Congestion Control Using
Saturation Feedback for
Multihop Packet Radio Networks

A Thesis Presented to
The Faculty of the College of Engineering and Technology
Ohio University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by

Donald E. Carter

November, 1991
Congestion Control Using Saturation Feedback for Multihop Packet Radio Networks

Donald E. Carter

November 19, 1991
Contents

1 The Packet Radio Network 1

1.1 Summary of Packet Radio Protocol ............ 2

1.1.1 Spread Spectrum Transmission ............... 2

1.1.2 Packet Acknowledgement Protocols .......... 4

2 Flow Control to Minimize Congestion 7

2.1 Congestion Control Algorithms ............... 7

2.1.1 Preallocation of Buffers .................... 8

2.1.2 Packet Discarding ......................... 8

2.1.3 Isarithmic Congestion Control ............... 9

2.1.4 Flow Control .............................. 9

2.1.5 Choke Packets ............................ 10

2.2 The Adaptive Flow Control Algorithms ........ 10

2.2.1 Theoretical Limits on Throughput with Fixed Transmit Probability ................. 11
2.2.2 Adaptive Transmit Probability ........................................ 11
2.2.3 Actual Network Throughput .......................................... 12
2.2.4 Minimum Queue Length Routing ..................................... 13
2.2.5 The Saturation Feedback Algorithm ................................. 13
2.2.6 Channel Access Protocol for Dedicated Channel Acknowledgement with Saturation Feedback .......................... 18

3 The MHSS Simulation Model ............................................ 20
  3.1 Modifications to the MHSS Computer Code ....................... 21
    3.1.1 Modifications to MHSS1.FOR ...................................... 21
    3.1.2 Modifications to SCHEDULE1.FOR .............................. 22
    3.1.3 Modifications to SEND DATA.FOR ............................. 22
    3.1.4 Modifications to SEND ACK.FOR .............................. 23
    3.1.5 Modifications to PROC RX ACK.FOR .......................... 24
    3.1.6 Modifications to GLOBAL.TXT ................................. 24
    3.1.7 Modifications to NODES.TXT .................................... 24

4 Tests of the Modified Code ........................................... 26
  4.1 Original Tests with Fixed Saturation Thresholds and Saturation Delays ...................................................... 26
  4.2 Results for Fixed Threshold and Delay using Random Minimum Hop Routing .............................................. 27
4.3 Optimum Delay dependent on Network Connectivity .... 31
4.4 Number of Neighbors Indicates Network Connectivity in the
   Vicinity of Each Node ............................. 35
4.5 Adaptive Determination of the Optimum Delay Based On
   Number of Neighbors .............................. 37
   4.5.1 Tests of the Adaptive Delay Calculation .......... 37
4.6 Conclusions of the Effectiveness of the Algorithm ......... 41

5 Tests Combining Saturation Feedback Techniques and the
   Minimum Queue Length Routing Algorithm ............ 42
   5.1 Effects of Varying the Threshold at Which Saturation is Re-
       ported ........................................... 43

6 Conclusions and Suggestions for Further Study ........... 47

A Listings of Modified Modules ......................... 49
   A.1 MHSS.FOR ........................................ 50
   A.2 SCHEDULE1.FOR ................................. 53
   A.3 SEND DATA.FOR ................................. 55
   A.4 SEND ACK.FOR ................................ 58
   A.5 PROC RX ACK.FOR ............................. 60
   A.6 GLOBAL.TXT ................................. 64
List of Figures

2.1 Sample network topology. ........................................ 17
2.2 Channel access protocol. ....................................... 19

4.1 Message delay versus sat delay. .............................. 28
4.2 Mean queue length versus sat delay. ........................ 29
4.3 Message delay versus sat threshold. ......................... 30
4.4 Message delay versus sat delay at 10% connectivity. ...... 32
4.5 Message delay versus sat delay at 15% connectivity. ...... 33
4.6 Message delay versus sat delay at 20% connectivity. ..... 34
4.7 Optimum sat delay versus number of neighbors. ........... 36
4.8 Sat delay (neighbors) at 10% connectivity. ................. 38
4.9 Sat delay (neighbors) at 15% connectivity. ................. 39
4.10 Sat delay (neighbors) at 20% connectivity. ............... 40

5.1 Message delay versus lambda for various threshold functions. . 46
A packet radio network is a collection of radio receivers and transmitters that have the capability to receive and transmit packets of digital information to surrounding nodes, and also have the capability to receive and transmit packets of information to distant nodes by forwarding packets to the next node on a path towards the ultimate destination node of the packet.

In this study the nodes are assumed to be mobile and dynamic in that nodes and radio links may appear and disappear randomly. This can happen in a packet radio network because of land topology, equipment failure, transmission propagation variations, etc. The packet radio simulation computer code **MHSS** (Multi Hop Spread Spectrum) can simulate these conditions and analyze the performance of a packet radio network model. This computer code was used to evaluate the congestion control algorithm which is the subject of this research. A comprehensive description of research in the
area of packet radio and packet radio network modeling is presented in the January 1987 issue of the *Proceedings of the IEEE Special Issue on Packet Radio Networks* [1],[2].

### 1.1 Summary of Packet Radio Protocol

The radio network used in this model is termed **receiver directed, code and time division multiple access**. The transmitter employs a technique known as *spread spectrum* [3] in which the information is transmitted over a wider frequency bandwidth than would be required to transmit the information.

#### 1.1.1 Spread Spectrum Transmission

In spread spectrum transmissions the bandwidth used to transmit the information is wider than would be normally necessary. This might seem wasteful at first but some significant advantages can be gained from the spread spectrum technique.

One advantage of spread spectrum is that the noise sources on a specific frequency are not as likely to affect the communication reliability because each data bit is spread across a wide frequency range and noise on any particular frequency is only a small part of the information used to reconstruct the data. In general, random noise on specific frequencies will average to-
wards zero and the data will average to the transmitted data bit. Also, error correcting codes are employed to correct data errors when possible. This has the obvious military advantage that jamming a particular transmission frequency will not disrupt communications.

One way to visualize the spread spectrum technique is to think of a logical exclusive or of the data and a random appearing but known bit pattern that is higher in frequency than the data pattern. This known but apparently random bit pattern can be thought of as a spreading key. This key then effectively transmits the data over a broad spectrum of frequencies (broad band).

If the intended receiver has the key it can effectively reconstruct (demodulate) the transmitted data pattern. This is known as receiver capture. If the receiver averages the data received during a bit slot it can effectively reconstruct the transmitted data in even adverse conditions because the noise will be filtered out.

This ability to capture data transmitted with the receiver key also allows the receiver to reject transmissions spread with another key. The transmissions with other keys will tend to average to zero while the transmissions with the proper key will average to the transmitted data pattern. This allows multiple transmissions to occur concurrently within a given spread spectrum
bandwidth. This gives rise to the term **Receiver Directed, Code and Time Division Multiple Access (RD-C/TDMA)**

Using the RD-C/TDMA protocol multiple transmissions of information to different receiving nodes can occur simultaneously therefore recovering some of the bandwidth that would have been wasted. Spread spectrum adds the benefit of more reliability and less vulnerability to jamming.

### 1.1.2 Packet Acknowledgement Protocols

The transmitting node of a data packet needs to know if the transmitted packet was received successfully in order to determine whether to queue the data packet for retransmission. RD-C/TDMA packet radio networks pose some problems for packet acknowledgement protocols. If the transmitting node can listen for retransmission of the data packet it has an implicit acknowledgement of reception. In RD-C/TDMA networks the transmitting node does not know the code channel that the data packet will be relayed on and for this reason a special acknowledgement packet is needed that is sent from the receiving node to the transmitting node to indicate reception of the data packet.

If acknowledgement transmissions are optimized for reliability and efficiency by somehow making acknowledgement transmissions different than data transmissions they would be open for intentional jamming. For this
reason the acknowledgement transmissions should be indistinguishable from data transmissions.

If acknowledgement transmissions are merely queued for transmission to the sending node they will be competing against collisions from other transmissions thus causing as many failures in acknowledgement transmissions that occur in data transmissions. The MHSS simulation code used in this work makes use of some acknowledgement protocol enhancement techniques.

**Dedicated Acknowledgement Timeslot**

Since in any transmission success requires that the node that the transmission is destined to is in the receive state the first optimization would be for the node that sent the data packet to expect an acknowledgement transmission at a specified timeslot following the transmission timeslot. The MHSS simulation code expects acknowledgement transmissions in the timeslot immediately following the data transmission timeslot. This protocol guarantees that the node to which the acknowledgement is sent is at least listening for the acknowledgement.

**Dedicated Acknowledgement Code Channel**

Another acknowledgement optimization used in the MHSS simulation code is for acknowledgement transmissions to occur on a different code channel
than data transmissions. This means that each node has a dedicated data
code channel and acknowledgement code channel that is known by all nodes
transmitting to that node. In this way acknowledgement transmissions are
not competing against data transmissions and the only transmissions on the
acknowledgement code channel will be from the acknowledging node.

These optimizations help significantly to guarantee the success of ac-
knowledgement transmissions with very high probability and thus minimize
duplicate packet transmissions caused by failures in acknowledgement trans-
missions.
Chapter 2

Flow Control to Minimize Congestion

Under various conditions of instantaneous load and certain topological conditions some nodes in a packet radio network can become overloaded with traffic and become congested. Various methods have been used to minimize or in some cases totally eliminate this congestion.

2.1 Congestion Control Algorithms

Congestion is a condition of severe delay caused by an overload of data traffic in a network[4]. When congestion occurs provisions must be made to attempt to deal with the problem using some strategy to reduce the traffic to an acceptable level.

Tanenbaum [5] discusses five strategies that have been used for congestion control. Each algorithm trades off performance for control of possible
2.1.1 Preallocation of Buffers

Preallocation of buffers employs a strategy where each virtual circuit is started with a call request packet that winds its way through the network allocating buffers as it propagates towards its final destination. When the packet reaches its destination a circuit has been set up and the actual data transfer takes place using the path and data buffers set up by the call request packet. This algorithm completely eliminates congestion because the necessary resources are allocated before any data transmission takes place. This technique increases the communication delay because no data transmission can take place until the complete circuit is set up. Network resources are also dedicated and may be idle for much of the time that they are allocated and therefore reduces substantially the maximum throughput of the network. This technique is not suitable for packet radio networks because it requires a stable topology and the packet radio network is inherently dynamic.

2.1.2 Packet Discarding

Another congestion control algorithm is packet discarding. Instead of allocating network resources in advance, no attempt to reserve any resource is made in advance. If a node cannot accept packets that arrive it simply ignores...
them. This implies that packets might need to be sent many times in order to reach the final destination. Various methods have been studied to decide which packets to discard and which packets to keep. This algorithm can increase packet delay and in the case of packet radio networks the additional transmissions can cause interference that disrupt other connections.

2.1.3 Isarithmic Congestion Control

Congestion occurs because there are more packets to transmit than the network can accept. This congestion can be localized or global in a packet radio network. The isarithmic control algorithm limits the congestion in a subnet by issuing permits that must be obtained before a packet can be sent. By limiting the number of permits the total traffic can be controlled and thus congestion minimized. This technique does not eliminate localized congestion however because many nodes may get permits and simultaneously attempt to communicate with the same node.

2.1.4 Flow Control

Flow control is an attempt to distribute the network traffic evenly between the resources available. Flow control can help keep one node from saturating another by limiting the rate at which it sends data to any one node. Flow control can waste network resources and add delay by limiting communica-
tions to a node to a predetermined maximum rate even though the node could accept data at a higher rate because other nodes might be idle at this particular time. This technique would be better suited in a network where the traffic were not bursty like it is in most networks.

2.1.5 Choke Packets

Limiting traffic to a particular node by flow control to limit congestion does so at the expense of reducing throughput. One way to limit the congestion is to allow data to be transmitted without control until a node reports that it is being heavily used and at that time sends a choke packet back to the node reporting that further communications should be throttled and limited to a lower rate for some time.

The subject of this research was to add and test a congestion control algorithm to the MHSS simulation code written by Dill [6]. The method that I chose was a combination of the flow control and choke packet techniques. Details of this technique are discussed later in this chapter referring to the Saturation Feedback Algorithm.

2.2 The Adaptive Flow Control Algorithms

In a real packet radio network it is desirable and necessary to make the flow control algorithms adaptive because of the bursty and inconsistent nature of
the network. The network connectivity and data rates vary with time and
the network should be able to adjust and allow the network resources to be
as fully utilized as possible while protecting against destructive congestion.
The adaptive techniques used by the MHSS simulation model are discussed
in the next sections.

2.2 Theoretical Limits on Throughput with Fixed Transmit Probability

From the work of Dill it was found that for an analytical model of a multihop
CDMA network using receiver directed codes and a random Bernoulli connec-
tivity topology that the maximum average network throughput in pack-
ets/sec., $\Gamma_{\text{max}}$, was

$$\Gamma_{\text{max}} = \frac{0.44N}{K_s}$$  \hspace{1cm} (2.1)

where $N$ is the number of nodes in the network, $K_s$ is the average number
of transmissions initiated by each arriving message. The constant 0.44 is the
optimal transmit probability for channel access protocol.

2.2.2 Adaptive Transmit Probability

An adaptive algorithm to improve network throughput was devised by Dill
which allows the transmit probability of each node to vary, within limits,
to a value that depends on the ratio of his current transmit queue length to
the average queue length of his neighbors. This allows nodes that have more packets to transmit than their neighbors more transmit opportunities. All simulations in the current work were done with adaptive transmit probability with the limits set at 0.1 minimum and 0.9 maximum. This allows the transmit probability to adapt to network conditions and deviate from the optimum value of 0.44 for a perfectly homogeneous network and thus improve network overall throughput.

2.2.3 Actual Network Throughput

The actual throughput of a network may fall short of the maximum theoretical throughput because of bottlenecks at certain nodes that may, under certain topological conditions, be a link that is needed often to reach other areas of the network. When these nodes start to become heavily used or saturated the performance of the network can degrade quickly. When many nodes attempt to communicate with a single node, interference makes it more difficult for a saturated node to move its traffic towards its destination. Also, a saturated node should attempt to transmit more often than receive in an attempt to rid itself of some of the packets in its transmit queue.

In the work of Dill it was found that for connectivity fractions of 20% or greater for a Bernoulli random network of 100 nodes that the network performance approximated that predicted by the analytical model. When the
routing path was selected randomly from a list of minimum hop paths towards the destination of the packet. When the connectivity fraction dropped to less than 20% it was found that the throughput of the network began to fall short of the theoretical limit. This was found to be caused by the fact that many paths had only one minimum hop path. In these cases many nodes in the network may be lightly loaded while a few links are extremely congested.

2.2.4 Minimum Queue Length Routing

When the number of minimum hop paths to a desired destination was greater than unity, Dill used a method which chose the path that had the smaller transmit queue length. This adaptive route selection algorithm works well to evenly distribute the network traffic to the least busy minimum hop path. This method used the packet acknowledgement transmission to return the size of the receiving nodes transmit queue therefore this method required little transmission overhead to implement. It was up to the transmitting node to keep track of the queue lengths of the surrounding nodes.

2.2.5 The Saturation Feedback Algorithm

The subject of this thesis is to further improve the homogeniety of the network traffic and reduce congestion where certain topologies have large amounts of traffic routed through certain nodes. The method described here
attempts to combine the flow control and choke packet techniques previously discussed. This new technique termed *Saturation Feedback* is triggered by the transmit queue length and attempts to adaptively control the delay between transmissions to a saturated node after the receipt of a choke packet. It is hoped that this new method will improve the delay and throughput characteristics of the network where some topologies give rise to congestion points within the network. This algorithm must be adaptive and attempt to route traffic through a less congested area when possible.

The minimum hop, minimum queue length algorithm improves network homogeneity when the network connectivity is 20% or greater but ceases to be effective at lower connectivities because it has fewer minimum hop nodes to choose from. When the number of minimum hop paths equals one, the algorithm offers no improvement at all. If only one minimum hop path exists this algorithm will always transmit and likely just cause network congestion in the vicinity of a saturated node.

The algorithm that I propose attempts to deal with this problem by not transmitting to a node that is reporting that it has more traffic than it can handle at this time (i.e., it is saturated). The idea is that a node should not transmit to a node that has reported that it is saturated thus not causing interference in the vicinity of the saturated node which may limit the satu-
rated nodes ability to successfully send packets on their way. This might also interfere with acknowledgement packets destined to the saturated node thus even a packet that has been successfully forwarded might get retransmitted by the saturated node because no acknowledgement was received.

The node that has selected a packet destined to a saturated node should, instead of transmitting this packet, place the packet at the end of its transmit queue and select another packet in the queue in an attempt to locate a packet that is destined to a non-saturated node. If no packet can be found that is destined to a non-saturated node then do not transmit a packet and set the next state to receive.

Another benefit would be that while a node is waiting to transmit to a saturated node its buffer may begin to fill with traffic destined to the saturated node. When the buffer begins to fill this node will eventually report a saturated status to the sending nodes. Thus the effects of a saturated node can ripple back through the network and other available paths around the point of congestion can be tried. This alternate path selection occurs because each node attempts to find a minimum hop path to the destination of the packet it is transmitting. If it picks a saturated node as the minimum hop path it is placed back at the end of the transmit queue and another packet is checked. If another minimum hop path exists it is likely that it will be
discovered by the path selection algorithm which either makes its selection randomly of the possible minimum hop paths or picks the minimum hop path with the smaller transmit queue.

The proposed algorithm is summarized below:

- If a packet is destined to a saturated node, then don’t transmit this packet. Instead, place this packet at the end of the queue and try to find another packet to transmit. This eliminates transmissions that are likely to add to the congestion at a saturated node.

- If no packet in the transmit queue is destined to a non-saturated node then enter the receive state. This attempts to minimize congestion in the vicinity of the saturated node.

- As queues begin to fill with traffic destined towards a saturated node, the saturation information will ripple back through the network and alternate routes will be attempted at each node if another minimum hop path exists.

As an illustration of how the saturation feedback algorithm might work, assume a network topology as in figure 2.1.

Node A is receiving many packets destined to node D. Using random minimum hop routing node A would distribute the packets destined for node
Many Packets Destined For Node D

Many Packets Destined For Node E

Figure 2.1: Sample network topology.
D evenly between nodes B and C. If node C is also receiving heavy traffic destined for node E it could easily become congested or "saturated". To help alleviate this problem, when node C sends node A a "saturated" message node A will route all traffic destined for node D thru node B. This action will then allow node C to send its packets on and also allow packets destined from node E via node C to continue.

2.2.6 Channel Access Protocol for Dedicated Channel Acknowledgement with Saturation Feedback

The channel access protocol for a node in each timeslot for the modified MHSS simulation code is shown in figure 2.2. This protocol takes into account the possibility that a node could be saturated and is a modification of the original MHSS channel access protocol. The protocol does guarantee that if an acknowledgement is to be transmitted it is transmitted on a dedicated acknowledgement code channel at a specific timeslot.
BEGIN
IF obligated to listen for acknowledgement, THEN
listen for acknowledgement on own ack channel.
ELSE IF obligated to transmit an acknowledgement, THEN
transmit acknowledgement on ack code channel.
ELSE
examine transmit queue and look for messages to transmit

IF the transmit queue is empty, THEN
enter the receive data state for this timeslot.
ELSE IF the transmit queue is not empty, THEN
roll a random number to decide whether to transmit or receive.

IF the decision is to transmit, THEN

IF the message at the top of the transmit queue
is destined to a saturated node, THEN
place this message at the end of the queue and loop through all packets
looking for a packet destined to a non-saturated node.
ELSE transmit the message to the node chosen by the routing algorithm.
END IF

ELSE IF the decision is to receive, THEN
enter the receive state for this timeslot.
END IF
END IF

END IF
END IF
END IF
END

Figure 2.2: Channel access protocol.
Chapter 3

The MHSS Simulation Model

In order to evaluate the performance of the adaptive algorithm discussed in the next chapter a computer model of the network was used. This model, Multi-Hop Spread Spectrum (MHSS) was used. Details of the modeling code can be found in [6].

The MHSS model can simulate a multihop, CDMA/TDMA packet radio network of up to 100 nodes with varying network topology and traffic patterns. This model implements the exact routing, transmission, acknowledgment and channel access protocol for each node timeslot by timeslot. Since some of the networks protocols are adaptive and nonlinear a purely analytical model would be too complicated and not practical.

The MHSS simulation code was written in VAX FORTRAN, version 4.0, and consisted of approximately 1600 lines of code. A complete description of the original code can be found in [6] dissertation in Chapter 5. This program
was transferred to a Vaxstation 3200 computer where the original code was modified to implement the algorithm and the simulations were run.

The MHSS simulation code was used to test the saturation feedback algorithm under various conditions of connectivity, message arrival rate, adaptive transmit probability, etc.

### 3.1 Modifications to the MHSS Computer Code

After extensive study of the code, it was necessary to modify 7 of the original 38 modules to implement the proposed algorithm.

#### 3.1.1 Modifications to MHSS1.FOR

The module MHSS1.FOR is the main program of the MHSS simulation code. This module initializes the various arrays and calls the simulation subroutines to perform the simulation based on the users input parameters. This module has the ability to make several simulation runs while iterating over a range of input parameters such as iterating the packet arrival rate from some user specified minimum to maximum value with a user specified increment.

This code was modified various times to fix or vary certain aspects of the simulation desired. Another modification to this module was to initialize an array that contained the number of neighbors that each node has. This
array was used to calculate the adaptive optimum delay used when a node reported that it was saturated.

3.1.2 Modifications to SCHEDULE1.FOR

The module SCHEDULE1.FOR schedules the activity that each node will perform for each time slot. Each node can be in one of four states; send data, receive data, send acknowledgement, or receive acknowledgement.

The modification to this module consisted of checking the saturation counters of each node's neighbors and if not zero decrementing the counter by one for each time slot processed. These saturation counters are used to delay transmission to a saturated node when the node has reported that it is saturated.

3.1.3 Modifications to SEND DATA.FOR

The module SEND DATA.FOR sends a data packet from a node's transmit queue when scheduled to transmit in the current timeslot. A packet is selected from the transmit queue and the routing path is determined by the mode specified by the user as random minimum hop or minimum queue length minimum hop.

This module was modified to check the saturation counter of the node that this packet is destined to and if this node has reported saturated and
the counter has not reached zero then this packet is placed at the end of the transmit queue. Another packet is then selected from the transmit queue and its saturation counter is then checked. This process continues until a packet is found that is destined to a node that is not saturated. If no packet in the transmit queue is destined to a node that is not saturated then no transmission is made and the next state is set to receive data.

3.1.4 Modifications to SEND ACK.FOR

The module SEND ACK.FOR sends acknowledgement packets when a data packet was successfully received. These acknowledgement packets are used to inform the transmitting node that the data packet that it sent to this node has been received. This transmission also contains information that indicates the saturation status for this node. In this way the saturation status can be relayed to the transmitting node causing no additional overhead on the data channel.

This module was modified to check the current number of packets in the receiving nodes transmit queue and if this number is greater than the parameter SAT THRESHOLD then the saturation flag is set in the acknowledgement packet. This indicates to the transmitting node that it should delay further transmissions to this node.
3.1.5 Modifications to PROC RX ACK.FOR

The module PROC RX ACK.FOR receives the acknowledgement packets from the node that it last sent a data packet to. It then marks this packet as successfully transmitted and removes it from the transmit queue.

This module was modified to check the saturation flag contained in the acknowledgement packet. If this saturation flag is true an adaptive SAT DELAY is calculated and recorded for this node. No further transmissions to this node will occur until this SAT DELAY has elapsed. The amount of delay is determined adaptively for each node and is a function of the number of neighbors for this node.

3.1.6 Modifications to GLOBAL.TXT

The module GLOBAL.TXT is a common declaration containing all of the high level control information.

The SAT THRESHOLD and SAT DELAY declarations were added to this module. These contain the user specified threshold at which saturation is reported and a factor used for the adaptive delay calculation, respectively.

3.1.7 Modifications to NODES.TXT

The module NODES.TXT contains the definition of the database for each node. The definitions of the saturation flag and the array of saturated nodes
were added to this module. Listings of the modified modules are included in Appendix A.
Chapter 4

Tests of the Modified Code

4.1 Original Tests

The first tests of the modified MHSS code were a series of runs to search for the optimum performance for various fixed thresholds (the number of packets in the receiving nodes transmit queue above which the node reports saturated) and fixed delays (the number of timeslots that a node delays transmission of a packet to a saturated node). The MHSS code limited the maximum number of packets in a transmit queue to 200 packets.

Initially the threshold was set to 20 (10% of the maximum queue length) and the delay was varied from 0 to 220 timeslots. It was found that the interval message delay, average queue length, and the standard deviation of the queue length improved as the saturation delay was increased and reached a maximum improvement with a delay of 180 timeslots. After this value of
delay the network performance degraded rapidly with increasing delays. A
graph of the delay results are shown in Figure 4.1 and a graph of the mean
queue length and standard deviation are shown in Figure 4.2.

After it was determined that the best performance for a threshold of 20
was obtained at a delay of 180 the threshold was varied with a fixed delay of
180. It was found that the initial guess of 20 for the threshold was a good
choice. A graph of mean interval delay versus threshold for delay fixed at
180 is shown in Figure 4.3.

4.2 Results for Fixed Threshold and Delay
using Random Minimum Hop Routing

Compared to the unmodified code using random minimum hop routing for a
connectivity of 20% the following improvements were observed. The average
interval message delay improved from 633.5 to 403.6 timeslots. The average
queue length improved from 32.8 to 21.0 packets. The standard deviation of
the average queue length improved from 12.9 to 4.7 packets.

These improvements were substantial and show that the algorithm works
effectively to improve network performance and traffic homogeniety.
Figure 4.1: Message delay versus sat delay.
Figure 4.2: Mean queue length versus sat delay.
Lambda = 5.25, Sat_Delay = 180, Connectivity = 20%

Figure 4.3: Message delay versus sat threshold.
4.3 Optimum Delay dependent on Network Connectivity

The previous tests were simulated with a network connectivity of 20%. After additional simulations for different network connectivities it was found that the optimum delay was dependent on network connectivity.

In a real packet radio network each node does not know the connectivity of the network. Since the optimum delay value is strongly dependent on the network connectivity the algorithm would not be very useful if it could not be made adaptive to the network connectivity. If the wrong value was chosen for the delay parameter it could adversely affect the network performance.

In general it was found that performance improved with increasing delay up to a point termed threshold of instability after which the network performance degraded rapidly with increasing delay. Figures 4.4, 4.5, and 4.6 show the curves of mean interval delay versus delay for connectivities of 10%, 15%, and 20% respectively.

From these curves the thresholds of instability were determined for each of the simulated connectivities. It was found that as the connectivity of the network increased, the optimum delay also increased. Intuitively, it seems logical that if a node has more neighbors from which to choose another route around a saturated node, and the choice is made by a random method, then
Figure 4.4: Message delay versus sat delay at 10% connectivity.
$\text{Lambda} = 4.80, \text{Sat\_Threshold} = 20, \text{Connectivity} = 15\%$

Figure 4.5: Message delay versus sat delay at 15% connectivity.
Figure 4.6: Message delay versus sat delay at 20% connectivity.
for higher connectivities packet transmissions to a saturated node should be
delayed longer thus allowing more chance to choose another possibly better
route around the saturated node.

4.4 Number of Neighbors Indicates Network Connectivity in the Vicinity of Each Node

It was observed that in order for this algorithm to be useful in a real packet
radio network it would be necessary to estimate the network connectivity in
order to determine the optimum value for the delay.

For a network of 100 nodes and an average connectivity of 10% each node
would have an average of 10 neighbors. A network connectivity of 15% would
have an average of 15 neighbors per node, 20% 20 neighbors, etc. A plot of
threshold of instability versus number of neighbors is shown if Figure 4.7.

It was found that a straight line would fit the points well. This implied
that each node could estimate the optimum delay value based on the number
of neighbors it had. This property would allow the algorithm to be adaptive
to the packet radio network and thus could be used effectively in a real
distributed network where the average connectivity was unknown.
Figure 4.7: Optimum sat delay versus number of neighbors.
4.5 Adaptive Determination of the Optimum Delay Based On Number of Neighbors

The MHSS code was modified again to implement the adaptive delay calculation algorithm. Since the optimum delay seemed to be a linear function of each nodes' number of neighbors, a section of code was added to calculate the delay. This delay was calculated as a constant multiplied by the number of neighbors this node had; i.e.,

\[ D = kN \]  

where \( D \) was the optimum delay in timeslots, \( N \) was the number of neighbors that this node has, and \( k \) was an arbitrary constant. It appeared that an approximate fit to the points would be

\[ y = 10N + 20 \]

4.5.1 Tests of the Adaptive Delay Calculation

A series of MHSS simulations were run where the constant \( k \) was iterated from 0 to 15 for connectivities of 10%, 15%, and 20%. The results of these simulation runs are summarized in figures 4.8, 4.9, and 4.10 for their respective connectivities.
Figure 4.8: Sat delay (neighbors) at 10% connectivity.
Figure 4.9: Sat delay (neighbors) at 15% connectivity.
Figure 4.10: Sat delay (neighbors) at 20% connectivity.
These graphs reveal that under actual network simulations where each node is calculating its own delay as a function of its neighbors that a good choice for the delay for various connectivities would be

\[ D = 8N \]  

(4.3)

4.6 Conclusions of the Effectiveness of the Algorithm

It was found that the new adaptive algorithm improved the network delay and queue length characteristics significantly when using the random minimum hop routing algorithm. Improvements on the order of 40\% were found in the 20\% connectivity MHSS simulations.
Chapter 5

Tests Combining Saturation Feedback Techniques and the Minimum Queue Length Routing Algorithm

All simulations up to this point were done using random minimum hop routing techniques. Under these conditions significant improvements in the network efficiency were obtained using the saturation feedback algorithm.

A series of initial simulations were run using the new algorithm in conjunction with Dill's minimum queue length routing algorithm. The results indicated that marginal benefit was obtained by combining the saturation feedback and minimum queue-length routing techniques.

After further thought it seemed that if the minimum queue length routing algorithm was selecting the node with the smallest queue length that only small second order improvements would be obtained. In general, if the node
selected was saturated and its queue length was the smallest of the available minimum hop paths then all other minimum hop paths would also be saturated. The only benefit that could be obtained was that of not transmitting in the vicinity of the saturated node. Some small improvements were indeed seen but it could not be asserted that they were statistically significant the results depended heavily on the network topology and the random number seeds chosen.

Since the simulations were done with arrival rates that are slowly increasing and forcing the network into saturation the feedback obtained from the saturated node probably couldn’t be used effectively because the entire network was essentially saturating homogeneously. The technique might offer more significant improvement if tested in a more dynamic situation where the entire network wasn’t being saturated homogeneously.

5.1 Effects of Varying the Threshold at Which Saturation is Reported

A series of simulations were tried where the threshold was varied and also made adaptive. It was thought that by altering the point at which saturation was reported might affect how well the algorithm performed. As the network was slowly driven into saturation the Message Delay was monitored as the network entered saturation. The various saturation curves were plot-
ted for the following definitions of saturated (the point at which saturation is reported).

• No saturation is ever reported ($T = 1000$).

• Saturation is reported at 20 packets in the transmit queue ($T = 20$).

• Saturation is reported when the receiving node has $2.0$ times the packets in his transmit queue as does the sender ($T = 2.0 \times \text{Sender}$).

• Saturation is reported when the receiving node has $1.5$ times the packets in his transmit queue as does the sender ($T = 1.5 \times \text{Sender}$).

• Saturation is reported when the receiving node has $1.5$ times the packets in his transmit queue as does the sender and will transmit if there are only saturated nodes to send packets to ($T = 1.5 \times \text{Sender and Xmit}$).

• Saturation is reported when the receiving node has $1.5$ times the packets in his transmit queue as does the sender or more than $20$ packets in the queue ($T = 1.5 \times \text{Sender or more than 20}$).

• Saturation is reported when the receiving node has greater than $20$ packets in his transmit queue. If he only has packets to send to saturated nodes, send the oldest packet anyway. ($T = 20$ and Xmit).
No significant improvements were found by this variation or by making the threshold a function of the ratio of queue lengths between the nodes in communication. Tests of ratios of 1.5 and 2.0 were tried with no significant improvements over the fixed threshold of 20 packets. Figure 5.1 summarizes these tests.
Figure 5.1: Message delay versus lambda for various threshold functions.
Chapter 6

Conclusions and Suggestions for Further Study

The results of the many time consuming simulation runs (each monte carlo simulation run took hours and sometimes days to complete) indicate that the saturation feedback algorithm greatly improves network efficiency when the random minimum hop routing algorithm is employed.

When the minimum queue length minimum hop routing algorithm was employed no significant improvements were indicated using the MHSS simulation where the network packet radio network was slowly driven into saturation. These results speak favorably for the minimum queue length routing algorithm employed by [6] and suggests that this is also a good technique.

The improvements offered by combining minimum queue length minimum hop routing techniques with the saturation feedback technique might be seen in a real packet radio network where the network traffic was bursty and the
network was not homogeneously being driven into saturation. The saturation feedback technique might be able to ripple back information about congested areas and alternate paths might be found more distant from a congested node.

This suggests that additional work could be done by modifying the simulation code to simulate a bursty transient network and study the effect the algorithms have in this type of environment.
Appendix A

Listings of Modified Modules
A.1 MHSS.FOR

C***********************************************************************
C* MHSS1 - This is the main routine of the multi-hop spread spectrum
C* communication network simulation program.
C*
C***********************************************************************

C** Global variable declarations.
include '( global )/nolist'
include '( nodes )/nolist'
include '( runstats )/nolist'
include '( connectivity )/nolist'
integer*4 i
integer*4 which_seeds
real*4 lambda_knee, lambda_temp
real*4 qlavg_0 /0./, qlavg
common / mhss1_local / lambda_temp, qlavg_0

C** Read in control parameters for this run from the input control file
call readinp

C** Main monte carlo loop.
do imc = 1, number_mc
   call setup_rn ! set up random number seeds for this loop.
   ! Generate the hearing, or connectivity matrix.
   if( matrix_type .EQ. 0 ) then
      call bernoulli
   else if( matrix_type .EQ. 1 ) then
      call euclidean
   end if

C** Loop on successively increasing values of lambda.
   lambda_temp = lambda_low
   lambda = lambda_low
   Do while( lambda .le. lambda_hi )
      !Output progress to terminal for monitoring of job.
      write( 6,999 ) imc, lambda, sat_delay
      999 forrmat(1x,'Monte Carlo # ',i4,' Lambda = ',f6.2,' ST= ',i5)
c      do while( sat_delay .le. 20 )

      !Output progress to terminal for monitoring of job.
      write( 6,999 ) imc, lambda, sat_delay
      999 forrmat(1x,'Monte Carlo # ',i4,' Lambda = ',f6.2,' ST= ',i5)
! Initialize parameters for this trial.
call setup_poisson !initialize the poisson distribution.
call initmq       !initialize the message queue.
call communicants !fill the communicant table.
call routing      !fill the routing table.
call matstats     !generate statistics on the connectivity.
!initialize link qualities
   do i = 1, 100
      node(i).tosscount = 0
   end do
   do j = 1, 100
      node(i).quality(j) = 1.0
      if(connect(i,j) .ne. 0) then
         node(i).neighbors = node(i).neighbors + 1
      end if
   end do
!
! Run the simulation.
now = -warm_up !run initial slots to fill queues.
mga = 0
mda = 0
sum_delay = 0.
sum2_delay = 0.
sumphops = 0.
sumtx = 0.
write(30,111)
111 format( '1' )
do jj = 1, 5
   write(30,1001) header_text(jj)
1001 format( a132 )
end do
if( traffic_type .eq. 0 ) then !initialize queues at time 0
call TRAFFIC
else if( traffic_type .eq. 1 ) then
call traffica
end if
do now = now+1, time_limit !run a bunch of timeslots
   call schedule1 !schedule node activity for this slot
   call channel1 !resolve the channel for this slot.

! Collect and output the statistics for this run.
if ( jmod( now, sampling_interval ) .eq. 0 ) then
call runstats

!Output progress to terminal for monitoring of job.
c write( 6,998 ) imc, lambda, now
998 format( 10x, 'MC = ', i4, ' lambda = ', f6.2, ' Time =', i9,
+ ' timeslots.' )
end if

if( traffic_type .eq. 0 ) then
  
  if( jmod( now, 10 ) .eq. 0 ) then
  do k = 1, nbr_nodes
    qlavg = qlavg + node(k).tq_length
  end do
  qlavg = qlavg / nbr_nodes
  lambda = amin1(lambda_hi, + amax1( 0.1, (lambda + 30. + qlavg_0 - 2.*qlavg) ))
  lambda = 5.0
  qlavg_0 = qlavg
  call setup_poisson
  end if

  call TRAFFIC
else if( traffic_type .eq. 1 ) then
  call traffica
end if

end do !timeslot loop

lambda_temp = lambda_temp + lambda_inc
lambda = lambda_temp
end do !lambda increment loop
c sat_delay = sat_delay + 1
c end do !end sat_delay inc loop

c !end monte carlo loop.
c call summary_stats
stop
end
A.2 SCHEDULE1.FOR

subroutine schedule1

  include '( global )/nolist'
  include '( nodes )/nolist'
  include '( txrx )/nolist'
  integer*2 tx_dice

!process channel traffic and advance node state for next slot.
  do i = 1, nbr_nodes

    do ii=1,nbr_nodes
      if(node(i).sat_nodes(ii) .gt. 0) then
        node(i).sat_nodes(ii) = node(i).sat_nodes(ii) -1
      end if
    end do

    if( node(i).cur_state .eq. rx_data ) then
      !check the data channel for reception
      if( chnl(i).captured .ne. 0 ) then !a message was received
        call proc_rx_data( i ) !so process it, ...
        node(i).cur_state = tx_ack !and send an ack.
      else !nothing was received, so set next state randomly.
        node(i).cur_state = tx_dice(i)
      end if
    else if( node(i).cur_state .eq. rx_ack ) then
      call proc_rx_ack( i ) !whether or not an ack was received.
      !set next state randomly
      node(i).cur_state = tx_dice(i)
    else if( node(i).cur_state .eq. tx_data ) then
      !set next state to listen for ack
      node(i).cur_state = rx_ack
    else if( node(i).cur_state .eq. tx_ack ) then
      !no obligation, so set next state randomly
      node(i).cur_state = tx_dice(i)
    end if

  end do

  chnl(i).numtx = 0
  chnl(i).pollution = 0
chnl(i).captured = 0
chnl(i+ack_offset).numtx = 0
chnl(i+ack_offset).pollution = 0
chnl(i+ack_offset).captured = 0

end do

!Place message traffic on channel for this slot.
do i = 1, nbr_nodes
if( node(i).cur_state .eq. rx_data ) then !
   node(i).channel = i

else if( node(i).cur_state .eq. rx_ack ) then !
   node(i).channel = i+ack_offset

else if( node(i).cur_state .eq. tx_data ) then !
call send_data( i )

else if( node(i).cur_state .eq. tx_ack ) then !
call send_ack (i)

end if
end do
return
end
subroutine send_data( i )

include '( global )/nolist'
include '( nodes )/nolist'
include '( txrx )/nolist'
include '( message_q )/nolist'
include '( routing_table )/nolist'
integer*2  d, mp, nh, msg_ix, i, k, kk
del_txid, tries, maxtries

tries = 0
maxtries = node(i).tq_length

! get pointer of next available message to be sent.
call gettx( i, d, mp )

!* select the next hop for a message and place it on tx queue.
!* place message on transmit list (if connectivity exists).
10 if( rt(i,d).ways .ge. 0 ) then !next hop exists.

! select next hop for message.
if( route_type .eq. 0 ) then
! use random choice from possible min-hops.
   nh = rt(i,d).next(rt(i,d).ways*ran(seed6)+1)
else ! use best link quality from possible min-hops.
   nh = rt(i,d).next(1) ! assume first in list is best.
do k = 2, rt(i,d).ways ! check all possible next hops.
   kk = rt(i,d).next(k)
if( route_type .eq. 1 ) then
! use success quality as decision criteria
   if( node(i).quality(nh) .lt. node(i).quality(kk) ) then
      nh = kk ! best found so far
   end if
else if( route_type .eq. 2 ) then
! use queue length as decision criteria
   if( node(nh).tq_length .gt. node(kk).tq_length ) then
      nh = kk ! best found so far
   end if
end if

end do
end if ! selection of next hop

if(msg_q(mp).sat_counter.gt.0) then

! dec the sat counter and enqueue to end of tx queue

c      write(6,*) now, ' SEND_DATA** Node ', i,
+ ' didn't send to neighbor ', nh, ' dest ', msg_q(mp).dest
temp_msg = msg_q(mp)
temp_msg.sat_counter = temp_msg.sat_counter - 1
del_txid = i
call delete_msg(mp,del_txid)
call gettx(i,d,mp)
call enq_msg_tx(temp_msg,del_txid)
tries = tries + 1
if((mp .ne. -1) .and. (tries .le. maxtries)) then
  goto 10
end if
node(i).cur_state = rx_data
return

end if

c***************************************************************************
c* If attempting to send to a saturated node, wait a while by not sending
  this and re-queing this message.
c***************************************************************************

if(node(i).sat_nodes(nh).gt.0) then

c      node(i).sat_nodes(nh) = node(i).sat_nodes(nh) - 1
temp_msg = msg_q(mp)
del_txid = i
call delete_msg(mp,del_txid)
call gettx(i,d,mp)
call enq_msg_tx(temp_msg,del_txd)
tries = tries + 1
if((mp .ne. -1) .and. (tries .le. maxtries)) then
  goto 10
end if
node(i).cur_state = rx_data
return
c
  goto 9090
end if

9090 if(mp .eq. -1) then
  node(i).cur_state = rx_data
  return
end if
node(i).channel = nh

!increment mutual interference level in neighborhood of node i.
call pollute1(i)

!place message on channel for selected next hop node.
msg_ix = chnl(nh).numtx + 1
chnl(nh).numtx = msg_ix
chnl(nh).msg(msg_ix).ptr = mp
chnl(nh).msg(msg_ix).txid = i
chnl(nh).msg(msg_ix).addressee = nh
c
  msg_q(mp).sent = .true.
msg_q(mp).xmits = msg_q(mp).xmits + 1
node(i).msg_from_ack = mp
node(i).node_from_ack = nh
if( WRITE_ARRIVSTATS) then !write message trace to output file.
  write( 97, 97 ) now, i, nh, d, msg_q(mp).tag
97      format( ix, 'send_data ', 518 )
end if
else !no connectivity to destination exists for this message
write(6,*) 'no connectivity.'
end if
return
end
A.4 SEND ACK.FOR

subroutine send_ack( i )

include '( global )/nolist'
include '( nodes )/nolist'
include '( txrx )/nolist'
include '( message_q )/nolist'
integer*2 ptr, addressee, i

addressee = node(i).node_to_ack

!increment mutual interference level in neighborhood of node i.
call pollute1( i )

!place ack on channel.
msg_ix = chnl(addressee + ack_offset).numtx + 1
chnl(addressee+ack_offset).numtx = msg_ix
chnl(addressee+ack_offset).msg(msg_ix).txid = i
chnl(addressee+ack_offset).msg(msg_ix).addressee = addressee

*******************************************************************************
* section added to set sat_flag and sat_dumped flags in acknowledge indicating the status of the transmit buffer
* d.e.carter
*******************************************************************************

* if the transmit queue length is above the saturated threshold
* set the proper flag in the message queue to be interpreted
* when the ack is processed

if(node(i).tq_length .gt. sat_threshold) then
  node(addresssee).sat_flag = .true.
c  else if(node(i).tq_length .gt. (1.5*node(addresssee).tq_length))then
  node(addresssee).sat_flag = .true.
c  write(6,*)(now, 'SEND_ACK** Node ', i, ' is saturated')
else
  node(addresssee).sat_flag = .false.
cend if

* if the transmit queue is full set the proper flag in the message queue to be interpreted when the ack is processed
if(node(i).tq_length .gt. sat_limit) then
    node(addressee).sat_dumped = .true.
    write(6,*) 'SEND_ACK** Node ', i, ' is full'
else
    node(addressee).sat_dumped = .false.
end if

*****************************************************************************
* end of additions               d.e.carter
*****************************************************************************

node(i).node_to_ack = 0

if( WRITE_ARRIVSTATS ) then !write message trace to output file.
    write( 97, 91 ) addressee, i
    format( 1x, 'send_ack ', 4I8 )
end if

return
end
A.5 PROC RX ACK.FOR

subroutine proc_rx_ack( i )

include '( global )/nolist'
include '( nodes )/nolist'
include '( txrx )/nolist'
include '( message_q )/nolist'
integer*2 capt, txid, i
integer*2 del_ptr, del_txid
integer*2 next_dec, orig_dest_dec
logical*2 finished, deleted

c write(6,*) 'In Proc_rx_ack... i = ', i

capt = chnl(i+ack_offset).captured
if( capt .ne. 0 ) then !an ack was received.
  txid = chnl(i+ack_offset).msg(capt).txid
if( node(i).node_from_ack .eq. txid ) then
!ack is from right guy.
deleted = .false.

c**********************************************************************
c* check the sat_flag in the message to see if this node is filling his
c* buffer above the saturated_threshold. d.e.carter

if(node(i).sat_flag) then
  node(i).sat_nodes(txid)=sat_delay*node(i).neighbors
end if

if((node(i).sat_flag) .and. (node(i).tq_length .gt. 0)) then
  ! search xmit queue for other messages destined for this node
  ! and inc sat_counter of each of them.

  orig_dest_dec = msg_q(node(i).msg_from_ack).dest
  next_dec = node(i).tq_head !get pointer to first in txq
  finished = .false.

c write(6,*) now, 'RX_ACK** Neighbor ', txid, ' of node ',
+ i, ' for dest ', msg_q(node(i).msg_from_ack).dest
c    write(6,*), ' processing other nodes in tx queue of node ',i
do while (.not. finished
    if(msg_q(next_dec).dest .eq. orig_dest_dec) then
      msg_q(next_dec).sat_counter =
      + msg_q(next_dec).sat_counter + 1
      write(6,*), ' packet to ', msg_q(next_dec).dest,
      + 'counter = ', msg_q(next_dec).sat_counter
    end if
    next_dec = msg_q(next_dec).back_ptr
    if(next_dec .eq. -1) then
      finished = .true.
    end if
  end do
end if

******************************************************************************
c* check the sat_dumped flag and if set move this message to end of the
c* tx queue with a sat_counter = 1.
c******************************************************************************

if(node(i).sat_dumped) then !move to end of queue
  node(i).sat_nodes(txid)=dumped_delay
c    write(6,*), 'Node reporting sat_dumped... ',txid
del_ptr = node(i).msg_from_ack
del_txid = i
temp_msg = msg_q( node(i).msg_from_ack )
temp_msg.sat_counter = 1
temp_msg.sent = .false.
call enq_msg_tx( temp_msg, del_txid )
!place message on end of queue
call delete_msg( del_ptr, del_txid)
deleted = .true.
end if

******************************************************************************
c* if the sat_dumped flag is not set, the packet was received ok, now delete
c* this message from the tx queue.
c******************************************************************************
if( .not. deleted ) then
    del_ptr = node(i).msg_from_ack
    del_txid = i
    temp_msg = msg_q( node(i).msg_from_ack )
    !copy message to temp buffer.
    call delete_msg( del_ptr, del_txid)
end if
else
    del_ptr = node(i).msg_from_ack
    del_txid = i
    temp_msg = msg_q( node(i).msg_from_ack )
    !copy message to temp buffer.
    call enq_msg_tx( temp_msg, del_txid )
    !place message on end of queue.
    call delete_msg( del_ptr, del_txid)
end if
else if( capt .eq. 0 ) then
    !message was not heard, so move to end of queue.
    del_ptr = node(i).msg_from_ack
    del_txid = i
    c write(6,*) 'MESSAGE NOT HEARD... msg_q = ',node(i).msg_from_ack, i
    temp_msg = msg_q( node(i).msg_from_ack )
    !copy message to buffer.
    call enq_msg_tx( temp_msg, del_txid )
    !place message on end of queue.
    call delete_msg( del_ptr, del_txid)
else
    write(6,*) 'messup in PROC_RX_ACK AT TIME:', now
end if
if( WRITE_ARRIVSTATS ) then !write message trace to output file.
    write( 97, 97 ) now, txid, i, temp_msg.dest, temp_msg.tag
    format( 1x, 'recv_ack ', 518 )
end if
node(i).node_from_ack = 0
node(i).msg_from_ack = 0
return
end
A.6 GLOBAL.TXT

C************************************************************************
C* GLOBAL - This common declaration contains all of the high level
C* control information necessary for running the MHSS simulation.
C*
C*************************************************************************

C** Set maximum number of nodes to be 100.
PARAMETER( MAX_NODES = 100 )
PARAMETER( MAX_LINKS = (MAX_NODES**2 - MAX_NODES)/2 )
PARAMETER( MAX_HOPS = 10 )

INTEGER*4 NBR_NODES, MATRIX_TYPE, NUMBER_MC, IMC
INTEGER*4 SEEDSEEED, SEEDSEED_0
INTEGER*4 SEED1, SEED2, SEED3, SEED4, SEED5, SEED6, SEED7, SEED8
INTEGER*4 SEED1_0, SEED2_0, SEED3_0, SEED4_0, SEED5_0, SEED6_0
INTEGER*4 SEED7_0, SEED8_0
INTEGER*4 MSG_COUNT, NOW, TIME_LIMIT
INTEGER*4 SAMPLING_INTERVAL
REAL*4 PROB_TX
INTEGER*2 WRITE_CONNECTIVITY, WRITE_COMMUNICANTS, WRITE_ROUTING
INTEGER*2 WRITE_ARRIVSTATS, TXPROB_TYPE, ROUTE_TYPE
INTEGER*2 RUN_KLEITMAN, MI_THRESH, TRAFFIC_TYPE, WARM_UP
integer*2 sat_threshold, sat_limit, sat_delay, dumped_delay
CHARACTER*132 HEADER_TEXT(10), RUN_COMMENT
REAL*4 AVG_DEGREE, LAMBDA, LAMBDA_LOW, LAMBDA_HI, LAMBDA_INC
REAL*4 PQUAL, QQUAL, PTPW, QTPW, TP_LO, TP_HI
DATA NBR_NODES /50/
DATA AVG_DEGREE /10.0/
DATA MATRIX_TYPE /1/

COMMON /GLOBAL/ NBR_NODES, AVG_DEGREE, MATRIX_TYPE, SEED1, SEED2,
+ SEED3, SEED4, SEED5, SEED6, SEED7, SEEDSEED,
+ SEED1_0, SEED2_0, SEED3_0, SEED4_0, SEED5_0,
+ SEED6_0, SEED7_0, SEEDSEED_0, NUMBER_MC,
+ WRITE_CONNECTIVITY, WRITE_COMMUNICANTS,
+ WRITE_ROUTING, RUN_KLEITMAN, LAMBDA, LAMBDA_LOW,
+ LAMBDA_HI, LAMBDA_INC, MSG_COUNT, NOW, MI_THRESH,
+ TIME_LIMIT, SAMPLING_INTERVAL, PROB_TX,
+ HEADER_TEXT, TRAFFIC_TYPE,
+ PQUAL, QQUAL, PTPW, QTPW, WRITE_ARRIVSTATS,
+ TXPROB_TYPE, RUN_COMMENT, ROUTE_TYPE, WARM_UP, IMC,
+ TP_LO, TP_HI,
+ sat_threshold, sat_limit, sat_delay, dumped_delay
A.7 NODES.TXT

!definition of database for each node.

structure /node_definition/
    integer*2 tq_head, !head of tx queue (pointer to msg_q)
    tq_tail, !tail of tx queue (pointer to msg_q)
    node_to_ack, !node to which I owe an acknowledgement
    node_from_ack, !node which owes me an acknowledgement
    msg_from_ack, !ptr to message which owes me an ack
    tq_length, !length of transmit queue
    channel, !channel (dest ID) selected for next tx
    neighbors, !no neighbors for this node
    tosscount
    byte cur_state
    real*4 quality(100), !quality of link to node i
    tx_prob
    logical*1 sat_flag, !indicating node’s tx queue is above flag
    sat_dumped !indicating node’s tx queue is full
    integer*2 sat_nodes(100)!array of sat flags for each node
end structure

record /node_definition/ node(100)

!define parameters for node states
integer*2 rx_data /0/,
    tx_data /1/,
    rx_ack /2/,
    tx_ack /3/

common /node/ node, rx_data, tx_data, rx_ack, tx_ack
Bibliography


