Quest Journals
Journal of Electronics and Communication Engineering Research
Volume 8 ~ Issue 5 (2022) pp: 49-63
ISSN(Online) : 2321-5941
www.questjournals.org

Research Paper

Resolving Congestion Control And Transmission Issue Using Aodv And Self-Organized Ad-Hoc Network In Wsn Environment

[1]Mohanarangan S. and [2]Sivakumar D.

[1]*Assistant Professor, Departmnet of Computer Science and Engineering, Arunai Engineering College, Thiruvannamalai, 606603, India*

[2]*Professor, Department of ECE, Easwari Engineering College, Chennai, 600025, India*

Abstract: ***D****ynamic routing protocol deployed within NETWORK the issue need to solve first is sorted out. Depending on the network's size, its topology and any local constraint static routing could be the most adequate solution. Regarding size, a very small network, with only one or two IP routers wouldn't greatly benefit from a change in dynamic routing. One concept that need to keep in mind is the difference between an internal routing protocol and an external routing protocol. The first type is used with the same autonomous system (AS) and the second type to exchange routes with a different autonomous system. A set of networks directed by an equivalent organization is provided by autonomous system., which share the identical routing policy. It's found two or more autonomous systems within the same organization, if networks belong to different departments, or if they are managed by different teams or have different locations.*

Keywords: *Adhoc On-demand Distance Vector(AODV), Dynamic Source Routing (DSR), AutonomousSystem (AS)*

R*eceived 12 May, 2022;* ***R****evised 24 May, 2022;* ***A****ccepted 26 May, 2022 © The author(s) 2022.*
P*ublished with open access at* ***www.questjournals.org***

I. INTRODUCTION

Synthetic Aperture Radar (SAR) is a powerful and well The WSN is a wireless network consisting of spatially distributed autonomous devices using sensors to cooperatively monitor physical environmental conditions, such as temperature, sound, vibration, pressure motion or pollutants, at different locations [8]. WSN has significantly different communication constraints. The devices in such type of network are deployed in a huge numbers, they need the ability to assist each other to communicate data to a centralized collection point which is called a sink or a base station. The smallest devices are composed of a sensing unit, a radio, a processor integration of the sensor and having a power unit. The devices are capable of monitoring of a wide variety conditions such that temperature, humidity, soil makeup, pressure, vehicular movement, lighting conditions and noise levels, etc [9].

A typical example of pervasive computing applications is WSN, which has a broad range of applications such as military reconnaissance, environment monitoring, disaster relief and agriculture. The foremost aim of this type of network is to improve its life time and energy efficiency, load balancing packet transfer from sink to network as sensor of network is to conserve battery power. In WSN the powered mainly consumed for three purposes: data transmission, signal processing and hardware operation. With the rapid development and increasingly mature technology of MEMS (Micro Electro Mechanism System), wireless communications and modern networks merge into wireless sensor networks (WSN). It has created various innovative sensor network applications in near future. Today's sensor nodes are capable of sensing more than one parameter with the aid of multiple sensor boards mounted on a single radio board. It is more efficient, reliable and cost effective to use multi sensing unit instead of multiple nodes with multiple functionality [10].

Congestion is a problem in wireless sensor networks. Some techniques are used to reduce the congestion in WSN. Fusion's Techniques mitigate congestion, queue occupancy detects congestion, hop-by-hop flow control improves the efficiency of the network and source rate limiting as will improves the fairness. Fusion improves efficiency by 3 times and eliminates starvation. Different types of data generated by the sensors have various priorities. Hence it is necessary to ensure desired transmission rate for each type of data

based on the given priority to meet the demands of the base stations. In such a network, the sensor nodes could in fact generate simple periodic events to unpredictable bursts of messages. Congestion occurs even more likely when concurrent data transmissions over different ratio links interact with each other or when the reporting rate to the base station increases. When the number of nodes in the entire network increases the congestion might occur frequently.

In smart agriculture, infrastructure health monitoring and disaster site monitoring, information of distributedsensor nodes is gathered continuously using multi-hop wireless sensor networks (WSNs) to a base node [11].In WSN and also in current WiFi systems, fundamental transmission collision problems occur among nodes within two-hop distances, the so-called "Hidden nodeproblem" and the "Exposed node problem," the hidden node problem using Carrier Sense Multiple Access with CollisionAvoidance (CSMA/CA) and Request to Send / Clear to Send (RTS/CTS) mechanisms in MAC protocols, which partially mitigates the problem when the medium is not heavily loaded [12]. Regarding the exposed node problem, the IEEE 802.11 RTS/CTS mechanism will not function properly when the nodes are not synchronized .described that the throughput of wireless ad hoc networks can be improved by not using a handshake mechanism such as RTS/CTS. The IEEE 802.15.4 standard also supports CSMA/CA as in the IEEE 802.11 MAC protocol, but the hidden and exposed node problems persist as a critically important issueTo date, many related studies have been conducted of hidden-node and exposed-node problems[13]. According to a well-organized MAC layer solution survey , current MAC layer solutions are classifiable as pure contention-based , busy tone signal based, power-aware, multiple channel-based, and directional antenna-based protocols. Kosek-Szott described in an earlier that for current MAC layer solutions, many issues have remainedunsolved or have not been addressed completely, and that insufficient interest has been devoted to the exposed node problem.The hidden node and exposed node problems are complementary in that both arise when two nodes in two-hop locations donot know of one another. As described herein, we address the transmission collision problems above and the packet trafficcongestion problem while information gathering for large-scale WSNs [14].

II. LITERATURE REVIEW

S. Choobkar and R. Dilmaghani [1] described to avoid collision among hidden transmitters that decide to send data to a sink via `One' relay node concurrently. It is to add a predefined handshaking signal called COL (Collision) which is applicable for any packetised-preamble MAC protocol. If receiver recognises a collision it sends COL message to the channel and afterwards all senders postpone data transfer process and save energy. Performance analysis of proposed method proves its viability and effectiveness for the problem.

Akiya Kamimura, Kohji Tomita [2] presented a self-organizing network coordination framework for WSNs that realizes an adaptive time-division transmission by nodes and also traffic congestion handling in a decentralized manner. Specifically, the framework is based on a decentralized time division technique using a simplified pulse-coupled oscillator model. By coordinating the transmission timing adaptively, each node sends messages without collisions. "Hidden node problems" as well as "Exposed node problems" will be prevented, in principle, when using our method. Additionally, to reduce traffic congestion in a decentralized manner, time slots in the transmission cycle on each node are used efficiently by additional algorithms: an "empty time slots utilization algorithm" and a "takeover algorithm of neighboring nodes' transmission slots". These are introduced for efficient large data gathering applications. We simulated a 60-node data gathering application and evaluated its superiority to a conventional WSN method using CSMA/CA on IEEE 802.15.4 standard. We also conducted hardware experiments using nine developed WSN nodes and confirmed our framework's feasibility in real situations targeted at real-time landslide detection with distributed WSN nodes.

Katarzyna Kosek-Szott [3], grouped in several categories and are described in the order of their publication date. To give the reader a deep understanding of the progress made in the area of alleviating the hidden node problem a brief summary of the key ideas as well as a detailed comparison of different protocols are presented. Open research directions are also discussed to serve as a starting point for future protocol design and evaluation.

Xiaolong Li and Qing-An Zeng [4] we use a discrete Markov chain model to an-alyze the throughput of CSMA/CA protocols in a wireless local area network (WLAN) considering the capture phenom-enon, which means the packet with the strongest power may capture the receiver even in the presence of other overlap-ping packets. The model of capture effect over the theoretical throughput of CSMA/CA protocols in the presence of path loss, shadowing, and Rayleigh fading are obtained and com-pared with the model without capture effect.

L. Jiang and J. Walrand [5] we introduce an adaptive carrier sense multiple access (CSMA) scheduling algorithm that can achieve the maximal throughput distributively. Some of the major advantages of the algorithm are that it applies to a very general interference model and that it is simple, distributed, and asynchronous. Furthermore, the algorithm is combined with congestion control to achieve the optimal utility and fairness of competing flows. Simulations verify the effectiveness of the algorithm. Also, the adaptive CSMA scheduling is a modular MAC-layer algorithm that can be combined with various protocols in the

transport layer and network layer. Finally, the paper explores some implementation issues in the setting of 802.11 networks.

Yoshiaki Taniguchi and Go Hasegawa and Hirotaka Nakano [6] presented a self-organizing transmission and sleep scheduling mechanisms for periodically gathering sensor data at a base station in wireless sensor networks. In our proposed mechanisms, we use a conventional self-organizing communication mechanism based on phase-locking in a pulse-coupled oscillator model to propagate sensor data from the edge of the network to the base station in order of hop counts from the base station. In addition, we propose two mechanisms to avoid collisions among sensor nodes with the same hop count, a randomization-based mechanism and a desynchronizationbased mechanism based on anti-phase synchronization in the pulse-coupled oscillator model. Through simulation experiments, we show that our proposed mechanism significantly improves the data gathering ratio and energy-efficiency in comparison with the conventional mechanism.

Yuki Kubo and Kokuke Sekiyama[7] a novel communication timing control for wireless networks and radio interference problem. Communication timing control is based on the mutual synchronization of coupled phase oscillatory dynamics with a stochastic adaptation, according to the history of collision frequency in communication nodes. Through local and fully distributed interactions in the communication network, the coupled phase dynamics self-organizes collision-free communication. In wireless communication, the influence of the interference wave causes unexpected collisions. Therefore, we propose a more effective timing control by selecting the interaction nodes according to the received signal strength.

III. METHODOLOGY

1.Network Simulator

1.1Introduction

NS (Version 2) is an open source network simulation tool. It is an object oriented, discrete event driven simulator written in C++ and Otcl. The primary use of NS is in network researches to simulate various types of wired/wireless local and wide area networks; to implement network protocols such as TCP and UPD, traffic source behavior such as FTP, Telnet, Web, CBR and VBR, router queue management mechanism such as Drop Tail, RED and CBQ, routing algorithms such as Dijkstra, and many more[31].

Ns2 is written in C++ and Otcl to separate the control and data path implementations. The simulator supports a class hierarchy in C++ (the compiled hierarchy) and a corresponding hierarchy within the Otcl interpreter (interpreted hierarchy).

The reason why ns2 uses two languages is that different tasks have different requirements: For example simulation of protocols requires efficient manipulation of bytes and packet headers making the run-time speed very important. On the other hand, in network studies where the aim is to vary some parameters and to quickly examine a number of scenarios the time to change the model and run it again is more important.

In ns2, C++ is used for detailed protocol implementation and in general for such cases where every packet of a flow has to be processed. For instance, if you want to implement a new queuing discipline, then C++ is the language of choice. Otcl, on the other hand, is suitable for configuration and setup. Otcl runs quite slowly, but it can be changed very quickly making the construction of simulations easier. In ns2, the compiled C++ objects can be made available to the Otcl interpreter. In this way, the ready-made C++ objects can be controlled from the OTcl level.

1.2. OTCL basis

This chapter introduces the syntax and some basic commands of the Otcl language that are used by ns2 [31]. It is important to understand how Otcl works before moving to the part that deals with the creation of the actual simulation scenario.

1.3. Assigning values to variables

In tcl, values can be stored to variables and these values can be further used in commands:

```
set a 5
set b [expr $a/5]
```

In the first line, the variable a is assigned the value "5". In the second line, the result of the command [expr $a/5], which equals 1, is then used as an argument to another command, which in turn assigns a value to the variable b. The "$" sign is used to obtain a value contained in a variable and square brackets are an indication of a command substitution.

1.4. Procedures

One can define new procedures with the proc command. The first argument to proc is the name of the procedure and the second argument contains the list of the argument names to that procedure. For instance a procedure that calculates the sum of two numbers can be defined as follows:

```
proc sum {a b} {
expr $a + $b
}
```

The next procedure calculates the factorial of a number:

```
proc factorial a {
if {$a <= 1} {
        return 1
            }
#here the procedure is called again
expr $x * [factorial [expr $x-1]]
}
```

It is also possible to give an empty string as an argument list. However, in this case the variables that are used by the procedure have to be defined as global. For instance:

```
proc sum {} {
global a b
expr $a + $b
}
```

1.5. Files and lists

In tcl, a file can be opened for reading with the command:

```
settestfile [open test.dat r]
```

The first line of the file can be stored to a list with a command:

```
gets $testfile list
```

Now it is possible to obtain the elements of the list with commands (numbering of elements starts from 0) :

```
set first [lindex $list 0]
set second [lindex $list 1]
```

Similarly, a file can be written with a puts command:

```
settestfile [open test.dat w]
puts $testfile “testi”
```

1.6. Callingsubprocesses

The command exec creates a subprocess and waits for it to complete. The use of exec is similar to giving a command line to a shell program. For instance, to remove a file:

execrm $testfile

The exec command is particularly useful when one wants to call a tcl-script from within another tclscript. For instance, in order to run the tcl-script example.tcl multiple times with the value of the parameter “test” ranging from 1 to 10, one can type the following lines to another tcl-script:

```
for {set ind 1} {$ind<= 10} {incrind} {
set test $ind
exec ns example.tcl test
}
```

2. CREATING THE TOPOLOGY

To be able to run a simulation scenario, a network topology must first be created. In ns2, the topology consists of a collection of nodes and links [32].

Before the topology can be set up, a new simulator object must be created at the beginning of the script with the command:

```
set ns [new Simulator]
```

The simulator object has member functions that enable creating the nodes and the links, connecting agents etc. All these basic functions can be found from the class Simulator. When using functions belonging to this class, the command begins with "$ns", since ns was defined to be a handle to the Simulator object.

2.1. Nodes
New node objects can be created with the command:

```
set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]
```

The member function of the Simulator class, called "node" creates four nodes and assigns them to the handles n0, n1, n2 and n3. These handles can later be used when referring to the nodes. If the node is not a router but an end system, traffic agents (TCP, UDP etc.) and traffic sources (FTP,CBR etc.) must be set up, i.e, sources need to be attached to the agents and the agents to the nodes, respectively.

2.2. Agents, applications and traffic sources
The most common agents used in ns2 are UDP and TCP agents. In case of a TCP agent, several types are available. The most common agent types are:
- Agent/TCP – a Tahoe TCP sender
- Agent/TCP/Reno – a Reno TCP sender
- Agent/TCP/Sack1 – TCP with selective acknowledgement

The most common applications and traffic sources provided by ns2 are:
- Application/FTP – produces bulk data that TCP will send
- Application/Traffic/CBR – generates packets with a constant bit rate
- Application/Traffic/Exponential – during off-periods, no traffic is sent. During on-periods, packets are generated with a constant rate. The length of both on and off-periods is exponentially distributed.
- Application/Traffic/Trace – Traffic is generated from a trace file, where the sizes and interarrival times of the packets are defined.

In addition to these ready-made applications, it is possible to generate traffic by using the methods provided by the class Agent. For example, if one wants to send data over UDP, the method

```
send(intnbytes)
```

can be used at the tcl-level provided that the udp-agent is first configured and attached to some node.
Below is a complete example of how to create a CBR traffic source using UDP as transport protocol and attach it to node n0:

```
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
```

```
$cbr0 set packet_size_ 1000
$udp0 set packet_size_ 1000
$cbr0 set rate_ 1000000
```

An FTP application using TCP as a transport protocol can be created and attached to node n1 in much the same way:

```
set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$tcp1 set packet_size_ 1000
```

The UDP and TCP classes are both child-classes of the class Agent. With the expressions [new Agent/TCP] and [new Agent/UDP] the properties of these classes can be combined to the new objects udp0 and tcp1. These objects are then attached to nodes n0 and n1. Next, the application is defined and attached to the transport protocol. Finally, the configuration parameters of the traffic source are set. In case of CBR, the traffic can be defined by parameters rate_ (or equivalently interval_, determining the inter arrival time of the packets), packet Size_ and random_ . With the random_ parameter it is possible to add some randomness in the inter arrival times of the packets. The default value is 0, meaning that no randomness is added.

2.3. Traffic Sinks

If the information flows are to be terminated without processing, the udp and tcp sources have to be connected with traffic sinks. A TCP sink is defined in the class Agent/TCPSink and an UDP sink is defined in the class Agent/Null [32].

A UDP sink can be attached to n2 and connected with udp0 in the following way:

```
set null [new Agent/Null]

$ns attach-agent $n2 $null

$ns connect $udp0 $null
```

A standard TCP sink that creates one acknowledgement per a received packet can be attached to n3 and connected with tcp1 with the commands:

```
set sink [new Agent/Sink]
$ns attach-agent $n3 $sink
$ns connect $tcp1 $sink
```

There is also a shorter way to define connections between a source and the destination with the command:

```
$ns create-connection <srctype><src><dsttype><dst><pktclass>
```

For example, to create a standard TCP connection between n1 and n3 with a class ID of 1:

```
$ns create-connection TCP $n1 TCPSink $n3 1
```

One can very easily create several tcp-connections by using this command inside a for-loop.

2.4. Links

Links are required to complete the topology. In ns2, the output queue of a node is implemented as part of the link, so when creating links the user also has to define the queue-type.

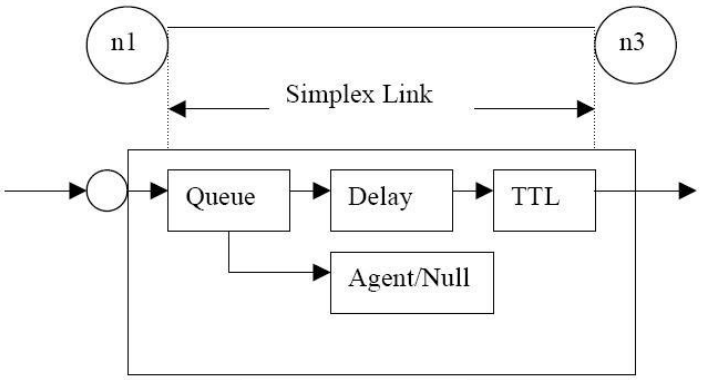

Figure 1 Link in ns2

Figure 2 shows the construction of a simplex link in ns2. If a duplex-link is created, two simplex links will be created, one for each direction. In the link, packet is first enqueued at the queue. After this, it is either dropped, passed to the Null Agent and freed there, or dequeued and passed to the Delay object which simulates the link delay. Finally, the TTL (time to live) value is calculated and updated.
Links can be created with the following command:

$ns duplex/simplex-link endpoint1 endpoint2 bandwidth delay queue-type

For example, to create a duplex-link with DropTail queue management between n0 and n2:

```
$ns duplex-link $n0 $n2 15Mb 10ms DropTail
```

Creating a simplex-link with RED queue management between n1 and n3:

```
$ns simplex-link $n1 $n3 10Mb 5ms RED
```

The values for bandwidth can be given as a pure number or by using qualifiers k (kilo), M (mega), b (bit) and B (byte). The delay can also be expressed in the same manner, by using m (milli) and u (mikro) as qualifiers. There are several queue management algorithms implemented in ns2, but in this exercise only DropTail and RED will be needed.

3. TRACING AND MONITORING

In order to be able to calculate the results from the simulations, the data has to be collected somehow. NS2 supports two primary monitoring capabilities: traces and monitors. The traces enable recording of packets whenever an event such as packet drop or arrival occurs in a queue or a link. The monitors provide a means for collecting quantities, such as number of packet drops or number of arrived packets in the queue. The monitor can be used to collect these quantities for all packets or just for a specified flow (a flow monitor) [33].

3.1. Traces

All events from the simulation can be recorded to a file with the following commands:

```
settrace_all [open all.dat w]

$ns trace-all $trace_all

$ns flush-trace

close $trace_all
```

First, the output file is opened and a handle is attached to it. Then the events are recorded to the file specified by the handle. Finally, at the end of the simulation the trace buffer has to be flushed and the file has to be closed. This is usually done with a separate finish procedure. If links are created after these commands, additional objects for tracing (EnqT, DeqT, DrpT and RecvT) will be inserted into them.

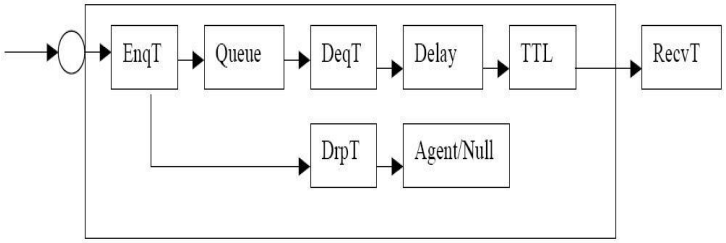

Figure 2 Link in ns2 when tracing is enabled.

These new objects will then write to a trace file whenever they receive a packet. The format of the trace file is following:

```
+ 1.84375 0 2 cbr 210 ------- 0 0.0 3.1 225 610

- 1.84375 0 2 cbr 210 ------- 0 0.0 3.1 225 610

r 1.84471 2 1 cbr 210 ------- 1 3.0 1.0 195 600

r 1.84566 2 0 ack 40 ------- 2 3.2 0.1 82 602

+ 1.84566 0 2 tcp 1000 ------- 2 0.1 3.2 102 611

- 1.84566 0 2 tcp 1000 ------- 2 0.1 3.2 102 611

+ :enqueue
- : dequeue
d : drop
r : receive
```

The fields in the trace file are: type of the event, simulation time when the event occurred, source and destination nodes, packet type (protocol, action or traffic source), packet size, flags, flow id, source and destination addresses, sequence number and packet id. In addition to tracing all events of the simulation, it is also possible to create a trace object between a particular source and a destination with the command:

```
$ns create-trace <type><file><src><dest>
```

where the type can be, for instance,

- Enque – a packet arrival (for instance at a queue)
- Deque – a packet departure (for instance at a queue)
- Drop – packet drop
- Recv – packet receive at the destination

Tracing all events from a simulation to a specific file and then calculating the desired quantities from this file for instance by using perl or awk and Matlab is an easy way and suitable when the topology is relatively simple and the number of sources is limited. However, with complex topologies and many sources this way of collecting data can become too slow. The trace files will also consume a significant amount of disk space.

3.2. Monitors

With a queue monitor it is possible to track the statistics of arrivals, departures and drops in either bytes or packets. Optionally the queue monitor can also keep an integral of the queue size over time [33].
For instance, if there is a link between nodes n0 and n1, the queue monitor can be set up as follows:

```
set qmon0 [$ns monitor-queue $n0 $n1]
```

The packet arrivals and byte drops can be tracked with the commands:

```
setparr [$qmon0 set parrivals_]

setbdrop [$qmon0 set bdrops_]
```

Besides assigning a value to a variable the set command can also be used to get the value of a variable. For example here the set command is used to get the value of the variable “parrivals” defined in the queue monitor class.
A flow monitor is similar to the queue monitor but it keeps track of the statistics for a flow rather than for aggregated traffic. A classifier first determines which flow the packet belongs to and then passes the packet to the flow monitor.

The flow monitor can be created and attached to a particular link with the commands:

```
setfmon [$ns makeflowmon Fid]

$ns attach-fmon [$ns link $n1 $n3] $fmon
```

Notice that since these commands are related to the creation of the flow-monitor, the commands are defined in the Simulator class, not in the Flow monitor class. The variables and commands in the Flowmonitor class can be used after the monitor is created and attached to a link. For instance, to dump the contents of the flow monitor (all flows):

```
$fmon dump
```

If you want to track the statistics for a particular flow, a classifier must be defined so that it selects the flow based on its flow id, which could be for instance 1:

```
setfclassifier [$fmon classifier]

set flow [$fclassifier lookup auto 0 0 1]
```

4. CONTROLLING THE SIMULATION

After the simulation topology is created, agents are configured etc., the start and stop of the simulation and other events have to be scheduled [34]. The simulation can be started and stopped with the commands

```
$ns at $simtime “finish”

$ns run
```

The first command schedules the procedure finish at the end of the simulation, and the second command actually starts the simulation. The finish procedure has to be defined to flush the trace buffer, close the trace files and terminate the program with the exit routine. It can optionally start NAM (a graphical network animator), post process information and plot this information.

The finish procedure has to contain at least the following elements:

```
proc finish {} {

global ns trace_all

$ns flush-trace

close $trace_all

exit 0

}
```

Other events, such as the starting or stopping times of the clients can be scheduled in the following way:

```
$ns at 0.0 “cbr0 start”
$ns at 50.0 “ftp1start”
$ns at $simtime “cbr0 stop”
$ns at $simtime “ftp1 stop”
```

If you have defined your own procedures, you can also schedule the procedure to start for example every 5 seconds in the following way:

```
proc example {} {
global ns
set interval 5
….
…
$ns at [expr $now + $interval] “example”
}
```

IV. RESULTS AND DISCUSSION

The proposed system compares the logic of two routing protocols in terms of its efficiency with respect to throughput, energy efficiency, reduced network delay and enhanced lifetime. The proposed system is constructed based on the following two routing protocols such as Adhoc On-demand Distance Vector (AODV) and the Dynamic Source Routing (DSR). These two metrics are taken into an account for estimating the whole process in Network Simulator environment. We construct the proposed system in two ways, such as Static and Dynamic Node Formations. In static concern, the network region is formed with fixed number of nodes and the dynamic mode the user can input the number of nodes. The communication process belongs to individual route metrics and the route selection process varies based on the static and dynamic procedures, in which the static routing protocol follows the same pathway and execution norms while executing the respective program. Instead the dynamic routing protocol execution is entirely different based on the respective source and destination node selection, the path varies according to sequential manner. For dynamic routing application, we utilize the AODV routing protocol and the static routing structure follows the DSR routing protocol logic. The required graphs will be emulated based on the functions constructed with respect to the user's input parameters and the following graphs will be provided to prove the proposed system efficiency metric, such as throughput, energy efficiency, network delay analysis and lifetime. Both the TCL codes need to be executed separately in XWinServer environment and the individual graphs for both AODV and DSR will be generated as well as the comparison graphs also be emulated for both the executions in single graphical format.

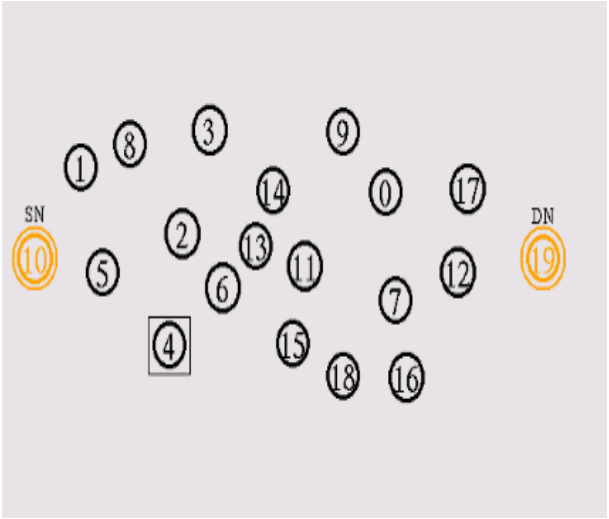

Figure 3 is shows the nodes at initial state, number 10 is start node and 19 is destination node.

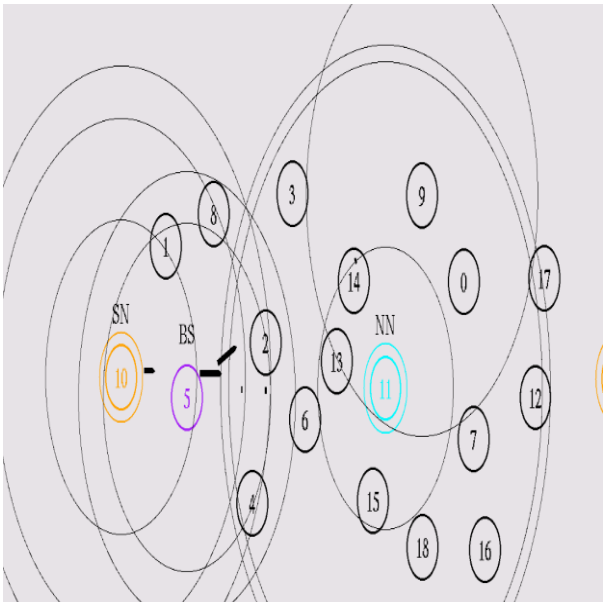

Figure 4 is shows 10th node is try to sending a data to destination through in between nodes by using the algorithm shortest distance of the path.

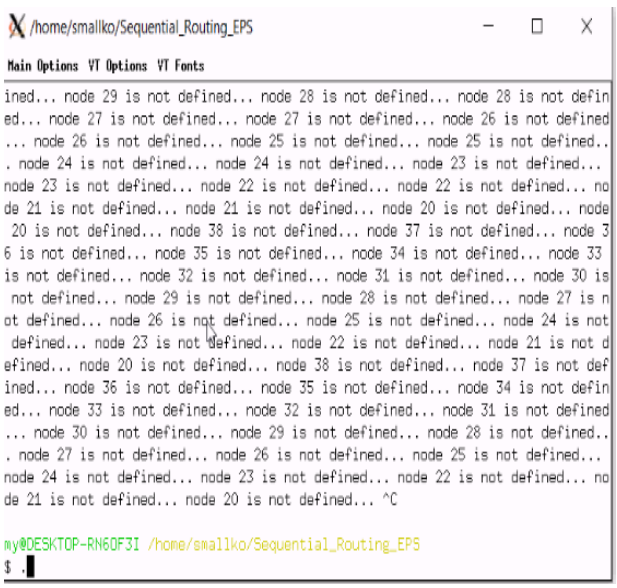

Figure 5 represents the listing the node numbers of data transmitting from SN to DN.

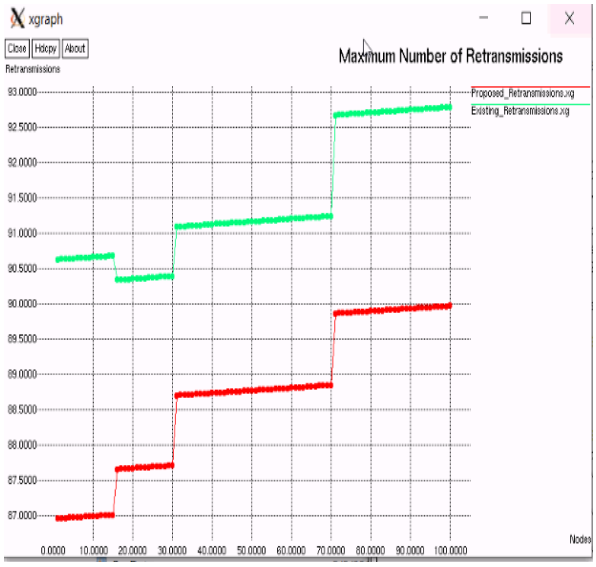

Figure 6 represents the maximum number of retransmissions graph.

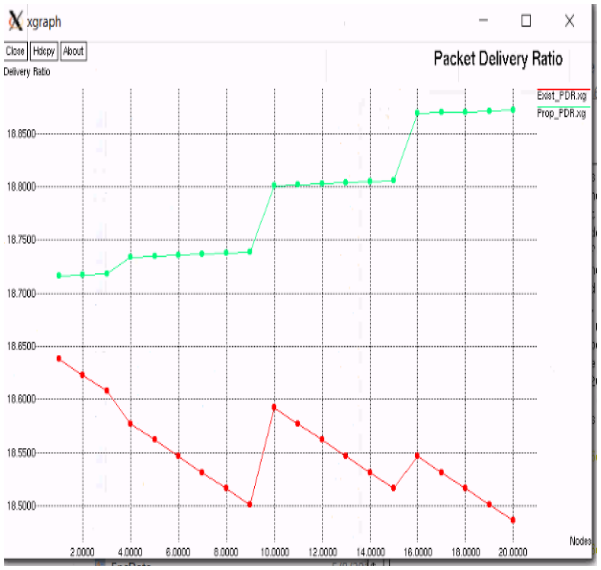

Figure 7 represents the graph which is packet delivery ratio of existing and proposed system by denoting red colour and green colour respectively

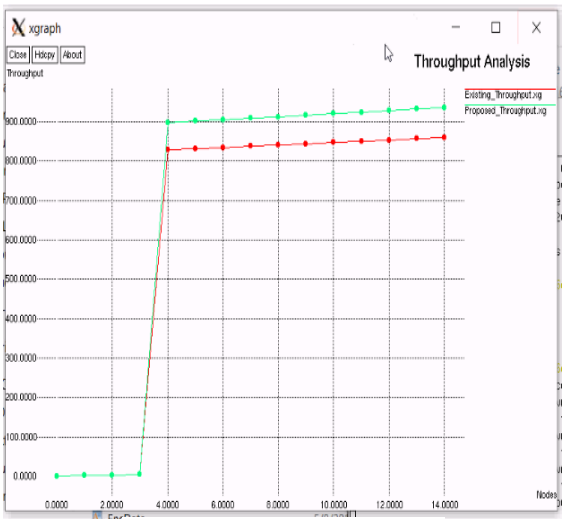

Figure 8 Throughput Analysis

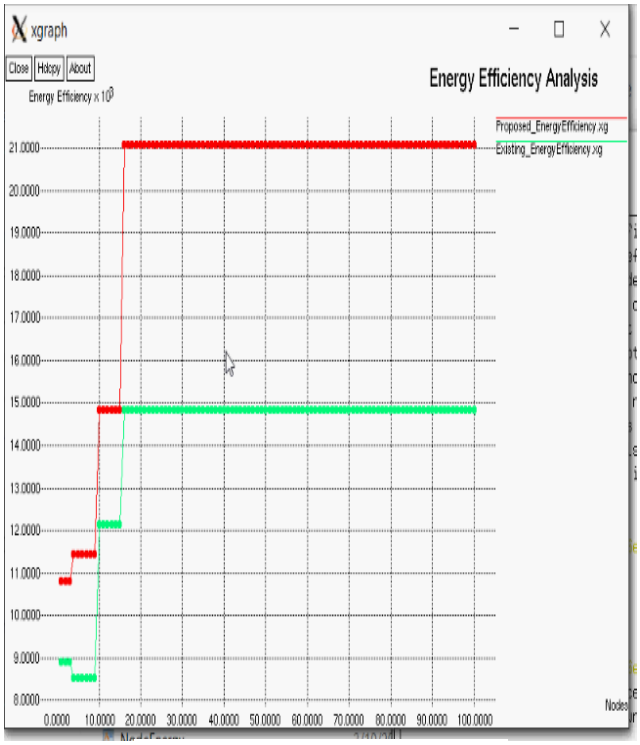

Figure 9 Energy Efficient Analysis

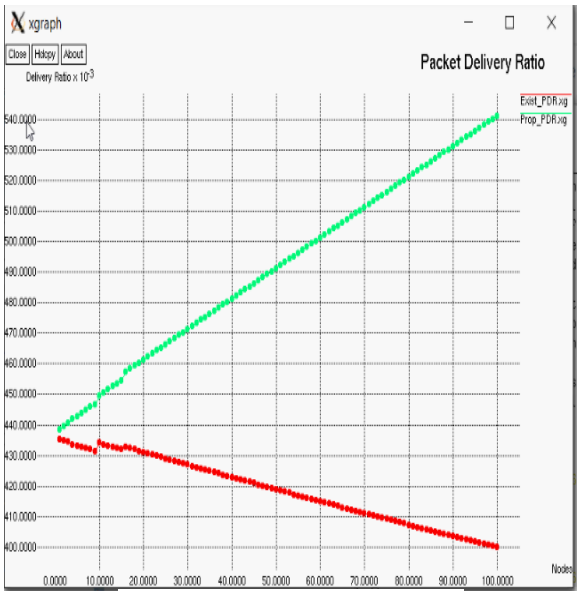

Figure 10 Packet Delivery Ratio

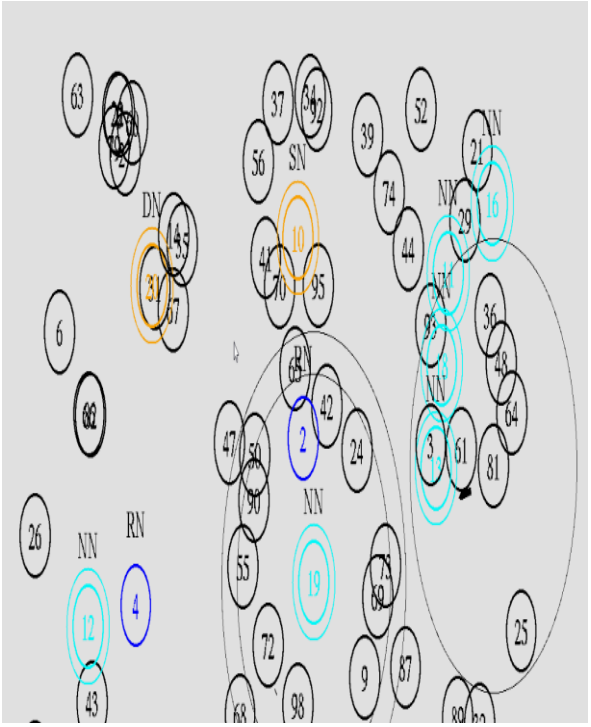

Figure 11 represents the highlighting the nodes which are participating in the transmission of data from SN to DN.

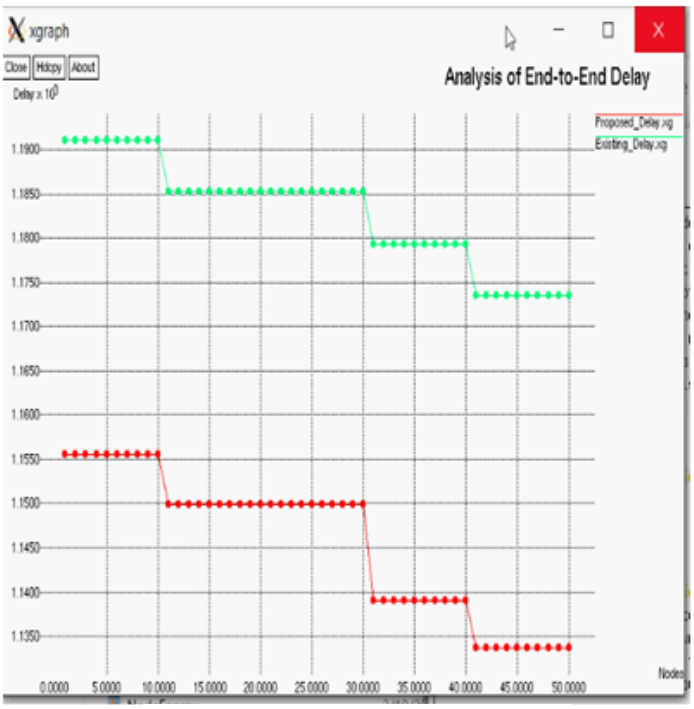

Figure 12 Analysis of End to End Delay

V. CONCLUSION

A packet transmission cycle of each node shared in the network will be changed adaptively according as per the quantity of nodes and their topology so the transmission interval is maintained efficiently.We presented a drawn out distributed algorithm for proficient data transfer for delivering traffic congestion when the stored data on the node are expanded. By comparing the CSMA/CA method and SoNCF showed the achievability of SoNCF utilizing software simulations. It is no packet loss and successfully moved information between nodes by simulation of the extended strategy.

REFERENCES

[1]. ME El-Telbany, MA Maged, "Wireless sensor networks for extreme environments: remote sensing and space industry", International Journal of Hybrid Intelligence, vol. 1, no. 1, 2019.

[2]. Akiya Kamimura, Kohji Tomita,"A self-organizing network coordination framework enabling collision-free and congestion-less wireless sensor networks", Journal of Network and Computer Applications,Vol. 93, pp. 228-244, 2017.

[3]. Katarzyna Kosek-Szott,"A survey of MAC layer solutions to the hidden node problem in ad-hoc networks",Ad Hoc Networks,Vol. 10, no. 3, pp. 635-660, 2012.

[4]. Xiaolong Li, Qing-An Zeng, "Performance Analysis of the IEEE 802.11 MAC Protocol over a WLAN with Capture Effect", IPSJ Digital Courier, Vol. 1, Pages 545-551, 2005.

[5]. L. Jiang and J. Walrand, "A Distributed CSMA Algorithm for Throughput and Utility Maximization in Wireless Networks," in IEEE/ACM Transactions on Networking, vol. 18, no. 3, pp. 960-972, June 2010.

[6]. Yoshiaki Taniguchi & Go Hasegawa & Hirotaka Nakano, "Self-organizing Transmission Scheduling Considering Collision Avoidance for Data Gathering in Wireless Sensor Networks", Journal of Communications. vol. 8, no. 6, pp. 389-397, 2013.

[7]. K Tomita, A Kamimura, "Transmission reordering in self-organizing network coordination framework", International Conference on Context-Aware Systems and Applications, pp 14–24, December 2018.

[8]. AminaKhan,SumeetGuptaSachin,KumarGupta, "Multi-hazard disaster studies: Monitoring, detection, recovery, and management, based on emerging technologies and optimal techniques",International Journal of Disaster Risk Reduction,vol 47, 2020.

[9]. Degesys, J., Rose, I., Patel, A., Nagpal, R., "DESYNC: Self-Organizing Desynchronization and TDMA on Wireless Sensor Networks," 2007 sixth International Symposium on Information Processing in Sensor Networks, pp. 11–20, 2007.

[10]. Taniguchi, Y., Hasegawa, G., Nakano, H., 2013. "Self-organizing Transmission Scheduling Considering Collision avoidance for Data Gathering in Wireless Sensor Networks," Journal of Communications, vol. 8, no. 6, pp. 389–397.

[11]. D Helen, D Arivazhagan, "Refining an Ad-hoc Network Performance by Executing Multiple Channel in Mac Protocol", International Journal of Computer Science and Information Technologies, Vol. 5, Issue. 4 , pp 5458-5460, 2014.

[12]. Bianchi, G., Fratta, L., Oliveri, M.,"Performance evaluation and enhancement of the CSMA/CA MAC protocol for 802.11 wireless LANs," Personal, Indoor and Mobile Radio Communications (PIMRC'96), 2, 15–18, 1996.

[13]. Hwang, L.J., Sheu, S.T., Shih, Y.Y., Cheng, Y.C., "Grouping strategy for solving hidden node problem in IEEE 802.15.4 LR-WPAN," First International Conference on Wireless Internet (WICON'05), pp. 26–32, 2005.

[14]. Jiang L., Walrand, J., "A Distributed CSMA Algorithm for Throughput and Utility Maximization in Wireless Networks," IEEE/ACM Transactions on Networking, vol. 18, no. 3, pp. 960-972, 2010.