Science of Computer Programming 4 (1984) 249–256
North-Holland

A SIMPLE LIVELOCK-FREE ALGORITHM FOR PACKET SWITCHING

R.K. SHYAMASUNDAR

National Centre for Software Development and Computing Techniques, Tata Institute of Fundamental Research, Bombay 400 005, India

Communicated by K. Apt
Received April 1983
Revised April 1984

Abstract. Store-and-forward deadlock (SFD) occurs in packet switched computer networks when, among some cycle of packets buffered by the communication system, each packet in the cycle waits for the use of the buffer currently occupied by the next packet in the cycle. In this paper, a deadlock-free algorithm as well as a livelock-free algorithm for packet switching is obtained using the strategy of the banker's algorithm. Furthermore, the solution obtained is interpreted for the hyper-fast banker's problem.

1. Introduction

In most cases, the occurrence of network deadlock has a horrendous impact upon network users. When the deadlocks are discovered, they are frequently corrected in an ad hoc fashion. Since changes in existing implementations can be costly, it is preferable to incorporate deadlock avoidance (or recognition and recovery) procedures into the design as early as possible and in an orderly fashion. A primary concern with any deadlock avoidance procedure is that it has minimal effect upon system performance during normal operating conditions.

SFD occurs in packet switching computer networks when some group of packets exists in the network communication system such that among this group no packet has arrived at its destination and none can make a next hop towards its destination. The simplest case of SFD is direct SFD, in which given two adjacent network nodes N_i and N_j, N_i's packet buffers are full of packets destined for N_j and N_j's buffers are full of packets for N_i; as a result, no packet can move. In more complex SFD's a cycle of nodes is involved each holding packets for the next node in the cycle.

Several algorithms have been proposed in the literature (see Merlin and Schweitzer [5], Toueg and Ullman [7], Gelernter [1]) for avoiding deadlock as well as livelock. In this paper, we first obtain a deadlock-free algorithm using the strategy for the banker's algorithm. A simple correctness proof is given for this algorithm. Though the deadlock-free algorithm is not new, it has resulted in a simpler presentation with a simple correctness proof. The algorithm obtained is further transformed to

a livelock-free algorithm. We also interpret the solution obtained for the hyperfast banker's problem (Lauer et al. [4]).

Before we discuss the algorithms, we formalize the packet switching network as given in [7].

Definitions

A packet switching network is a directed graph $G = (V, E)$; the vertices V represent processors and the edges E represent communication links. Messages called packets are to be passed between processors. Each network has an associated constant, b, the number of buffers at each vertex; a buffer can hold exactly one packet. Associated with each packet is an acyclic route $r_1, r_2, \ldots, r_q$ which is a path in G; r_1 is the source and r_q is the destination of the packet. We assume a fixed routing procedure as in Kleinrock [3], where a packet's route is determined at the source node. We may also assume that the route of a packet is included as part of the message in the packet, although in practice the packet could hold only the source and destination with each processor in the network responsible for deducing the next vertex to which the packet is to be passed. Also associated with the network G is a constant k, the length of the longest route taken by a packet in G. If we want to state explicitly the two constants associated with a network G, we write that G is a (b, k)-network. Note that k need not be the length of the longest path in the network; we may never wish to send messages between distant nodes.

The moves made by the network can be classified in the following way:

(1) *Generation:* A vertex V accepts a packet p created by a process residing in V and places in an empty buffer.

(2) *Passing:* A vertex V transfers a packet in one of its buffers to an empty buffer of vertex W, where $V \rightarrow W$ is an edge and the route for the packet has W following V. The buffer of V holding the packet becomes empty.

(3) *Consumption:* A packet in a buffer of V such that the destination for the packet is V, is removed from that buffer and the buffer is made empty (consumption is assumed to take only a finite amount of time).

For the sake of simplicity, we assume that the edges are of length 1. We say that the algorithm (controller) is *deadlock-free* for a given network if it does not permit this network to enter a state in which one or more packets can never make a move permitted by that algorithm (controller) as long as no additional packets are generated. In other words, given a network and a set of source-destination pairs, is there a corresponding set of routes such that the unrestricted flow of packets is *deadlock-free.*

Another interesting situation is termed *livelock*—a situation in which unfair scheduling of packets prevents one or more packets from reaching their destination. An algorithm is said to be *livelock-free* if there is no possibility of livelock in the network. It must be noted that livelock-freeness implies deadlock freeness.

As our algorithm is based on the banker's algorithm we will briefly discuss the Banker's algorithm as discussed in Habermann [2].

2. Banker's algorithm

Consider a nonemtpy set of processes, P, each of them engaged in a finite transaction for the completion of which it may need a (varying but bounded) number of units of some shared resource at its exclusive disposal (the units are all equivalent). A process may 'borrow' one or more units which are then added to its current 'loan', it may 'return' one or more units, which are then subtracted from its current loan. The act of borrowing is restricted by the condition that, for each process, the loan will never exceed a pre-stated 'need', i.e. the maximum number of units that may be simultaneously needed by that process for the completion of its transactions. The act of returning is restricted by the constraint that for no process the loan can ever become negative; upon completion of a transaction, the corresponding loan returns to zero.

Treating the capital available as the total number of resources available, customers as processes, and the loans claimed as the resources required by the respective processes, we formalize the borrowing just described as follows:

Let STOCK represent the total resources available of one type. Let $P[0]$, $P[1], \ldots, P[i], \ldots, P[N-1]$ be the processes requiring NEED$[P[j]]$ units of the resource. Let ALLOC$[P[j]]$ indicate the resources already allocated to process $P[j]$ by the system and CLAIM$[P[j]]$ = NEED$[P[j]]$ − ALLOC$[P[j]]$. Obviously, for any realizable state the following conditions are to be satisfied:

(a) For all processes NEED$[P[j]] \leqslant$ STOCK, i.e. no process claims more resources than are available;

(b) For all processes ALLOC$[P[j]] \leqslant$ NEED$[P[j]]$, i.e. no process will try to seize more resources than it has claimed;

(c) Sum of the resources already allocated is at most equal to STOCK.

Safe State: When a realizable state does not contain the deadlock danger, it is called a safe state; otherwise it is called an unsafe state. Now, let us formalize the notion of safe state with the following condition in mind:

The claim by process $P[k]$ must not exceed the sum of the free resources and those resources which will become free 'in due time'.

Obviously, under the above constraint the claim by $P[k]$ could be realized in a finite time. If the above condition is satisfied for all processes in some state then we say that the state is safe. A safe state implies that the claims of the process can be linearly ordered such that the claim of each process does not exceed the sum of the free resources and those resources which will become free 'in due time'.

The condition is formalized as follows:

For all $0 \leqslant i < N$,

CLAIM$[P[i]] \leqslant$ unused STOCK
+the resources already allocated to
processes (going to become free 'in due time')
$P[0], P[1], \ldots, P[i-1]$.

The above condition can be written as:

For all $0 \leq i < N$,

$$\text{CLAIM}[P[i]] \leq \left(\text{STOCK} - \sum_{j=0}^{N-1} \text{ALLOC}[P[j]]\right) + \sum_{j=0}^{i-1} \text{ALLOC}[P[j]].$$

In other words,

$$\forall 0 \leq i < N, \text{CLAIM}[P[i]] < \left(\text{STOCK} - \sum_{j=i}^{N-1} \text{ALLOC}[P[j]]\right). \quad \text{(A)}$$

It must be noted that it is only necessary that the condition is true for *one* permutation of process numbers. The Banker's Algorithm tries to find a permutation of process numbers keeping the above relation invariant (for a proof see [2]).

3. Deadlock-free packet switching algorithm

In a fixed routing (b, k) network path lengths of packets can be at most k (if N is the number of nodes in the network then obviously $k \leq N-1$ assuming the length of all edges to be 1). In the following, we reduce the problem of packet switching to the problem of investigating whether a given pattern of loans and needs is safe.

Let us consider vertex V. The buffers of V could be filled by packets whose destinations are at a distance of at most k. A packet of length zero coming into vertex V implies that it is a packet for consumption at V, i.e. the destination of the packet is that node itself. At each node of the (b, k) network, we can partition the packets as follows:

At each vertex V, the packets can be grouped into $k+1$ groups $(j_0, j_1, \ldots, j_k)$ where j_i gives the number of packets stored in V whose route from V to its destination is of length i.

The conditions under which the (b, k)-network will be *safe* is given below:

Assertion A1. The pattern of generation and passing of the packets is deadlock-free (safe) in (b, k)-network, if the following condition holds at all nodes of the network:

Let V be a vertex and p be a packet with a route of length j from V to its destination. p will be accepted by V iff for all i, $0 \leq i \leq j$,

$$i < b - \sum_{r=i}^{k} j_r$$

where j_r is the number of packets stored in V whose destinations are r hops from node V.

Note. It must be noted that we do not consider packets being sent along some non-zero path for itself (i.e. the routes are acyclic). Also, we assume that $b > k$ and the decision whether a packet p will be accepted by a vertex V or not is made

according to local information alone. Further, it may be observed that the relation given here is identical in structure to (A) of the Banker's Algorithm.

Proof. Let us assume that the assertion holds and there is a deadlock. The presence of a deadlock implies the existence of a cycle of nodes in the network, each holding packets for the next node in the cycle (note that we are using a fixed routing for packets). The essential idea of our proof is to show that if V_1 is stuck on packet p_1 of length d_1 having V_2 as its next destination, then V_2 must be stuck on p_2 of length d_2 such that $d_2 < d_1$.

Let V_1 get stuck on p_1 of length d_1 from its destination. Let V_2 be the next destination in the route of p_1. Since there is a deadlock, there is at least one deadlocked packet in V_2. Let p_2 be a packet in V_2 whose distance d_2 to its destination is minimal with respect to the other deadlocked packets in V_2. That is if j_i's are the number of packets of length i from their respective destinations at V_2, then

$$d_2 = \min\{d | j_d > 0\}.$$

At V_2, the partition of the packets is given by

$$(0, \ldots, 0, j_{d_2}, j_{d_2+1}, \ldots, j_k). \tag{1}$$

It must be noted $d_2 \neq 0$, since $d_2 = 0$ implies V_2 is stuck on a packet of length zero. However, this contradicts the hypothesis that the packet received by the node is consumed and hence the corresponding buffer is vacated in a finite amount of time.

Let p_s be the last packet accepted by V_2. Let d_s be the distance from V_2 to the destination node of p_s. By definition of d_2, we have $d_2 \leq d_s$. From the assertion, d_s will be accepted by V_2 provided for all i, $0 \leq i \leq d_s$,

$$i < b - (j_i + \cdots + (j_{d_s} - 1) + \cdots + j_k).$$

Hence for $i = d_2$ we get

$$d_2 < b - \left(\sum_{r=d_2}^{k} j_r \right) + 1,$$

i.e.

$$d_2 \leqslant b - \sum_{r=d_2}^{k} j_r. \tag{2}$$

Since p_1 is not accepted by V_2, then there exists i_0 such that $0 \leqslant i_0 \leqslant d_1 - 1$ and

$$i_0 \geqslant b - \sum_{r=i_0}^{k} j_r. \tag{3}$$

Suppose, for contradiction $d_2 > d_1 - 1$. Then, we have $i_0 \leqslant d_1 - 1 < d_2$. From (1), we get

$$\sum_{r=i_0}^{k} j_r = \sum_{r=d_2}^{k} j_r. \tag{4}$$

From (3) and (4) we get

$$i_0 \geq b - \sum_{r=d_2}^{k} j_r. \tag{5}$$

But $d_2 > d_1 - 1 \geq i_0$. Hence $d_2 > b - \sum_{r=d_2}^{k} j_r$ contradicting (2). Hence we conclude

$$d_2 \leq d_1 - 1 \quad \text{or} \quad d_2 < d_1.$$

As $<$ is a partial order, it follows that if there is a deadlock, then there must be some node stuck on packets of length zero. However, as the latter is not possible, there could not have been a deadlock.

Hence the assertion. □

Deadlock-free algorithm

1. Each node of the (b, k) network makes the moves preserving Assertion A1.
2. Initially all the nodes satisfy A1.

Note. It must be noted that we do not consider communication failures.

4. Livelock-free algorithm

In the above algorithm livelock can occur for the following reasons:

(1) *Packet passing at each node:* Packets at each node (of varied destination lengths) already received may be arbitrarily selected for passing—which can lead to livelock;

(2) *Packet reception at each node:* Packets may be received at each node from its neighbours arbitrarily subject to satisfying the relation given in A1—arbitrary selection could lead to livelock.

Algorithm. For the sake of simplicity, we consider packet passing and packet reception explicitly in the following scheme:

Packet passing: Each vertex has a local timestamp: It must be noted that there is no need to synchronize the clocks of all the nodes of the network—it is sufficient to have one local timestamp for each node. A packet p can be passed from node V to its next node provided it satisfies the following conditions:

(i) The relation given by A1 is preserved;

(ii) The packet has been waiting for the longest time (it is assumed that a node receives at most one packet or a message at a time—thus there is no question of 'after you–after you' occurring).

Packet reception: Whenever node V_1 wants to send a packet to V_2 (V_1 sends a message to V_2 to that effect), vertex V_2 associates a timestamp with each message. Now, V_2 agrees to receive packet p from V_1 provided the following conditions are satisfied:

(i) the relation given by A1 is satisfied;

(ii) V_1 is the neighbour of V_2 that has seen waiting for the longest time (compared to the other neighbours of V_2 which might also be waiting to pass packets to V_2); again from the earlier assumptions it follows that 'after you–after you' cannot occur.

Note. It may be observed that both the situations are alike except that in the former, waiting packets are considered and in the latter waiting nodes (immediate neighbours) are considered. Further, it may be noted that a packet can be successfully passed if and only if one node satisfies the conditions for a packet passing and the other satisfies the conditions of packet-reception.

Proof of correctness. Consider a node V. Let the packets in its buffers be denoted by $\{i_1, i_2, \ldots, i_k\}$ where i_j indicates the number of packets whose destinations are at a distance of j hops from node V. As there is no deadlock, we can assert that there exists at least one packet say p of length j hops, which can reach its destination in a finite amount of time. Further, if the node to which the packet is passed from V is W, then we know that conditions for packet passing at V and conditions for packet reception at W are to be satisfied. Hence, the packet received by V in place of p cannot override the other packets as we know that there is no deadlock. Hence it follows that no packet can be livelocked for ever.

5. Hyperfast bankers algorithm (Lauer et al [4])

In this section, we will show how a generalized hyperfast Banker's solution can be obtained from the solution. Let us first brief the hyperfast banker as described in [4].

The banker builds in his office as many windows as he has money and he lets the people queue at a window depending on the amount of money they ask for. Cyclically, he looks at how much money he may loan and opens the windows corresponding to loans not greater than this amount. At each open window, he serves a single customer, if any, by giving him a dollar and by sending him to the next window, i.e., that corresponding to a loan one less, and then he closes the window. Whenever a customer applies for a new loan, he verifies whether the loan can be given safely (so that there is no deadlock) and blocks them whenever it is unsafe (the term 'safe' is understood as discussed earlier).

Now, we can generalize the scheme as follows: There is a network of banks, each bank functioning as described above. Further, each bank has its counter in each of the other nodes and the issue of the loan to a customer can be distributed through its various counters. Now, we can interpret the hyperfast banker's scheme in terms of the livelock-free packet switching network as follows:

Each banker at a node looks at how much money he may loan and describes a path of the counters at the various nodes from where the money can be collected.

The customer returns the money to the bank where he had applied for the loan. As in the hyperfast banker's scheme, each node will not initiate any fresh path unless it gets back the money. The solution follows from the correspondence of the two schemes.

6. Concluding remarks

In this paper, a simple algorithm has been proposed based on the strategy of Banker's algorithm. The strategy has resulted in a simple proof of correctness of the algorithm. It may be observed that the deadlock-free algorithm obtained using A1 that uses the strategy of Banker's algorithm is the same as the DF uniform controller as discussed in [7].

The connection of the banker's problem with the allocation of nonshareable, non-preemptive and reusable resources in an operating system are well known. By deriving the packet-switching algorithm from the banker's algorithm and the generalization to the hyperfast banker's scheme, we have shown that the generalization serves as interesting resource management model for distributed systems.

Acknowledgment

It is a pleasure to thank the referees for their suggestions and critical comments.

References

[1] D. Gelernter, A DAG-based algorithm for prevention of store-and-forward deadlock in packet networks, *IEEE Trans. Comput.* (10) (1981) 709–715.
[2] A.N. Habermann, Prevention of system deadlocks, *Comm. ACM* (7) (1969) 373.
[3] L. Kleinrock, *Queueing Systems Vol. II: Computer Applications* (Wiley, New York, 1976).
[4] P.E. Lauer, P.R. Torrigiani and R. Devillers, A COSY Banker: specification of highly parallel and distributed resource management, TR 151, University of Newcastle upon Tyne (1980).
[5] P.M. Merlin and P.J. Schweitzer, Deadlock avoidance in store and forward networks—I: Store-and-forward deadlock, *IEEE Trans. Comm.* **28** (3) (1980) 345–354.
[6] A.S. Tanenbaum, Network protocols, *ACM Comput. Surveys* **13** (4) (1981) 353–489.
[7] S. Toueg and J.D. Ullman, Deadlock-free packet switching networks, *SIAM J. Comput.* **10** (3) (1981) 594–610.