CITI Technical Report 92-8

NESTMOD Simulation of DQDB Metropolitan Networks

Gerald A. Winters
gerald@engin.umich.edu

David W. Bachmann
bachmann@austin.ibm.com

Toby J. Teorey
teorey@citi.umich.edu

ABSTRACT

Local area computer networks are well-established, and highly-interconnected, heterogeneous local networks are coming into prominence. NESTMOD is a combined analytical modeling and simulation tool, based on the existing tools NetMod and NEST, that is responsible for representing the topology of interconnected networks as well as providing library routines that describe the behavior of the interconnected networks. As an example of current technology, the metropolitan area network, DQDB (distributed queue dual bus), has been added to NESTMOD. The purpose of this paper is to describe the DQDB simulation and compare the simulation results with well-known analytical results.

October 1992

Center for Information Technology Integration
University of Michigan
519 West William Street
Ann Arbor, MI 48103-4943

NESTMOD Simulation of DQDB Metropolitan Networks

Gerald A. Winters
David W. Bachmann
Toby J. Teorey

October 1992

1. Introduction

This section gives a brief description of NESTMOD (described fully in "NESTMOD: the NetMod-NEST Interface" [1]) and explains its role. It is a tool comprised of the tools NetMod [2] and NEST [3].

NetMod is a HyperCard-based set of analytic models of network topologies. The purpose of NetMod is to allow the user to construct arbitrary networks, possibly composed of LANs, DQDBs, network interconnect devices, and workstations or nodes.[1] Characteristics such as medium capacity and packet output rate for workstations can be set to individually tailor the system. Once the topology and characteristics have been set, NetMod can provide a steady state analysis of the proposed network in terms of its basic performance characteristics: utilization, mean packet delay, and average queue length. When dynamic behavior analysis is desired, then NESTMOD invokes NEST to conduct the simulation.

NEST is a generalized network simulation tool. It allows simulation of arbitrary networks and network topologies. Only point-to-point connections can be specified, so NEST requires a number of detailed routines as input, such as the behavior of each of the components that comprise the network.

The integration of NetMod and NEST is called NESTMOD. This union is intended to provide the user with maximum flexibility as well as ease of use. A typical scenario would be for the user to call NetMod to set up a blank window, compose an arbitrary network using the icon menus, and then specify network topology and device/transmission medium characteristics. NetMod would then send a description of the network to the analysis server, and the server would build and run the simulation. See Figure 1.

In order to increase the suite of networks available to NESTMOD, the metropolitan area network DQDB has been added. The remainder of this paper discusses certain aspects of DQDB, such as architecture and protocol, as well as verifying the success of adding DQDB to NESTMOD.

1. We will use the terms workstation and node interchangeably throughout the remainder of this paper and intend them to be the same.

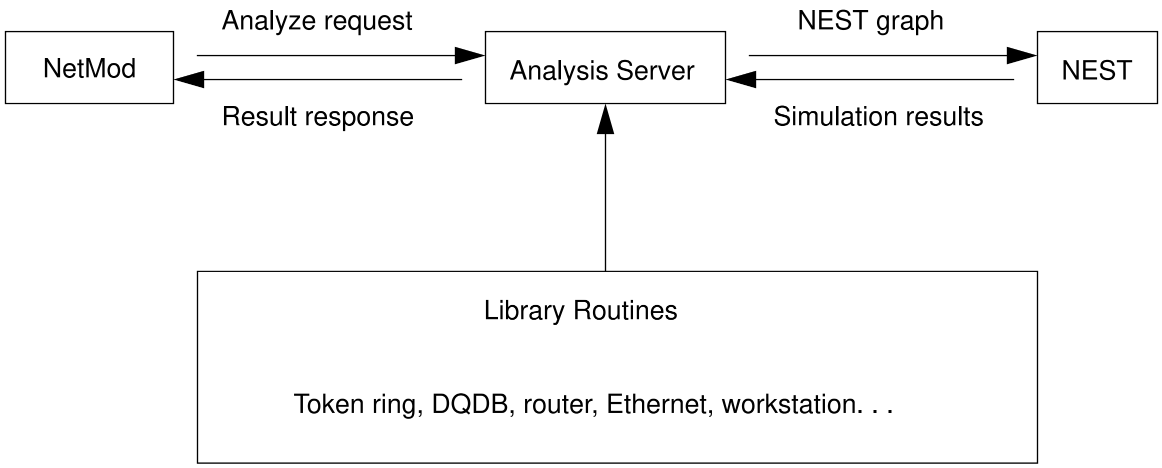

Figure 1: Overview of NESTMOD

2. DQDB

2.1 Architecture

The dual bus architecture of DQDB is shown in Figure 2. Each bus carries information at the same rate in opposite directions. This 45 Mbs DQDB actually operates at a total speed of 90 Mbs. Each node, with connections to both busses, has read and unidirectional write capability with connections that are logically adjacent to the bus and do not pass serially through the nodes. This architecture has the advantage that passive node failures have no effect. Writing to the bus is done only in "blank" slots so that a logical OR suffices to combine the 1-bits of the data with the all-0 slot space on the bus. Active node failures are detected through hardware by means of input-to-output checking [4]. When such a situation is detected, the checker can simply bypass the errant node. In contrast to ring architectures, data passes serially through each node and must be retransmitted and removed as necessary. Therefore, in ring architectures both active and passive node failures can be catastrophic because both types of failures involve some type of system healing. Healing involves system reconfiguration that disrupts and consumes resources. Furthermore, additional node failures can produce network islands. This architectural difference offers advantages to DQDB both in terms of robustness and reliability.

Located at the head end of each bus is a special node called a frame generator. Frames are generated synchronously at a rate that is determined by the transmission speed. When frames reach the end of the bus, they are simply dropped. A logical view of a frame can be seen in Figure 3. Frames are further subdivided into fixed length slots (53 octets) that are the units of communication between nodes. The slot control fields we shall be concerned about in this paper are the *busy* and *request* fields (see Figure 4). The busy field indicates that a slot contains data and cannot be used for transmission by other nodes. The request field is used to reserve empty slots. Upstream neighbors can be signaled by a downstream node to let a slot pass by for use. The slots form the basis for determining the node's position in the distributed queue. The node at the beginning of the queue may use the next free slot, and new requests are placed at the end of the queue. The DQDB media access protocol is the subject of the next section.

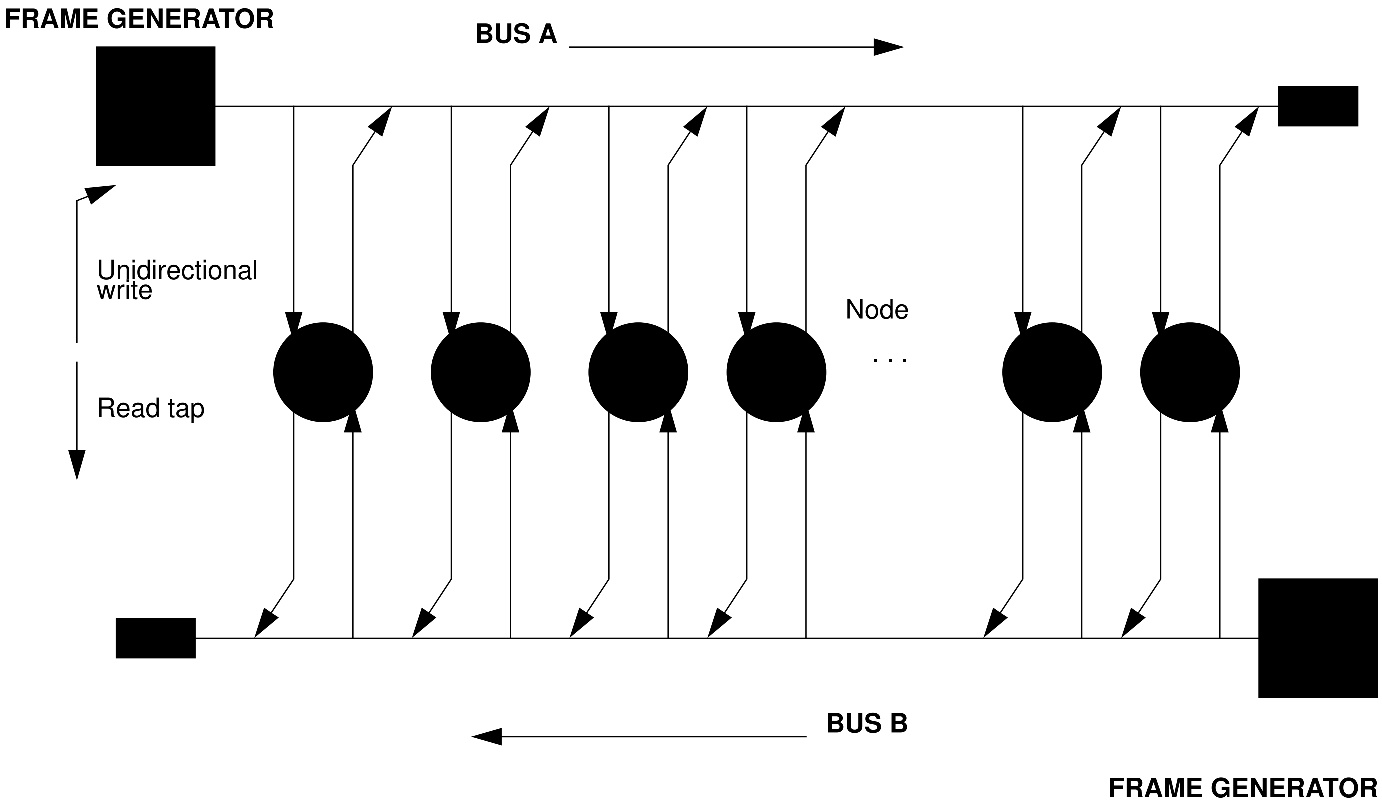

Figure 2: Logic Diagram for the DQDB Network Architecture

2.2 Protocol

Media access, or scheduling, is achieved by sending reservations "upstream" when a node wishes to transmit. The receiving node is always "downstream" from the sending node, in DQDB. Let's assume for a moment that a node wished to transmit a message on bus A (refer to Figure 2) to some node downstream. This makes bus B the upstream direction. In order for a node to establish itself in the distributed queue for bus A, it must send a reservation to its upstream neighbors by setting the request bit of a slot on bus B. Each station keeps an up/down counter running continuously, one for each bus. When a reservation request goes by on bus B, the counter is incremented by one. When an empty slot goes by on bus A, the counter is decremented by one. A nonzero value in the counter, indicating the position at the end of the distributed queue, means that there are outstanding requests for empty slots. If the up/down counter is zero, then there are no outstanding requests and the bus may be used at the next available slot. In the case where a node wishes to send a message on bus B, a second up/down counter is used, and the previous discussion follows symmetrically.

When a node wishes to transmit, it reads the up/down counter for the proper direction. If there are no contents, the next empty slot may be used. Thus the token latency associated with ring protocols can be eliminated. Under conditions of light load, the DQDB delay is short, which is a desirable property shared with CSMA/CD protocols. If the up/down counter is nonzero, then the sample is copied to a countdown counter and the up/down counter is set to zero. As empty slots pass by, the countdown counter is decremented and when it reaches zero, the next empty slot may be used. Un-

DQDB Frame

FRAME HEADER	Slot 0	Slot 1	. . .	Slot n	STUFFING FIELD

Figure 3: DQDB Frame

DQDB Slot

Access Control Field	Segments
1 octet	52 octets

Access Control Filed (1 octet)

Busy	Slot type	Reserved	Previous slot received	Request
1 bit	1 bit	1 bit	1 bit	4 bits

Figure 4: DQDB Slot Format

der heavy load conditions, the efficiency approaches 100 percent [5]. This is similar to the situation with token based protocols. A diagram of the counters is shown in Figure 5.

The combination of quick access under light load and predictable queueing delays under heavy load make the DQDB protocol suitable for networks of metropolitan dimensions.

3. Simulation Layout

The purpose of this section is to give a brief description of the routines that were provided to the analysis server in support of the actual simulation. Recall that the analysis server is responsible for describing to NEST the behavior of the different components that make up the network. In the case of DQDB, there were four basic components/modules that were provided: workstation, frame generator, network card, and wire. The simplest routine was the wire routine that described the propagation delay for the buses. As messages pass along the wire, a propagation delay of 1 microsecond is added for each network card the message passes. Therefore, the wire module assumes that nodes are approximately 180 meters apart.

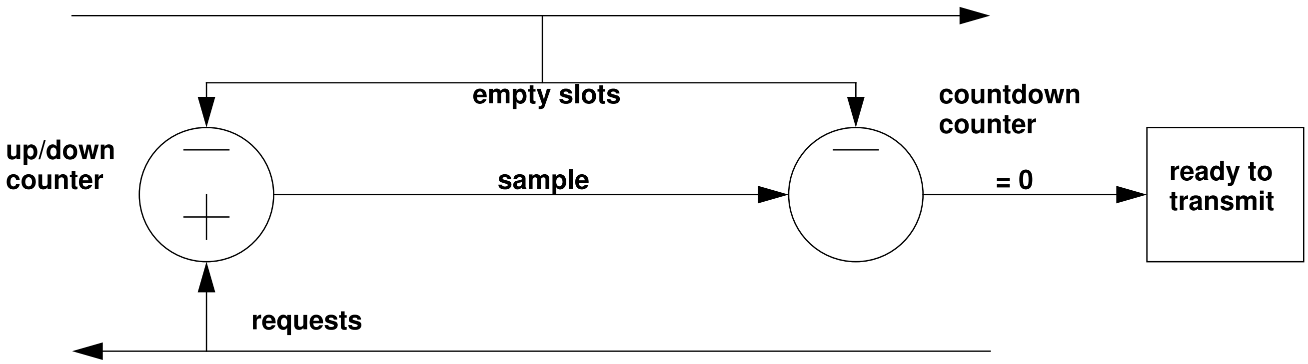

Figure 5: Media Access Counters

The frame generator routine is also fairly simple. In fact, this routine might be more aptly named "slot generator." Frames are not needed for purposes of simulation; all that is needed are slots to pass by the network cards. Thus the frame generator is responsible for generating empty slots at a rate of 1 every 9.4 microseconds (53 octets $*$ 8 / 45 Mbs).

The workstation routine's chief responsibility is generating packets. The workstations follow a Poisson process for generating packet arrivals with mean passed from NetMod. The packet sizes are generated according to an exponential distribution with mean value also passed from NetMod.

The network card module encapsulates the DQDB media access protocol and is responsible for breaking the packets from workstations into slots and managing the local queues. This module also "time stamps" the messages in order to compute delays. For example, when a packet is placed into its local queue, it is time stamped. When the packet finally reaches the destination, this time can be used along with the time stamp to compute the delay experienced by the packet. This information can then be stored in a global data structure and tabulated for all messages upon completion of the run.

Finally, a special graph data structure must be initialized for NEST that indicates the topology of the network. The graph, along with the modules that describe the network behavior, constitute the input that the analysis server must provide to NEST in order to run the simulation. Included in the appendices is a sample of the code as well as a schematic overview showing the connection between modules.

4. DQDB Analysis and Verification

In this section, we present numerical results from our simulations of the DQDB protocol and the analytic model given in Potter and Zukerman's paper, "Analysis of a Discrete Multipriority Queueing System" [6]. We will compare our simulation results with the results obtained from Potter and Zukerman to verify our simulation model. They show that their analytic model is sufficient for predicting packet delay by comparing its accuracy to simulation runs. Therefore, we use their analytic model as a benchmark for verifying our simulation model. This model for packet delay, $D(\bar{b})$, is given as follows:

$$D(\bar{b}) = \frac{v}{2}\left(1 + \frac{p}{1-p}\right) + \frac{\bar{b} - \left[\bar{b}\left(1 + C_b^2\right) + 1\right]/2}{1 - \frac{M-1}{M}p} + \frac{1}{2}$$

where

- $\bar{b}$ = average packet length, where b is a random variable representing the packet length in slots
- p = bus utilization
- M = number of workstations
- C_b^2 = the squared coefficient of the variation of b
- $v = \bar{b}(C_b^2 + (\lambda C_a^2)/M)$
- a = random variable representing the number of packet arrivals within a slot to any local queue
- λ = packet arrival rate within a slot
- C_a^2 = the squared coefficient of the variation of a

This formula, $D(\bar{b})$, gives the mean packet delay in slots and does not include propagation. It is important for us, at this point, to state that this is analytic model makes the assumption that the request channel has infinite capacity. Of course this is not possible to implement in a real network and one would expect that this model would report delays lower than actual network delays. Our results seem to support his idea as we experienced delays between 11% and 28% greater than the delay from the analytic model. This underscores the importance of having a good simulation model for DQDB, given the simplifying assumptions required by current analytical models.

In Table 1 we present our numerical results along with the results from the closed form analytical model obtained from Potter and Zukerman [6]. Inspection of Table 1 shows similar results when comparing the simulation runs versus the analytic results. The length of the simulation runs were set at 100 milliseconds (generating 20,000 slots). Five runs were conducted for each utilization value and the average of the runs were computed and compared to the analytic results. The system consisted of M = 7 nodes spaced one slot apart with varying packet rates per run. We held the packet size constant at 8000 bits throughout the runs. Because our simulation model delay was generally close to the analytic model delay under varying utilization values, we are confident that our simulation model is a reasonably accurate predictor of DQDB performance, although further simulations and appropriate statistical tests need to be done [7].

Table 1. DQDB Simulation Versus Analytic Results (single priority case)

Input Workload		Packet Delay (ms)			
Packet Rate (pps)	Util.	Sim.	Sim. Avg.	Analytic	% error
1300	89%	2.3332	2.0168	1.7860	+13
		1.7165			
		1.8437			
		2.1202			
		2.0706			
1100	75%	1.0047	1.0092	0.7890	+28
		1.0205			
		1.0394			
		1.0112			
		0.9700			
900	62%	0.5564	0.5618	0.5070	+11
		0.5717			
		0.5585			
		0.5415			
		0.5809			
450	31%	0.3535	0.3318	0.2820	+18
		0.3370			
		0.3226			
		0.3306			
		0.3153			

5. Conclusion

In this paper we have reported our results for adding the simulation of DQDB to NESTMOD. To judge the success of this endeavor, we compared our simulation results to published analytic results. Based on the low percentage of error between the analytic and simulation results, we feel confident in our ability to model DQDB.

References

[1] U. Amin, D. Bachmann, K. Deboo, and T.J. Teorey. "NESTMOD: the NetMod-NEST Interface." *Proceedings of the 1991 CAS Conference*, pages 239-254, October 1991.

[2] D. W. Bachmann, M.E. Segal, and T.J. Teorey. "NetMod: a Design Tool for Large-scale Heterogeneous Campus Networks." *IEEE Journal on Selected Areas in Communications,* 9(1):15-24, January 1991.

[3] A. Dupuy, J. Schwartz, Y. Yemini, and D. Bacon. "NEST: A Network Simulation and Prototyping Testbed." *Communications of the ACM*, 33(10):66-74, October 1990.

[4] J.L. Hullett. "New Proposal Extends the Reach of Metro Area Nets." *Data Communications*, pages 139-147, February 1988.

[5] J.F. Mollenauer. "Standards for Metropolitan Area Networks." *IEEE Communication*s, pages 15-19, April 1988.

[6] Philip G. Potter and Moshe Zukerman. "Analysis of a Discrete Multipriority Queueing System Involving a Central Shared Processor Serving Many Local Queues." *IEEE Journal on Selected Areas in Communications,* 9(2):194-202, February 1991.

[7] T.J. Teorey. "Validation Criteria for Computer System Simulations." *Proceedings of the ACM-SIGSIM on the Simulation of Computer Systems*, pages 161-175, August 1975.

Appendix A: C-code for DQDB Network Card Simulation

```
/*
        Simulation of DQDB in NEST; LIBRARY function for DQDB.
        This module imitates a DQDB network card.
*/

#include <stdio.h>
#include <math.h>
#include "dqdb.h"
#include "../library.h"

dqdbnode(nodeid)
    ident        nodeid;                       /* My position along the network */
{
    ident sender,dest;
    char *message;
    timev    sendt,arrvt;
                                               /* Local queues */
    struct Queue qA[QUEUE_LENGTH,qB[QUEUE_LENGTH], *q;
    int readatA, putatA, readatB, putatB,
        *readat, *putat;                       /* Indices in queue of messages yet to be sent */
    int countdownA, countdownB;                /* Count down counters */
    unsigned countreqA, countreqB;             /* Request counters */
    exern void init_queue_pointers(), dqdb_packet_handler(), put_in_dqdb_queue ();

    stop_time(); /* Stop simulation time during workstation initialization */

    init_queue_pointers (&putatA, &readatA);
    init_queue_pointers (&putatB, &readatB);
    RESET_COUNTDOWN_COUNTER (countdownA);
    RESET_COUNTDOWN_COUNTER (countdownB);
    countreqA = countreqB = 0;

    while (totalPasses)
        {
                    /* Nodes sleep while waiting for messages */
            sender=recvmt(&dest,&msgtype,&message,&sendt,&arrvt);

            if (MESSAGE_FROM_WS(msgtype))          /*Message is from workstation */
                {
                if (BUSA_PACKET(-msgtype,nodeid))  /* Determine the proper bus*/
                    {
                    q = qA;
                    putat = &putatA;
                    readat = &readatA;
                    }
                else
                    {
                    q = qB;
                    putat = &putatB;
```

```
                readat = &readatB;
                }                          /* Queue the message until bus access is gained */
            put_in_dqdb_queue(nodeid,putat,readat,q,message);
            }
        else if (BUSA_SLOT(nodeid,sender))        /*Slot is on bus A */
                                                   /* The slot handler manages bus access */
            dqdb_slot_handler(nodeid,sender,msgtype,&countreqA,countreqB,
                    &countdownA,&countdownB,&putatA,&readatA,putatB, readatB,qA,
                    arrvt,speed,message);
        else
            dqdb_slot_handler(nodeid,sender,msgtype,&countreqB,countreqA,
                    &countdownB,countdownA,&putatB,&readatB,putatA,readatA, qB,
                    arrvt,speed,message);

        }         /* while(totalPasses) */

    } /* routine dqdbnode */
```

Appendix B: Module Interconnection

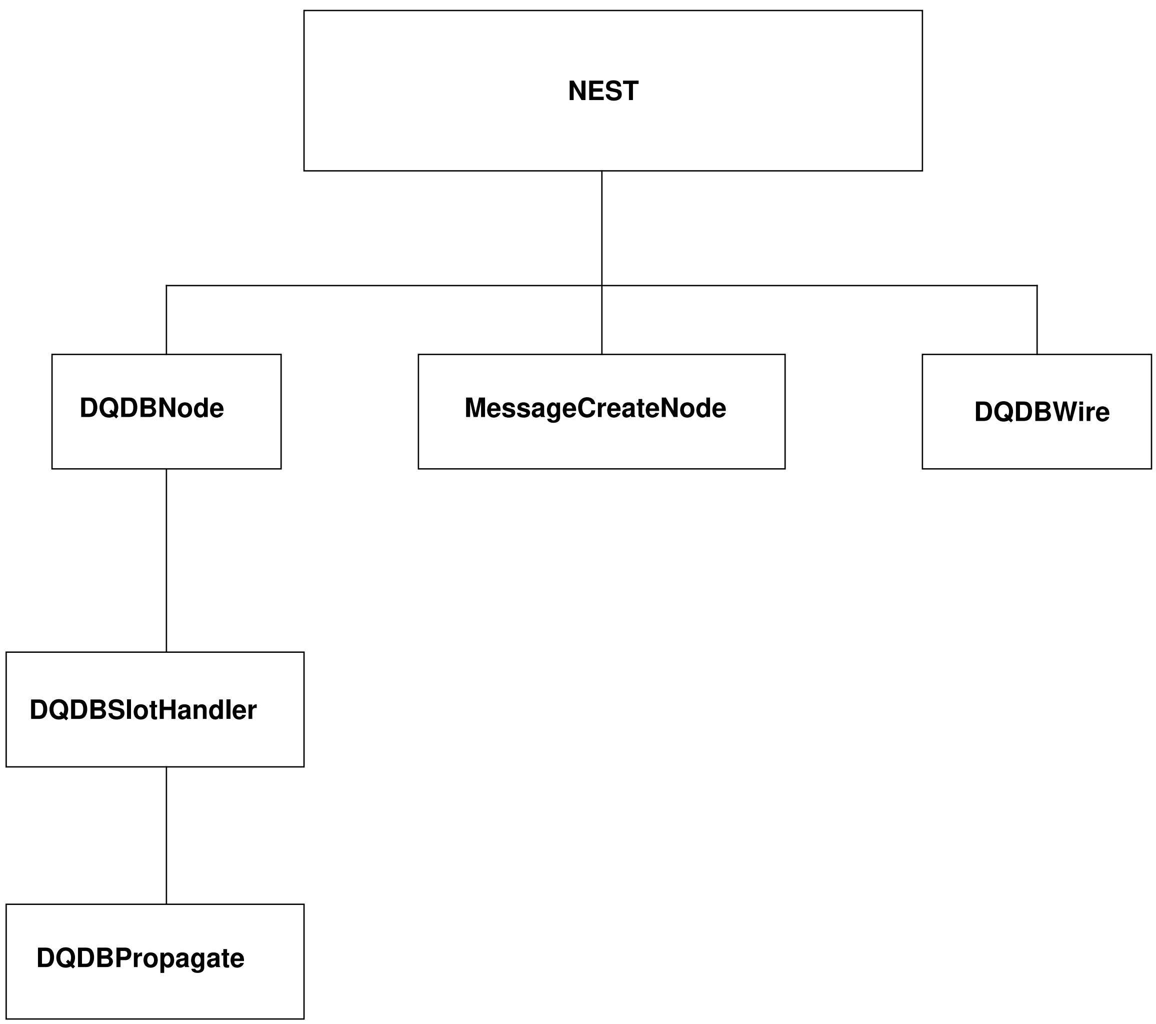

Figure 6: Schematic View of the Interconnection Between Main Modules