A Starvation-free Access Protocol for a Full-duplex Buffer Insertion Ring Local Area Network

RAHUL SIMHA[1]

Dept. of Computer Science
College of William & Mary
Williamsburg, VA 23185

YORAM OFEK

IBM Research Division
TJ Watson Research Center
Yorktown Heights, NY 10598

Abstract

Of several existing designs for Local Area Networks, the Buffer Insertion Ring has been shown to provide higher throughputs, lower mean delays and greater spatial reuse than competing designs such as the Token Ring and Slotted Ring networks. However, one disadvantage is that the normally unregulated access scheme of the Insertion Ring allows for the phenomenon of "starvation" - when a network node has to wait too long before it can access the ring. In our work, we demonstrate that starvation is a serious problem and present a protocol to prevent it. It is shown that the new protocol is correct, stable and does not substantially degrade the otherwise efficient operation of the ring.

[1]This work was done while this author was working at IBM in the summer of 1988.

1 Introduction

As the demands of network users grow in magnitude and complexity, there is a corresponding need to provide efficient architectures and access mechanisms for Local Area Networks [6]. A special class of such networks, ring networks [11], has been the subject of several research efforts in the past decade [6, 11]. In this paper, we focus on the Buffer Insertion Ring [7] and examine the issue of "fair" access to the network.

Buffer Insertion rings offer certain advantages: the atomic operations of the ring are essentially asynchronous and distributed; there is no need for a complicated clocking mechanism and hence it is suitable for large distances. In addition, the buffers allow for concurrent access to the ring and, in this manner, provide spatial reuse of capacity along the ring. In terms of disadvantages, there is a synchronization overhead with every packet and there may be delays caused by packets travelling through several buffers. Furthermore, there is also some concern about address-checking delays that messages incur in traversing the ring [13]. We note, however, that several studies of ring networks have consistently concluded that the Buffer Insertion Ring exihibits higher throughputs and lower mean packet delays than other designs of ring networks as well as alternative LAN architectures [1, 8, 11, 12, 19]. In addition, recent results [13] suggest that the problem of address-checking may be circumvented. However, in contrast to designs such as the Token Ring, which inherently provide fair access to the medium, the architecture and access mechanism of the Buffer Insertion Ring allows for the phenomenon of *starvation*. This occurs when a network node is prevented from accessing the medium due to heavy usage by other nodes. We note that, in the literature, this phenomenon is also known as *lock-out* and is occasionally alluded to by its antonym, *hogging*.

In this paper, we argue that starvation is indeed a serious drawback that can occur in large magnitudes with the type of message traffic found in LANs. We present a media-access protocol for a full-duplex buffer insertion ring that prevents starvation. We prove the correctness and stability of the protocol and show that it provides a minimum guaranteed throughput. The protocol has several attractive properties: it is decentralized, uses only local information and makes efficient use of the architecture. Furthermore, the protocol operates during certain types of failures, continuing to provide fair access to the medium,

The organization of this paper is as follows. In the next section, we present a brief outline of the operation of the basic unidirectional buffer insertion ring and then, in more detail, we describe the architecture of a full-duplex system with separation of control and transport. Section 3 contains a discussion on the phenomenon of starvation and our access protocol. Section 4 contains a discussion of the important properties of the protocol and following that, in Section 5, we present simulation results. Finally, Section 6 contains our concluding remarks.

2 Principles of Operation

In this section we describe the architecture and basic operation of a full-duplex buffer insertion ring [4]. We assume that the reader is familiar with the essential principles of a buffer insertion ring (see [6, 7, 11, 14] for details) and we restrict ourselves to a simple description below. We focus, instead, on the full-duplex operation and our mechanism for the exchange of control information between network nodes.

In a unidirectional buffer insertion ring [7], each node has only a single transmitter and receiver and the nodes are connected, each transmitter to the next node's receiver, in the form of a ring. To communicate with another node on the ring, a node packetizes a message and transmits the message on the ring asynchronously provided that there is no possibility of collision with another message. This is achieved through the use of a buffer (or register, as it is sometimes known) at each node, in the following manner.

Consider a node which has a message x to transmit. If the buffer at the node contains a message y then y is transmitted. When the buffer has been emptied of y, i.e. y has been transmitted, then the node begins the transmission of message x on the ring, provided that, earlier, during the transmission of y, there was no incoming traffic. If, during the transmission of x, there is any incoming traffic then that traffic is switched into the buffer while the transmission of x continues concurrently. Also, if a node has no messages to transmit, then messages that have to travel through the node bypass (or cut through) the insertion buffer and, consequently, do not experience any delay introduced by buffering.

Observe that each node operates in a completely asynchronous and distributed fashion, a property we preserve in our protocol. In addition, several concurrent transmissions can take place independently, a feature not found in several other architectures [3, 16, 17]. Below, we describe our full-duplex architecture and the method of transmitting control information.

2.1 Full-duplex Operation

In our full-duplex version of the basic buffer insertion ring (see [4] for further details), each node possesses two transmitters and two receivers. Figure 1 shows the arrangement of the nodes in the system. The nodes are connected by full-duplex links and thus, logically, we say that there are two independent rings and every node has a transmitter-receiver pair in each of the two rings. The two rings, which we call the I-ring (or Inner ring) and the O-ring (or Outer ring) for simplicity, carry messages in opposite directions. We note the following distinguishing features of the system:

- Both rings are used at all times, in contrast to some other designs [17] where a second ring is used as as backup ring.
- The routing is assumed to be fixed, shortest-hop routing; thus, each node transmits to half the nodes on the I-ring and the other half using the O-ring. For example, in figure 1, node 4 uses the I-ring to transmit to nodes $3, 2, 1$ and 9, whereas the O-ring is used for transmissions to $5, 6, 7$ and 8.
- Messages are removed from the ring by the destination node.
- Each message packet, in addition to the information and synchronization fields, contains the source and destination addresses and a special field called the *Reservation* field to be used in the protocol, which is described in a later section.
- We assume that the packets are of a fixed, constant size. Later, modifications are suggested for variable-sized packets.

Figure 1: **A full-duplex buffer insertion ring**

We define a node i to be *upstream* of node j on the I-ring if i transmits messages on the I-ring that reach j or that must pass through j to some other node. Node j is said to be *downstream* of node i on the I-ring. Upstream and downstream on the O-ring are similarly defined.

2.2 Control Messages

We now describe the manner in which control information is dispatched around the network. In contrast to several other designs, wherein control information is imbedded in regular messages into space reserved specifically for it [10, 16], in our full-duplex buffer insertion ring, control information is propagated in separate control messages. We observe the following salient points:

- A control message is a short sequence of bits containing a special sequence of bits, the sending node's identity and when necessary, synchronization bits.

- A control message has preemptively higher priority over regular messages and thus, if a regular message is in transmission, the transmission is interrupted and the control message is sent. Following the sending of the control message, the transmission of the interrupted regular message is resumed. At the receiving end, when a message is being sent through the node or being switched into the buffer, the occurence of the special sequence of bits mentioned above (a sequence of bits that does not appear in regular messages) causes the control message to be routed into a control message buffer. Note that the special sequence is required only when the transmission of a regular message is interrupted, in which case the control message *imbeds* itself into a regular message.

- Control messages are sent in the same direction along the ring as regular traffic. We note that, just as with regular messages, a fixed, shortest-hop routing is used for the control messages. Thus, for a node on the I-ring to send a control message to its nearest upstream neighbour (upstream on the I-ring), the node sends the control message on the O-ring. Therefore, control messages follow the same path as regular messages.

- Although extra hardware is required to support such separation of control and transport, we believe that the benefits incurred outweigh the additional hardware cost:

  - The propagation of control information, sometimes extremely critical, is very fast and does not depend on the flow of regular message traffic. Thus, in the event of any failure or system anomaly (such as starvation), the anomaly is swiftly isolated and information regarding the anomalous situation is rapidly propagated through the system.
  - Anomalies such as starvation are caused by upstream nodes and, quite often, by the *nearby* upstream nodes. The control messages travel upstream on the opposite ring and thus, reach the upstream nodes quickly.
  - We note that in systems wherein control information is transmitted in a field of a regular message, space for the field is usually reserved in every message for this information. Thus, when there is no information to be carried, bandwidth is wasted. If the frequency with which control information is required to be disbursed is small, then any additional overhead incurred in separate control messages is recovered in bandwidth gained by not having to reserve space in regular messages.

Figure 2: **Starvation**

In the ensuing section we identify the problem of starvation and, following that, describe a fair protocol that makes efficient use of the control message separation discussed above.

3 Starvation

3.1 The Phenomenon of Starvation

We now demonstrate that for message traffic that characterizes typical LAN traffic, there can be starvation of high magnitude. Our result, that starvation can occur in large magnitudes, is obtained through simulation. While several analytical results are available on buffer insertion rings [2, 8, 12, 21], these are mostly approximations for metrics such as the mean packet delay. For the quantities of interest here, an exact analysis appears to be extremely difficult, and hence we use simulation results.

As described in the above section on the operation of the ring, a node with a message to transmit has to wait till its insertion buffer is empty. We say that a node is *starved* if the node has to wait "too long" for its insertion buffer to be empty, i.e., upstream nodes send too many messages past the given node in quick succession. For example, in figure 2, node A is continuously transmitting to node C. As a result, node B finds its insertion buffer always full with an upstream message and is unable to transmit (for the purposes of the protocol, a precise definition of "too long" is needed

Load	$\max_{i,k} w_i^{(k)}$		
	B=1	B=10	B=100
0.2	0	10	100
0.6	2	20	101
1.0	3	26	300
1.4	3	30	300
1.8	4	40	300
2.2	7	100	300
2.6	8	68	300
3.0	9	89	390
3.4	11	119	456
3.8	12	130	426
4.2	22	118	849
4.6	20	147	663
5.0	17	124	691
5.4	19	158	900
5.8	19	157	1700
6.2	26	175	2114

Table 1: **Existence of starvation**

and this is presented in the next section). It might appear that starvation occurs as a pathological situation that the algebra of the system permits. In the above example, it seems likely that node A will eventually finish transmitting and node B can then start sending messages. It might also be reasonably argued that, at moderate loads and Poisson type traffic, the possibility of node A transmitting several such messages is indeed low. While that is certainly the case, we argue that under more realistic traffic conditions, the period of waiting can be extremely long.

We observe that LAN traffic is bursty and characterized by transfers of blocks of data [14] - files, disk sectors, memory pages etc. We consider the following simple model of such traffic: there are bulk arrivals of fixed size B to each node in the system and destinations are chosen uniformly at random. The bulk size, B, is in terms of the number of messages constituting the block of data.

Next, let $w_i^{(k)}$ be the number of messages from upstream that node i must wait to pass through its insertion buffer before being able to transmit the k^{th} message *while* the k^{th} message is at the head of the queue at node i (on the I-ring). In other words, $w_i^{(k)}$ counts the number of upstream messages that fill the insertion buffer of node i in succession before the buffer empties. Thus, we see that, $w_i^{(k)}$ is an estimate of the waiting time before successful transmission once a packet is ready to be transmitted.

In table 1 we show $\max_{i,k\leq K}\{w_i^{(k)}\}$ for finite $K = 50,000$ in a system of 10 nodes. The values are tabulated for different loads and bulk sizes. Observe that with a bulk size of $B = 1$, the maximum waiting time is low, as expected. This corresponds, loosely, to Poisson-type arrivals. With a relatively small bulk size of 10 messages, the maximum waiting time is already high and with $B = 100$, a reasonable unit of file transfer, the waiting time is inordinately high - even at moderate loads. With increasing bulk sizes, these large waiting times (periods of starvation) occur more frequently and thus, we are motivated to examine methods of reducing such starvation.

In the next section we present a protocol that removes this problem and ensures fair access to the ring. We note that there are other definitions and measures of fairness that are concerned with prioritized traffic, users [20] and the establishment of connections [14]. While it is important to provide mechanisms for handling messages with a range of priorities, in this paper we do not consider such traffic and focus instead on the basic access to the medium.

3.2 A Starvation-free Protocol

We now present a protocol for regulated, starvation-free access to our full-duplex buffer insertion ring. We have argued that the buffer insertion in its full-duplex version offers several advantages and is an attractive alternative to other LAN designs. It is, therefore, desirable that the protocol preserve these advantages and, in addition, exploit the control message architecture. We list below some of the goals in designing our protocol:

- The protocol should make efficient use of the full-duplex architecture and control message separation.
- The new access mechanism should operate in a distributed manner and require only local information.
- The protocol should be stable and provide a minimum guaranteed throughput.
- The execution of the protocol should not cause the throughput to fall nor should the mean message delay increase.
- The protocol should be as resilient to failures as possible.

In order to present the criterion by which starvation is determined as well as the access protocol, we introduce some definitions and notation. Since the operation of the two rings, the I-ring and the O-ring, are identical, for simplicity the description that follows pertains to only one ring (the I-ring, for example). We distinguish between the following two terms:

- *Transmit.* We say that a node *transmits* a message if the node either transmits a message originating at the node or transmits onto the ring a message from upstream that is present in its buffer.
- *Insert.* We say that a node *inserts* into the ring when a message locally originating at the node is at the head of the queue, finds the insertion buffer empty and is able to transmit onto the ring successfully.

Next, let

$n =$ the number of nodes in the system.

$w_i^{(k)}$ = the number of upstream messages that the k^{th} message (arriving to the head of the queue) at node i must wait for, without being able to transmit.

$t_i^{(k)}$ = the time of arrival of the k^{th} message to head of the queue at node i.

$T_i^{(k)}(m)$ = the time at which m upstream messages (and no local messages) are transmitted from node i after $t_i^{(k)}$, i.e., after the arrival of the k^{th} message to the head of the queue at node i.

$\tau_i^{(k)}(m)$ = the time at which m messages (both upstream or local) are transmitted from i after $T_i^{(k)}(n)$.

Next we define a cycle:

- *Cycle.* A *cycle* is the period between the pairs $\tau_i^{(k)}(ln)$ and $\tau_i^{(k)}((l+1)n)$ for each n and $l = 1, 2, 3, ...etc.$

Finally, let

$I_i^{(k)}(l)$ = the number of inserts that node i makes in the cycle delimited by $\tau_i^{(k)}(ln)$ and $\tau_i^{(k)}((l+1)n)$.

The above notation is better explained using figure 3. Here $i = 2, k = 11$ and $m = n = 5$. We see that $t_2^{(11)}$ occurs when the 11^{th} message generated at node 2 arrives at the head of the queue, $T_2^{(11)}(5)$ occurs when 5 upstream messages have gone by and $\tau_2^{(11)}(5)$ occurs when 5 transmissions (in this case, none are local) take place from node i thereafter. Finally, $\tau_2^{(11)}(10)$ records the time after $T_2^{(11)}(5)$ at which 10 messages have been transmitted.

We define two nodal states: *normal* and *starved* with transitions taking place between these two states as follows:

- *normal*$\rightarrow$*starved*: when $w_i^{(k)} = n$ for any k, i.e., when a message is made to wait for at least n upstream messages to go by. Thus, a node is declared *starved* when there is a message at the head of its transmit queue and the insertion buffer successively fills up with n consecutive upstream messages.

- *starved*$\rightarrow$*normal*: Assume that a node has moved into the *starved* state, i.e., at $T_i^{(k)}(n)$. Then the node returns to *normal* at $\tau_i^{(k)}((L+1)n)$ where L is such that $L = \min_l\{l | I_i^{(k)}(l) \geq 2\}$, i.e., at the end of the first cycle in which the node is able to make two *inserts*.

The state transitions are illustrated through the example in figure 3. We see that in the third cycle, node 2 is able to insert twice and hence returns to the *normal* state. We note that, in the above definitions and example, the superscript k is maintained in spite of higher-numbered messages being inserted. The emphasis is that the notation is associated with the starvation of the k^{th} message. At this juncture, it is important to observe that the state transitions are made asynchronously and independently based on locally observed phenomena. There is no information that is used from other nodes or messages in the ring. We now describe the exact mechanism for countering starvation.

Figure 3: **An example**

3.2.1 Request messages

The mechanism for countering starvation is based on the following fundamentally simple idea: when a node is starved, the node sends control messages (called REQuest messages) upstream in order to cause the upstream nodes to suspend their insertions long enough for node i to insert into the ring. Equivalently, node i requests the upstream nodes to send it an "empty" or "dummy" message, which node i can then write over and thus, in this fashion, insert its pending message into the ring.

Node i, when *starved*, sends a REQ message upstream (and hence on the other ring) at time $T_i^{(k)}(n)$ and at times $\tau_i^{(k)}(n), \tau_i^{(k)}(2n), ..., \tau_i^{(k)}(Ln)$, where L is defined as before. Two comments may be made here. First, we will establish later that no matter how many of the other $(n-1)$ nodes in the system are requesting for empty messages, an empty message destined for node i will arrive from upstream no later than n messages after it made the request. For example, in the case of the request message sent at $T_i^{(k)}$, an empty message *reserved* for node i will arrive at i no later than $\tau_i^{(k)}(n)$. Secondly, bandwidth savings are obtained through the spatial reuse permitted by the explicit reservation; this is discussed in section 5.

3.2.2 Responding to REQuests

We now describe how the upstream nodes respond to requests made by downstream nodes. In order to do this, we need the following definition:

- $F_I(i)$: Denote by $F_I(i)$ the farthest upstream node that can send messages to node i along the I-ring.

Thus, in figure 1, $F_I(9) = 4$. Note that $F_O(i)$ is similarly defined for the O-ring. We note that, due to the shortest-hop routing scheme, $F_I(i)$ is no further than $\frac{n}{2}$ nodes away from node i.

The manner in which an upstream node j responds to requests from a downstream node i is presented in two categories: one for all upstream nodes j such that $j \neq F_I(i)$ and the other, remaining case, $j = F_I(i)$. We denote the request message from node i as REQ(i).

Node j, $j \neq F_I(i)$, receives REQ(i)

if (node j's insertion buffer is empty)
    and (node j is not *starved*)
    and (node j is not committed to sending an empty message)
    then
        node j pulls the REQ(i) message off the ring;
        node j commits to sending node i an empty message;
        node j prepares to send an empty message reserved for node i
    else
        node j forwards the REQ(i) message upstream
endif

Thus, a node $j, j \neq F_I(i)$, sends an empty message to i if and only if it is in a position to do so, i.e., if node j itself is not starved, its insertion buffer is empty and it has no other reservation commitments made. The purpose here is to satisfy a request message as early as possible in its travel upstream while ensuring that there can be no instability. Note that the empty message is *reserved* for node i by using the reservation field described earlier.

Node j, $j = F_I(i)$ receives REQ(i)

```
The REQ(i) is pulled off the ring and a flag is set indicating
a pending empty message to be sent to node i.
repeat
    if (node j's buffer is empty)
        and (no commitment has been made)
        then
            node j commits to send an empty message reserved
                for node i;
            node j prepares to send an empty
                message reserved for node i
        elseif (the message in the buffer is not reserved already)
            then
                the message in the buffer is reserved for node i
            else
                node j waits for the message currently in the
                buffer to be transmitted
            endif
    endif
until reserved empty message can be sent to i;
```

Note that $F_I(i)$ is the last node to see a request message from node i. Clearly, if the insertion buffer is empty then an empty message reserved for i can be sent to node i immediately. However, the insertion buffer may contain a message from further upstream. In this case, the upstream message must be destined for a node that is upstream of node i, from our routing arrangement (also, from the definition of $F_I(i)$, any node upstream of $F_I(i)$ transmits only to nodes upstream of i). Thus, if this upstream message is not already reserved, then it can be reserved for node i's use after it has reached its destination. If node $F_I(i)$'s insertion buffer contains a message that is already reserved, then node $F_I(i)$ repeats the algorithm after that message is transmitted, i.e. it waits to reserve an empty message for i at the next possible opportunity. In the next section we show that this waiting period is bounded.

We note that when the starved node, i, finally receives the empty message reserved for it, node i marks the empty message as 'not reserved' if it has no commitments to make, otherwise the empty message is reserved for a node downstream of i. Then, if node i has a message to be inserted, it is copied into the empty message and transmitted.

3.2.3 Some Fairness Theorems

Thus far, we have only presented and described the protocol. It is important to be able to prove some fairness properties about the protocol. Under the mitigating assumption that, due to the high priority and short lengths of control messages, control messages require negligible time to propagate compared to regular messages, we show the following:

- *A correctness property.* A node waits no longer than n messages after making a request to receive its reserved empty message.
- *A stability property.* Every node is guaranteed to be able to transmit, on the average, once every n messages. Note that, in the worst possible situation where all nodes have messages to transmit, this implies that each ring performs no worse than a Token Ring network.

THEOREM 1: A node waits no longer than n messages (that pass through its insertion buffer) after sending a request REQ to receive its reserved empty message.

PROOF: Consider a node i, on the I-ring, that sends a REQ(i) message upstream. If the request can be satisfied along the way by some node $j, j \neq F_I(i)$, then, clearly, the empty message arrives at node i no later than $\frac{n}{2}$ messages later. However, in the worst case, the REQ(i) message will travel all the way to node $F_I(i)$. In this case the $\frac{n}{2}$ insertion buffers between $F_I(i)$ and i are full and thus, node i will see at most all of these $\frac{n}{2}$ messages pass through its insertion buffer. Now, node $F_I(i)$ may have to wait for at most $\frac{n}{2}$ messages, each already reserved, to pass through *its* insertion buffer, in order to reserve an empty message for node i. This is an additional $\frac{n}{2}$ messages that node i might have to wait for before receiving its empty message. Thus, node i waits no more than $\frac{n}{2} + \frac{n}{2} = n$ messages. Note that $F_I(i)$ does not wait for more than $\frac{n}{2}$ messages above because, according to the protocol, the nodes j between i and $F_I(i)$ cannot request again until the times $\tau_j(n), \tau_j(2n), ...etc$, i.e., until they have transmitted n messages themselves.

We note that, in the worst case, all nodes are starved and are continually making requests. Then nearly every message inserted is copied onto a reserved empty message. Since, every message travels only half way around the ring, each message is used by two nodes in one cycle. Thus, on the average, each node transmits twice in every cycle on each ring. Therefore, we see that the average capacity of the full-duplex system is four times that of a single Token ring.

THEOREM 2: Each node is guaranteed to be able to insert, on the average, once every n messages.

PROOF: From Theorem 1, a node waits no longer than n messages with each request. Thus, when a node returns to the *normal* state, it will have made a fraction of at least $\frac{L}{n+(L-1)n} = \frac{1}{n}$ insertions on the average. If the node never returns to the *normal* state, then it makes one insertion every n messages except for the first message inserted after going into the *starved* state. This first insertion might require a wait period of at most $2n$ messages. Thus, the long term average is still at least once every n messages.

In regular traffic, the performance is much better. This conclusion is amply borne out by simulation results: in the section on performance results, we will see that the average latency (time it takes for the reserved empty message to arrive) is actually much smaller than n.

In the above protocol, we have set $w_i^{(k)} = n$. In practice, $w_i^{(k)}$ can be varied within limits. A large value of $w_i^{(k)}$ indicates an optimistic policy in which starvation is not declared too early, whereas, a small value of $w_i^{(k)}$ might cause a premature transition into the *starved* state. The advantage of an optimistic threshold, as will be evidenced later, is that in waiting longer, an empty message might arrive due to randomness in traffic and thus, no bandwidth need be wasted in reservation.

Clearly, $w_i^{(k)}$ can be larger than n, but there is some question as to how small it could be made without introducing potential instability into the system. We now show that it is possible to take $w_i^{(k)} = \lceil \frac{n}{2} \rceil$. There is, however, a small addition required in the protocol: a node moves to the *starved* state if it cannot insert *or make a reservation* for $\lceil \frac{n}{2} \rceil$ messages.

THEOREM 3: Each node can send requests after every $\lceil \frac{n}{2} \rceil$ messages and inserts at the minimum rate of at least once every $\lceil \frac{n}{2} \rceil$ messages.

PROOF: We consider the case when n is even. For odd n, the behavior is no worse than the corresponding even ring of size $n+1$. In the case that n is even, $i = F_I(F_I(i))$. Consider the case that node i is starved and requests empty messages periodically every $\frac{n}{2}$ messages (i.e. at $T_i^{(k)}(\frac{n}{2}), \tau_i^{(k)}(\frac{n}{2})...etc$). In the worst case, $F_I(i)$ is also starved and requests empty messages from i. Since there are $n-2$ remaining nodes and n buffers, $F_I(i)$ receives an empty message within a finite amount of time. This is used by $F_I(i)$ but reserved for node i. Note that this can be done at $F_I(i)$ because the message itself is reserved for $F_I(i)$ when it arrives at node $F_I(i)$. When node i receives this message it can be reserved for $F_I(i)$ and thus both i and $F_I(i)$ receive empty messages every $\frac{n}{2}$ messages.

Thus we see that each node inserts into each ring at least once every $\frac{n}{2}$ messages transmitted. We observe that, in the worst case situation, where every node wishes to insert, the throughput at every node (on each of the two rings) is twice that of corresponding nodes on a Token ring. Therefore, the total throughput at each node, in this case, is four times that of a similarly loaded Token ring.

4 Properties of the Protocol

In this section, we outline some of the salient features of the protocol, some of which make the usage of this protocol together with the full-duplex architecture an attractive alternative to other LAN designs:

- *Decentralized operation.* A node determines starvation independently and sends requests asynchronously. There is no central scheduler and thus the system is more failure resistant.

- *Local information.* A node only observes variables local to the node in order to determine starvation and also to determine the times at which requests are dispatched. Furthermore,

Figure 4: **Minimization of bandwidth wastage**

there are only a few variables - a counter to count upstream messages and local insertions, and some state variables.

- *Correctness and stability.* The protocol was shown to be correct and to provide a relatively high minimum throughput.

- *Efficiency.* The protocol exploits the full-duplex architecture and contol message separation. Control messages reach the source of starvation quickly and requests may be satisfied very early. In a sense, the request messages search for the closest available empty message. We note that the protocol is easily modified to suit architectures in which the control information is imbedded into regular messages. For example, a similar protocol may be used in a full-duplex Slotted Ring with n slots.

- *Spatial reuse.* We note that a reserved empty message may waste bandwidth as it travels along the ring unused. However, this wastage is minimized since the empty messages are marked with the identity of the node they are reserved for. In this case, the reserved empty message may be used along the way. We demonstrate this through an example in figure 4. Assume that node 3 has a message to transmit to node 2 on the I-ring and node 4 sends an empty message reserved for node 1 on the ring. This message, although reserved for node 1, is used by node 3, since the message empties before it reaches node 1.

Figure 5: **Protocol operation during a node failure**

- *Failures.* We now describe how the protocol operates during node failures (for details on other types of failures we refer the reader to [4]) through an example. Consider the failure of node 6 in figure 5. Then node 5 receives no upstream messages from 6 and thus, is never starved. Then all requests from nodes $4, 3, 2$ and 1 are pulled off the ring by node 5, which can send empty messages since it is itself never starved.

The protocol presented here is for fixed-sized packets. In the case that variable-sized packets are permitted, some modification is necessary. We note that, if a packet is smaller than the fixed size permitted then it may be transmitted immediately. In the case that the packet is larger, we suggest two possibilities. In the first, the larger packet is broken up and transmitted [18] in a few fixed size packets. In the second scheme, the cost of breaking up a packet is avoided, and instead, a large packet waits in a separate buffer for a large enough empty message to arrive. We are at present unaware of the relative merits of these two schemes and note that a thorough performance study is required.

5 Performance Results

In this section we present our simulation results: graphs of the mean packet delay against differing loads and a table of the mean reserved message latency. We compare not only our protocol with the unregulated buffer insertion ring, but also with other standards such as the Token Ring [5, 11] and the QPSX Dual Bus [9, 15, 16]. We use the following labels on each of the plots:

- $T1$ - the single Token Ring.
- $T2$ - two Token rings operating in parallel. Since we use full duplex links in our architecture, a fair comparison with the token-passing ring requires a comparison with two such token-passing rings operating concurrently.
- $B1$ - the unregulated buffer insertion ring.
- $B2$ - the buffer insertion ring using our protocol with $w_i^{(k)} = n$, i.e., when the wait period before transiting to the *starved* state is n.
- $B3$ - the buffer insertion ring using our protocol with $w_i^{(k)} = 2n$. As discussed earlier, a node may move into the *starved* state "too soon" and thus unnessarily request an empty message. A more optimistic scheme will permit a longer waiting time and hence, a more cautious transition into the *starved* state. In some cases, particularly with small bulk sizes, this may result in bandwidth savings.
- $QPSX$ - the QPSX dual bus architecture and access protocol (also referred to as DQDB) [9, 16].

For each system considered above, we considered idealized operating conditions. For example, in all of the above systems we assume that messages are removed by destination. In the simulation of the Token Ring, we assume that there is no delay in passing the token. In the case of QPSX, we allow for spatial reuse along the the bus and assume that a node does not wait to set a request bit (see [16]). For the buffer insertion ring, we assume no additional overhead in address decoding and control messages. Next, we assume that destinations are chosen at random from a uniform distribution over the set of possible destination nodes. Finally, we take the bulk arrival process at each node to be a geometric random variable with fixed bulk sizes and choose $n = 10$. We note that the load factor in each graph has been normalized so that the throughput of a single Token ring is unity and also, the delay is measured in terms of the number of messages.

In table 2, we tabulate the average latency (the average number of messages before the arrival of a reserved empty message when a request was made) with differing loads and bulk sizes of 10 and 100 messages respectively. We note that although the worst case latency is 10, the number of nodes, the results show that the average is far less.

Figure 6 shows the estimated mean packet delay, against increasing load for the system with unit bulk sizes. We observe that QPSX performs better than both the Token Rings but that all of the buffer insertion rings exhibit dramatically better performance than the other designs. This behavior, both in order and degree, is found in the plots with higher bulk sizes of 10 and 100 in figures 7 and 8 respectively. We note that the protocol introduces no degradation in throughput and the difference in mean packet delay, with or without the protocol, is negligible. From this, we may infer that the restrictions on transmission imposed by the protocol do not perceptibly reduce the performance of the system in any way. Next, we observe that for a small bulk size, the optimistic scheme, $B3$, performs slightly better than $B2$, the one described in earlier sections. This

Figure 6: **Delay vs. Load for unit bulk size**

Figure 7: **Delay vs. Load for a bulk size of ten messages**

Figure 8: **Delay vs. Load for a bulk size of hundred messages**

Load	Latency	
	B=10	B=100
0.2	1.555	3.731
0.6	1.737	2.668
1.0	1.796	2.668
1.4	1.835	2.374
1.8	2.040	2.690
2.2	2.050	2.719
2.6	2.127	2.717
3.0	2.248	2.759
3.4	2.258	2.609
3.8	2.324	2.683
4.2	2.364	2.728
4.6	2.466	2.602
5.0	2.506	2.708
5.4	2.561	2.743
5.8	2.643	2.723
6.2	2.715	2.785

Table 2: **Average latency of reserved messages**

suggests that the waiting time threshold (before transiting to the *starved* state) be a dynamically variable system parameter such that large values are used in the presence of small bulk sizes and small values are used for large bulk sizes.

6 Conclusions and Future Work

In this paper, a full-duplex architecture with control message separation for buffer insertion rings was presented. It was shown that the problem of starvation occurs in realistic LAN traffic. A protocol was introduced to ensure fair access to the network and it was shown that the protocol is correct, stable and possesses several attractive properties in addition to preserving the usual advantages of buffer insertion rings over other designs. Simulation results indicated that the introduction of the protocol caused negligible degradation in performance and, in any case, exhibited significantly better performance than competing systems.

For future work, we observe that there are several issues of interest worth pursuing here. Firstly, it is desirable to modify the design for handling variable-sized packets efficiently. Next, we note that, in order to support traffic with real-time constraints such as voice traffic (isochronous traffic), a further modification is required wherein certain timing guarantees must be made. In addition, there concerns about address-checking delays that need to be addressed and, although recent results [13] suggest that this problem may be overcome, it is desirable to study this effect in the context of the protocol presented here. Finally, we note that an investigation into the effects of various system parameters on overall performance such as the number of nodes, varying bulk sizes, and different classes of traffic would constitute an important direction for future research in this area.

Acknowledgements:

The authors would like to thank Israel Cidon and Inder Gopal of the IBM TJ Watson Research Center, New York, for several valuable comments and helpful discussions.

References

[1] W.Bux, "Local Area Subnetworks: A Performance Comparison", *IEEE Trans. on Communications*, Oct 1981.

[2] W.Bux and M.Schlatter, "An Approximate Method for the Performance Analysis of Buffer Insertion Rings", *IEEE Trans. on Communications*, pp. 50-55, Jan 1983.

[3] W.Bux, "Modeling Token Ring Networks - A Survey", *IBM Research Report*, RZ-1615, 1987.

[4] I.Cidon and Y.Ofek, "Metaring: A Ring with Fairness and Spatial Reuse", *IBM Research Report*, 1989.

[5] D.Dykeman and W.Bux, "An Investigation of the FDDI Media Access Control Protocol", *IBM Research Report*, RZ-1591, 1987.

[6] D.C. Flint, "The Data Ring Main - An Introduction to Local Area Networks", *Wiley*, 1983.

[7] E.R.Hafner, Z.Nenadal and M.Tschanz, " A Digital Loop Communication System", *IEEE Trans. on Communications*, June 1974.

[8] W.Hilal and M.T.Liu, "Analysis and Simulation of the Register-Insertion Protocol", *Proc. Computer Networking Symposium*, 1982.

[9] J.L.Hullet, "New Proposal Extends the Reach of Metro Area Nets", *Data Communications Magazine*, Feb 1988.

[10] A.A.Lazar, A.Patir, T.Takahasi and M.Zarki, "MAGNET: Columbia's Integrated Network Testbed", *IEEE J. Selected Areas in Communications*, pp. 859-871, Nov 1985.

[11] M.T.Liu and D.M.Rouse, "A Study of Ring Networks", Ring Technology Local Area Networks", I.N.Dallas and E.B.Spratt (Editors), *Elsevier Science Publishers*, 1984.

[12] W.M.Loucks, V.C.Hamacher, B.Preiss and L.Wong, "Short-packet Transfer Performance in Local Area Rings", *Proc. of IEEE Infocom.*, San Francisco, 1984.

[13] M.Xu and J.H.Herzog, "Concurrent Transmission and Short Loop Latency Protocol", *Local Computer Networks Conference*, 1988.

[14] M.Misson, J-J.Mercier and A.El.Oussoul, "A Fair Management of Communication Establishment for a Ring LAN", *Local Computer Networks Conference*, 1988.

[15] J.F.Mollenauer, "Networking for Greater Metropolitan Areas", *Data Communications Magazine*, Feb 1988.

[16] R.M.Newman, Z.L.Budrikis and J.L.Hullet, "The QPSX Man", *IEEE Communications Magazine*, Apr 1988.

[17] F.E.Ross, "FDDI - A Tutorial", *IEEE Communications Magazine*, May 1986.

[18] J.F.Shoch, "Packet Fragmentation in Inter-Network Protocols", *Computer Networks*, Feb 1979.

[19] W.Stallings, "Local Network Performance", *IEEE Communications Magazine*, Feb 1984.

[20] W.Stallings, "Fairness in LANs: Is the Performance Price Worth It?", *Data Communications Magazine*, Feb 1988.

[21] A.Thomasian and H.Kanakia, "Performance Study of Loop Networks Using Buffer Insertion", *Computer Networks*, Dec 1979.