PROJECT 3
Evaluate MAC random transmission protocol using NS-2

BY
AMIT PRATAP SINGH
AMITPRAT@BUFFALO.EDU

PROJECT REPORT
SUBMITTED IN THE PARTIAL FULFILMENT OF REQUIREMENT FOR
CSE 589 : MODERN COMPUTER NETWORKS
COMPUTER SCIENCE AND ENGINEERING
UNIVERSITY AT BUFFALO
FALL 2013

Overview

In this project, we simulated the ad-hoc wireless networks with 100 sources nodes and 1 sink node. The source nodes continuously keeps on sending packets to sink node at specified time interval(0.02) . The network architecture consists of one sink node which receives the packets transmitted by source nodes. The source nodes are all within one-hop communication range of the sink node. The only reason for unsuccessful transmission is due to collision among multiple packets sent by multiple source nodes. The source nodes generate and transmit data as follows:

1) A data packet is generated every T seconds and has to be delivered to the sink before the next packet is generated.
2) The source nodes are equipped with only an RF transmitter, it is impossible for them to sense the channel or receive any acknowledgements (or negative acknowledgement) from the sink node.
3) In order to increase the chance of successful transmission, each node transmits X copies of each packet at random instants before the next packet is generated.
4) The source node picks X random instants of time within the interval [0, T] (i.e the interval between the time the current packet is generated to the time the next packet will be generated), and transmits the data packet at each of the X instants of time.

Network Simulation with NS2

NS2 is the complete package of various tools which helps us to simulate the real network scenarios and analyze the behaviour using logging, graph generator and graphical simulations . NS2 provides substantial support for simulation of TCP/UCP, routing and multicast protocols. NS2 is a object oriented interpreter written in C++ with OTcl interpreter as front end. Most of the scripts are generated using tickle(tool command language).

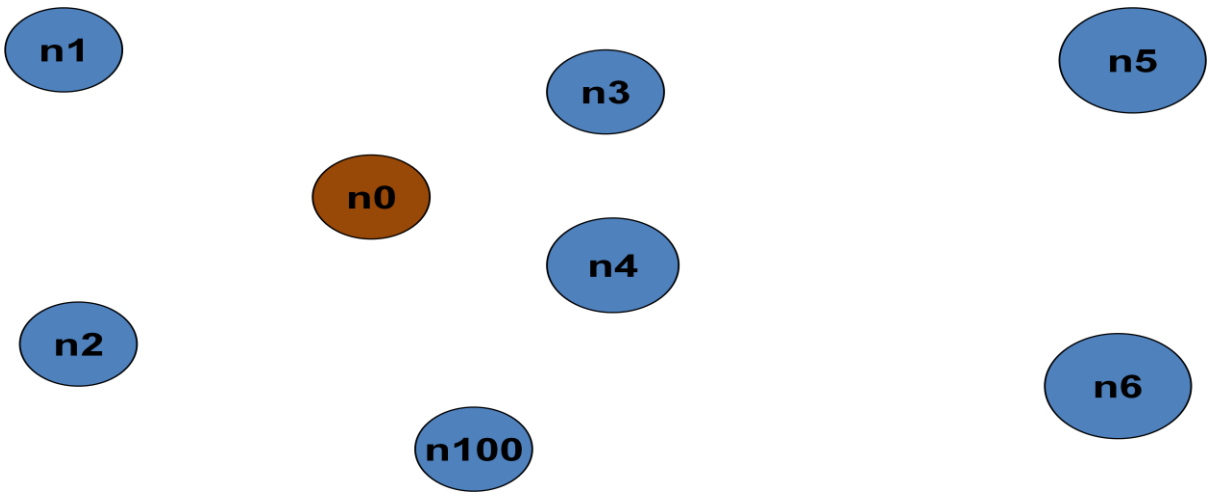
Tickle Script : We used this script to create the network topologies and specify the network parameters to be used for simulation. It allows to change the parameters (through parameter binding) for various simulations without modifying the underlying protocol code. NS2 creates the OTCL linkage between tickle script and protocol(C++) code to access their parameters and call the methods from the script. In this simulation, we used the sink agent as Loss Monitor which allows to record the sent data(bytes) information. Loss Monitor objects are the subclass of agent object that implements the traffic sink which maitains some information about received data such as nlost(number of bytes lost),npckts(number of packet received) etc.

Node Configuration Parameters : Below listed are network parameters that were used for the simulation.

Val(chan)	Channel/WirelessChannel	Channel Type
Val(prop)	Propagation/TwoRayGround	Radio-propagation model
val(netif)	Phy/WirelessPhy	Network Interface Type
val(mac)	Mac/MncPrj	Mac Protocol
val(ifq)	Queue/DropTail/PriQueue	Interface Queue Type
val(ll)	LL	Link Layer Type L(inherits

		from LinkDelay).It has a composed component ARP which works as ARP procedure, mapping the protocol address (such as IP address) to Hardware address (such as MAC address);
val(ant)	Antenna/OmniAntenna	Wireless Antenna Type
val(ifqlen) 50	50	Interface Queue Length
val(nn)	101	Total mobile nodes(100 source + 1 sink node)
val(rp)	DumbAgent	Routing Protocol Used
val(x)	50	Terrain Dimension
val(y)	50	Terrain Dimension
val(stop)	100	Time when simulation stops
val(repeatTimes)	1-10	Total number of repeat counts of sent packet
val(packetsize)	128Bytes	traffic packet Size
val(interval)	0.02s	Packet send interval
val(rate)	11Mb	Transfer rate

Creating Topology:



This is sample instance of created topology using god(General Object Descriptor) with 100 sources nodes and 1 sink node [n0 is sink node while the rest are source nodes].
These nodes are distributed at random positions within 50mx50m terrain area.

Each node sends 128 bytes packet(additional duplicate packets are also adjusted in this time interval) to sink node(n0) after every 20 ms for the entire duration of simulation(100sec). These nodes communicate using UDP protocol with CBR(constant bit rate) as data traffic. Initially each node starts sending data at different time intervals within 0.02 duration. Each node finishes sending its all packet copies before the next packet is generated(0.02sec time duration).

Parameter Binding:

In this project, we bind the local *repeatTimes* variable of tickle script with *repeatnum* variable used in MAC protocol file to send the multiple copies of packet. The value of *repeatCount* can directly be modified in tickle script to affect the corresponding C++ variable(*repeatnum*).

```
set val(repeatTimes) 1 ;#duplicate copies of packets
$val(mac) set repeatnum $val(repeatTimes)
```

(script segment of tickle file for parameter binding)

```
tcl.evalf("Mac/MncPrj set repeatnum");
bind("repeatnum",&repeatnum);
```

(c++ code part to bind the tickle private with its local variable)

Interpreting Trace File:

The generated trace file contain verbose of information about each packet. Generally, we need to extract some specific information from trace file to calculate throughput/delay/block ratio etc. The popular way is to write some scripts/programs (specially in awk/perl/java) to analyze string patterns, record variable counts etc to deduce the final outcome.
In this project, awk script(*awk_scr.awk*) is used to extract the information from generated trace file (*trace_file.tr*). [The general notion of awk usage is gawk -f *awk_scr.awk trace_file.tr* . *Ps - You may need to install gawk utility to compile the script*]

(trace file fields under the observation which may be differ from other older ns2 formats)

ACTION	[s|r|D]: s -- sent, r -- received, D -- dropped
WHEN	the time when the action happened
WHERE	the node where the action happened
LAYER	AGT -- application, RTR -- routing, LL -- link layer (ARP is done here) IFQ -- outgoing packet queue (between link and mac layer) MAC -- mac, PHY -- physical
flags	
SEQNO	the sequence number of the packet
TYPE	the packet type cbr -- CBR data stream packet DSR -- DSR routing packet (control packet generated by routing) RTS -- RTS packet generated by MAC 802.11

	ARP -- link layer ARP packet
SIZE	the size of packet at current layer, when packet goes down, size increases, goes up size decreases
[a b c d	a -- the packet duration in mac layer header b -- the mac address of destination c -- the mac address of source d -- the mac type of the packet body
flags:	
[.....]:	[source node ip : port_number destination node ip (-1 means broadcast) : port_number ip header ttl ip of next hop (0 means node 0 or broadcast)]

Sample Analysis Script Output:

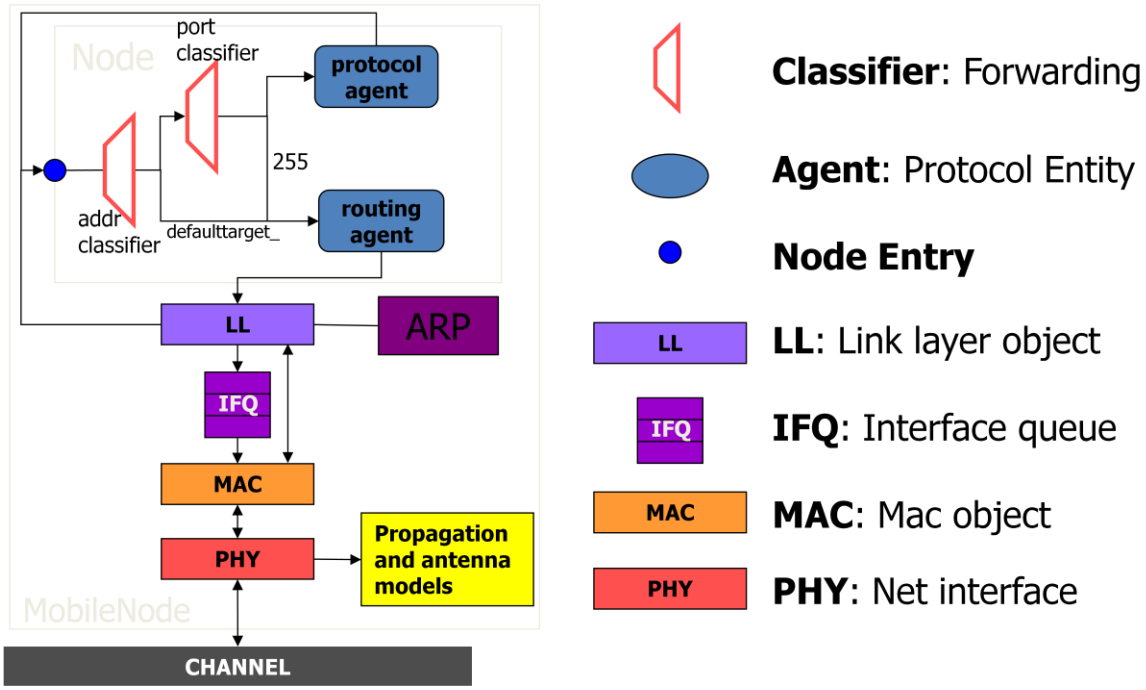
==========

GeneratedPackets = 50000
ReceivedPackets = 476950
SentPackets = 499555
Packet Delivery Ratio (X = Recv/Generated) = 95.39%

==========

Mac Protocol

Layering Architecture



Layering Architecture of a Single Node in Wireless Mesh (to specify the MAC

Mac Layer

Important Mac Parameter set:

MacState state_: deciding the state of the mac or channel, such as IDLE, RECV, SEND, COLL, RTS,CTS, ACK, POLLING.
basicRate_: transmission rate for control packets such as RTS, CTS, ACK and Broadcast;
dataRate_: transmission rate for mac layer data packets;
mhRecv_: receiving timer;
mhSend_: sending timer;
rx_state_: receiving or incoming state (MAC RECV or MAC IDLE);
tx_state_: sending or outgoing state

The major functionalities of Mac-Layer includes the followings:-

SendDown(Packet *) : Sending the packet to channel (Physical layer), stamping the packet with interface arguments. This function is called from within recv function which receives the packet from upper layer and sends the packet to lower layer.
SendUP(Packet *) : This function is called by lower layer(physical channel) . This decides whether the sent packet has sufficient energy levels and check it for errors and then sets the channel state to idle.
recv(Packet *p, Handler *h) : This function is called by up level agent or down-level object. If it is outgoing packet (e.g. hdr_cmn::direction()==hdr_cmn::DOWN) , then call send() function otherwise it is incoming packet.
send(Packet *p, Handler *h) : This function places packet on lower level channel (WirelessPhy) disregard of what type of packet this it. This function calls the sendhandler() to let the upper level guy know about it. In our modification, we repeatedly call sendhandler and notify upper level guy each time so that at least one out of these duplicate packets are successfully acknowledged.
recvHandler() : This function effectively holds the waiting packet to be processed. These packets are held by field pkt_tx. This pktRx would be checked for error and energy level to identify the collision or fading to discard the packets. Here the modifications were made to create the duplicate copy of pkt_tx and send it repeatedly or free it.
sendHandler() : This function tells the lower layer(channel) about sending data packet and then frees the pkt_Tx_ and reset the other parameters and tells the upper level about it if the packet is sent or dropped. Here also the modifications are done to try sending the packet down each time and notifying receiver hoping if out of repeatTimes, one packet will be successfully sent.
sendTimer() : When this timer expires, this function would be called which in turn will check for transmission state tx_state_. If it is MAC_SEND , the it means that packet has been sent out but not acknowledged and hence it tells the upper level guy otherwise if state is IDLE then it does nothing.
recvTimer() : The packet received and waiting to be processed is handled is held by pktRx_. The pktRx_ would be checked for collision, error and power and then will be forwarded to corresponding handler function.

Mac Protocol

Most MAC protocols are based upon principle of carrier sensing and Collision avoidance (CA) . But the hidden terminal/Signal Fading/Exposed terminal play dominant role in CSMA/CA based implementations. Another way of implementation can using control handshaking which involves the exchange of RTS/CTS parameters before sending actual data. But it is generally used only for relatively large packets. Some MAC protocols may specify some threshold packet size after which only they employ RTS/CTS mechanism. The collision of RTS/CTS packets and multiple recipients of a CTS packet within the node communication range again creates the problem for control handshaking. To avoid this problem, directional antennas can be used to limit the reception of packet to particular destination.

Retry Counters: In this implementation, we use the retry counter approach for efficient MAC protocol implementation. These retry counters can be long or short which again depends upon various simulation scenarios(such as network traffic, nodes involved in simulation). Here, we use the retry counters within the range of [1-10]. Each node sends the duplicate copy of packet at random time intervals within 20ms time duration. This is maximize the chances to packet reception which may be lost due to RxRx or TxTx collision.

Simulation Results

Average Delivery Probability (P)= (Total Number of Received packets/Generated Packets)

1) Number of Nodes : 10 Source Nodes + 1 Sink Node ; Duration : 100s

Repeat Count	Generated Packets	Received Packets	P
1	50000	49968	0.99
2	50000	97460	1.94
3	50000	144830	2.89
4	50000	192265	3.84
5	50000	239600	4.79
6	50000	287061	5.74
7	50000	334234	6.68
8	50000	381796	7.63
9	50000	428881	8.57
10	50000	476162	9.52

2) Number of Nodes : 20 Source Nodes + 1 Sink Node ; Duration : 100s

Repeat Count	Generated Packets	Received Packets	P
1	100000	99887	0.99
2	100000	194789	1.94
3	100000	289587	2.89
4	100000	384630	3.84
5	100000	479204	4.79
6	100000	574012	5.74
7	100000	858792	6.67
8	100000	953673	7.62
9	100000	858792	8.58
10	100000	951116	9.51

3) (Number of Nodes : 50 Source Nodes + 1 Sink Node ; Duration : 100s)

Repeat Count	Generated Packets	Received Packets	P
1	250000	249280	0.99
2	250000	493550	1.97
3	250000	729225	2.91
4	250000	940220	3.76
5	250000	1154220	4.61
6	250000	1368450	5.47
7	250000	1577240	6.30
8	250000	1794000	7.17
9	250000	2005200	8.08
10	250000	2205750	8.80

4) (Number of Nodes : 100 Source Nodes + 1 Sink Node ; Duration : 100s)

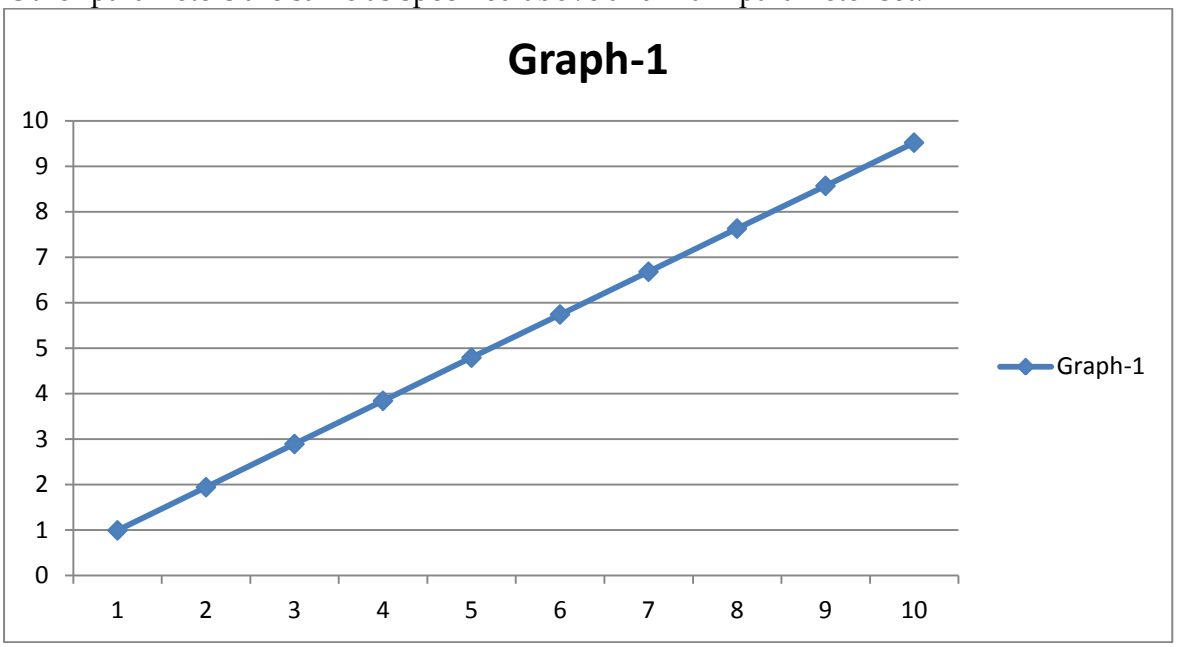
Repeat Count	Generated Packets	Received Packets	P
1	500000	496489	0.99
2	500000	968833	1.93
3	500000	1413600	2.82
4	500000	1864600	3.72
5	500000	2281000	4.56
6	500000	2640600	5.62
7	500000	3018400	6.03
8	500000	3369200	6.73
9	500000	3645000	7.29
10	500000	3900000	7.80

Table/Plot

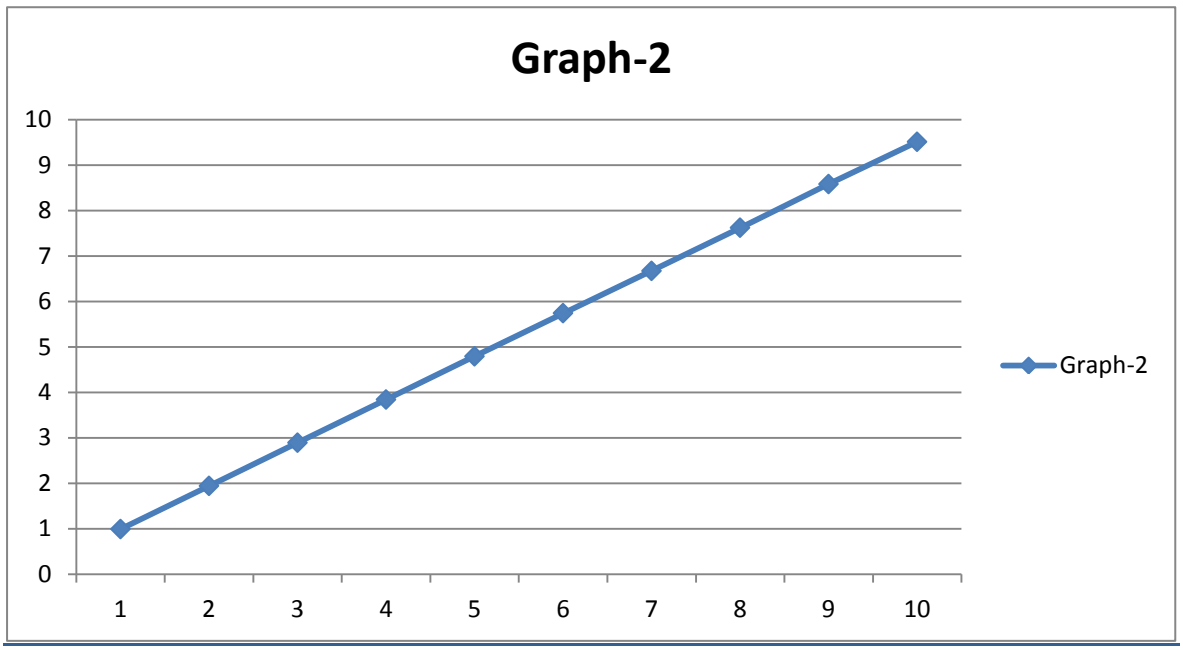
Blocking Probability = Ratio of { Received Packet at Sink Node & Total Generated Packets }.
The analyzed traffic in result is Constant Bit Rate (CBR) using UDP agent as upper level protocol and Dumb Agent as routing level protocol.
Here the graphs are plotted using upper table values. The different plots use different number of nodes[10,20,50,100] in experiment.
(repeat count along x-axis , P along y-axis)

1) Number of Nodes = 10 sources nodes + 1 Sink Node

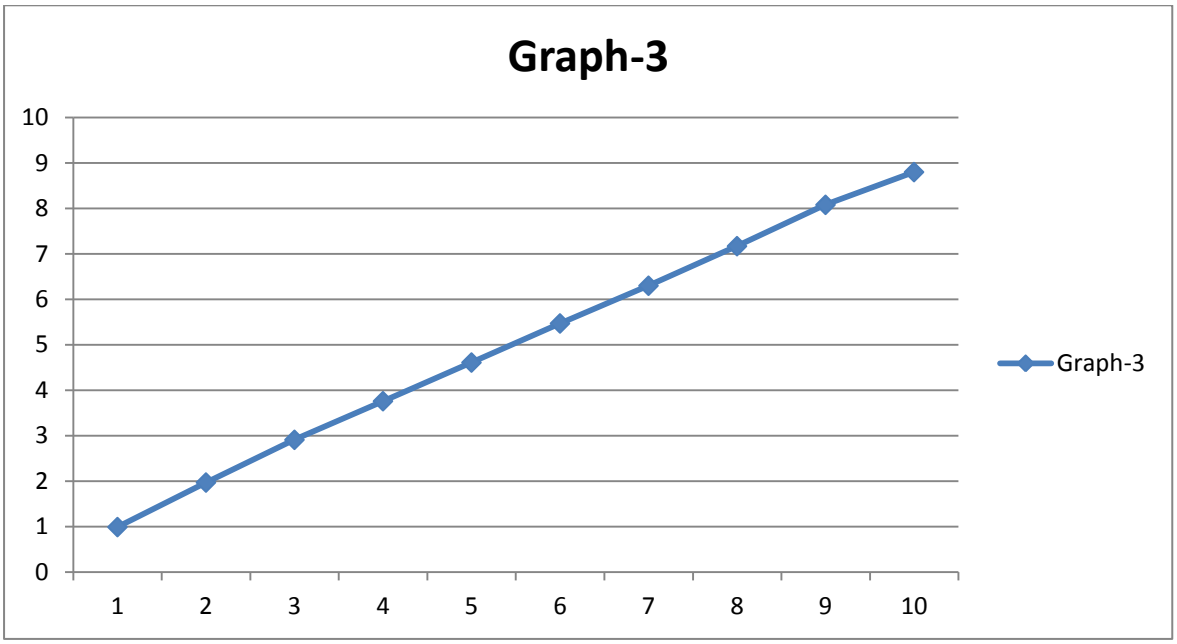
Other parameters are same as specified above and main parameter set.



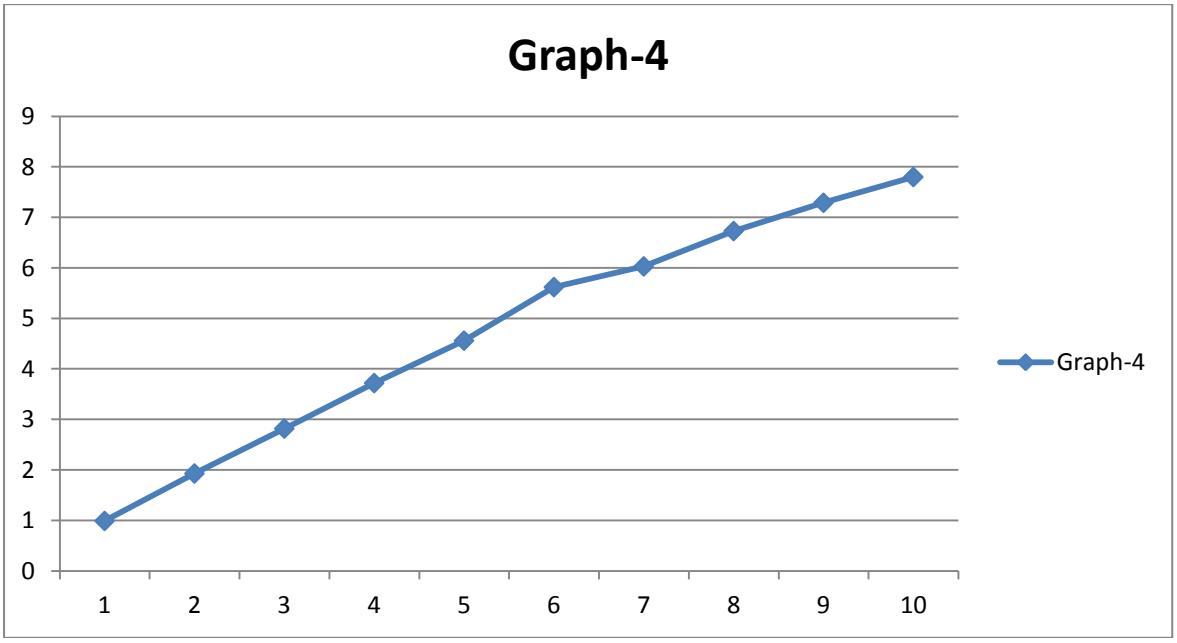
2) Number of Nodes = 20 sources nodes + 1 Sink Node



3) Number of Nodes = 50 sources nodes + 1 Sink Node



3) Number of Nodes = 100 sources nodes + 1 Sink Node



Reasoning

Observation Result :

1) The blocking probability increases with the increase in number of sent copies. For the small number of nodes, the ratio increases almost linearly with the increase in repeat count.
2) The blocking probability increases slowly for large number of nodes. Further, the ratio (P) increases with lesser rate as we increase repeatCount for large number of nodes.

Analysis

1) For the small number of nodes, the blocking probability increases almost linearly with the increase in repeatCount. However the percentage of total packet recieved/sent packets decreases little bit but the value of not significant. The reason for this small decrease is the TxTx/RxRx collision at sender and receiver side. When the repeat count increases, the receiver (sink node) tries to receive the large number of packets and hence results in more collisions [even more sometime, MAC is already being busy in processing earlier packet and hence simply discards the newly received packet]. Similar behaviour on sender side.
But as the number of nodes is less, the effective RxRx/TxTx collision is not so much and hence the increase in the value of P is almost linear but congestion becomes significant when number of nodes becomes higher.
2) The congestion in channel becomes significant with increase in number of nodes and hence blocking probability increases much slowly. Here, as the number of nodes are higher, both sender tries to sent multiple copies of the packet in small duration and hence it results in more packet drop. Similarly, the receiver also receives very large number of packets in small duration[0.02s] and hence it results in more packet drop.(either due to RxRx/TxTx collision or deliberate drop by mac layer as it is already busy processing other packets). If the number of nodes and repeat increases beyond certain limit, it may result in decrease in value of P.

Conclusion

The discussed protocol is modified protocol version of mac-simple to improve the chances of successful delivery. The basic approach applied is the use of retry counters. In the given problem scenario, there is single antenna on sender side and hence it can't receive acknowledgement of packet. And hence the idea is to send the duplicate copies of same packet to hope for that at least one copy will be received successfully.
The count of retry counters were controlled form tickle script and modified in the range of [1-10] to observer the received packet patterns. Our comprehensive simulation results, graphs and reasoning for achieved results shows that proposed protocol efficiently achieves the goal. The observed ratio patterns of the P clearly indicates that the reception ratio can certainly be increased with the increase in number of sent copies but if the repeatCount increases beyond the optimal value, it results in more collision and hence more packet loss.

References

http://www.isi.edu/nsnam/ns/tutorial/
http://www.isi.edu/nsnam/ns/
*http://www.isi.edu/nsnam/ns/ns-documentation.html_*
ns2.35/mac/mac-simple.cc,channel.cc,mac-802_11.cc