Proceedings of the 1988 Winter Simulation Conference
M. Abrams, P. Haigh, and J. Comfort (eds.)

On modeling Local Area Networks

Louis Y. Tsui
Department of Industrial and Systems Engineering
The University of Michigan-Dearborn
Dearborn, MI 48128-1491

Onur M. Ulgen
Department of Industrial and Systems Engineering
The University of Michigan-Dearborn
Dearborn, MI 48128-1491

ABSTRACT

The study of network throughput and utilization requires the modeling of packets being transmitted along the media. Modeling such a system can be quite logically demanding, especially that packets can move along the media in opposite directions at almost the speed of light; and when collisions are allowed, a packet may collide with another that is beyond its control.

Two modeling approaches are presented in this paper. One results in programs that closely follow the behavior of the physical system, yet require significant CPU time to execute. The second approach results in programs that may seem to be 'unnatural', yet more time-efficient.

Examples on token-ring and CSMA/CD networks are given in pseudo codes, followed by comparison of run times.

1. INTRODUCTION

In the past decade, local area networks (LANs) have grown increasingly popular. As more expensive equipment and more complex systems are being considered, the demand for tools to evaluate network performance also rises.

When developing complex models, a good portion of the analyst's time will be spent on understanding the rules as how the system behaves. As a result, the models that first come to mind usually have a clear correspondence to the physical system. However, such models may require detailed information on an object-by-object basis; thus, require large amount of CPU time to obtain the results.

In their paper, Henriksen and Schriber present two simplified approaches on modeling conveyor systems [2]. In many aspects, modeling packets moving along a cable is similar to modeling objects moving along a conveyor. However, modeling of packets appear to be more challenging because packets may move in both directions along a cable, while the movement of objects along a conveyor is unidirectional. Also, the objects can be put on the conveyor once an adequate space is found, and they will reach their destination successfully (assuming the objects do not fall off the conveyor). In contrast, packets can get on a cable anywhere but later they may collide with other packets and cause transmission to be aborted; thus, the "adequate space" for the packets does not exist any more.

Another issue that makes the modeling of networks challenging is that two different time scales must be dealt with. The packet interarrival time is in magnitude of seconds or minutes, e.g. the frequency that the user hits the return key. The packet propagation time is in magnitude of micro-seconds (at the speed of light or 186,000 miles/sec). Unless one assumes that the propagation time is zero, the program may be very time-consuming.

Two approaches are presented in this paper. The first approach is straightforward. It appears to be the simpler approach but, generally, it is also the inefficient one. The second approach is the more efficient approach but requires more insight to the process and a little more programming effort. Two network topologies are considered here. Models of a token-ring network and a CSMA/CD network are built for illustration. Section 2 of the paper briefly describes the token-ring protocol. Section 3 gives a straightforward approach to modeling token-ring LANs. Section 4 gives the more efficient approach to modeling the token-ring LANs. Section 5 gives the brief description of the CSMA/CD protocol. Section 6 discusses some of the difficult modeling issues associated with CSMA/CD protocol. Section 7 gives a straightforward approach to modeling the CSMA/CD LANs. Section 8 gives the more efficient approach to modeling the CSMA/CD LANs. The comparison of results and conclusions are given in Section 9.

2. THE TOKEN-RING PROTOCOL

The IEEE formed the 802 Committee in order to define the LAN standards. The 802.5 standard defines the token-ring protocol, which utilizes ring topology and token passing as access method [1]. In a token-ring LAN such as the one IBM announced in 1985, all nodes are connected to form a ring (see Fig. 1). Examples of a node can be a personal computer, a printer, or a file server. Each node is connected to the ring via two cables; one on which it receives data from its upstream neighbor and another one on which it sends data to its downstream neighbor. A special data packet called token circulates around the network. Only one token exists on the network at any one time and only the node owning the token is granted the right to send data to other nodes of the network. A token-holding time is set to prevent any node hogging the network. When the node is done sending the data or the token-holding time has run out, the token owner passes the token to its downstream neighbor so that the other nodes can transmit.

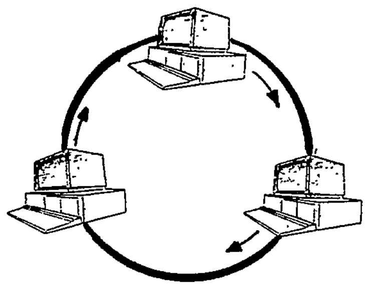

Fig. 1 An Example Token-Ring LAN

In a LAN, the performance measures include variables such as network throughput, network utilization, and response time. In a token-ring LAN, the node must receive and capture the token before the data can be transmitted. If node A is ready to transmit before node B is ready, but node B receives the token first, node B gets to transmit the data first even though its request to transmit comes after that of node A. Therefore, if the token itself is not modeled and the ring is modeled as a server with the FIFO queue discipline, the response time of each node can not be measured correctly.

3. NODE-BY-NODE APPROACH

A straightforward approach to modeling the token-ring LAN would be to model the token circulating about the ring, from one node to the next (thus the "node-by-node" approach). If a node is ready to transmit when the token arrives, the node will keep the token and start to transmit until it is done or the token-holding time has run out. The node will then put the token back onto the ring so that the other nodes can transmit.

Figure 2 gives a Pascal-like pseudo-code model of a ring with three nodes. The pseudo code can be easily translated into languages such as GPSS, SLAM or SIMAN.

```
Segment Token;

Procedure TransmitFrom (Queue);
/*   this procedure sends data from Queue
     until it is done or the token-holding
     time has run out
*/
  Begin
     Capture the token;
     Get data from Queue to form packets;
     Wait until the packets are transmitted
       or the token-holding time runs out;
     Update Queue status;
     Release the token;
  End;

/*   The following sections simulate the
     arrival of data to the nodes to be
     transmitted. One procedure for each
     node, and they may execute
     concurrently
*/
Procedure NodeA;
  Begin
     Generate data to Queue1 according to
       its interarrival time;
     Assign length attribute;
  End;

Procedure NodeB;
  Begin
     Generate data to Queue2 according to
       its interarrival time;
     Assign length attribute;
  End;

Procedure NodeC;
  Begin
     Generate data to Queue3 according to
       its interarrival time;
     Assign length attribute;
  End;

Begin  /*  The following simulates the
           token movement */

     Generate a token;

Node1:
     If Queue1 is not empty
       Then TransmitFrom(Queue1);
     Wait until token arrives Node 2:

Node2:
     If Queue2 is not empty
       Then TransmitFrom(Queue2);
     Wait until token arrives Node 3:

Node3:
     If Queue3 is not empty
       Then TransmitFrom(Queue3);
     Wait until token arrives Node 1;

     Goto Node1
End.
```

Fig. 2 A Node-by-Node Model of a Token-Ring LAN

The above model is easy to understand and simple to come up with but, as described below, it is very slow to run, especially when the workload on the network is low. Depending on the material of the cable, the signal can travel from 600 feet to 1000 feet per micro second. Given that the local area network resides in an office building or on a plant floor, it may take only a fraction of a micro-second for the token to go from one node to the next. If the packets are generated at an aggregate rate of one packet per second, and the travel time from node to node is one microsecond, more than 99% of the clock update process will be used to keep track of the position of the token. Therefore, such a model can be quite undesirable.

4. NEXT-MAJOR-EVENT APPROACH

Since the location of the token does not contribute to any statistics of interest, especially when the network is idle, a second look at the above model suggests the following approach. In this approach, the circulation of the token is not simulated when the network is idle. Instead, the model logic figures out the location of the token only when a node becomes ready to transmit. Therefore, the time is only spent on major events that are of direct interest to the analysis (thus the "Next-Major-Event" approach). The approach requires the identification of the major events and updates the model at these event times only. After the event that the network becomes idle, it is likely that the next sequence of events will be the token arriving at each node with no data to transmit, followed by the event that a node becomes ready to transmit the data. The efficiency of the model can be improved significantly if the simulation can go from idle state to ready state directly, all that is required is being able to figure out the location of the token when a node becomes ready to transmit. Hence, in the following model, the variable Count is used to indicate the number of stations ready to transmit. It is set to zero initially. When Count becomes zero, the network becomes idle. Instead of continuing to move the token around, the model records the time and location where the token is currently at and put the token aside. The next major event is when a node becomes ready to transmit, Count becomes 1 which indicates that the token must be put back onto the ring. The location can be computed from the elapsed time, the last location of the token and the travel time of a bit around an idle ring (walk time). When Count is non-zero, the token is either held by a node or in the transition from one node to the next, the node simply has to wait for the token to arrive before it can start transmission. By this approach, the need to update the location of the token when the network is idle is eliminated, the savings in CPU time should be obvious. Figure 3 gives the three-node ring modeled in pseudo-code using the next-major-event approach.

```
Segment Token;

/*   Global variables

     Name     Range    Definition
     -----    -----    ------------------
     Count    0..3     the number of
                       nodes ready
                       to transmit, set
                       to 0 initially.
     Node     1..3     the node number
                       where the token
                       was at when the
                       network became
                       idle.
     Time     real     the last time the
                       network became
                       idle.
*/

Procedure TransmitFrom(Queue, Node#);

/*   This procedure sends data from Queue
     at Node# until it is done or the
     token-holding time has run out
*/
  Begin
     Capture the token;
     Get data from Queue to form packets;
     Wait until the packets are transmitted
       or the token-holding time runs out;
     Update Queue status;
     If Queue becomes empty Then
        Count := Count - 1;
     If Count = 0 Then
        Begin
          Node := Node#;
          Time := TimeNow;
          Remove the token from the
            network;
        End
      Else Release the token;

  End;

/*   The following procedures, NodeA, NodeB
     and NodeC run concurrently
*/
Procedure NodeA;
  Begin
     Generate data to Queue1 according to
       its interarrival time;
     If Queue1 was empty Then
        Count := Count + 1;
     If Count = 1 Then
        Begin
          Calculate where the token is;
          Calculate the travel time t that
            the token will reach the next
            node, say NodeX;
          Wait for t time units;
          Generate a token at NodeX;
        End;
  End;

Procedure NodeB;
  Begin
     Generate data to Queue2 according to
       its interarrival time;
     If Queue2 was empty Then
        Count := Count + 1;
     If Count = 1 Then
        Begin
          Calculate where the token is;
          Calculate the travel time t that
            the token will reach the next
            node, say NodeX;
          Wait for t time units;
          Generate a token at NodeX;
        End;
  End;
```

```
Procedure NodeC;
  Begin
     Generate data to Queue3 according to
     its interarrival time;
     If Queue3 was empty Then
        Count := Count + 1;
     If Count = 1 Then
        Begin
          Calculate where the token is;
          Calculate the travel time t that
            the token will reach the next
            node, say NodeX;
          Wait for t time units:
          Generate a token at NodeX;
        End;
  End;

Begin  /* the token movement */

     Generate a token;

Node1:
     If Queue1 is not empty Then
        TransmitFrom(Queue1, 1)
     Wait until token arrives Node 2;

Node2:
     If Queue2 is not empty Then
        TransmitFrom(Queue2, 2);
     Wait until token arrives Node 3;

Node3:
     If Queue3 is not empty Then
        TransmitFrom(Queue3, 3);
     Wait until token arrives Node 1;

     Goto Node1
End.
```

Fig. 3 A Next-Major-Event Model of a Three-Node Token-Ring LAN

5. THE CSMA/CD PROTOCOL

The CSMA/CD protocol is the IEEE 802.3 standard, which is the standardized version of Ethernet, developed by Xerox Corporation, utilizing bus topology and the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) access method. This access method mimics the way a roomful of people talk to each other. There is no central control, any person who wishes to talk will listen first, and if someone else is talking, he/she would defer until that person is done. If more than one person start to talk simultaneously, they stop when they hear the voice of others and wait for a while before they try again. The roomful of people are the nodes, each node can communicate with others by sending packets through the bus, usually a coaxial cable. Before a node starts to transmit packets (talk), it must test if the carrier on the bus is active (listen to see whether someone else is talking). If the carrier is active, the node will wait until the carrier becomes inactive, then, after a short period (interframe spacing time), the node will start transmitting the packets. During the transmission period, the node listens to see if another node is also transmitting, i.e. collision detection. If it detects signals transmitted from other nodes, it realizes a collision has happened. Since the packet it sent becomes garbled, it will not send the remaining part of the packet. Instead, it enforces the collision by sending a jamming signal to notify all the other nodes that a collision has occurred, then stops transmission and backs off for a random amount of time (backoff time). When the backoff time expires, the node starts carrier sense and tries to send the packet again. If collisions happen again and again on the same packet, the node will discard the packet after a certain number of retries and reports it as an unsuccessful event [4]. Figure 4 gives an example of the CSMA/CD LAN.

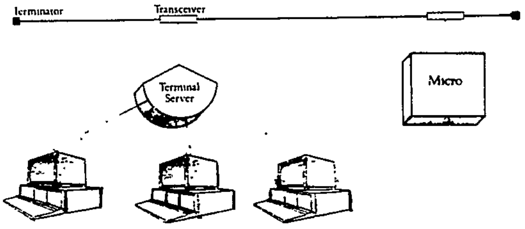

Fig. 4 An Example CSMA/CD LAN

6. MODELING ISSUES

The behavior of CSMA/CD network is much more complex than the token-ring network. In the most simplistic case, the bus can be treated as a server with FIFO queue discipline. Unfortunately, this approach completely eliminates the issues of collision and retransmission. In fact, part of the network traffic is caused by the retransmission of previously collided packets. As the workload of the network increases, so will the collision and retransmission rate. Therefore, the above approach will produce an incorrect throughput of the LAN when the workload is high.

Since most people are interested in the response time and the throughput of the LAN when it is heavily loaded, the bus can not be adequately modeled as a server. In reality, the bus appears available to some nodes and unavailable to others, depending upon the location of the transmitting node and the node requesting to transmit. If the carrier is sensed by the node ready to transmit, the bus is unavailable and the node defers. Otherwise, the node assumes the bus is available and starts transmission. It is only when the first bit of a packet reaches both ends of the bus will the carrier be sensed active by all the nodes. Therefore, only if no other nodes participated before the packet reaches both ends (call this time duration the contention period), can the transmission be guaranteed successful. Otherwise, collision will occur and will be detected when the transmitting node receives the packet from some other node. Since the contention period varies according to the position of the transmitting node, and collisions are caused by other nodes that are beyond the control of the transmitting node, modeling collision detection can be difficult.

7. SECTION-BY-SECTION APPROACH

Because the packets go in both directions and the nodes are not necessarily equally spaced, the node-by-node approach must be modified to a section-by-section approach in the bus topology LAN, where the bus is divided into sections of equal length and each section is attached to by at most one node. Each section becomes busy (carrier becomes active) as the packet propagates from the previous section. Nodes attached to the busy sections will sense the carrier and defer the sending of packets while nodes in the not-busy sections will transmit. If a packet enters a section that is already busy, collision will be detected by the node attached to that section. Thus, the node will stop transmission and backs off after sending the jamming signals.

Figure 5 describes the logic of the section-by-section approach where the bus is divided into N equal sized sections. The rightmost section is numbered 1 and the leftmost numbered N. Each node has one procedure and all node procedures run concurrently.

```
/* Global Variables

   Name         Range     Definition
   ------       ------    --------------------
    N           >= 1      the number of
                          sections in the bus

 bus[0..N+1]    >= 0      bus[i]=0 if carrier
                          can not be sensed in
                          section i;
                          bus[i]>0 if carrier
                          can be sensed in
                          section i;
                          bus[i]>1 if
                          collision can be
                          detected in section i

    Count       >= 0      the number of nodes
                          transmitting during
                          the contention period
*/
Procedure BusBecomesBusy (Var K: integer;
                          Ownsection: 1..N;
                          Tmax: 1:N);

  /* As the leading edges of the packet
     propagate along the bus, this
     procedure updates the status of each
     section. The procedure returns when
     collision is detected or the leading
     edges reach both ends of the bus.  If
     collision is detected, K gives the
     number of sections the leading edges
     have propagated.
  */
 VAR
   left, right: 0..N+1; /* array index */

 Begin
   bus[Ownsection] := 1;
   count := count + 1;     /* on the air */
   left := OwnSection;
   right := left;

   K := 0;
   While (K < Tmax) and
      (bus[Ownsection] = 1) Do
     Begin
       K := K + 1;
       Wait one time unit;
       left := Max (left - 1, 0);
       right := Min (right+1, N+1);
       bus[left] = bus[left] + 1;;
       bus[right] = bus[right] + 1;
     End
 End;

Procedure BusBecomesIdle (Ownsection:1..N);
  /* This procedure updates the status of
     each section as the trailing edges of
     the packet propagate along the bus.
  */
 VAR
   i: 0..N+1;  /*  array index  */

 Begin

   bus[Ownsection] := idle;
   count := count - 1;  /*  off  air */
   left := OwnSection;
   right := left;

   For i := 1 to Tmax do
     Begin
       wait one time unit;
       left := Max (left - 1, 0);
       right := Min (right+1, N+1);
       bus[left] = bus[left] - 1;;
       bus[right] = bus[right] - 1;
     End
 End;

Procedure ANode;

/* Local Variable

  Name         Range       Definition
  ------       ------      --------------------
 Ownsection    1..N        the bus section
                           number this node
                           is attached to.

     i         0..N+1      array index

    Tmax       0..N        the number of
                           sections to the
                           farther end of the
                           bus

    Tmin       0..N        the number of
                           sections to the
                           closer end of the
                           bus
*/
```

```
Begin  /*  the main segment  */

      /*  initialization  */
  Let Tmax and Tmin be the distance to the
     farther and the closer end of the bus
     in number of sections, respectively.

  Wait until a packet is generated by an
     application;
  While the transmit buffer is not empty Do
    Begin
      /* carrier sense */
     Wait until bus[OwnSection] <> busy;

      /* start transmission */
     BusBecomesBusy(K, Ownsection, Tmax);

      /*  signals have propagated to both
          ends of the bus  */
     If (K < Tmax) Then
       Begin    /* collision detected  */
         /* send the jamming signal while
            the leading edges of the
            original signal keep
            propagating
         */
        While (K < Tmax) Do
          Begin
            left := OwnSection + K;
            right := OwnSection - K;
            K := K + 1;
            Wait one time unit;
            left := Max (left - 1, 0);
            right := Min (right+1,N+1);
            bus[left] := bus[left] + 1;;
            bus[right] := bus[right] + 1;
          End;

        Wait until the remaining jamming
          signal is sent;
        BusBecomesIdle(Ownsection);
        Wait until backoff time expires;
       End
     Else  /*  Collision not sensed yet */
      If (count = 1) Then
        Begin /* The packet can be
                 transmitted  successfully
              */
          Wait until the remaining packet
            is transmitted;
          BusBecomesIdle(Ownsection);
        End
      Else  /* some node sent a packet,
               which has not reached this
               section yet
            */
        Begin
          /*  wait until it gets here  */
          Wait until Bus[OwnSection] > 1 ;

          /*  Send the jamming signal  */
          Wait until the complete jamming
            signal is sent;
          BusBecomesIdle(Ownsection);
          Wait until the back off time
            expires;
        End
    End
End.
```

Fig. 5 The Section-by-Section Model of A CSMA/CD LAN

The above approach resembles very much the inch-by-inch approach to simulating the progress of cartons down the conveyor [3]. Although there won't be as many packets on the cable as there are cartons on the conveyor, it is still time-inefficient. Since the signal can travel from 600 feet to 1000 feet per micro second, we can readily see that the time unit may well be a fraction of a micro second, and it requires at least one hundred time units for the packet to go from one end to the other in a network with one hundred nodes.

The section-by-section approach allows the model to detect the collision when the node detects it. However, little value is gained for most of the time when the packet moves form one section to the next. Therefore, if the model can identify and move directly to the next major event where appropriate actions are taken, the efficiency can be improved significantly.

8. NEXT-MAJOR-EVENT-APPROACH

After a node starts to transmit, if the next major event is defined at the instance when the packet reaches both ends of the bus, one can examine the number of nodes which started transmitting during this period and determine whether collision has taken place or will take place. If only one node is transmitting during this period, the transmission is a successful one; otherwise, collision is detected. The collision can always be scheduled later in time if it has not happened yet; however, if the collision should have been detected earlier, is it too late to react?

In order to answer the question, a closer look at the LAN must be taken. One requirement on the local area networks is that the propagation delay should be much less than the transmission delay. Since the jamming signal is relatively long (32 bits), even if the node detects the collision right after it started transmission, it would not be done sending the jamming signal when the original packet reaches both ends of the cable. Therefore, it is not too late to stop the transmission of jamming signal in time and produce the correct network utilization.

Since there is no need to trace the individual packet, there is no need to model the bus in this approach. Instead, a counter is used to tally the number of transmitting nodes. When a node starts to transmit, its node ID and start time are recorded in a data structure, and the counter is incremented by one. When a node stops transmitting, its node ID and stop time are recorded in another data structure, and the counter is decremented by one. Collision is indicated when the counter becomes greater than one. The time collision is detected by each node can be figured out from the distance between the nodes and the difference between the transmission start times. The carrier sense can be handled in a similar fashion. The utilization can also be calculated according to the transmission start time and stop time. Figure 6 gives a detailed model of this approach.

```
/* Global Variables

   Name          Range         Definition
  -------       ---------     --------------------
     N            >0           the number of nodes

  Carrier      (active,        active if all nodes
                inactive)      on the network can
                               sense the carrier,
                               inactive otherwise.

 XmitID[1..N]   1..N           the transmitting
                               node ID

 XmitTime[1..N]  >0            the time
                               transmission
                               started

 XmitCount        1..N         the number of
                               transmitting nodes

 StopID[1..N]     1..N         the node ID

 StopTime[1..N]   >0           the time when the
                               node stops
                               transmitting

 StopCount        1..N         the number of
                               entries in
                               StopID (StopTime)
*/

Segment ANode;

/* Local Variables

   Name             Range        Definition
   ----             ------       ------------
  OwnID             1..N         The node ID

  CarrierSensed     (true,       true if this
                     false)      node can
                                 sense the
                                 carrier
*/

Procedure SendJammingSignal (Var count:
                                   1..N);
/*  Local Variables

    Name        Range        Definition
    -----       -----       --------------------
    count       1..N         the total number
                             of nodes involved
                             in a collision

     i,j        1..N         loop index

     T          real         time collision is
                             detected

     Min        real         the earliest time
                             collision is
                             detected by a node

     Max        real         the time collision
                             is detected by all
                             nodes

     Tjam       real         the time required
                             to transmit the
                             jamming signal
*/
Begin
     /* find out when the last node
        detects the collision
     */
     max := -1;
     For i := 1 to count Do
     /* the transmitting node */
       Begin
         Min := maxint;
         For j := 1 to count Do
         /* all other nodes */
           Begin
             distance := the distance
               between XmitID[i] and
               XmitID[j];
             /* calculate the time
               collision is being realized
               */
               T := XmitTime[j] +
                 distance/propagation_speed;
               If Min > T Then Min := T;  /*
                      time collision detected
                      by the ith node */
           End;
         If Max < Min then Max := Min;
       End

     /* Max is the time the last node
           detects collision */
     Wait for Tjam+(Max-TimeNow) time units

     /* all nodes are done with sending the
          jamming signal */
     count := 0;
     carrier := inactive;
End;

Begin    /*  Main Program  */
 Wait until a packet is generated by an
     application;
 While the transmit buffer is not empty Do
  Begin

    /* carrier sense */
Ready:
   If (carrier = active)  Then  /* defer */
     Wait until carrier = inactive

   /* wait until the trailing edge of the
      last packet pass the node
   */
   t := -1;
   For i := 1 to StopCount Do
     Begin
       temp := (distance to StopID[i]) /
          propagation_speed + StopTime[i];
       If temp > t Then t := temp;
     End;
```

```
   If temp > TimeNow Then
     Wait for (temp - TimeNow) time units
    Else If (XmitCount > 0) Then
          /* in the contention period, see
             if carrier   can be sensed
          */
          Begin
            i := 1;
            CarrierSensed := false;
            While (Not CarrierSensed) and
               (i<=XmitCount) Do
              Begin
                distance := the distance to
                    Xmit[i].nodeID;
                If TimeNow-XmitTime[i] >=
                 distance/propagation_speed
                 Then CarrierSensed:=True;
                i := i + 1;
              End
            If CarrierSensed Then
               Goto Ready
          End

 /* start transmitting the packet */

 XmitCount := XmitCount + 1;
 XmitID[XmitCount] := ownID;
 Xmit[Xmitcount].Starttime := TimeNow;
 Wait until signal reaches both ends;

 /* collision detection */

 If XmitCount = 1 Then
   /* at this point, all nodes will sense
      the carrier and thus defer.  No
      collision past this point is possible
   */
   Begin
     carrier := active;
     XmitCount := 0;
     Wait until transmission is completed;
     carrier := inactive;
     StopCount := 1;
     StopID[1] := OwnID;
     StopTime[1] := TimeNow;
   End
 Else  /* more than one node is
          transmitting */
   Begin
     carrier := active;
     SendJammingSignal(count);
     Calculate (BackOffTime);
     Wait until BackOffTime expires;
     Goto Ready;
   End
END.
```

Fig 6. The-Next-Major Event Model of A CSMA/CD LAN

9. CONCLUSION

Two models for the token-ring LAN are programed in GPSS/H using the two different approaches. The ring consists of four nodes, each one is one (1) microsecond apart, every time the token is captured, it is held for one (1) millisecond, the requests to transmit are varied to achieve different network utilization. The programs are run for five (5) simulated seconds on Micro VAX II, the execution time of the node-by-node approach is more than two hundred times higher than that of the next-major-event approach when the network utilization is about 20% (1336.11 vs. 4.74 CPU seconds). At 43% utilization, the ratio is about ninety-seven (962.31 vs. 9.86 CPU seconds). At 63.5% utilization, the ratio is about thirty-nine times (577.57 vs. 14.39 CPU seconds). At 80% utilization, the ratio dropped to about seventeen times (307.9 vs. 17.74 CPU seconds). Finally, the ratio approaches to one as the network utilization approaches to 99%. The superiority of the next-major-event approach is clearly demonstrated.

REFERENCES

1. Token Ring Access Method and Physical Layer Specifications, IEEE Std 802.5-1985, The Institute of Electrical and Electronics Engineering, Inc, 1985.

2. James O. Henriksen and T. J. Schriber, "Simplified Approaches to Modeling Accumulating and Nonaccumulating Conveyor Systems," Proceedings of the 1986 Winter Simulation Conference.

3. James O. Henriksen, "You Can't Beat the Clock," Proceedings of the 1986 Winter Simulation Conference.

4. Dick Lefkon, "A LAN Primer," BYTE, July 1987.

AUTHOR'S BIOGRAPHIES

Louis Tsui is an Assistant Professor of Industrial and Systems Engineering at the University of Michigan-Dearborn. He received his B.S. in Computer Science in 1974 at the Tankiang University, Taipei, Taiwan, and M.S. and Ph.D. in Industrial and Operations Engineering at the University of Michigan, Ann Arbor, in 1977 and 1984, respectively. Dr. Tsui has been working on production scheduling, manufacturing process simulation, and computer network performance evaluation. His present research interests include decision support systems and software development. He is a member of TIMS/ORSA.

Onur M. Ulgen is an Associate Professor of Industrial and Systems Engineering at the University of Michigan-Dearborn. He received his B.S. in Mechanical Engineering in 1972 at the Bosphorus University, Istanbul, Turkey, and M.S. and Ph.D. in Industrial Engineering at the Texas Tech University, Lubbock, Texas, in 1975 and 1979, respectively. Dr. Ulgen has been an active consultant for a number of years in the application of simulation in manufacturing systems. His present research interests include computer simulation program generators, object-oriented programming, animation, and scheduling theory as applied to manufacturing systems. He is a member of TIMS/ORSA, IIE, SCS, and IEEE-SMC.