An algorithm for detecting and resolving store-and-forward deadlocks in packet-switched networks

by

CHAN Cheung-wing

(陳長榮)

A master thesis submitted in partial

fulfilment of the requirements for the degree of

Master of Philosophy

in

the Department of Electronics

The Chinese University of Hong Kong

Hong Kong

May, 1986

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institution of learning.

Abstract

Store-and-forward (S/F) deadlocks in a packet-switched network can be totally avoided with the use of deadlock avoidance protocols. These protocols put so much restrictions on the use of buffers that even under normal circumstances the buffer utilization is small.

We propose a deadlock control algorithm that is entirely invisible under normal circumstances. As soon as certain channels in the network have trouble in accepting and transmitting packets due to the lack of buffers, the deadlock detection phase of the algorithm is invoked. When a deadlock is identified, the deadlock resolving phase of the algorithm is executed. Once the deadlock is resolved, the control is removed. The algorithm can be used in conjunction with either the Complete Partitioning or the Sharing with Maximum Queue Lengths output buffer allocation strategies. A proof on the correctness of the algorithm is given. Simulation results show that the network can maintain a relatively high throughput even when deadlocks are being detected and resolved. In addition, several properties of deadlocks are found: i) deadlocks start to increase abruptly once the network operates beyond its capacity; and ii) under heavy load condition, increasing the buffer size will not delay the occurrence of deadlocks.

Acknowledgements

I would like to express my deepest gratitude towards my supervisor, Dr. Tak-shing Yum, for his valuable advice, guidance and encouragement.

Thanks are also due to Mr. Wing-kay Kan and Mr. Chi-ming Yau for their helpful discussions and suggestions on the use of simulation languages.

I am also indebted to Ms Shun-tim Chan, who patiently read the manuscript and offered many detailed comments.

Table of Contents

I. Introduction

Store-and-forward (S/F) packet-switched network [1-2] is one of the most popular computer networks nowadays. It may be thought of as a distributed pool of resources (channels, buffers, switching processors etc) which can be shared dynamically by a community of competing users wishing to communicate to each other. This dynamic sharing of resources has the advantages of greater speed, more flexible in setting up user connections in the network and more efficient use of network resources after the connection is established.

But unless careful control is exercised on the user demands, the dynamic sharing do not come without danger: the users may seriously abuse the network. In addition, if the offered load are allowed to exceed the network capacity, unpleasant congestion effects will occur which will rapidly neutralize the fast response and efficient advantages of the network. So properly monitoring and controlling the offered load is necessary and is called **flow control procedure**. M.Gerla [3] has made a good comparative survey in this field.

The main functions of flow control in a packet-switched network are:

1) prevention of throughput degradation and loss of efficiency due to overload.

2) fair allocation of resources among competing users, and

3) speed matching between the network and its attached

users.

One of the useful methods in flow control is the output buffer allocation strategy [4-6] in which five categories are classified. They are:

1) Complete Sharing (CS). Letting B_i be the number of packets in the ith queue and B be the total buffer size, we have the following constraint:

$$0 \leq B_i \leq B \qquad \forall i$$

2) Complete Partitioning (CP). Let N be the number of output queues. The constraint becomes:

$$0 \leq B_i \leq B/N \qquad \forall i$$

3) Sharing with Maximum Queue Length (SMXQ). Let b be the maximum queue size allowed (where b > B/N). We have:

$$0 \leq B_i \leq b \qquad \forall i$$

$$B_i \leq B$$

4) Sharing with Minimum Queue Length (SMA). Let c be the minimum buffer allocation which is guaranteed to each queue (typically, c $\leq$ B/N). The control then becomes:

$$\max(0, B_i - c) \leq B - N \cdot c$$

5) Sharing with Minimum and Maximum Queue Length. This strategy combines both SMXQ and SMA.

M.Irland [4] has made an analytic analysis in these partitioning strategy, as shown in Fig.1-4.

Anyway, flow control procedures can guarantee the network to have a good performance except that one catastrophic problem does not happen: **Deadlocks**. Once deadlocks happen, the entire network can be completely inoperative even under normal traffic condition. A good survey of potential deadlocks in S/F networks can be found in [7]. There are altogether six types of deadlocks in S/F packet-switched network :

1) Direct S/F deadlock, where two adjacent nodes of a network both are filled up with packets waiting for transmission to the other node. Then none of these packets can be transmitted since there is no free buffer available (and none will become available) at the receiving node, as depicted in Fig.5.

2) Indirect S/F deadlock, where more than two nodes are involved in the deadlock. It may occur even if the precautions against direct S/F deadlock have been taken. This can easily be demonstrated by a circular network, each node of which is filled with packets directed to the next node in some cycle orientation, as depicted in Fig.6.

3) Progency deadlock, where original messages spawn other ones, and buffer contention occurs between the original and progency messages. This occurs when positive or negative acknowledgments are created. e.g. if messages reverse direction after encountering a path failure.

4) Copy-release deadlock, where a message copy is stored at the

source node and the buffer is not released until an acknowledgement is received from the destination node. Buffer contention may arise among the original messages, stored copies and acknowledgements.

5) Pacing deadlock, where a local flow control protocol is used between a network node and attached terminals. Buffer contention may arise between the message flows into and out of the terminal, preventing the transmission of go-ahead commands.

6) Reassembly deadlock, whereby reassembly of packetized messages at the destination node cannot be completed. Two types of reassembly deadlocks have been recognized.

i) The first type is completely analogous to situations which may arise in primitive computer operating systems if requests for storage units are granted freely, as long as sufficiently many units are available. In the context of reassembly of messages from packets such an 'expedient' allocation strategy may cause the entire buffer pool to be filled with incompletely reassembled messages such that none of them can ever be completed.

ii) The second type of reassembly deadlock is somewhat more intricate. It occurs when reassembly of messages in a certain node N cannot be completed due to the following: all the reassembly buffers at node N are either occupied or reserved for awaited packets of partially reassembled messages. The neighbours of node N are filled with S/F packets also having destination N for which no reassembly buffers are being reserved. Thus the neighbours of N prevent the remaining packets of the partially reassembled messages at node N from reaching their destination.

In all types of deadlocks mentioned, direct and indirect S/F deadlocks are of particularly important and have received considerable attention in recent years. There are two main streams to tackle with these two types of deadlocks: prevention techniques and detection and resolution techniques.

1) Prevention Techniques

The typical approach of this technique is to institute some form of flow control based on buffer reservation. These solutions involve partitioning the buffer pool at a node into several classes and permitting only a restricted set of packets access to a given buffer class. The structure buffer pool technique proposed by Merlin [8] is the buffer reservation of this kind based on the number of hops.

In this technique, packets arriving at each node are divided into classes according to the number of hops they have covered. For example, packets entering a node from the host belong to class 0 of that node, since they have not yet covered any hops. The highest class H_{max} corresponds to packets that have traversed H_{max} hops, where H_{max} is the maximum path length in the network (a function of the topology and the routing algorithm). The highest class H_{max} also includes all the packets that have reached their destinations. The nodal buffer organization reflects this class structure as shown in Fig.7. Each packet class has the right to use a well-defined set of buffers. Class 0 can access only the buffers available in set 0. Class i+1 can use all the buffers available to class i. Finally, class H_{max} can access all the buffers available to class $H_{max}-1$. Hence, all the buffers in each node have been partitioned and well-structured and a buffer graph is created.

It can be easily shown that this techniques eliminates deadlocks of both the direct and indirect type. First a deadlock occurs if and only if there is a cycle in the buffer graph, i.e., there is a chain of arcs which starts from one buffer, and terminates at the same buffer. But in the buffer graph, no cycle can occur since each arc starts from a buffer of class i and points to a buffer of class i+1 (recall that a packet gains seniority at each hops; an illustration of this property is shown in Fig.8). Thus this technique is deadlock-free.

This technique is easy to implement and has an implicit hop-level flow control. But the main drawback is that this scheme requires that a certain minimum number of buffers be present in a nodes. For example, deadlock-free cannot be guaranteed if all the nodes contain fewer that $h_{max}+1$ buffers, where h_{max} is the maximum number of hops that any packet in the network will have to travel. Even if we make the assumption that the routing used in the network is deadlock-free, i.e., the same node is never visited more than once by a packet, h_{max} must still be no smaller than N-1 where N is the number of nodes in the network.

To minimize the number of buffer classes necessary in each node, other solutions have included buffer reservation based on the number of "valleys" [7], the route which the packet is travelling on [10] and the counting of negative hops [11].

Another interesting solution by Gelernter [12] attempts to prevent S/F deadlocks by a deadlock-free flow control procedure. This procedure has the advantages of 1) no restriction in the routing of packets, 2) no partitioning of buffer-pool and 3) the size of buffer pool at each node being independent of the network size. Unfortunately, as network congestion increases, this algorithm requires rerouting of packets and, in the worst case, it may even loss some of them.

(2) Detection and resolution techniques

J.Blazewicz [13] proposed a distributed deadlock-resistant flow control procedures which can detect and then remove direct S/F deadlocks at negligible cost. But for indirect S/F deadlocks, the deadlock detection and recovery is much more expensive.

Gambosi [14] proposed another deadlock detection and removal algorithm to resolve both direct and indirect S/F deadlocks. First Gambosi pointed out two important criteria that a deadlock control algorithm should fullfil:

1) Deadlock detection should be performed in a distributed fashion since any form of centralized control would certainly incur in an unreliable and inefficient operation of the

algorithm.

2) The algorithm should exhibit negligible overhead for nodes not involved in a deadlock. In other words, normal network operation should not be affected by any kind of deadlock control traffic.

He then proposed a two-part algorithm: one for deadlock detection and the other for deadlock recovery. The detection algorithm can detect deadlocks by constructing a Site Blocking Graph (SBG). Once a deadlock is detected, then based on SBG, a distributed deadlock recovery procedure is applied to resolve the deadlock. Nevertheless, the whole algorithm relies on a SBG whose construction is quite time-consuming.

At the first glance, the prevention techniques seem to be more favourable than the detection and resolution techniques. But after studing the Gambosi criteria and the Gelernter three advantages, we find most of prevention algorithms cannot fulfil these criteria (the structure buffer pool technique sacrifies the Gelernter three advantages and violates the Gambosi second criterion). In addition, intuitively, S/F deadlocks rarely happens under normal traffic condition in a properly designed network. But any deadlock prevention algorithm applied to the network will surely deteriorate its performance (e.g. network throughput, delay etc.). Thus, in this respects, the detection and resolution techniques are more desirable than the prevention techniques.

In this thesis, a different deadlock detection and resolution algorithm is proposed. The algorithm, first based on the output buffer allocation strategy being CP and later extended to SMXQ strategy, not only satisfies the Gambosi criteria, but also shares the three advantages of the Gelernter algorithm under normal traffic condition. Moreover, packets will not get lost with this new algorithm.

In the following, we shall first introduce a network model (Chapter II). We then describe the deadlock control algorithm for the CP strategy (Chapter III) and give two examples for illustration (Chapter IV). This is followed by a correctness proof of the algorithm (Chapter V). A modified deadlock control algorithm for the SMXQ strategy is then introduced (Chapter VI). Finally, using simulation, the performance of the algoithm is determined and some interesting properties of deadlocks are presented.

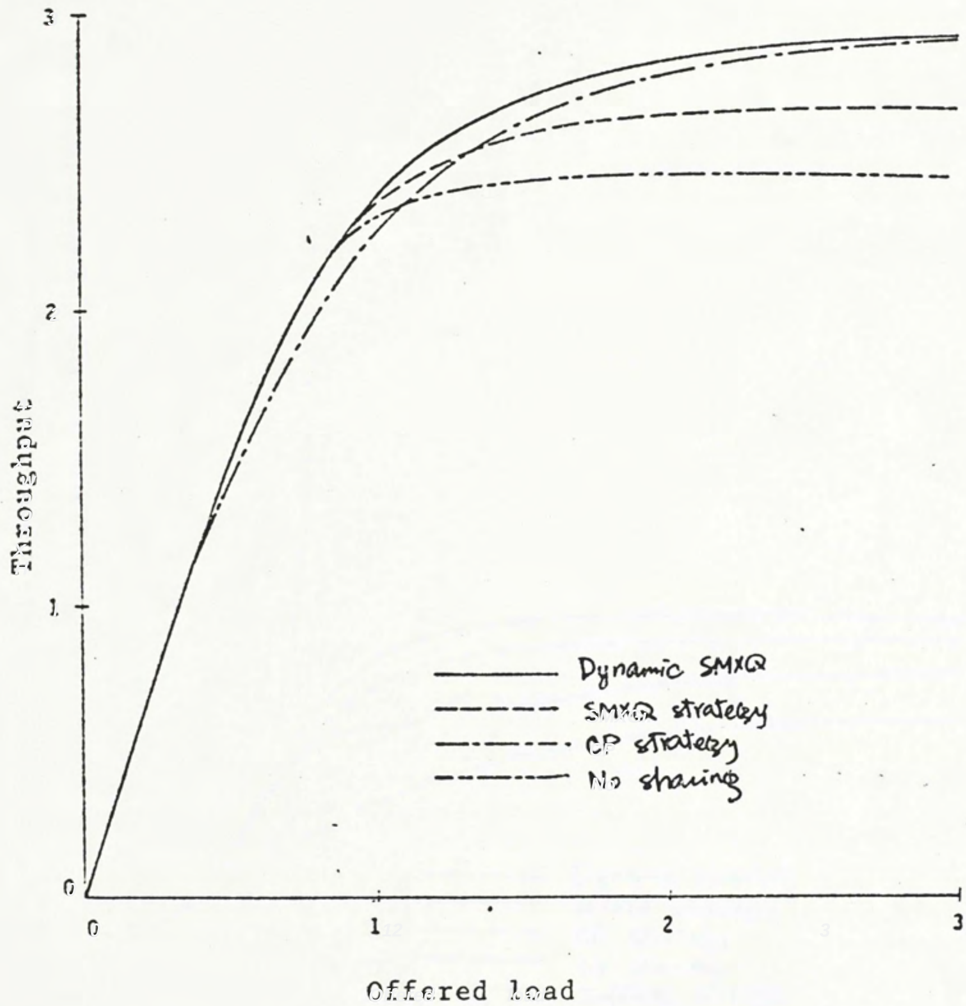


Fig.1 Throughput versus offered load: balanced load (from [4])

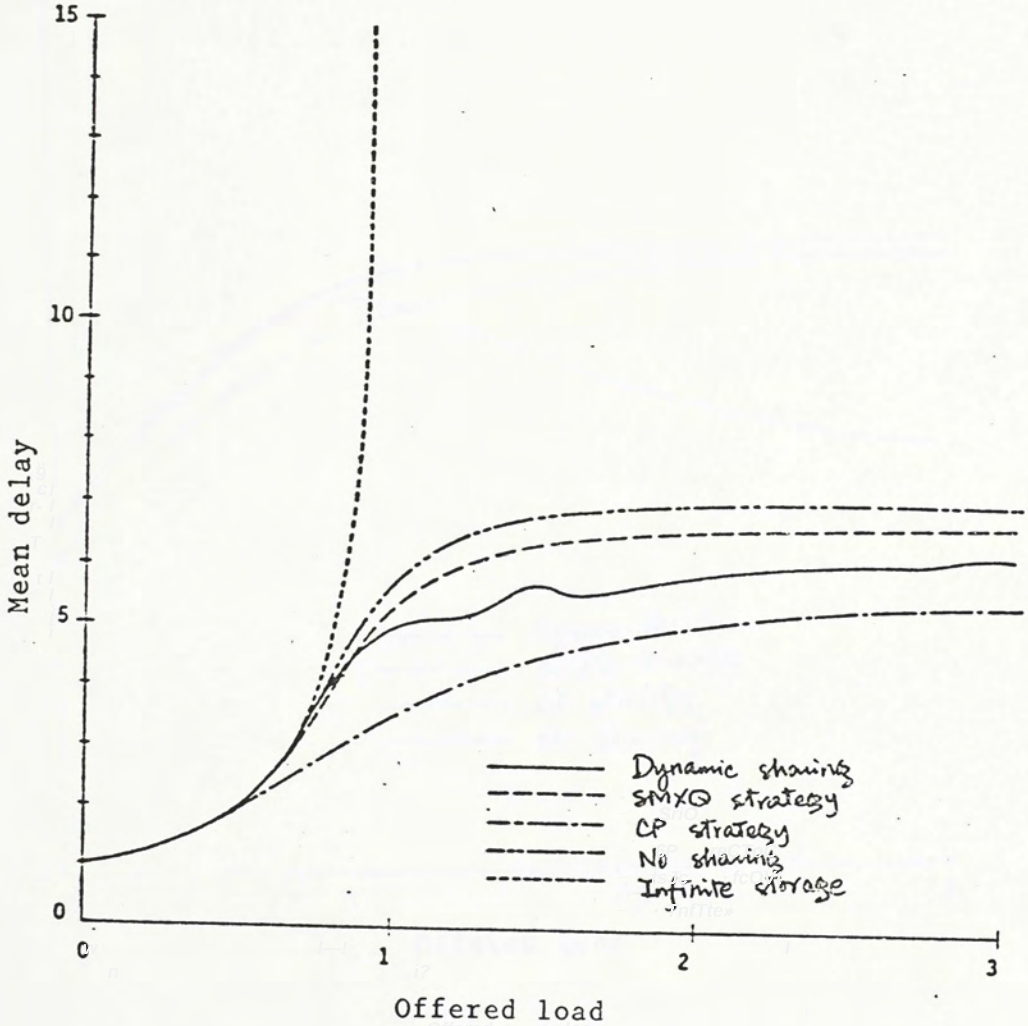


Fig.2 Mean delay versus offered load: balanced load (from [4])

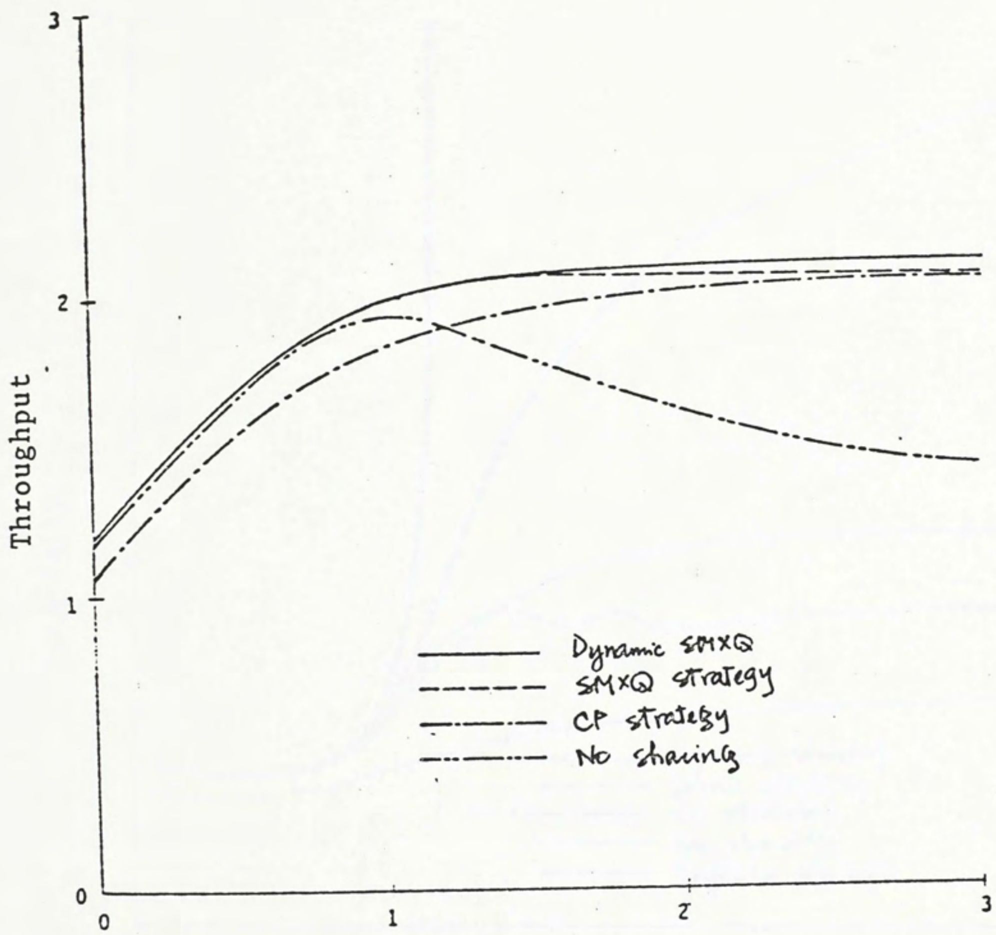


Fig.3 Throughput versus offered load: unbalanced load (from [4])

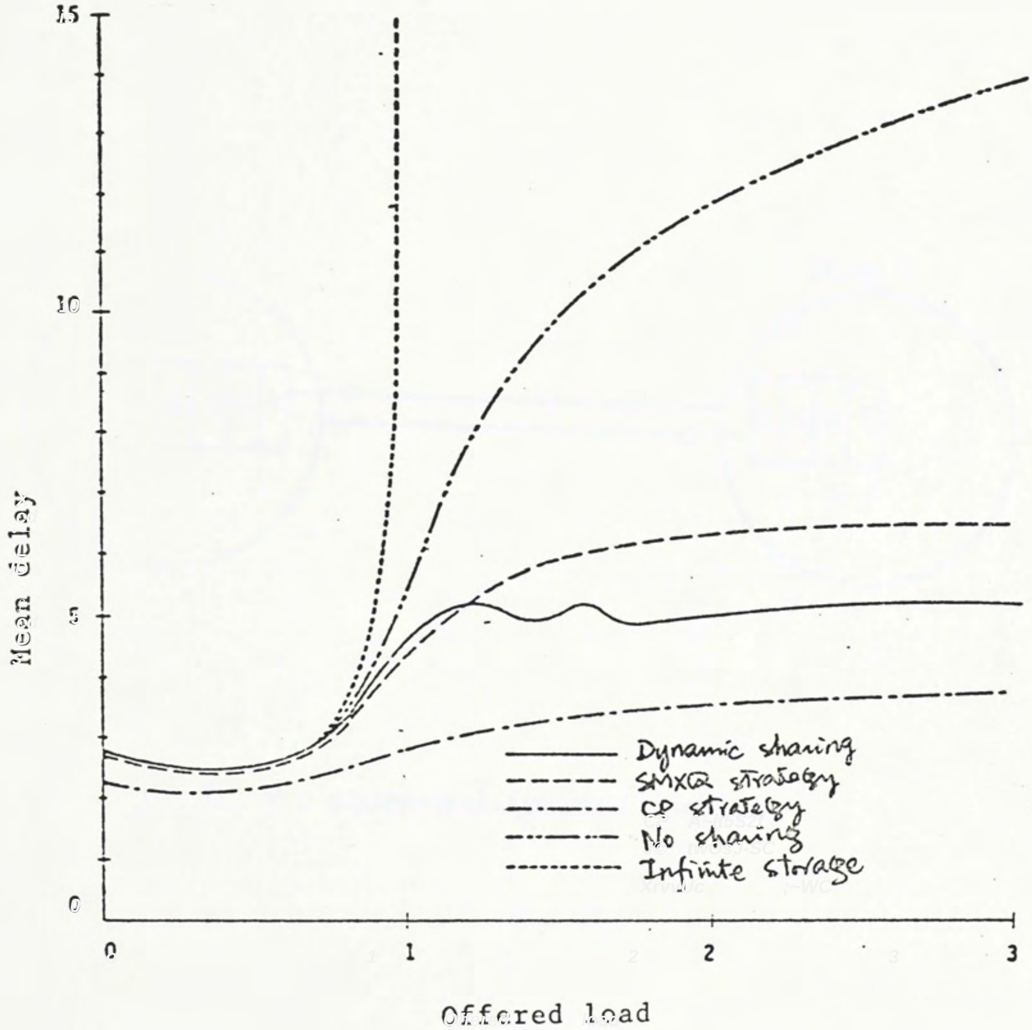


Fig.4 Mean delay versus offered load: unbalanced load (from [4])

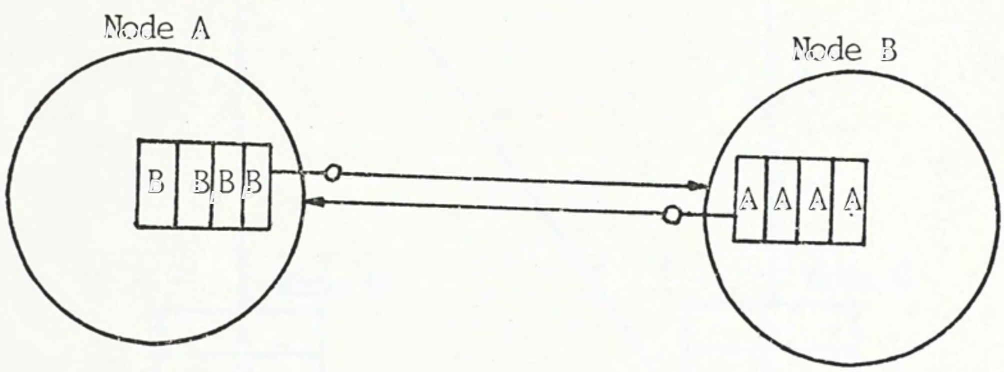


Fig.5 Direct store-and-forward deadlock

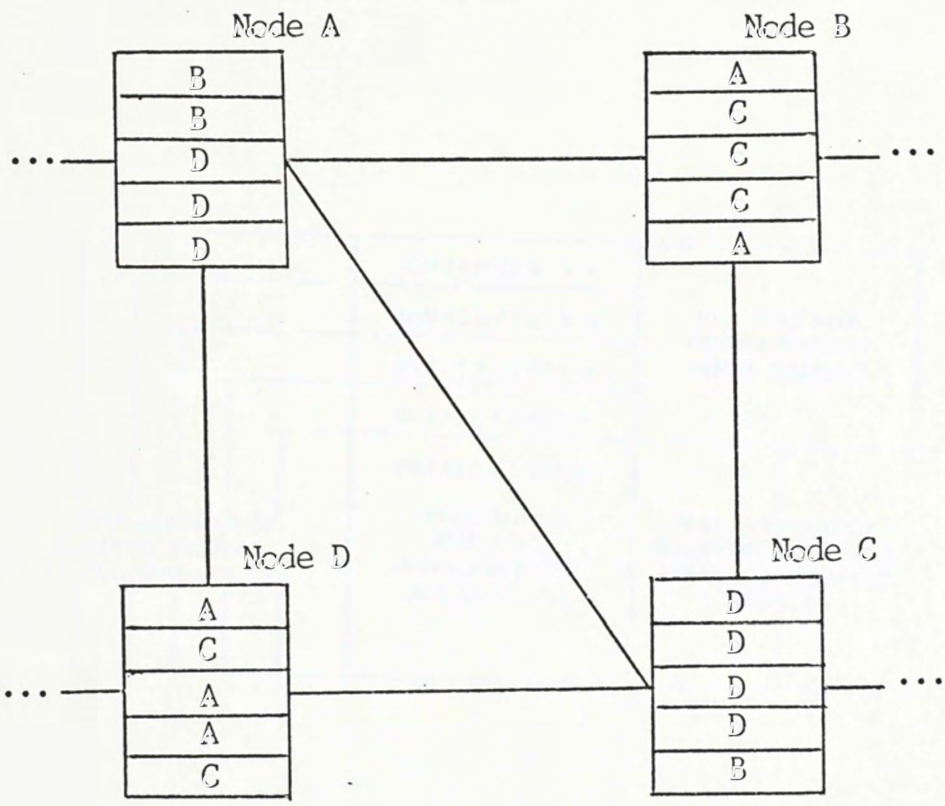


Fig.6 Indirect store-and-forward deadlock

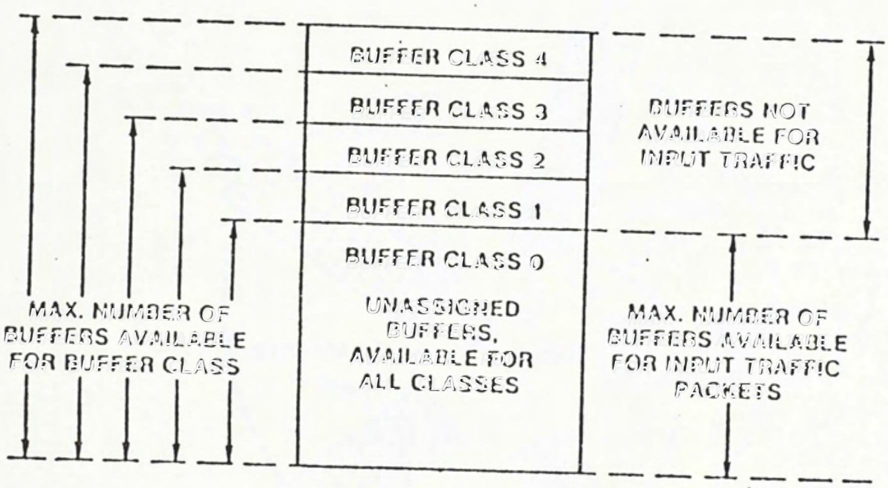


Fig.7 Structured buffer pool

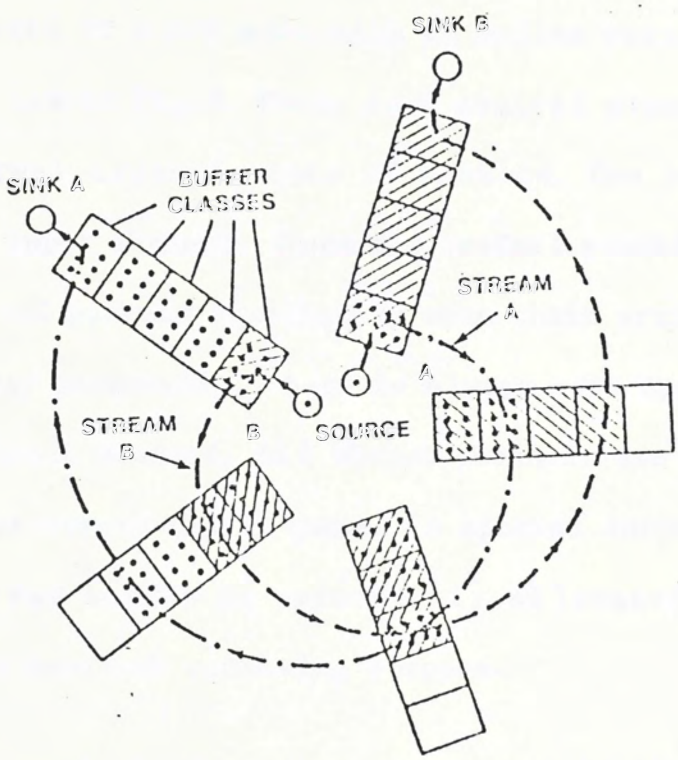


Fig.8 Access to buffer classes. Example for two data streams. Dotted area: buffer available for stream A; hatched area: buffer available for stream B.

II. Network Model

Consider a S/F packet-switched network with channels always reliable. Let all inputs to this network be fixed-size packets, each occupies one unit of buffer space. All packets are acknowledged or negative-acknowledged depending on whether they are accepted or not. The overhead of the acknowlegement traffic is neglected.

A typical model of a S/F node with CP buffer partitioning strategy is depicted in Fig.9. There is a central processor to handle all internal transmissions of packets. One buffer is reserved for each input channel. Since the central processor will process the received packets immediately upon their arrivals and will move them away afterwards, there is always room for further receipts at each input channel. All output channels are modelled as first-come-first-served (FCFS) queues. A special output buffer called the reserved buffer is permanently allocated at each output channel for deadlock resolving purpose.

The deadlock control algorithm is executed by the channel processors, each of which works independently. An output channel can be in one of the following three states: 1) Normal state (N), 2) Test state (T), and 3) Deadlock-resolving state (DR). A state transition diagram is depicted in Fig.10.

Depending on the channel state, outgoing packets are attached with different headers:

1) Normal header. It has a normal header identity, source and

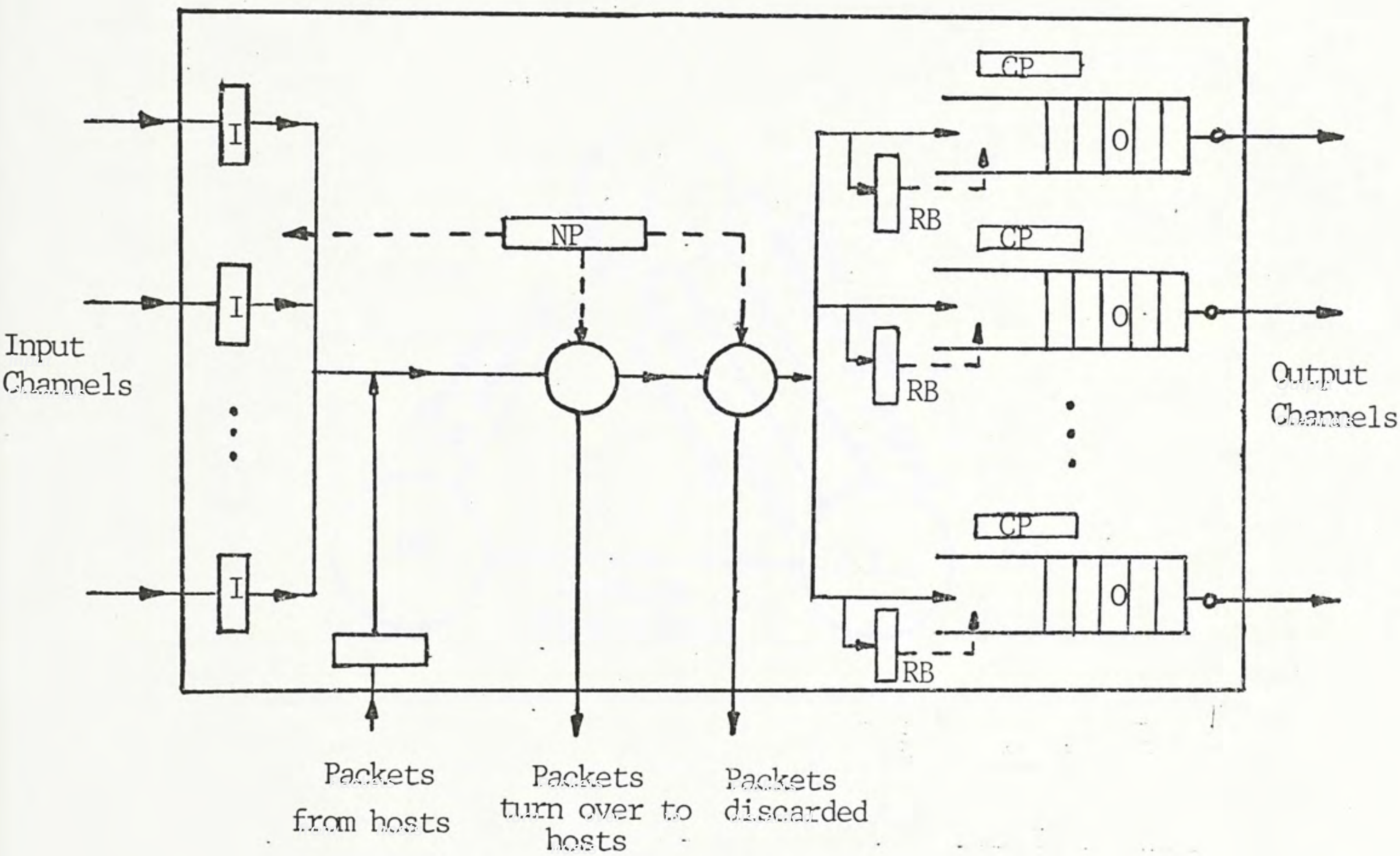
destination node addresses and a header check-sum.

2) Test header. Besides a header identity and all the normal header information, it also includes (i) an integer x indicating the total number of DR packets (packets with DR headers attached) to be transmitted and received upon detecting a deadlock and (ii) an I field recording a set of channel identities in state T.

3) DR header. It contains the integer x, a bit pattern for DR header identification and all the normal header information.

Upon receiving a packet from an adjacent node, the node processor will check whether the packet is destined for the present node or not. If yes, the processor will immediately turn the packet over to the local host. If not, the packet will be routed to one of the output queues, say queue i. If queue i is full, then except for the DR packets which are put into the reserved buffer associated with queue i, all other kinds of packets are discarded after extracting the header information. Similarly, packets from the local host are accepted if queue i is not full.

Note that a packet, once stored in a buffer, does not have to be physically moved. Movement of packets depicted in Fig.9 may be accomplished by software pointers.



I : Input buffer   O : Output buffer   RB : Reserved buffer

CP : Channel processor   NP : Nodal processor

Fig.9 Model of a S/F node

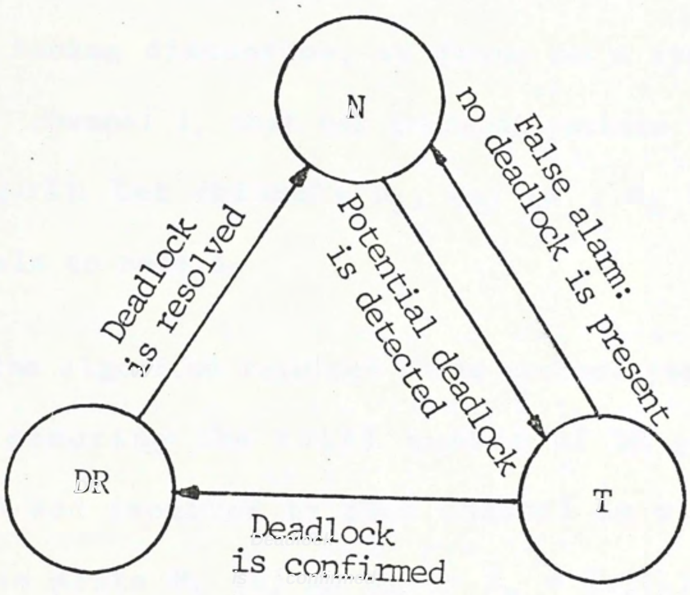


Fig.10 State transition diagram for CP strategy

III. The deadlock control algorithm with CP strategy

Conceptually, a S/F deadlock refer to the situation where there is a cycle of buffer requests among a set of nodes, all of which have no empty buffer left. Our algorithm is based on detecting the presence of these cycles and then resolving them efficiently. We will neglect, in our model, those packets destined to the local node as they will be turned over to the host immediately and will not impose demand on the output buffer.

In following discussion, we focus on a typical one-way channel, say channel i, that can transmit packets from node A to node B (Fig.11). Let channels $n_1, n_2, \ldots, n_R$ be the set of input channels to node A.

Here, the algorithm requires three control parameters: i) an integer y denoting the total number of DR packets to be transmitted and received by each channel in state DR before returning to state N, ii) an array $S_x = [s_x(1), s_x(2), \ldots, s_x(R)]$ where $s_x(r)$ records the integer x collected from channel n_r, and iii) a set of channel identities S_I. When channel i is in state N, these parameters are set to:

$$y = 0 \qquad (1)$$

$$S_x = [s_x(1), s_x(2), \ldots, s_x(R)]$$
$$= [M, M, \ldots, M] \qquad (2)$$

$$S_I = [\ ] \qquad (3)$$

where M is an integer larger than the output buffer size of channel i.

Normally, channel i is in state N. It will change to state T if a potential S/F deadlock is detected. If it is a false alarm, then channel i will go back to state N. Otherwise it will change to state DR which, after the deadlock is resolved, go back to state N. Therefore, the algorithm consists of three procedures, one for each of the three states.

(i) Procedure for state N

At state N, all normal and test packets received are placed in the output buffer of channel i. But if the buffer is full, the received packets are discarded.

Channel i will change from state N to state T if the two conditions are satisfied: 1) the buffer of channel i is full and 2) the head packet (the packet in the first position of the output queue) has waited in the output queue longer than a time-out period, say T_{out} secs.

Comments We declare that a potential S/F deadlock involving channel i is detected when channel i cannot receive and transmit any packet in a finite time T_{out} secs.

(ii) Procedure for state T

In state T, channel i will discard all normal packets received. When a test packet is received from, say channel n_r, the channel i processor will discard the packet body and extract the x and I fields from the packet header. It then checks whether its channel identity i is in the I field or not. If it is in, channel i will declare the detection of a deadlock, change its state to DR and set y to x. If i is not found in I, $s_x(r)$ and S_I are updated as follows:

$$s_x(r) \leftarrow x$$

$$S_I \leftarrow S_I \cup [i]$$

The receipt of a DR packet indicates that channel i is already involved in a deadlock. The DR packet will be accepted and placed in the reservdd buffer of channel i. Also y is set to x which is obtained from the DR packet header. Channel i then changes its state from T to DR.

When channel i is in state T, only test packets are transmitted. If the head packet is to be routed to channel j (Fig.11), then the x and I fields in its header are set as:

$$x = \min[k_{ij}, s_x(1), s_x(2), \ldots, s_x(R)] \quad (4)$$

$$I = S_I \cup [i] \quad (5)$$

where k_{ij} denotes the total number of packets in the output buffer of channel i to be routed to channel j. Once a new test header is created, S_x and S_I are reset according to eqns. (2) and (3) (i.e. all their entries are used once only).

On the other hand, when the head packet transmitted by channel i is accepted by channel j, channel i will change from state T to N.

Comments As mentioned before, the sufficient conditions for the existence of a deadlock are a) the presence of a cycle of buffer requests and b) all buffers in that cycle being full. It is equivalent to having a set of channels, all in state T, forming a loop as depicted in Fig.12. All channels in that loop will transmit test packets. The I field in the headers of these packets will, according to eqn (5), gradually accumulate the identities of these channels. Sooner or later, one or more channels in that loop will receive test packets with their own identities included in I. If that happens, these channels realize immediately that they are involved in a deadlock and change their state from T to DR. Moreover, they will, in turn, transmit DR packets to inform the other channels in the loop that a deadlock is present.

(iii) Procedure for state DR

At state DR, all normal and test packets received are discarded. When a DR packet is received, it is placed in the reserved buffer of channel i and joins the end of the output queue. When a copy of a DR packet from channel i is accepted by the neighbouring node, the buffer storing the DR packet is freed and treated as the reserved buffer for receiving other incoming DR packets.

Note that only those packets having the same destination channel as that of the head packet are selected as DR packets to be transmitted and there are always y or more of these in the output buffer. The integer x in these DR packet headers are set to y.

After y DR packets are transmitted and received, channel i will change back to state N.

Comments During the process of resolving a deadlock, the number of DR packets to be forwarded and received is y which is, after repeated use of eqn (4), equal to $\min[k_{ij}, k_{jk}, k_{kl}, \ldots, k_{n_r i}]$ where i, j, k, l, ... , n_r are the channel identities of the closed loop.

Meanwhile, since DR packets will only be transmitted to those channels involved in a deadlock, channel i in state N will not receive DR packets.

Figs. 13 to 15 show the flow-charts for the above procedures.

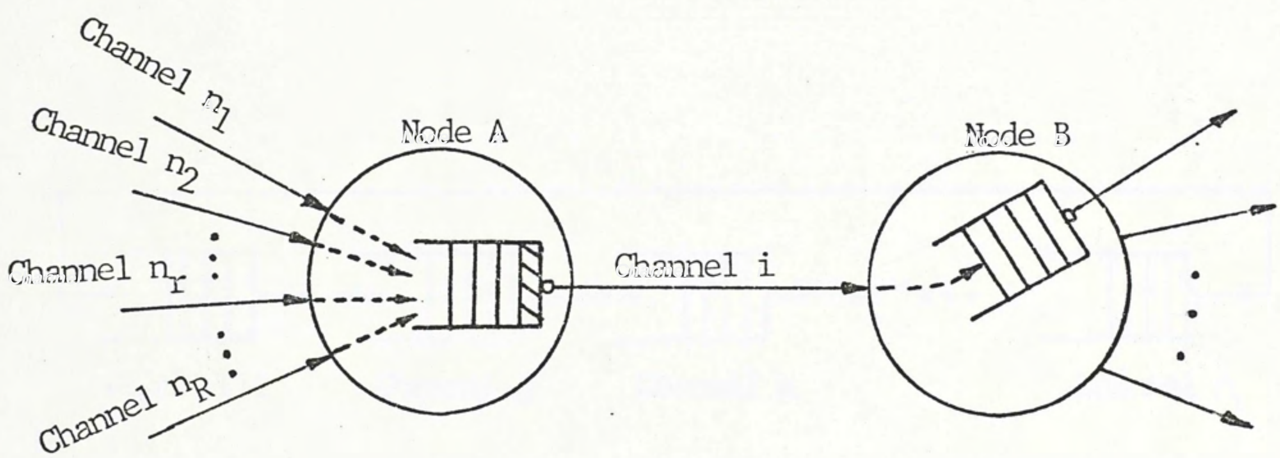


Fig.11 A typical channel

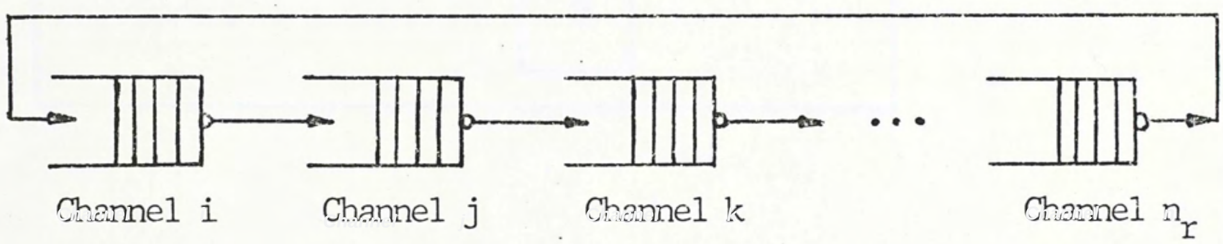
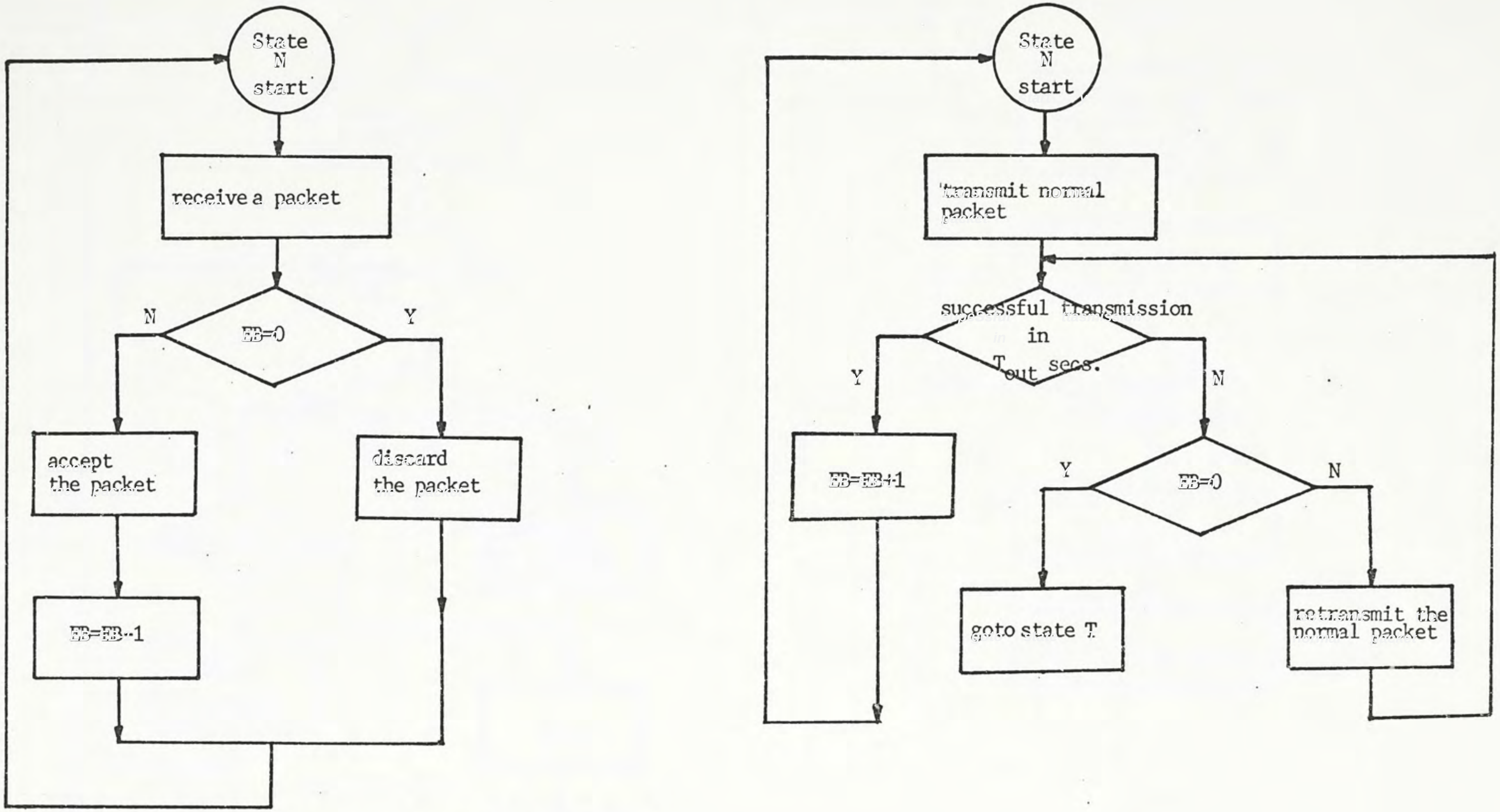


Fig.12 A closed loop of channels

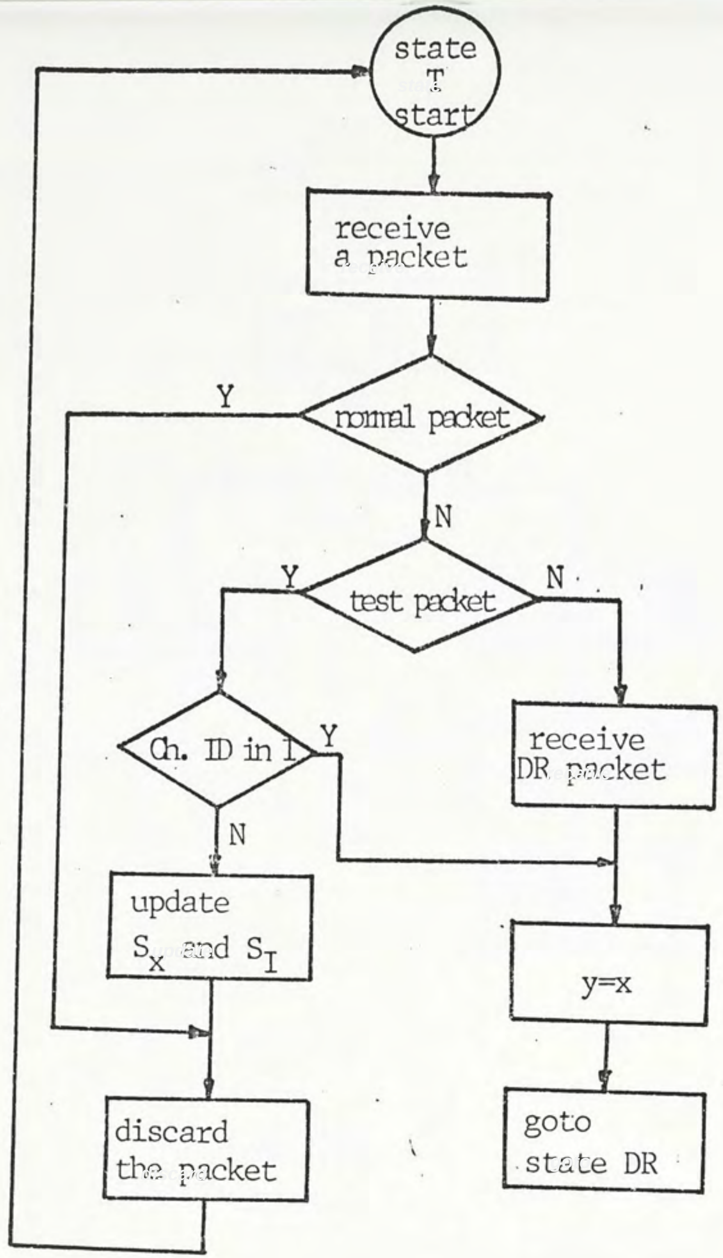


EB = the number of empty buffer

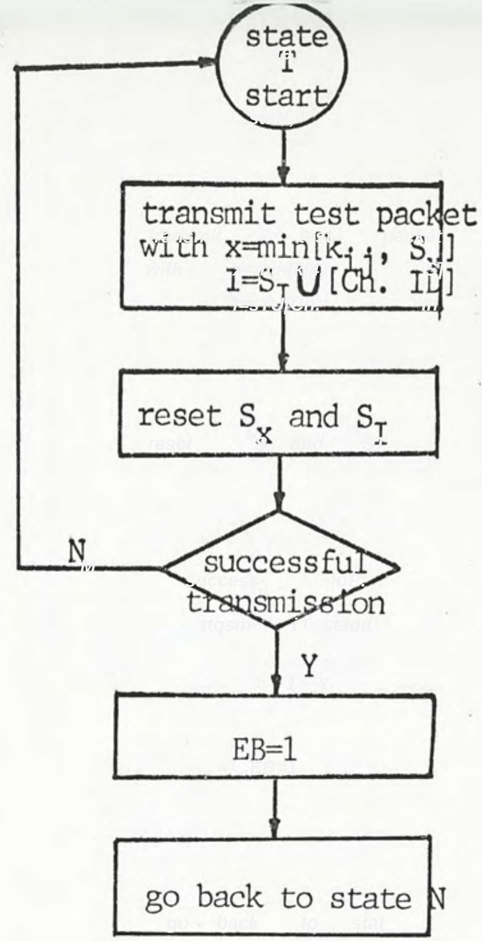
i) receiving side

ii) transmitting side

Fig.13 Flow-chart for state N

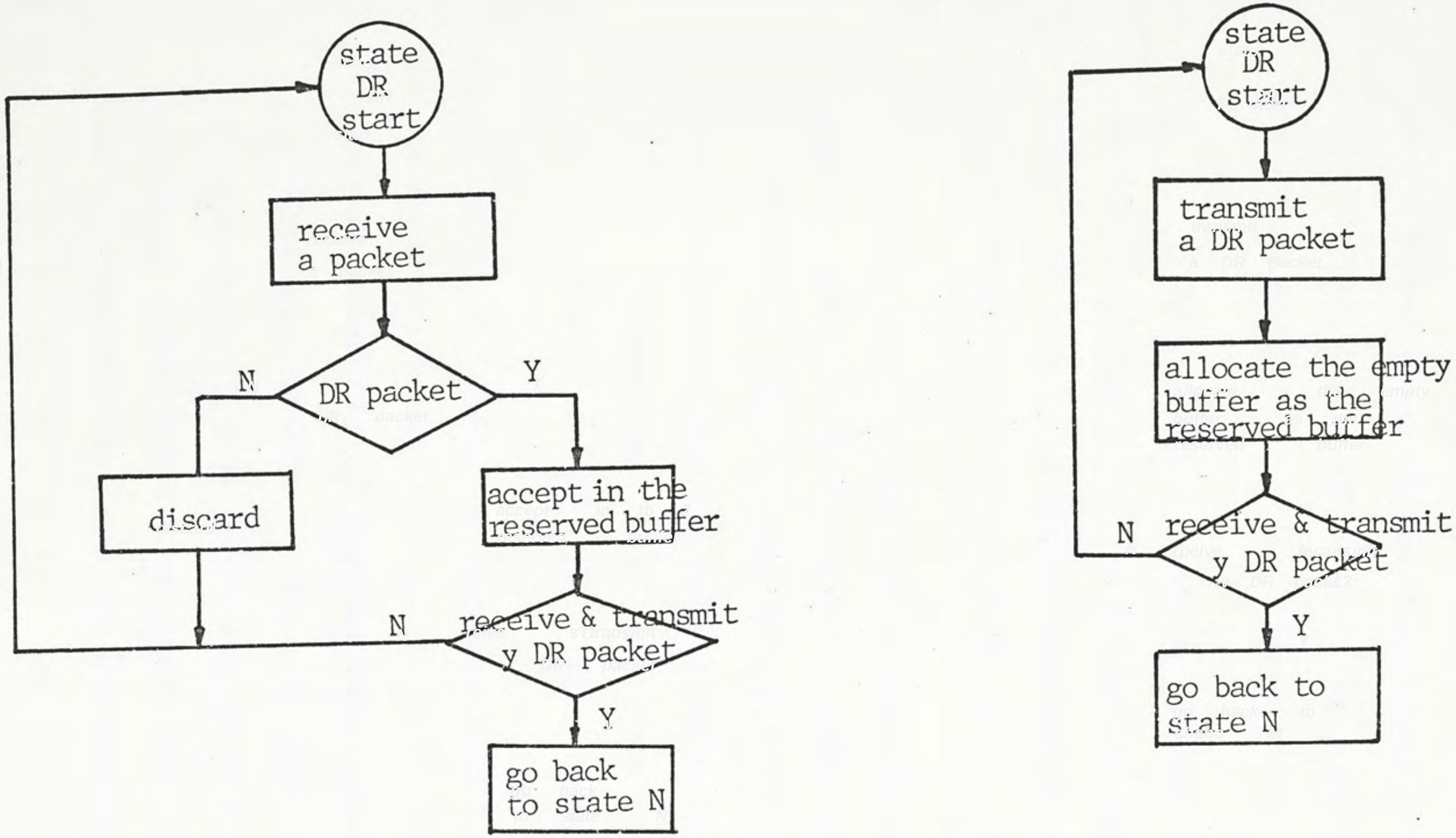


i) receiving side



ii) transmitting side

Fig.14 Flow-chart for state T



i) receiving side

ii) transmitting side

Fig.15 Flow-chart for state DR

IV. Examples

1) To clarify the operation of these procedures, consider the example shown in Fig.16. Initially let channel i be in state N. At t_1, channel i detects a potential deadlock and enters into state T. Let us assume that the head packet of channel i is to be routed to channel j; and let the total number of packets to be forwarded to channel j be four (i.e. $k_{ij} = 4$).

Let us say in $[t_1, t_2)$, no test packet is received and at t_2, channel i transmits a test packet. The x and I fields in the packet header are, according to equations (4) and (5), equal to 4 and [i] respectively since S_x = [M, M, ... , M] and S_I = [ ].

In $[t_2, t_3)$, let two test packets be received: one from channel n_1 with x=2 and I=[n_1] and the other from channel n_2 with x=1 and I=[n_2]. Since i is not in I, S_x and S_I are updated as [1, 2, M, ... , M] and [n_1, n_2] respectively. At t_3, another test packet is transmitted. This test packet will contain x = min[1,2,4] = 1 and I = [n_1, n_2] U [i] = [n_1, n_2, i].

In $[t_3, t_4)$, let a DR packet be received from channel n_1 with x=1 before a test packet is received from channel n_2. The DR packet is accepted in the reserved buffer and that causes channel i to enter state DR with y=1. The test packet is discarded upon its arrival. At t_4, a DR packet with x=y=1 is transmitted. Since now channel i has received and transmitted y DR packet, it knows that the deadlock is resolved and changes to state N.

2) Consider a set of four channels as shown in Fig.17. At t_0, these channels are all in state N with their head packets to be forwarded to their adjacent channels (i.e. A-->B, B-->C, etc). Let the packet transmission time and the time-out period be 1 and 3 time units respectively.

In $[t_0, t_1)$, channels A, B and D fail to forward their head packets due to the lack of buffers in the receiving ends. But channel C succeeds in forwarding its packet to D and therefore has an empty buffer left while making channel D's buffer full. At t_2 a packet from another adjacent channel (i.e. not from B) is accepted by C. Now all buffers are full and all head packets cannot be forwarded to their adjacent channels, a deadlock loop is formed (Fig.18a).

At t_3, channels A, B and D change from state N to T and channel C follows at t_5. The test packet header information during $[t_5, t_6)$ is shown in Fig.18b.

At t_7, channel D detects the deadlock by receiving a test packet with its own identity in the I field. It then switches to state DR, sets y=1 and starts to forward a DR packet to A (Fig.18c). After receiving a DR packet from D at t_8, channel A will change its state to DR and send a DR packet to B. Since y=1, A can change its state back to N. Similarly, channel B, C and D will also change their states to N after transmitting and receiving one DR packet (Fig.18d). The deadlock is therefore resolved.

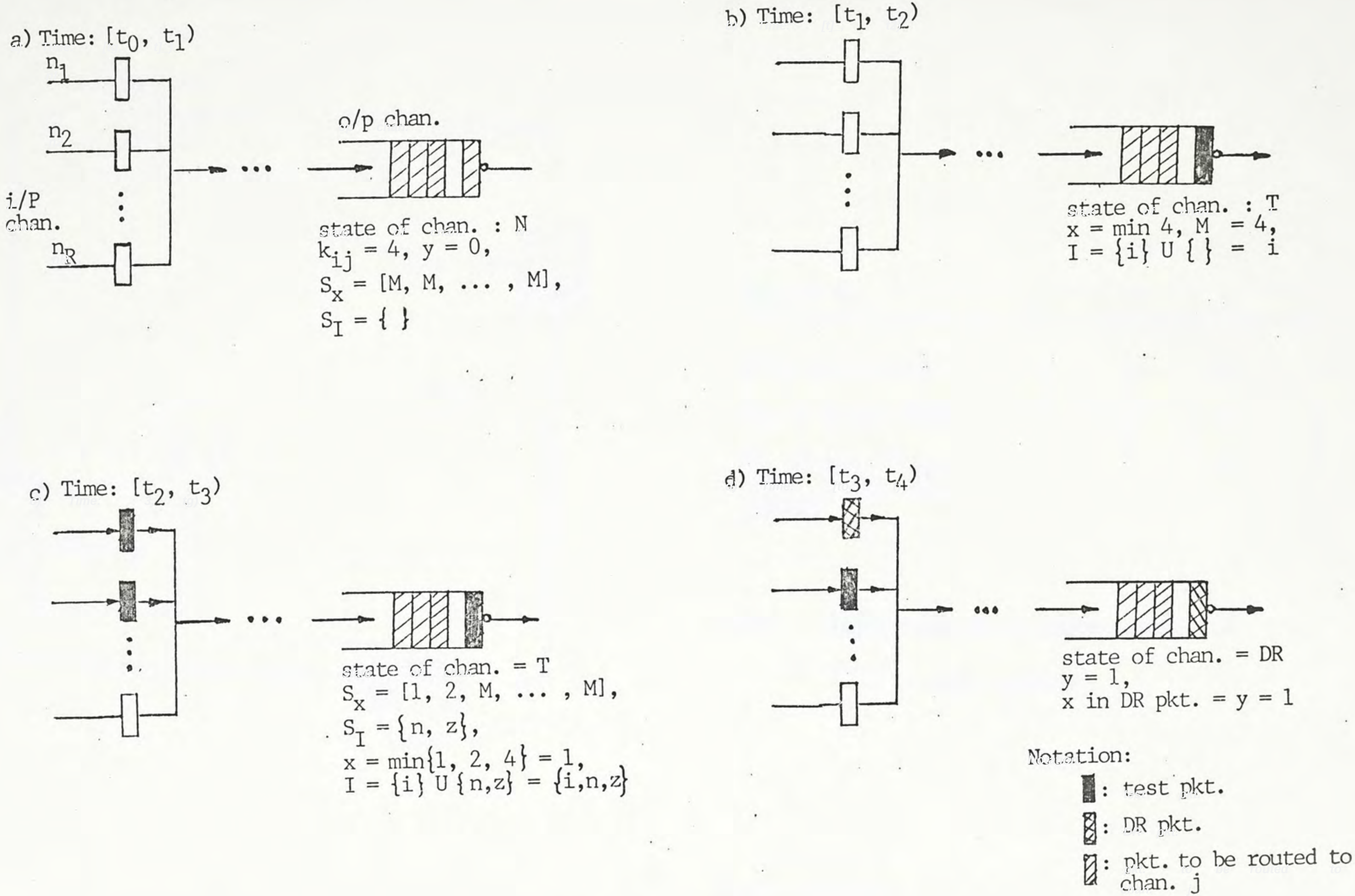
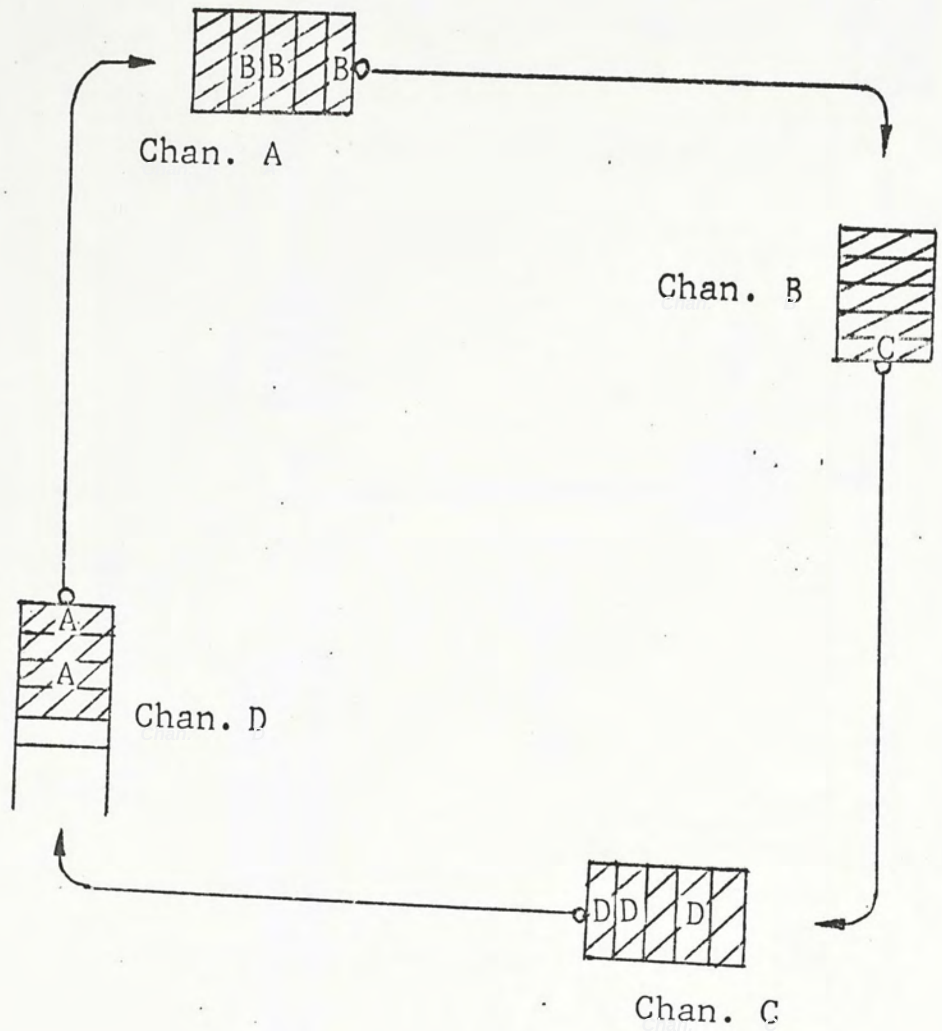


Fig.16 Example 1

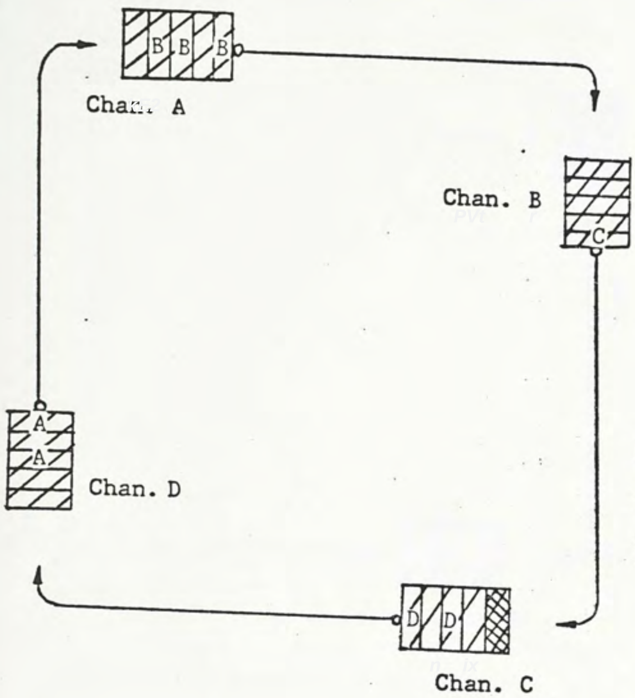


Channel No.	A	B	C	D
State of channel	N	N	N	N
No. of buffer left	0	0	0	1
No. of pkt. to be routed to the next channel	3	1	3	2

Note: [X] = pkt to be routed to chan. X

[ ] = empty buffer

Fig.17 Example 2

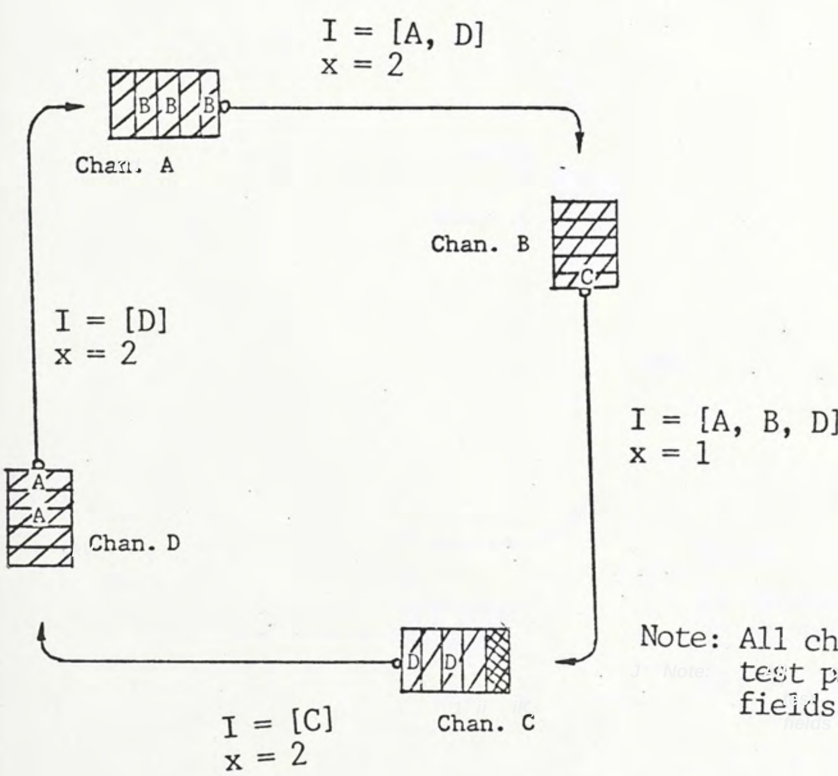


Note:

i) all channels are in state N

ii) [cross-hatched box] = packet received from another adjacent channel, but not from channel B

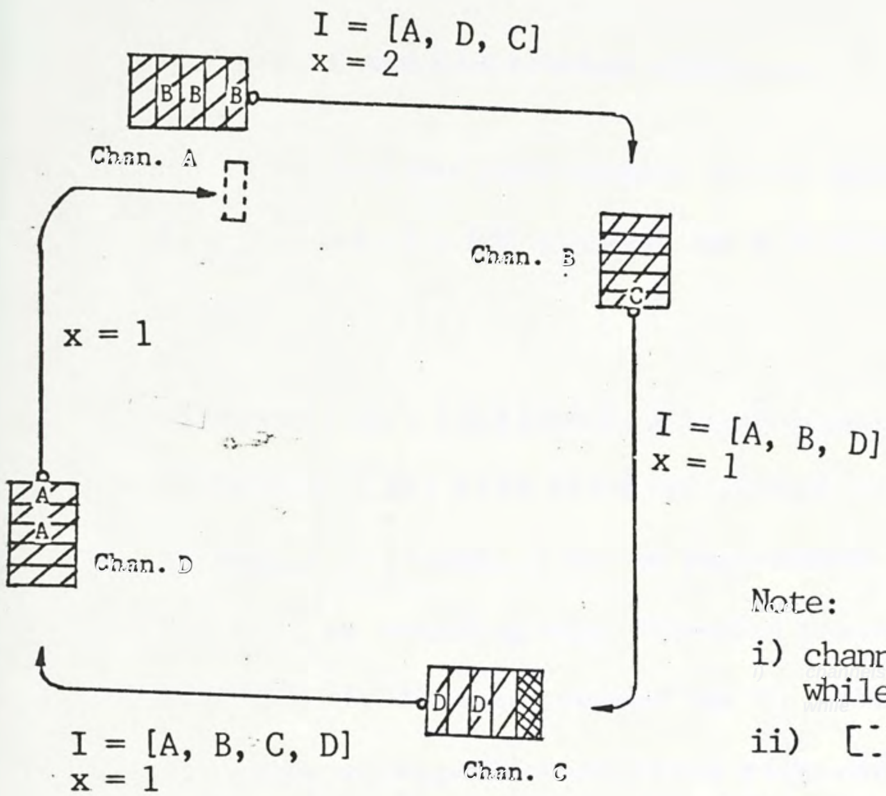
a) $t = t_2$



Note: All channels are in state T transmitting test packets with suitable x and I fields enclosed

b) $t_5 < t < t_6$

Fig.18
Cont./

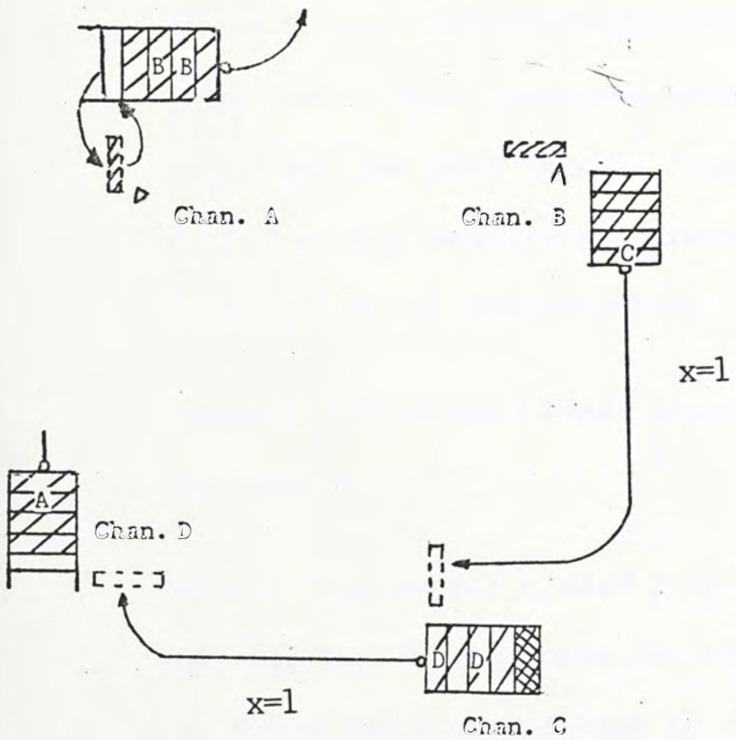


Note:

i) channels A, B and C are in state T while channel D is in state DR

ii) [ ] = emergency buffer

c) $t_7 < t < t_8$



Note:

i) channels B, C and D are in state DR while channel A is back to state N

ii) [ZZZ] x = emergency buffer contained DR packet from channel x

d) $t_9 < t < t_{10}$

Fig.18 The deadlock detection and resolution process in Example 2

V. Proof of the distributed algorithm

To prove the correctness of the distributed algorithm, we first model the S/F network as a directed graph described as follows.

Directed Graph (DG) Model: A transmission channel i, in state T or in state DR, with the head packet to be routed for another transmission channel j can be represented by vertex **i**, denoted as V_i, with an outgoing edge directed towards V_j. For channel i in state N, it is represented as V_i with no outgoing edge. To illustrate, Fig.19 shows three cascaded channels i, j and k in states DR, T and T respectively.

Based on the DG Model representation, we can, at any moment, use a directed graph to model the state of a S/F network. Fig.20 shows a four-node fourteen-channel subnetwork and its directed graph model. Note that each vertex can have at most one outgoing edge, but can have several incoming ones. Such a directed graph exhibits only those packet transmissions that are related to the deadlock. Those unrelated ones are neglected.

Lemma 1: Different closed loops in a directed graph are vertex disjointed.

Proof: Suppose the closed loops L_1 and L_2 have a common vertex, say V_i. Then V_i must have two outgoing edges: one belongs to loop L_1 while the other belongs to loop L_2. But it contradicts our DG Model representation of a vertex which has at most one outoging edge. By a similar argument, we can disprove the existence of two or more common vertexes in three or more closed loops.

Lemma 2: A S/F deadlock exists in a network if and only if a closed loop exists in the corresponding directed graph.

Proof: As mentioned in Chapter III, a deadlock exists when there is a cycle of buffer requests. This condition is revealed when a set of channels in state T have formed a closed loop (refer to Fig. 12). Then by using the DG Model representation, it immediately yields the corresponding closed loop in the directed graph. Q.E.D.

Based on the preceding lemmas, Fig.21 depicts the general situation for the presence of a deadlock in the directed graph. Note that V_i, V_j, V_k, ... , V_l, V_m, ... , V_n form a closed loop L and that V_w, V_x, V_y, ... , V_z form a path P.

Theorem 1: Every vertex in the closed loop L can detect the deadlock and enter into state DR while vertexes not in loop L will not.

Proof:

(i) Vertexes in the closed loop L. When the closed loop L is formed, all its vertexes are in state T. Let us assume that all their states remain unchanged and consider the detection of a deadlock by V_i. First V_i forwards its test packets to V_j and receives test packets from V_n and V_z. After this interchange of packets, the test packets sent by V_i will have their headers' I field set to [..., n, z, ...] U [i]. As the algorithm is distributed, every vertex in loop L does the same and eventually,

V_i will find its own identity in I of a test packet received from V_n. By the procedure for state T, V_i will change to state DR indicating the detection of a deadlock.

Next, let V_1 be the first vertex to detect the deadlock in the path Q from V_1 to V_i and consider V_i for its deadlock detection. According to the procedure for state DR, V_1 will send DR packets to V_m. Upon receiving the first DR packet, V_m will (by the procedure of state T) change to state DR and start transmitting a DR packet. This transmission of DR packet and change of state then propagate along the path Q and finally, V_i will receive a DR packet from V_n to indicate the detection of a deadlock.

(ii) Vertexes not in the closed loop L. Consider the vertex in path P which is not in loop L but has a directed edge, originated at V_w, towards loop L. All these vertexes must be in state T when path P is first formed. By Lemma 1, all the loops formed must be vertex disjointed. Therefore, path P cannot be the segment of any loop. So even though all vertexes in path P forward test packets with their own identities inserted in the I field of the packet headers, they will not receive test packets with their identities in I. In addition, these vertexes will not receive DR packets because none of them can enter into state DR. Q.E.D.

Note that during the detection of a deadlock, there must be at least one vertex which enters into state DR by the successful search of their node identities in I of the receiving test packets. These vertexes will forward DR packets to inform the next vertexes for the existence of a deadlock.

Theroem 2: The transmission of DR packets is deadlock-free.

Proof: When the vertexes of a closed loop detect a deadlock, their associated reserved buffers are always ready for receiving DR packets. Since in state DR, only DR packets can propagate in loop L, the transmission of DR packets is deadlock-free. Q.E.D.

Theorem 3: By the time a deadlock corresponding to loop L is resolved, all test packets generated by the vertexes of a loop L are discarded.

Proof: Consider V_i in state T which forwards all its test packets to V_j. Since V_j must be either in state T or in state DR, all test packets received are discarded. Moreover, V_j in state DR will make a possible change of state to N only when a DR packet is received from V_i. But V_i is in state DR and will not generate test packets. Hence, when V_j returns to state N, all test packets forwarded from V_i to V_j for the search of this deadlock are discarded. Q.E.D.

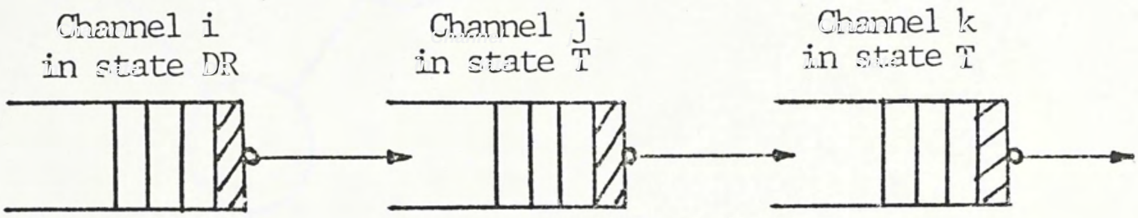
We can conclude from Theorem 1, 2 and 3 that:

a) Only those output channels involved in a deadlock can detect the deadlock.

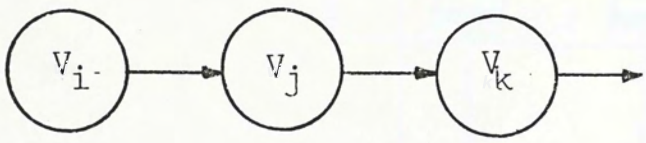
b) This deadlock can be resolved.

c) By the time when the deadlock is resolved, all test packets generated for the search of this deadlock are discarded; and so will not interfere with the detection of other potential deadlocks.

With these properties, the algorithm is proven.

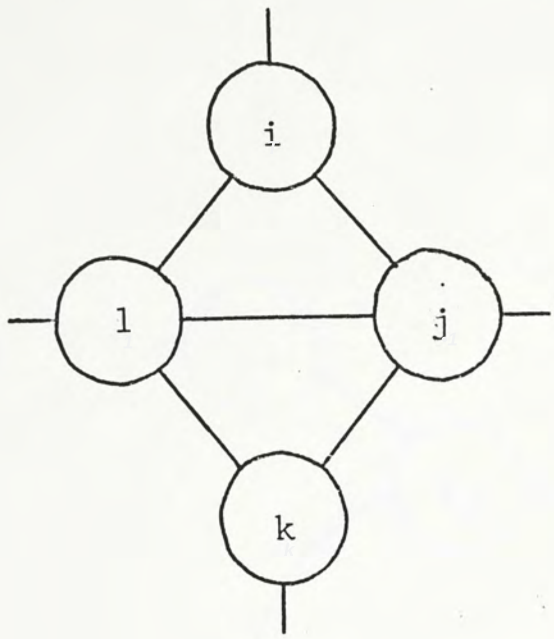


a) three cascaded channels

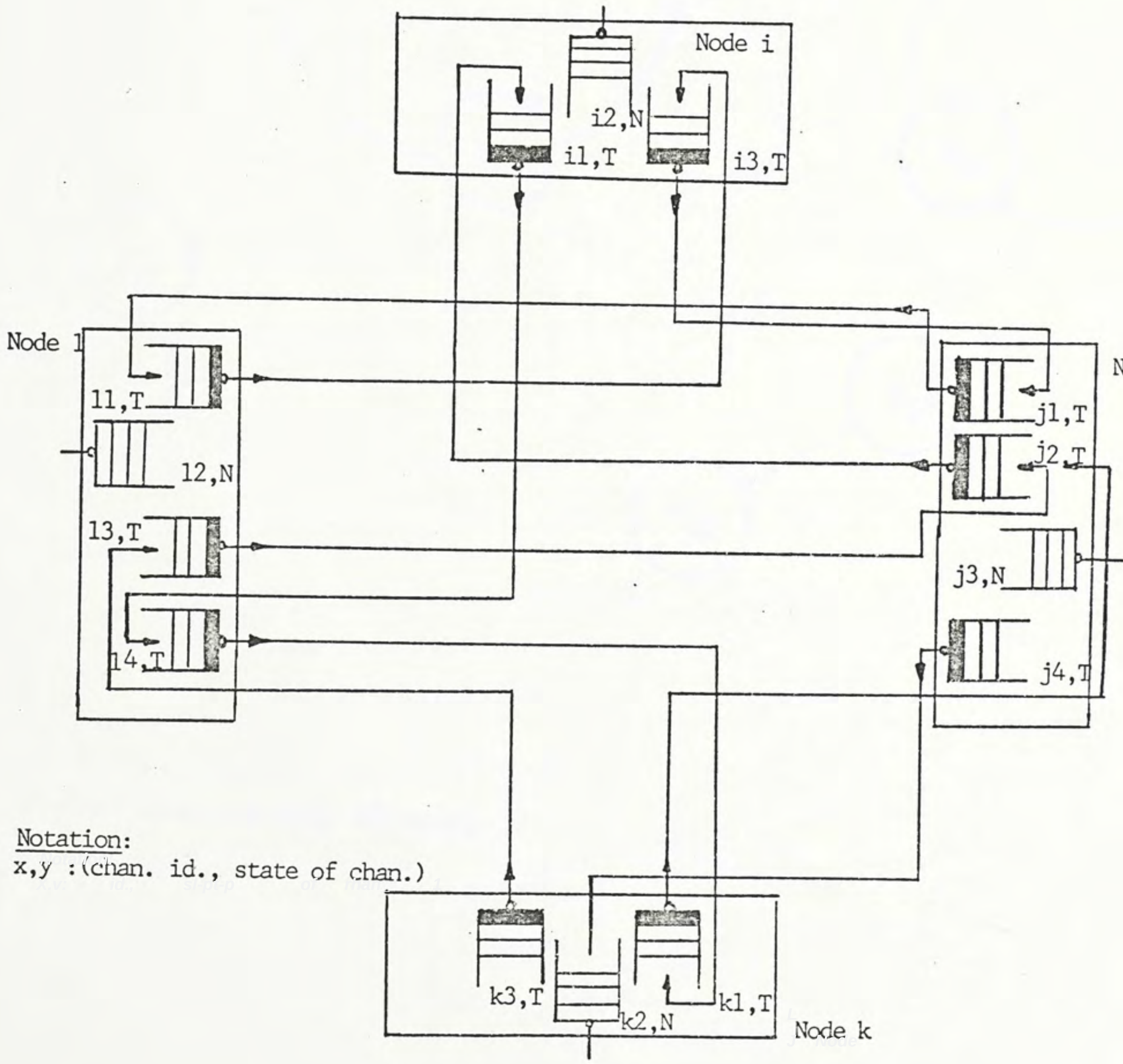


b) DG representation of a)

Fig.19 A DG model representation

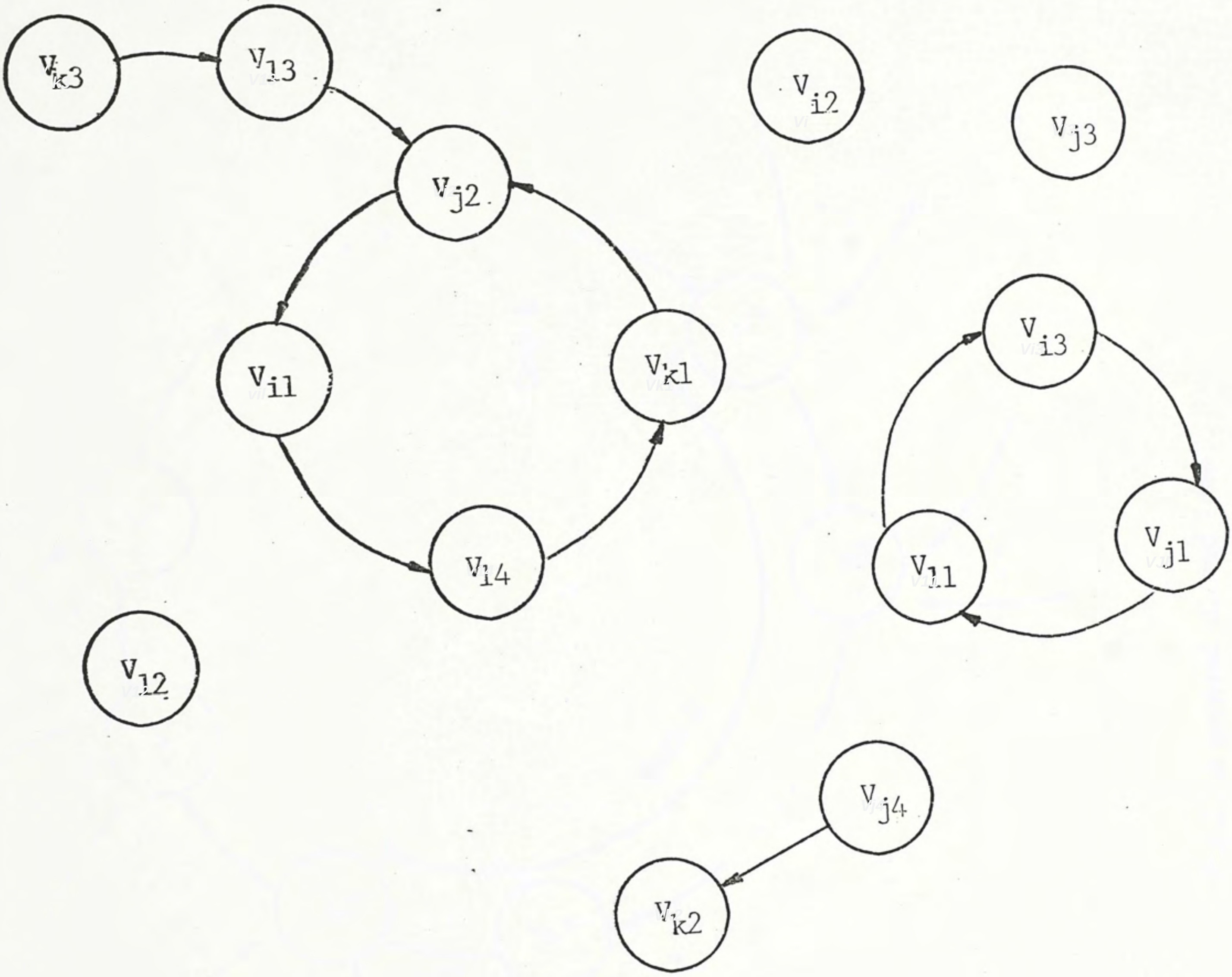


a) a section of a typical S/F network



b) the output channels of the subnetwork in a)

Fig.20
Cont./



c) the corresponding DG model

Fig. 20 The DG modelling of a S/F subnetwork

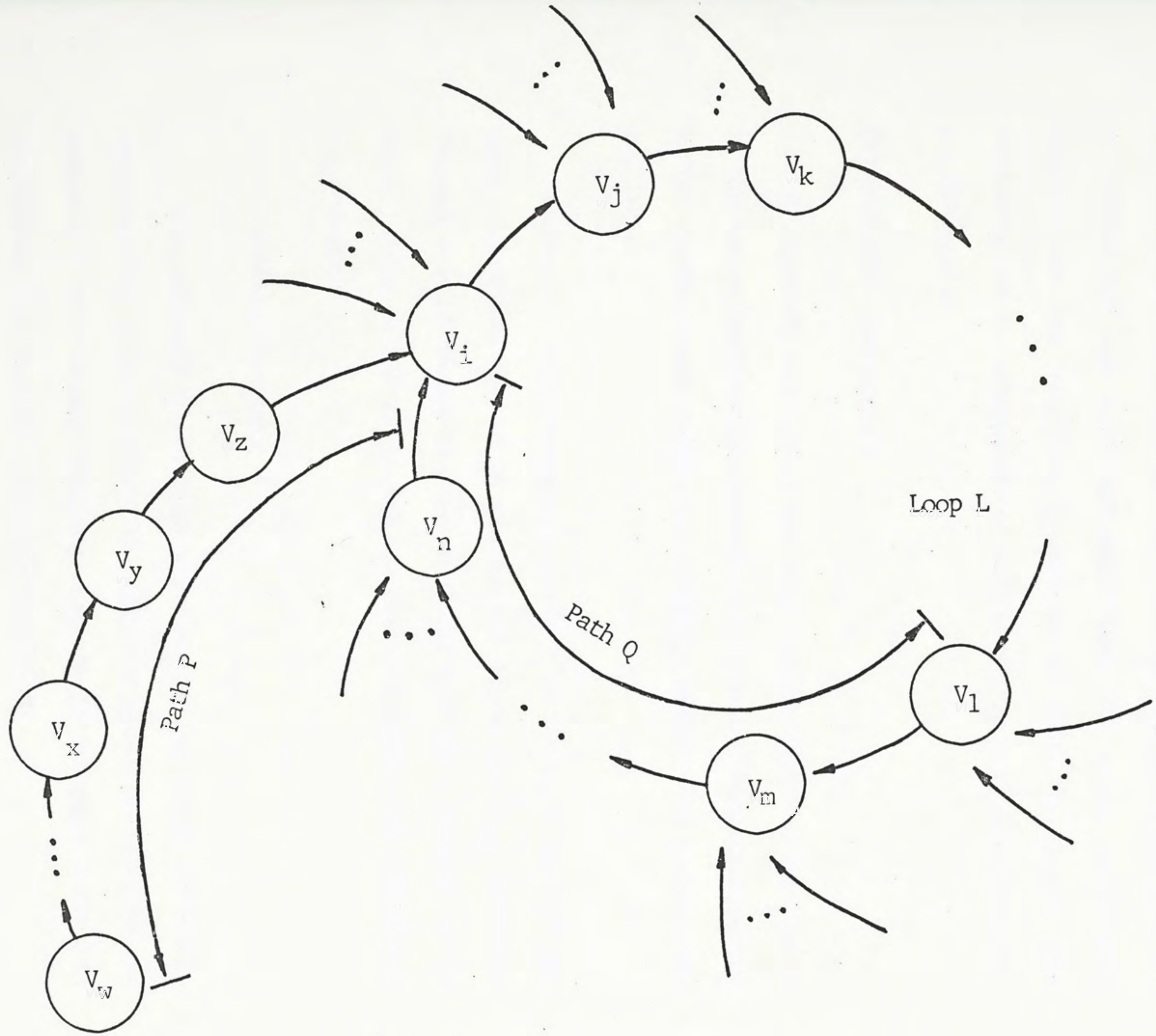


Fig.21 A general closed loop

VI. The deadlock control algorithm with SMXQ strategy

We now extend the algorithm so that the output buffers allocation strategy is SMXQ. Let B_i be the total number of packets on channel i and b be the maximum queue size allowed for each channel. Besides states N, T and DR, a new wait state, W, is needed for the modified algorithm. The state transition diagram for the modified algorithm is shown in Fig.22. The following are the procedures of each state for a typical channel, say channel i.

(i) Procedure for state N

At state N, all normal and test packets received are placed in the output buffers of channel i. If B_i=b or there is no empty buffer in the common buffer pool (CBP), the received packets are discarded.

Channel i will change to state T if (1) B_i=b and (2) the head packet has waited in the output queue for T_{out} seconds. Channel i will change to state W if (1) B_i<b (2) a request for an empty buffer in CBP fails and (3) the head packet has waited for T_{out} seconds.

(ii) Procedure for state W

When channel i receives a packet and an empty buffer is successfully allocated from the CBP, the packet is accepted and channel i will change back to state N. If no empty buffer is available, the received packet is discarded.

All outgoing packets in this state are attached with normal headers.

Once channel i is in state W, it checks whether all other outgoing channels in the same node are in states T or W. If yes, channel i will change its state to T and trigger all other channels in state W to change to state T.

(iii) Procedures for state T and state DR

A channel in state T or DR is not allowed to request buffers from the CBP for incoming packets. Besides that, the procedures are the same as that for the CP strategy in Chapter III.

Comments:

(1) To account for the additional restriction in the procedures for states T and DR, consider the deadlock loop in Fig.12. Let us say there is no such restriction as described in (iii). Let channel j, in state T or DR, obtains an empty buffer from the CBP. Then channel j may have a chance of accepting a test packet from channel i. Doing so will allow channel i to go back to state N and thus a deadlock cannot be detected.

(2) Since all outgoing packets of channel i are still attached with normal headers, a channel i in state W is represented, similar to state N, by V_i with no outgoing edge. Thus the directed graph created is the same as that for the CP case. The correctness proof of the algorithm therefore is also the same.

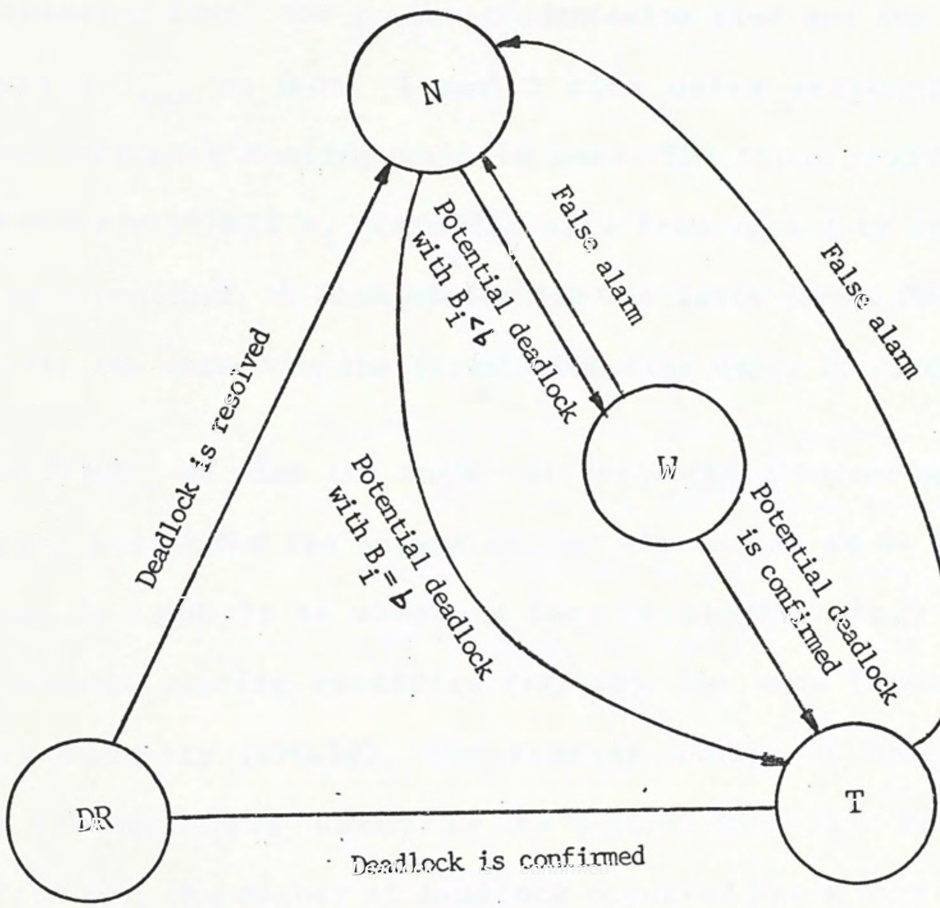


Fig.22 State transition diagram with SMXQ strategy

VII. Simulation results

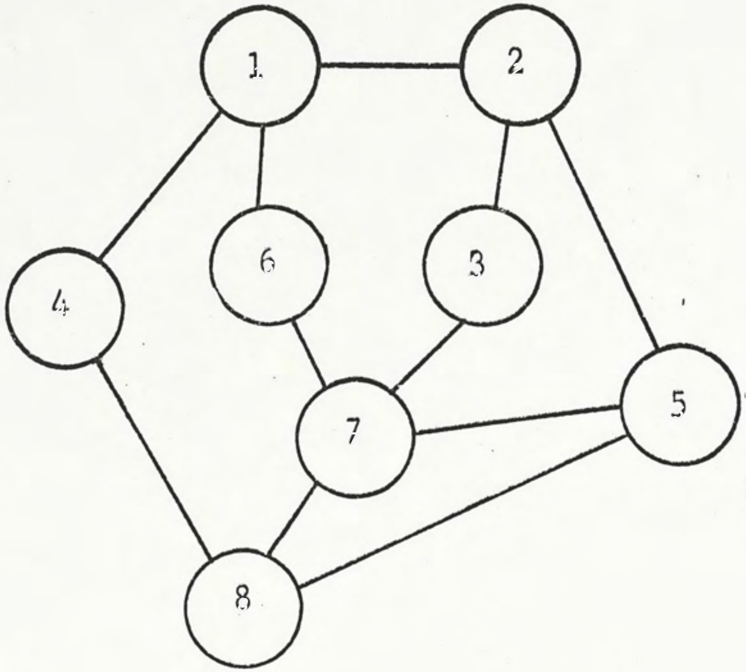
Fig.23 shows an eight-node network connected by eleven homogeneous full-duplex links. Each link is modelled as a FCFS M/D/1 queue with one reserved buffer permanently allocated and other buffers allocated according to the CP or SMXQ strategy. Let the processing time, the packet transmission time and the time-out period T_{out} be 0.01, 1 and 3 time units respectively. The shortest path routing rule is used. The input traffic is homogeneous with all r_{ij} (traffic rate from node i to node j) equal to a constant, r. Each simulation run lasts for 11,000 time units with the data from the first 1,000 time units discarded.

In Fig.24, we plot the number of deadlocks occurred against the input load r for the output buffer size equal to 4. The CP strategy is used. It is observed that deadlocks rarely occur under normal traffic condition ($r<0.45$). But once beyond the network capacity ($r>0.58$), the average number of deadlocks detected increases abruptly to about 200. In between ($0.45<r<0.58$), the number of deadlock occurred has a very large variance. Fig.25 shows the case with buffer size equal to 5. We observe that the curve is similar except that the high variance region is shifted to $0.59<r<0.75$.

Fig.26 shows the network throughput under normal traffic condition ($r<0.45$). The output buffer for each channel is 4. Here we show 3 curves: Curve C shows the network throughput with no deadlock control algorithm implemented while Curves A and B represent the throughput with the deadlock control algorithms implemented with the SMXQ and CP strategies respectively. It is readily seen that very high network throughput can be maintained when the network is not saturated. But for Curve C, the network breaks down. In addition, the SMXQ starts to give slightly higher throughput than the CP at $r>0.4$. When one-third of the channels are offered with twice the amount of input (i.e. under asymmetric traffic condition), similar result is found as shown in Fig.27.

Fig.28 shows the time for the first occurrence of a deadlock (starting from an empty system) versus the input load using the CP strategy. Under heavy loading, we see that the first deadlock occurrence time is nearly a constant, independent of the number of buffers available at each channel. On the other hand, under moderate traffic condition, increasing the buffer size can indeed delay the occurrence of deadlock. When the traffic is very light, deadlock is still very unlikely to occur even with a very small buffer size.

Fig.29 and 30 show the case for another eight-node network. Exactly the same phenomena are oberved.



Number of nodes: 8

Number of links: 11

Number of possible deadlock loops involving
- i) 4 nodes: 1
- ii) 5 nodes: 5
- iii) 6 nodes: 3
- iv) 7 nodes: 1

Packet routing stratgy: Shortest path

Fig.23 Network I

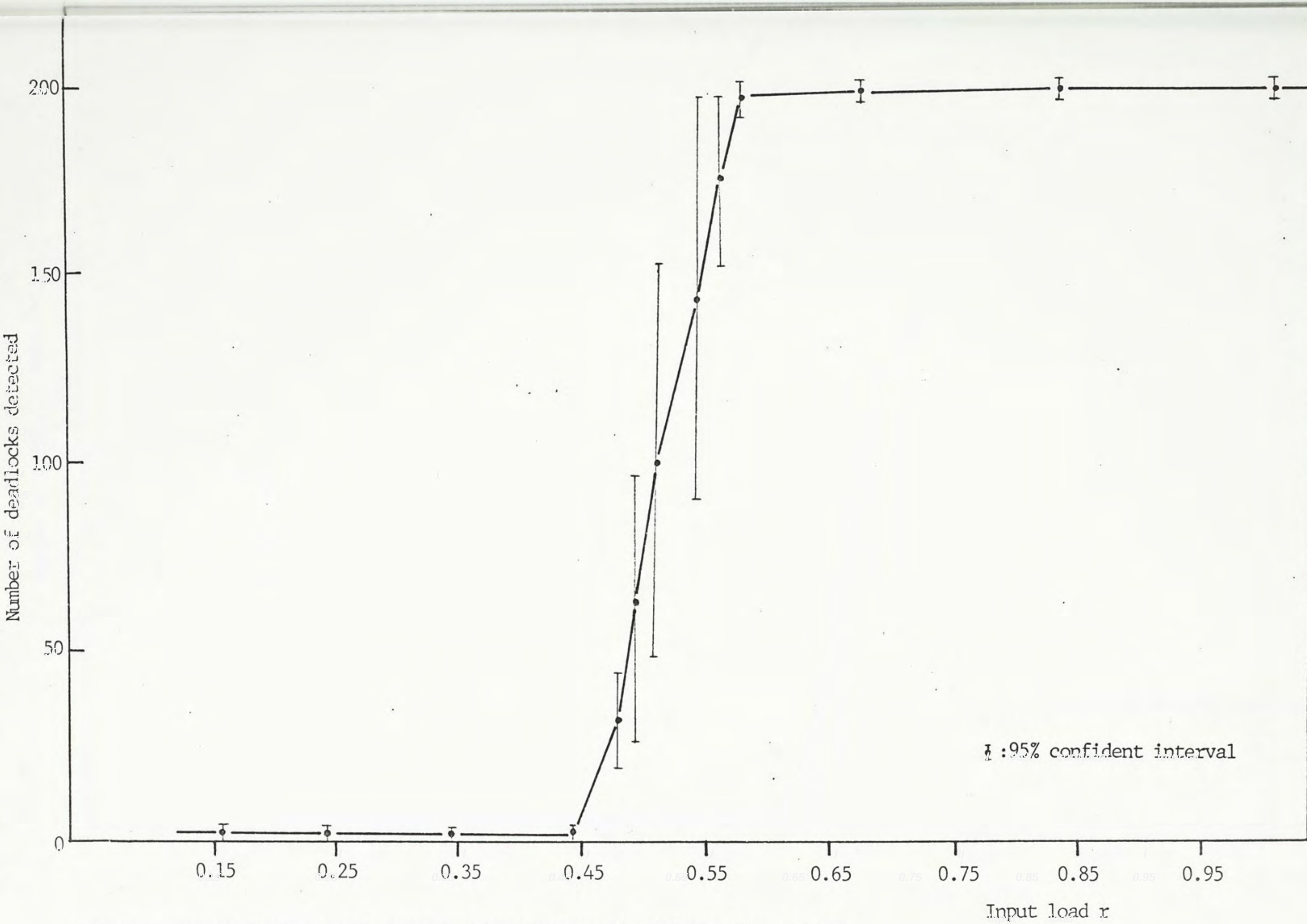


Fig.24 Number of deadlocks detected versus the input load for buffer size=4

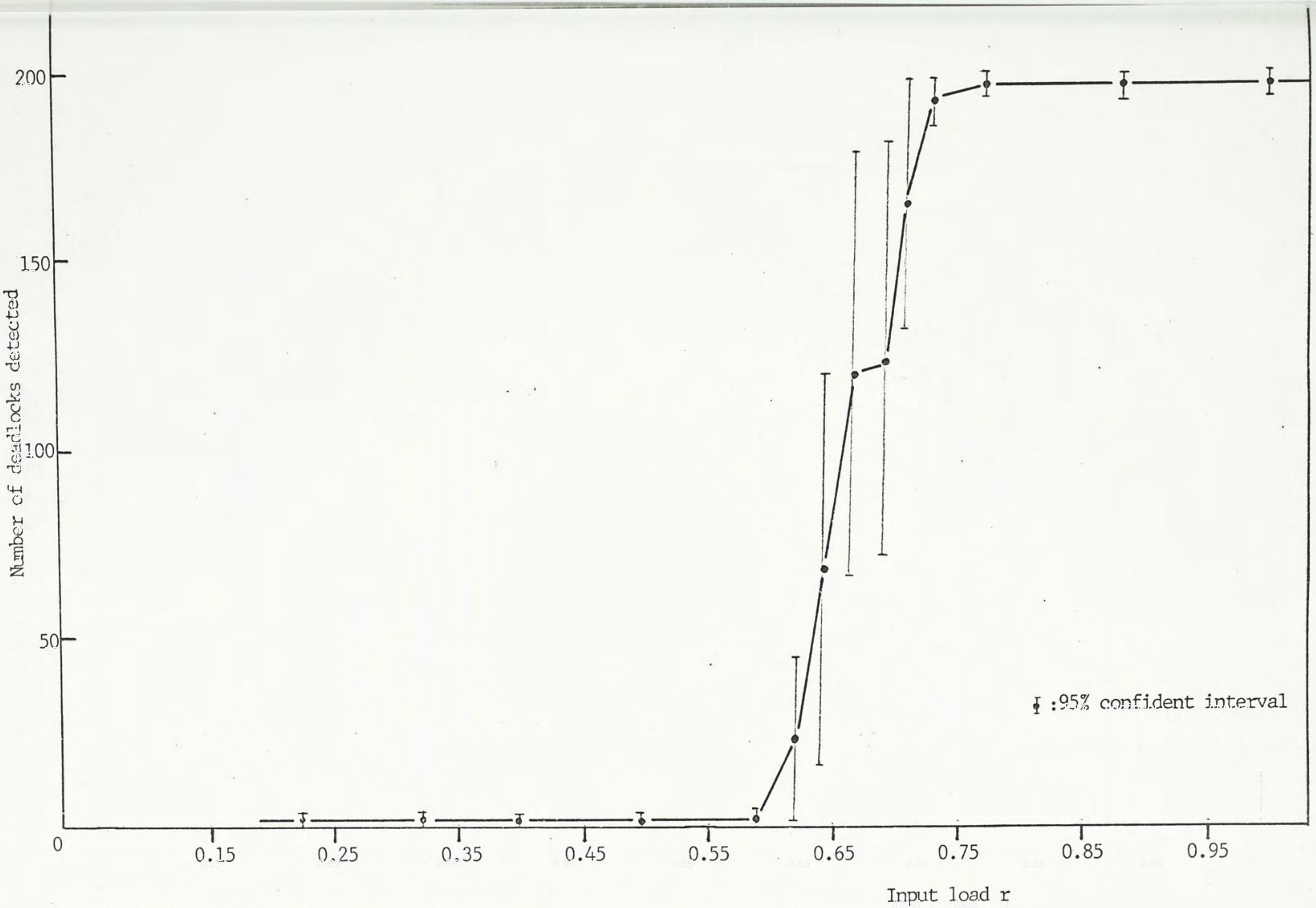


Fig.25 Number of deadlocks detected versus the input load for buffer size=5

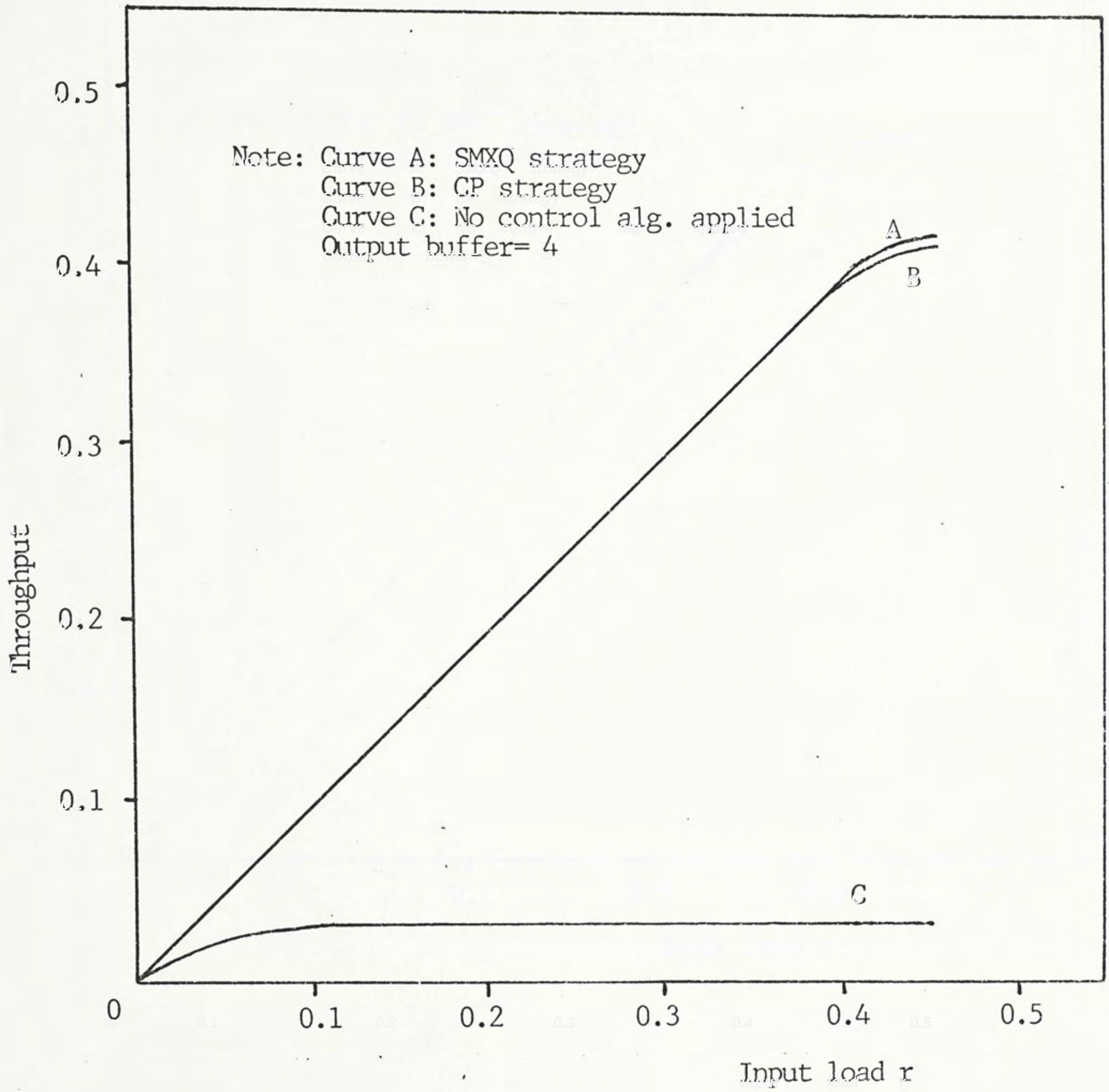


Fig.26 Network throughput versus the input load
(symmetric load condition)

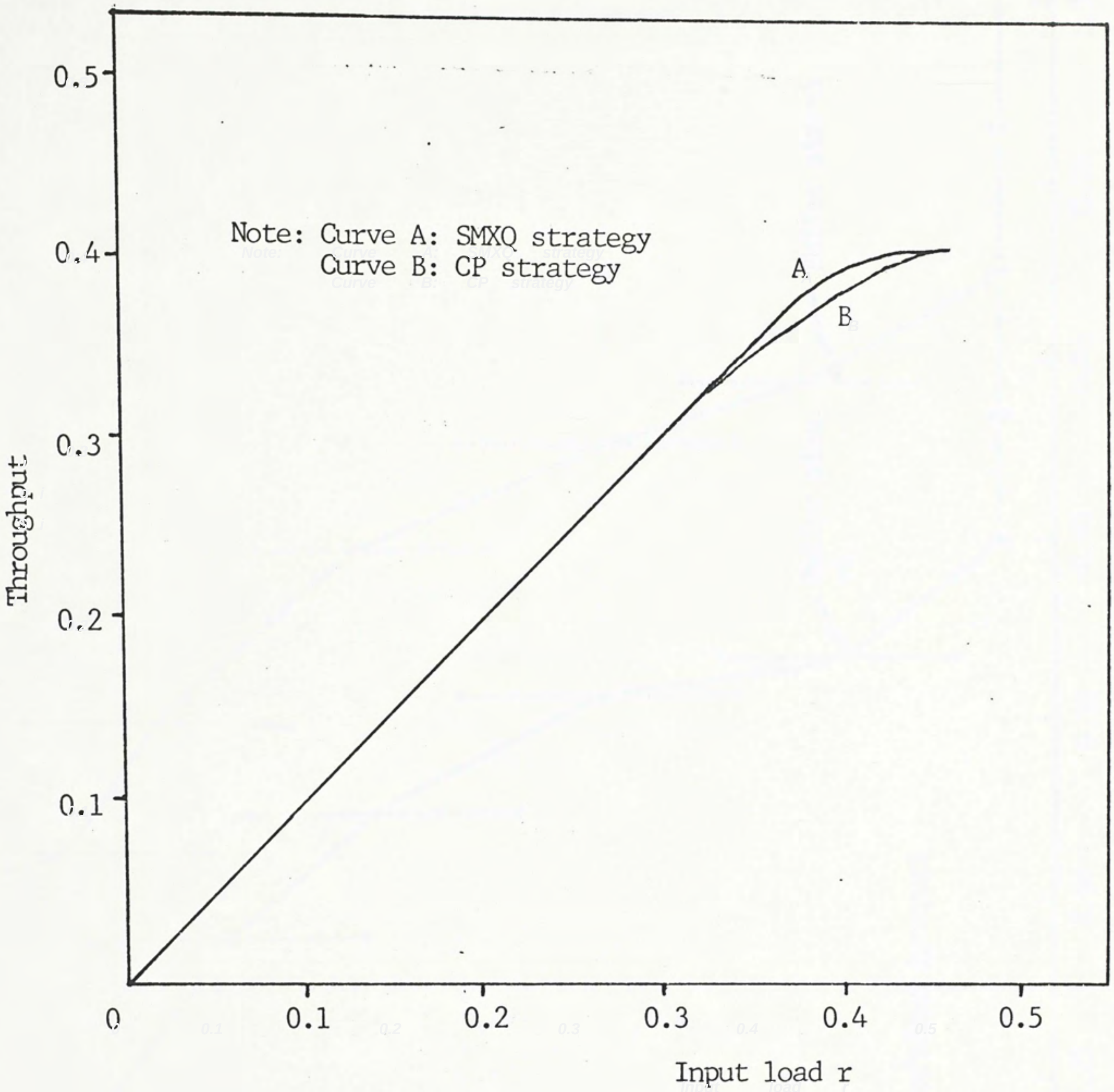


Fig. 27 Network throughput versus the input load r
(asymmetric load condition)

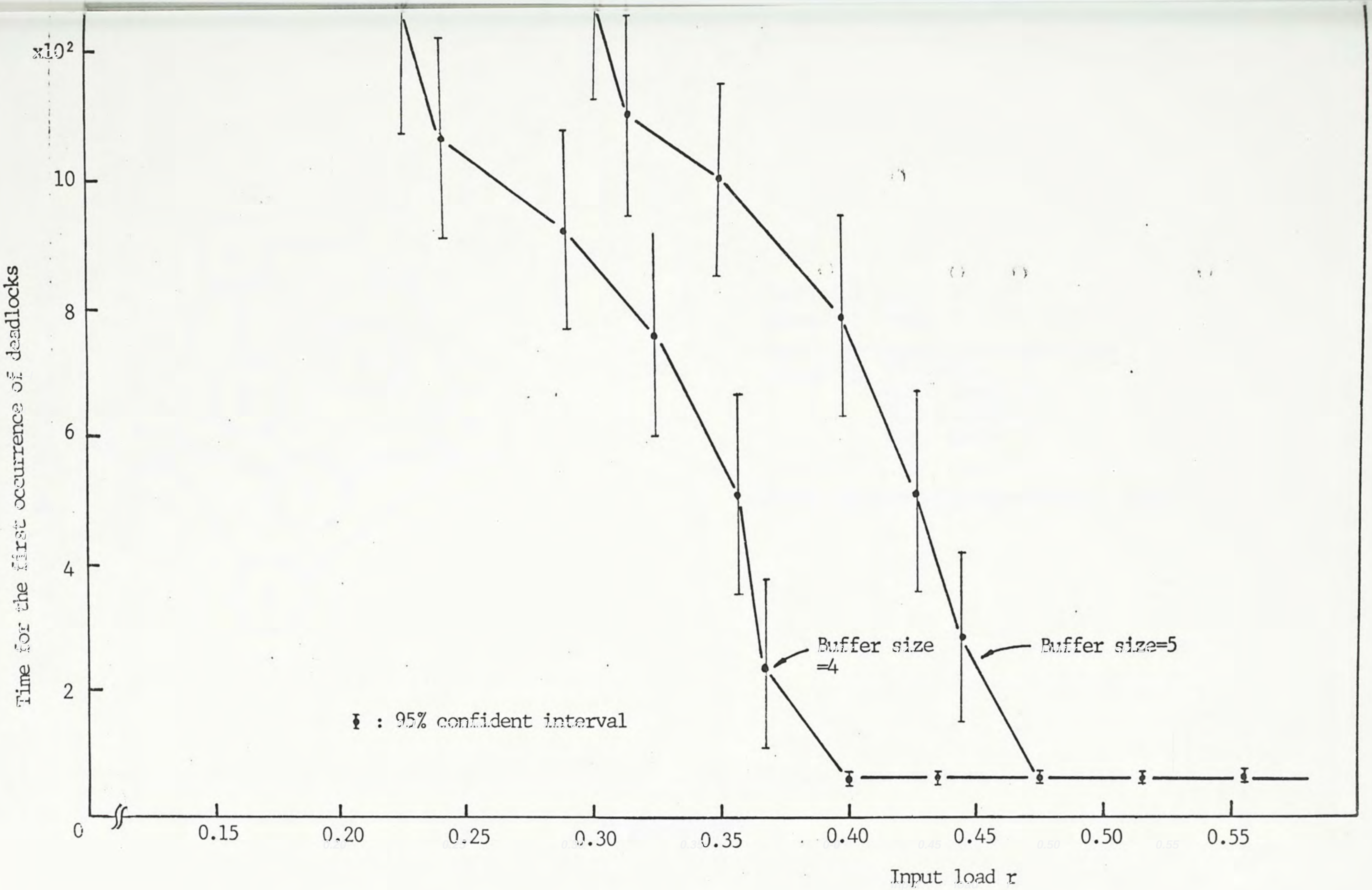
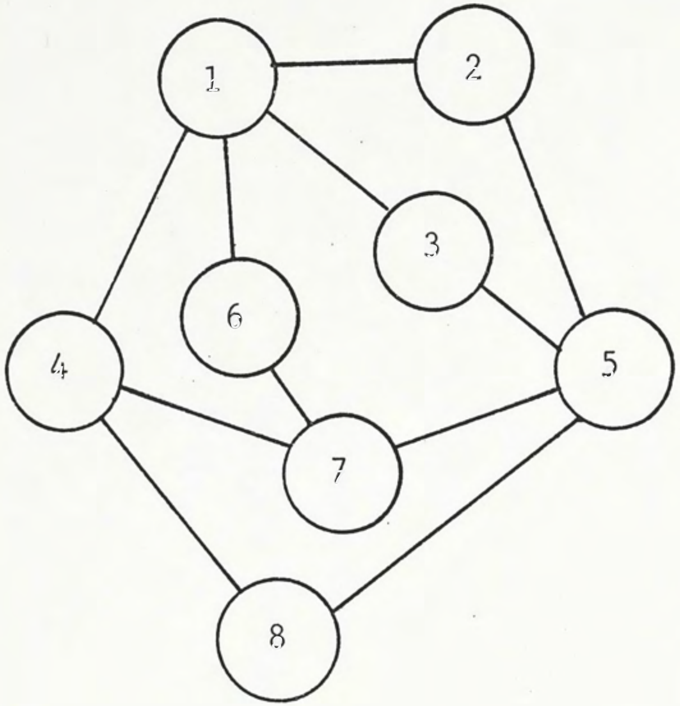


Fig.28 Time for the first occurrence of deadlocks versus the input load r



Number of nodes: 8
Number of links: 11

Number of possible deadlock loops
involving i) 4 nodes: 3
ii) 5 nodes: 6
iii) 6 nodes:1
iv) 7 nodes: 2

Packet routing strategy: shortest path

Fig.29 Network II

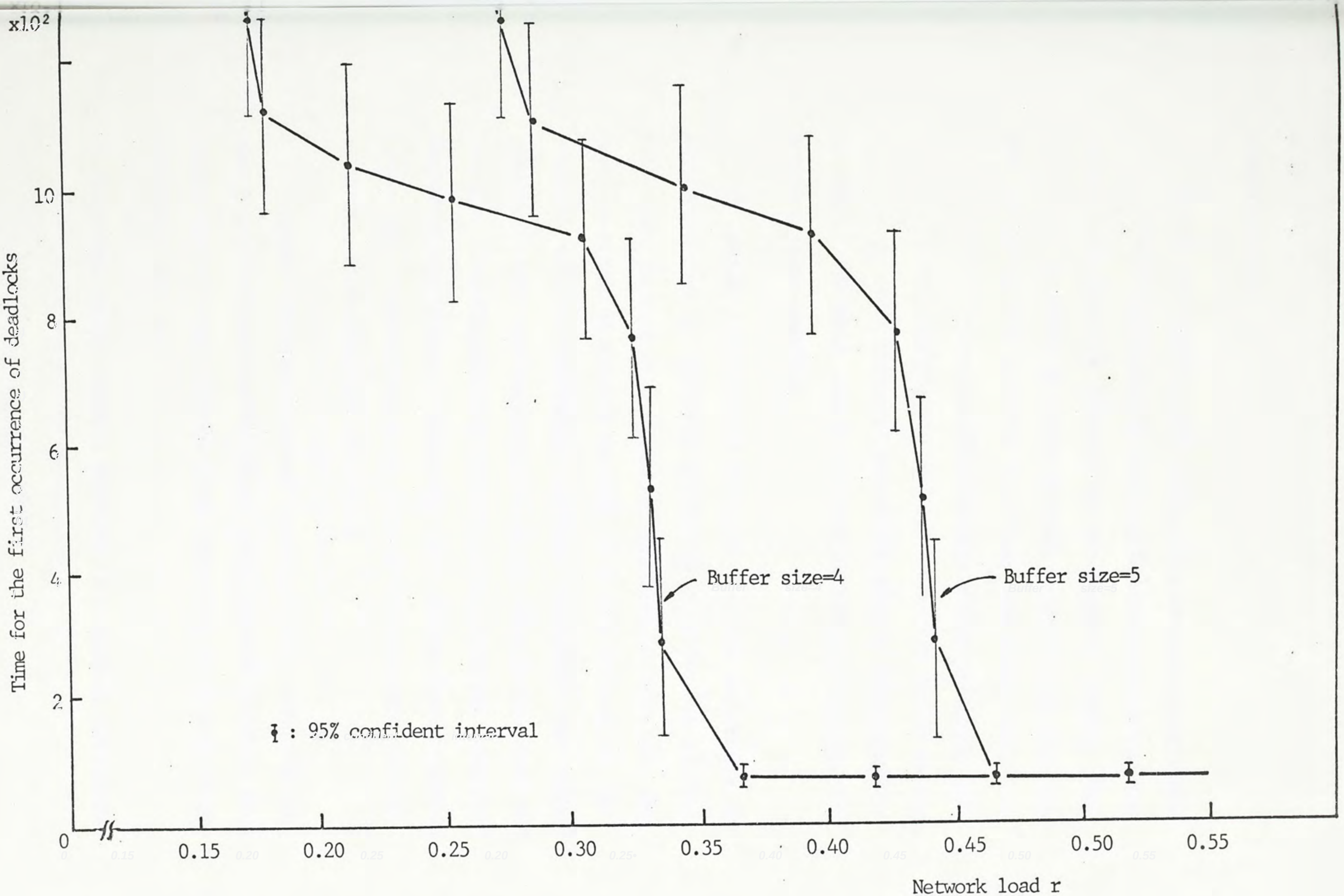


Fig.30 Time for the first occurrence of deadlocks versus the input load

VIII. Conclusion

Direct and indirect S/F deadlocks are some catastrophic problems that every network designer has to tackle with. They occur even when the network is not heavily loaded. Some prevention techniques have been proposed to solve these problems. But as far as the network resources and performance are concerned, they are unjustified because 1) even when deadlocks does not occur, any prevention algorithm will either impose restrictions on the dynamical use of resources or deteriorate the network performance and 2) S/F deadlocks rarely happen under normal traffic condition.

We propose a distributed deadlock detection and resolution algorithm that is entirely invisible under normal traffic condition. As soon as a potential deadlock is detected, the deadlock detection phase is invoked. If it is a false alarm, the algorithm will be inactivated; otherwise, the deadlock resolution phase will start to resolve the deadlock. Once the deadlock is resolved, the algorithm is removed. This algorithm not only satisfies the Gambosi criteia but also shares the Gelernter advantages. In addition, it can be used in conjunction with CP and SMXQ buffer allocation strategies.

To prove the correctness of the algorithm, a direct graph modelling is first used to depict a S/F network in which the algorithm is being invoked. Three properties are then observed and proved. They are:

1) Only those output channels involved in a deadlock can detect

the deadlock.

2) This deadlock can be resolved.

3) By the time when the deadlock is resolved, all test packets generated for the search for this deadlock are discarded; and so will not interfere with the detection of other potential deadlocks.

These properties directly lead to the deadlock and resolution properties of the algorithm.

Simulation results show that the network can maintain a relatively high throughput even when deadlocks are being detected and resolved. Furthermore, several properties of deadlocks are found: 1) deadlocks start to increase abruptly once the network operates beyond its capacity; and 2) under heavy load condition, increasing the buffer size will not delay the occurrence of deadlocks.

However, several areas are valuable for future investigations:

1) To extend the algorithm applicable to Sharing with Minimum Allocation (SMA) and the combined SMA-SMXQ strategies.

2) To investigate the deadlock phenomena analytically.

3) To compare the network performance among the algorithm proposed in this thesis as well as the techniques proposed by other authors.

IX. References

[1] A.S. Tanenbaum, Computer Network, Prentice-Hall, Inc., 1981, pp.137-150.

[2] L. Kleinrock, Queueing Systems, Vol-II:Computer Applications, Wiley-Interscience, 1976.

[3] M.Gerla and L.Kleinrock, "Flow control: a comparative survey", IEEE Trans. Comm., Vol. COM-28, Apr. 1980.

[4] M.I. Irland, "Buffer management in a packet-switch", IEEE Trans. Comm., Vol. COM-26, pp.328-337, Mar. 1978.

[5] T.P.Yum and C.Dou, "Buffer allocation strategies with blocking requirements", North-Holland Performance Evvaluation 4, pp.285-295, 1984.

[6] F.Kamoun and L.Kleinrock, "Analysis of shared finite storage in a computer network node environment under general traffic conditions", IEEE Trans. Comm., COM-28, pp.992-1003.

[7] K.D. Gunther, "Prevention of deadlocks in packet-switched data transport systems", IEEE Trans. Comm., Vol. COM-29, pp.512-524, Apr. 1981.

[8] P.M. Merlin and P.J.Schweitzer, "Deadlock avoidance in S/F network I:S/F deadlocks", IEEE Trans. Comm., Vol. COM-28, pp.345-354, Mar. 1980.

[9] P.M.Merlin and P.J.Schweitzer, "Deadlock avoidance in store-

and-forward networks-II: other deadlock types", IEEE Trans. Comm., Vol. COM-28, pp.355-360, Mar. 1980.

[10] W. Wimmer, "Using barrier graphs for deadlock prevention in communication networks", IEEE Trans. Comm., Vol. COM-32, pp.897-901, Aug. 1984.

[11] I.S. Gopal, "Prevention of store-and-forward deadlock in computer networks", IEEE Trans. Comm, Vol. COM-33, pp.1258-1264, Dec. 1985.

[12] D. Gelernter, "A DAG-based algorithm for prevention of S/F deadlock in packet networks", IEEE Trans. Comm., Vol. COM-30, pp.709-715, Oct. 1981.

[13] J. Blazewicz, et al, "Deadlock-resistant flow control procedures for S/F networks", IEEE Tran. Comm., Vol COM-32, pp.884-887, Aug. 1984.

[14] G. Gambosi, et al,"A detection and removal of deadlocks in S/F communication networks", <u>Performance of Computer-communication Systems</u> edited by W.Bux and H.Rudin, IBM Zurich Research Laboratory, pp.219-229, 1984.

Appendix

Simulation program listing and a typical output.

```
*READING NETWORK DEFINITION
$$$ GOOD COMPILATION
*NETWORK READY
*START SIMULATION
*PAUSE IN SIMULATION AT 1000
*CONTINUING SIMULATION
*START SIMULATION
*SIMULATION REACHED       1000
*SIMULATION REACHED       2000
*SIMULATION REACHED       3000
*SIMULATION REACHED       4000
*SIMULATION REACHED       5000
*SIMULATION REACHED       6000
*SIMULATION REACHED       7000
*SIMULATION REACHED       8000
*SIMULATION REACHED       9000
*SIMULATION REACHED      10000

END PES
```

```
START PES
DECLARE
REAL P1, P2, P3, P4, DEAD_TIME
INTEGER X, NUM_DEAD
INTEGER(NODE) SI(100), SX(100), Y, Z
CHAR(NODE) STATUS
INTEGER(LOCAL) ID(100), X1
NODES SOURCE, MQUEUE(22), MREC(22), MTRAN(22), SETUP1, SETUP2,
      CHECK, MBRANCH, MSINK1, MSINK2, MDELAY, SETUP,
      MBRANCH1, NEXTCHAN
MSG-TYPE PKT(FIXED LENGTH, 1 UNIT)

TOPOLOGY
 SOURCE          SETUP          <PKT,ALL>   1.0;
 SETUP           SETUP1         <PKT,ALL>   1.0;
 SETUP1          SETUP2         <PKT,ALL>   1.0;
 SETUP2          CHECK          <PKT,ALL>   1.0;
 CHECK           MSINK1         <PKT,ALL>    P1;
 CHECK           MBRANCH        <PKT,ALL>    P2;
 MBRANCH         MREC(X)        <PKT,ALL>   1.0;
 MREC(X)         MSINK2         <PKT,ALL>    P4;
 MREC(X)         MQUEUE(X)      <PKT,ALL>    P3;
 MQUEUE(X)       NEXTCHAN       <PKT,ALL>   1.0;
 NEXTCHAN        MDELAY         <PKT,ALL>   1.0;
 MDELAY          MTRAN(X)       <PKT,ALL>   1.0;
 MTRAN(X)        MBRANCH1       <PKT,ALL>   1.0;
 MBRANCH1        CHECK          <PKT,ALL>   1.0;

END PES
```

```
START PES
ROUTING METHOD  SHORTEST;
 ROUTING TABLE
  NODE1            NODE2          1;
  NODE1            NODE3          1, 5;
  NODE1            NODE4          4;
  NODE1            NODE5          1, 6;
  NODE1            NODE6          3;
  NODE1            NODE7          3, 18;
  NODE1            NODE8          4, 8;
  NODE2            NODE1          2;
  NODE2            NODE3          5;
  NODE2            NODE4          2, 4;
  NODE2            NODE5          6;
  NODE2            NODE6          2, 3;
  NODE2            NODE7          2, 4;
  NODE2            NODE8          6, 12;
  NODE3            NODE1          9, 2;
  NODE3            NODE2          9;
  NODE3            NODE4          10, 16, 20;
  NODE3            NODE5          10, 15;
  NODE3            NODE6          10, 17;
  NODE3            NODE7          10;
  NODE3            NODE8          10, 16;
  NODE4            NODE1          7;
  NODE4            NODE2          7, 1;
  NODE4            NODE3          7, 1, 5;
  NODE4            NODE5          8, 22;
  NODE4            NODE6          7, 3;
  NODE4            NODE7          8, 21;
  NODE4            NODE8          9;
  NODE5            NODE1          11, 2;
  NODE5            NODE2          11;
  NODE5            NODE3          13, 14;
  NODE5            NODE4          12, 20;
  NODE5            NODE6          13, 17;
  NODE5            NODE7          13;
  NODE5            NODE8          12;
  NODE6            NODE1          19;
  NODE6            NODE2          19, 1;
  NODE6            NODE3          18, 14;
  NODE6            NODE4          19, 4;
  NODE6            NODE5          18, 15;
  NODE6            NODE7          18;
  NODE6            NODE8          18, 16;
  NODE7            NODE1          17, 19;
  NODE7            NODE2          14, 9;
  NODE7            NODE3          14;
  NODE7            NODE4          16, 20;
  NODE7            NODE5          15;
  NODE7            NODE6          17;
  NODE7            NODE8          16;
```

```
000052    NODE8        NODE1      20, 7;
000053    NODE8        NODE2      22, 11;
000054    NODE8        NODE3      21, 14;
000055    NODE8        NODE4      20;
000056    NODE8        NODE5      22;
000057    NODE8        NODE6      21, 17;
000058    NODE8        NODE7      21;

END PES
```

```
START PES
     DEFINE
     SOURCE  INPUT RATE EXP(0.10);

     SETUP
     TYPE COMPUTE
     (PKT, ALL)  REQUEST LC[1]='N';

     SETUP1
     TYPE SOURCE_ADDR
     (PKT, ALL) EQUAL PROB INT[1<=LI[1]<=22];

     SETUP2
     TYPE DEST_ADDR
     (PKT, ALL) EQUAL PROB INT[1<=LI[2]<=22] AND
                           [LI[1]<>LI[2]],
                LI[3]=LI[1],
                LI[4]=9999;

     CHECK
     TYPE BRANCH
     (PKT,ALL) REQUEST GOTOIF 1 LI[3] LI[2],
                       P1=0.0,
                       P2=1.0,
                       EXIT,
                      LABEL 1
                       P1=1.0
                       P2=0.0;

     MSINK1
     TYPE SINK;

     MBRANCH
     TYPE BRANCH;

     MREC
     TYPE ARRAY(22) COMPUTE
     (PKT,ALL) REQUEST NGOTOIF 1 STATUS 'N',
                       GOTOIF 2 BUF_LENGTH 0,
                       P3=0.0,
                       P4=1.0,
                       EXIT,
                      LABEL 2
                       P3=0.0
                       P4=1.0
                       EXIT,
                      LABEL 1
                       NGOTOIF 6 STATUS 'T',
                       GOTOIF 3 LC[1] 'N',
                       GOTOIF 4 LC[1] 'D',
                       SEARCH ID LI[3] LB[1],
                       GOTOIF 5 LB[1] T,
```

```
                  EQ ARRAY SI ID,    *USER DEF. SUBRTN.
                  IN SX, XI,
                 LABEL 3
                  P3=0.0,
                  P4=1.0,
                  EXIT,
                 LABEL 4
                  Z=Z-1,
                 LABEL 5
                  STATUS='D',
                  Y=XI,
                  Z=XI,
                  EXIT,
                 LABEL 6
                  NGOTOIF 3 LC[1] 'D',
                  Z=Z-1,
                  NGOTOIF 7 Y 0,
                  NGOTOIF 7 Z 0,
                  STATUS='N',
                  NUM_DEAD=NUM_DEAD+1,
                  NGOTOIF 7 NUM_DEAD 1,
                  DEAD_TIME=CURR_TIME,.
                 LABEL 7
                  EXIT;

MQUEUE
TYPE ARRAY(22) QUEUE
QD FCFS CONSTANT(0.01)
BF BUF_LENGTH 4, EMER_LENGTH 1;

MDELAY
TYPE SERVICE CONSTANT(1.0);

NEXTCHAN
TYPE ROUTING
RD SHORTEST PATH LI[3];

MTRAN
TYPE ARRAY(22) COMPUTE
<PKT,ALL> NGOTOIF STATUS 'N',
          JP SUBROUTINE CHECK_SUCCESS(S:BOOLEAN,
             TOUT:INT),
          GOTOIF SMALLER 2 TOUT 3,
          NGOTOIF BUF_LENGTH 0,
          STATUS='T',
         LABEL 2
          EXIT,
         LABEL 1
          NGOTOIF 3 STATUS 'T',
          JP SUBROUTINE CHECK_SUCCESS(S:BOOLEAN,
             TOUT:INT),
          GOTOIF 4 S T,
```

```
            EQ ARRAY ID SI,
            JP SUBROUTINE FIND_KIJ(KIJ:INT), *USER DEF.SUB.
            MIN X1; KIJ, SX,
            CLEAR SI, SX,
            EXIT,
          LABEL 3
            Y=Y-1,
            NGOTOIF 5 Y 0,
            NGOTOIF 5 Z 0,
            STATUS='N',
            NUM_DEAD=NUM_DEAD+1,
            NGOTOIF 5 NUM_DEAD 1,
            DEAD_TIME=CUR_TIME,
          LABEL 4
            STATUS='N',
          LABEL 5
            EXIT;

SINK2
TYPE SINK;

NBRANCH2;
TYPE BRANCH;

STATISTICS REPORT ON SINK1;

RUN
STATUS='N'
GO 1000
CLEAR
GO 10000, 1000
PRINT DEAD_TIME, NUM_DEAD
DUMP
EXIT;

END;

END PES
```

```
NUMBER OF EVENTS:  295843.0
BATCH NUMBER    :         1
BATCH DURATION  :  10000.00
BATCH STARTED AT:   1000.00
CURRENT TIME    :  11000.00
```

REPORT ON NODE SINK1:

COUNT	RATE
98253	9.83

```
DEAD_TIME: 9857.00
NUM_DEAD :       1
```

	THROUGHPUT		INPUT		
	RATE	COUNT		RATE	COUNT
NODE:SOURCE(1)	9.86	98515	*		
NODE:SETUP1(1)	9.86	98515	*		
NODE:SETUP2(1)	9.86	98515	*		
NODE:CHECK(1)	19.85	198470	*		
NODE:MSINK1(1)	9.83	98253	*		
NODE:MBRANCH(1)	10.02	100,217	*		
NODE:MREC(1)	4.53	4532	*		
NODE:MREC(2)	4.62	4625	*		
NODE:MREC(3)	4.54	4536	*		
NODE:MREC(4)	4.56	4555	*		
NODE:MREC(5)	4.61	4613	*		
NODE:MREC(6)	4.50	4503	*		
NODE:MREC(7)	4.56	4557	*		

	THROUGHPUT			INPUT	
	RATE	COUNT		RATE	COUNT
NODE:MREC(8)	4.53	4526	*		
NODE:MREC(9)	4.60	4601	*		
NODE:MREC(10)	4.60	4598	*		
NODE:MREC(11)	4.61	4610	*		
NODE:MREC(12)	4.51	4507	*		
NODE:MREC(13)	4.62	4617	*		
NODE:MREC(14)	4.58	4577	*		
NODE:MREC(15)	4.52	4522	*		
NODE:MREC(16)	4.52	4518	*		
NODE:MREC(17)	4.55	4545	*		
NODE:MREC(18)	4.62	4622	*		
NODE:MREC(19)	4.53	4528	*		
NODE:MREC(20)	4.50	4501	*		

	THROUGHPUT RATE	THROUGHPUT COUNT		INPUT RATE	INPUT COUNT
NODE:MREC(21)	4.50	4496	*		
NODE:MREC(22)	4.53	4528	*		
NODE:MSINK2(1)	0.00	24	*		
NODE:MQUEUE(1)	4.53	4530	*	4.53	4530
NODE:MQUEUE(2)	4.62	4622	*	4.62	4622
NODE:MQUEUE(3)	4.54	4534	*	4.54	4534
NODE:MQUEUE(4)	4.56	4555	*	4.56	4555
NODE:MQUEUE(5)	4.61	4613	*	4.61	4613
NODE:MQUEUE(6)	4.50	4503	*	4.50	4503
NODE:MQUEUE(7)	4.56	4556	*	4.56	4556
NODE:MQUEUE(8)	4.53	4526	*	4.53	4526
NODE:MQUEUE(9)	4.60	4599	*	4.60	4599
NODE:MQUEUE(10)	4.60	4597	*	4.60	4597

	THROUGHPUT			INPUT	
	RATE	COUNT		RATE	COUNT
NODE:MQUEUE(11)	4.61	4610	*	4.61	4610
NODE:MQUEUE(12)	4.51	4506	*	4.51	4506
NODE:MQUEUE(13)	4.62	4616	*	4.62	4616
NODE:MQUEUE(14)	4.58	4575	*	4.58	4575
NODE:MQUEUE(15)	4.52	4521	*	4.52	4521
NODE:MQUEUE(16)	4.52	4518	*	4.52	4518
NODE:MQUEUE(17)	4.55	4545	*	4.55	4545
NODE:MQUEUE(18)	4.62	4622	*	4.62	4622
NODE:MQUEUE(19)	4.53	4527	*	4.53	4527
NODE:MQUEUE(20)	4.50	4501	*	4.50	4501
NODE:MQUEUE(21)	4.49	4494	*	4.49	4494
NODE:MQUEUE(22)	4.53	4526	*	4.53	4526
NODE:NEXTCHAN(1)	10.02	100193	*		

	THROUGHPUT RATE	THROUGHPUT COUNT		INPUT RATE	INPUT COUNT
NODE:MDELAY(1)	10.02	100183	*		
NODE:MTRAN(1)	4.53	4530	*		
NODE:MTRAN(2)	4.62	4622	*		
NODE:MTRAN(3)	4.54	4534	*		
NODE:MTRAN(4)	4.56	4555	*		
NODE:MTRAN(5)	4.61	4613	*		
NODE:MTRAN(6)	4.50	4503	*		
NODE:MTRAN(7)	4.56	4556	*		
NODE:MTRAN(8)	4.53	4526	*		
NODE:MTRAN(9)	4.60	4599	*		
NODE:MTRAN(10)	4.60	4597	*		
NODE:MTRAN(11)	4.61	4610	*		
NODE:MTRAN(12)	4.51	4506	*		

	THROUGHPUT			INPUT	
	RATE	COUNT		RATE	COUNT
NODE:MTRAN(13)	4.62	4616	*		
NODE:MTRAN(14)	4.58	4575	*		
NODE:MTRAN(15)	4.52	4521	*		
NODE:MTRAN(16)	4.52	4518	*		
NODE:MTRAN(17)	4.55	4545	*		
NODE:MTRAN(18)	4.62	4622	*		
NODE:MTRAN(19)	4.53	4527	*		
NODE:MTRAN(20)	4.50	4501	*		
NODE:MTRAN(21)	4.49	4494	*		
NODE:MTRAN(22)	4.53	4526	*		
NODE:MBRANCH1(1)	10.02	100217	*		

QUEUE AT NODE: MQUEUE(1)

	MEAN	STANDARD DEV
QUEUE LENGTH	1.84	0.125
QUEUE TIME	0.117	0.034

QUEUE AT NODE: MQUEUE(2)

	MEAN	STANDARD DEV
QUEUE LENGTH	1.76	0.117
QUEUE TIME	0.126	0.044

QUEUE AT NODE: MQUEUE(3)

	MEAN	STANDARD DEV
QUEUE LENGTH	1.82	0.112
QUEUE TIME	0.102	0.025

QUEUE AT NODE: MQUEUE(4)

	MEAN	STANDARD DEV
QUEUE LENGTH	1.46	0.144
QUEUE TIME	0.130	0.030

QUEUE AT NODE: MQUEUE(5)

	MEAN	STANDARD DEV
QUEUE LENGTH	1.22	0.117
QUEUE TIME	0.140	0.039

QUEUE AT NODE: MQUEUE(6)

	MEAN	STANDARD DEV
QUEUE LENGTH	1.69	0.102
QUEUE TIME	0.112	0.031

QUEUE AT NODE: MQUEUE(7)

	MEAN	STANDARD DEV
QUEUE LENGTH	1.78	0.121
QUEUE TIME	0.134	0.021

QUEUE AT NODE: MQUEUE(8)

	MEAN	STANDARD DEV
QUEUE LENGTH	1.58	0.120
QUEUE TIME	0.136	0.027

QUEUE AT NODE: MQUEUE(9)

	MEAN	STANDARD DEV
QUEUE LENGTH	1.48	0.151
QUEUE TIME	0.124	0.032

QUEUE AT NODE: MQUEUE(10)

	MEAN	STANDARD DEV
QUEUE LENGTH	1.33	0.140
QUEUE TIME	0.131	0.015

QUEUE AT NODE: MQUEUE(11)

	MEAN	STANDARD DEV
QUEUE LENGTH	1.63	0.110
QUEUE TIME	0.118	0.025

QUEUE AT NODE: MQUEUE(12)

	MEAN	STANDARD DEV
QUEUE LENGTH	1.75	0.138
QUEUE TIME	0.137	0.024

QUEUE AT NODE: MQUEUE(13)

	MEAN	STANDARD DEV
QUEUE LENGTH	1.54	0.122
QUEUE TIME	0.118	0.037

QUEUE AT NODE: MQUEUE(14)

	MEAN	STANDARD DEV
QUEUE LENGTH	1.55	0.121
QUEUE TIME	0.140	0.034

QUEUE AT NODE: MQUEUE(15)

	MEAN	STANDARD DEV
QUEUE LENGTH	1.61	0.137
QUEUE TIME	0.137	0.022

QUEUE AT NODE: MQUEUE(16)

	MEAN	STANDARD DEV
QUEUE LENGTH	1.90	0.119
QUEUE TIME	0.135	0.037

QUEUE AT NODE: MQUEUE(17)

	MEAN	STANDARD DEV
QUEUE LENGTH	1.72	0.144
QUEUE TIME	0.125	0.036

QUEUE AT NODE: MQUEUE(18)

	MEAN	STANDARD DEV
QUEUE LENGTH	1.35	0.138
QUEUE TIME	0.127	0.019

QUEUE AT NODE: MQUEUE(19)

	MEAN	STANDARD DEV
QUEUE LENGTH	1.73	0.103
QUEUE TIME	0.113	0.036

QUEUE AT NODE: MQUEUE(20)

	MEAN	STANDARD DEV
QUEUE LENGTH	1.10	0.102
QUEUE TIME	0.127	0.035

QUEUE AT NODE: MQUEUE(22)

	MEAN	STANDARD DEV
QUEUE LENGTH	1.53	0.128
QUEUE TIME	0.137	0.026