```

```
                    }
                }
              }
            }

 #..... Display HOP nodes from src to des .....

 tm=tm+0.05
 print "set inf0 [Connecting-Agent $node("a[0]") $sink("a[q+1]") "pks" "itv"]" > "Hop_count.tcl"
 print "$nso at "tm" \"$inf0 start\"" > "Hop_count.tcl"
 print "$nso at "tm+itval" \"$inf0 stop\"" > "Hop_count.tcl"

 if(flg==0)
 {
   print "$nso at "tm-0.05" \"$node("a[0]") color red\"" > "Hop_count.tcl"
   print "$nso at "tm-0.05" \"$nso trace-annotate \\\"Node "a[0]" Measure the Link Quality using
Passive Monitoring\\\"\"" > "Hop_count.tcl"
   print "$nso at "tm+itval-0.05" \"$node("a[0]") color orange\"" > "Hop_count.tcl"
   print "$nso at "tm+itval-0.05" \"$nso trace-annotate \\\"Node "a[0]" Send Monitoring Results
to Gateway \\\"\"" > "Hop_count.tcl"
 }

  if(flg==1)
  {
     print "$nso at "tm" \"$node("a[hct]") color blue\"" > "Hop_count.tcl"
     print "$nso at "tm" \"$node("a[hct]") label Destination\"" > "Hop_count.tcl"
     print "$nso at "tm" \"$nso trace-annotate \\\"Multiple UDP flows between a Gateway and
Randomly chosen mesh node "des"\\\"\"" > "Hop_count.tcl"
  }

 tm=tm+0.01
```

```
 for(q=1;q<hct;q++)
 {
  print "set inf"q" [Connecting-Agent $node("a[q]") $sink("a[q+1]") "pks" "itv"]" >
"Hop_count.tcl"
  print "$nso at "tm" \"$inf"q" start\"" > "Hop_count.tcl"
  print "$nso at "tm+itval" \"$inf"q" stop\"" > "Hop_count.tcl"
  tm=tm+0.01
 }

 if(flg==1)
 {
  print GN " " a[hct-1] " " tm+itval " " des > "ARS_tmp"
  print "set inff"q" [Connecting-Agent $node("a[hct-1]") $sink("a[hct]") "pks" 0.002]" >
"Hop_count.tcl"
  print "$nso at "tm+int(tm/2) " \"$inff"q" start\"" > "Hop_count.tcl"
  print "$nso at "tm+itval" \"$inff"q" stop\"" > "Hop_count.tcl"
  print "$nso at "tm+int(tm/2)" \"$nso trace-annotate \\\"Link Failure between Node "a[hct-1]" -
Node "a[hct]" \\\"\"" > "Hop_count.tcl"
 }
}
```