V. Ahuja

# Algorithm to Check Network States for Deadlock

*The problem of checking the states of a system for deadlock is treated for a single class of systems, or networks, and for a single class of resources, or buffers. An algorithm is described that, for a given state, requires* O$[m + n^2]$ *operations, where m and n are, respectively, the number of tasks and nodes in the state. (In general, m is greater than n.)*

## Introduction

In a computing system, two or more tasks may be unknowingly waiting for each other to release resources. Until the wait is resolved, the system resources in question are wasted and the progress of such tasks is blocked. Such a state of the system is generally called a *deadlock*.

Unless otherwise protected, computing systems with concurrent tasks and multiple resources are exposed to deadlock. Several approaches have been developed to provide such protection by checking each state of the system for deadlock. A typical algorithm [1] to check a system state for a deadlock takes O$[mn]$ comparison operations, where $m$ is the number of tasks and $n$ is the number of resource classes. In this paper we describe an algorithm that, for a class of communication networks to be described later, requires O$[m + n^2]$ comparisons. The algorithm is computationally better, since, in general, there are more tasks ($m$) than the number of resource classes ($n$).

Now consider a store and forward communication network, where a message is stored and then transmitted by each node on its route. Message transmission from a node does not start until a buffer has been obtained in the next node on its route. The problem is restricted to buffers as resources and assumes fixed message routes. A deadlock occurs if tasks in two or more nodes are waiting for each other for buffers and there is no free buffer.

The general problem of flow control in networks has been addressed by several authors [2, 3]. Here, we consider the problem of deadlock resulting from buffer depletion in networks. In the literature on the ARPA network [4, 5] such deadlocks have been described as direct or indirect store and forward lockups. Network deadlocks can be prevented by enforcing some buffer classification and allocation scheme [6]. Alternatively, one can examine a network design for exposure to deadlock. Elsewhere, we have described an approach that determines the exposure of networks to deadlocks by translating this problem into a network flow problem [7, 8]. In this paper, we develop an algorithm that checks a given network state for deadlock. The information obtained by executing the algorithm can also help in preventing the predicted deadlock.

In the next section, a model is described for a network in which deadlock can occur. In the succeeding section, some results for the algorithm are proved. The algorithm is then described, along with comments on its computational complexity.

## A model

The nodes, links, and buffers in a network can be represented in terms of a directed graph, where each node is labeled with the number of buffers in it. So, a *network* is a triple $N = (X, A, B)$, where $X$ is a nonempty finite set representing the nodes of the network, $A$ is a relation on $X \times X$ representing the directed links of the network, and $B$ is a function on $X$ to the set of positive integers; $B(x(i))$, or simply $b(i)$, is the number of buffers in node $x(i) \in X$. A buffer is either *free* or *allocated*.

A network with six nodes is shown in Fig. 1(a). There are two buffers in each node.

**Copyright** 1979 by International Business Machines Corporation. Copying is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract may be used without further permission in computer-based and other information-service systems. Permission to *republish* other excerpts should be obtained from the Editor.

The activity in the network can be defined in terms of the allocation of and request for resources. We allow a unit of activity to have one buffer allocated in a node and request one more in another node. In order to distinguish among tasks requiring buffers in the same node, we include an index for the buffer allocated to a given task. (This index is required to define some entities of the model; it is not used in the algorithm.) So, a *task* is defined by a triple, $\langle x(i), p, x(j)\rangle$, where $x(i)$, $x(j)$ are two adjacent nodes and $p$ is a positive integer not greater than $b(i)$. For a task $t = \langle x(i), p, x(j)\rangle$, call $x(i)$ the *allocated node*, $A(t)$; $x(j)$ the *requested node*, $R(t)$; and $p$ the *buffer index* of $t$.

In a network $N = (X, A, B)$, the *set T of all tasks* is given by

$$\bigcup_{i=1}^{n} \bigcup_{k=1}^{b(i)} (\{\langle x(i), k, x\rangle | x \text{ is adjacent to } x(i)\}),$$

where $n$ is the cardinality of set $X$.

Then a *state* of a network is defined by a subset $T'$ of $T$ such that $T'$ has no more than one task with the same starting node and buffer index. Our definition of state allows at most one task to be allocated to any given buffer.

Next, we introduce the concept of "mutual wait" among tasks. Consider a set of connected nodes such that their tasks are waiting for each other for a buffer. Furthermore, there is no free buffer in any of these nodes. Then, the tasks in these nodes are involved in an unresolvable wait that leads to a deadlock. This is defined next.

In a state $T'$ of a network, a nonempty set $T'' \subseteq T'$ of tasks is in a *mutual wait* if

1. For each task $t = \langle x(i), r, x(j)\rangle$ in $T''$, the tasks in the set $\{\langle x(i), p, x(j)\rangle | p \le b(i), x(j) \in X\}$ are in $T''$ and $x(j)$ is in the set $\{A(t') | t' \in T''\}$.
2. There is no free buffer in the set $\{x(i) | x(i) \in A(T'')\}$ of nodes.

According to the first part of the definition, if there is some task of a node in $T''$, then all the tasks in that node are in $T''$. It also assures that the requested node of each task in $T''$ is also in the set of nodes that have tasks in $T''$. The second part implies that there is no free buffer in the nodes that have tasks in $T''$.

Then, in a network, a *deadlock* or a *deadlock state* is a state $T'$ if there exists a nonempty set $T'' \subseteq T'$ of tasks that is in a mutual wait.

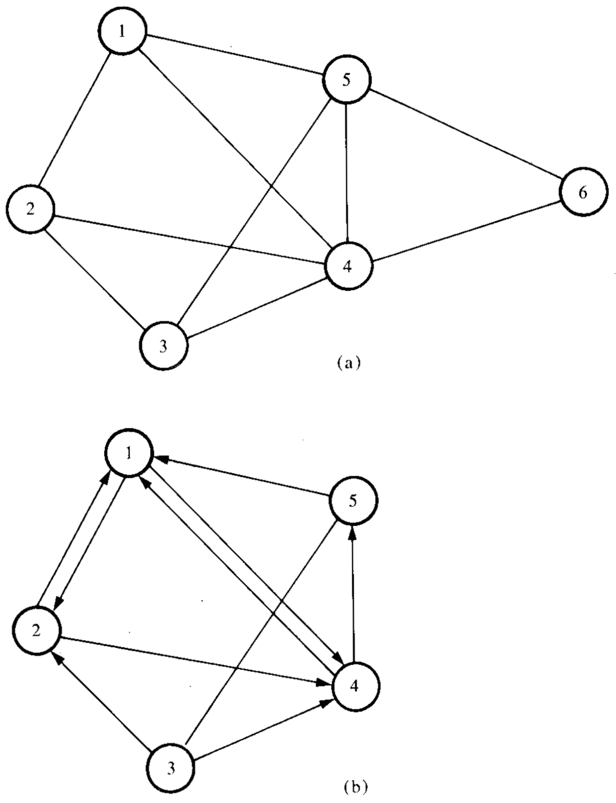

**Figure 1** A deadlock in the network at (a) is shown at (b). A task at the trailing end of the arrows is waiting for a buffer in the node at the leading end.

By definition, a set of tasks in mutual wait must be nonempty. Now if there is some node that has no free buffers, then its tasks must be waiting for buffers in at least one other node. By definition of mutual wait, this node must also not have a free buffer. Hence, in order for a state to be a deadlock, there must be at least two nodes without any free buffers.

Figure 1(b) illustrates a deadlock in the network of Fig. 1(a). The set (1, 2, 3, 4, 5) of nodes is in a deadlock, since each task in this set of nodes is waiting for some other node in the set and there is no free buffer.

The set of tasks in a mutual wait for this deadlock is

$$\begin{aligned} T' = \{&(1, 1, 4), \ (1, 2, 2), \\ &(2, 1, 1), \ (2, 2, 4), \\ &(3, 1, 2), \ (3, 2, 4), \\ &(4, 1, 1), \ (4, 2, 5), \\ &(5, 1, 1), \ (5, 2, 3)\}. \end{aligned} \quad (1)$$

This completes the description of a network model.

$$\mathbf{M} = \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

(a)

$$\mathbf{M} = \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 & 1 & 0 & 0 \\ 4 & 1 & 0 & 0 & 0 & 1 & 0 \\ 5 & 1 & 0 & 1 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

(b)

**Figure 2** Matrix **M** at (a) is initialized as shown at (b).

## Results

Several results are required to establish an algorithm for checking a network state. By definition, a network state is a deadlock if it has a set of tasks in a mutual wait. In this section, results are developed that use the description of various tasks in a state to determine whether a subset of its tasks is in a mutual wait.

First, we demonstrate the intuitive result that in a deadlock there must exist a set of tasks that are waiting for each other in a cyclic fashion.

**Theorem 1**

Let $T'$ be a deadlock state of a network so that the set $T'' \subseteq T'$ of tasks is in a mutual wait. Then any directed path constructed by the allocated and requested nodes of the tasks in $T''$ contains a cycle.

**Proof**

Begin construction of a directed path from any task in $T''$, say $t_1$. The first arc of the path is $(A(t_1), R(t_1))$. Let $R(t_1) = x(i)$. By definition of mutual wait, $x(i)$ is in the set $\{A(t) | t \in T''\}$. Let $t_2 \in T''$ be a task with a buffer in node $x(i)$. Then, by repeating the above argument, we get the next arc of the directed path.

Continuing the construction, the path must revisit a node since there are a finite number of nodes in the network. This implies that the directed path has a cycle. Since this is true for an arbitrary directed path, it is true for any directed path of allocated and requested nodes. This completes the proof. □

Next we derive a result for obtaining the set of nodes, if any, such that the tasks in those nodes are in a mutual wait. In order to facilitate the derivation, some additional definitions are required. First a binary relation $W$ is defined as a relation among any two tasks, $t_1$ and $t_2$, that holds only if $t_2$ occupies a buffer in a node that is also the requested node of $t_1$. So $t_1\ W\ t_2$ is true if $t_2 = \langle R(t_1), p, x\rangle$. Next, a binary relation $W^*$ is defined as a relation between any two tasks $t_1$ and $t_n$ that holds if there exists a sequence $t_1\ W\ t_a$, $t_a\ W\ t_b$, $\cdots$, $t_u\ W\ t_n$. So $t_1\ W^*\ t_n$ is true if either 1) $t_1\ W\ t_n$ or 2) $t_1\ W^*\ t_a$ and $t_a\ W\ t_n$ holds. Thus, the relation $W^*$ holds between any two tasks $t_1$ and $t_n$ if $t_1$ requires, directly or indirectly through other tasks, a buffer in the node with task $t_n$.

Consider a state $T$ of a network $N = (X, A, B)$ such that the set $X' \subseteq X$ of nodes has no free buffers. Also, let $X'' \subseteq X'$ be the set of all those nodes in $X'$ such that at least one task in each node in $X''$ satisfies the relation $t\ W^*\ t'$ and $t'$ is in a node with one or more free buffers. Then, the set $X' - X''$ of nodes has no free buffers, and also no task in them can obtain a buffer in the requested node. (This is true since no task $t$ in $X' - X''$ holds the relation $t\ W^*\ t'$, where $t'$ is in a node with a free buffer.) So each task in $X' - X''$ requires a buffer in nodes $X' - X''$, and there is no free buffer in any node in $X' - X''$. By definition, this implies that the set of tasks in nodes $X' - X''$ is in a mutual wait. This proves the next theorem.

**Theorem 2**

Let $N = (X, A, B)$ be a network and $T'$ be a state of $N$ such that there is no free buffer in the set $X' \subseteq X$ of nodes.

Also, let $T'' \subseteq T'$ be the set of tasks such that for each task $t$ in $T''$, $t\ W^*\ t'$ holds and the node $A(t')$ has a free buffer. Then the set of tasks in the nodes $X' - \{A(t) | t \in T''\}$ is in a mutual wait.

Finally, we prove a related result that can be used to remove tasks, and therefore nodes, from deadlock consideration. Consider a state of a network that has tasks waiting directly, or indirectly through other tasks, for a buffer in a node that has one or more free buffers. Then such tasks cannot be in a mutual wait. This is proved next.

**Theorem 3**
Let $N = (X, A, B)$ be a network and $T'$ be a deadlock state such that the set $T'' \subseteq T'$ of tasks is in a mutual wait. Then, a task $t_1 \in T'$ is not in $T''$ if $t_1\ W^*\ t_2$ and the node $A(t_2)$ has a free buffer.

**Proof (by contradiction)**
Assume that task $t_1$ is in $T''$ and $t_1\ W^*\ t_2$ is such that the node $A(t_2)$ has a free buffer. Let $t_1\ W^*\ t_2 = t_1\ W\ t_a,\ t_a\ W\ t_b,\ \cdots,\ t_n\ W\ t_2$.

Since $t_1$ is in $T''$ and $T''$ is in a mutual wait, $R(t_1)$ is in $\{A(t) | t \in T''\}$ and $R(t_1)$ has no free buffer. But by definition of the relation $W$, $R(t_1) = A(t_a)$. So, $A(t_a)$ has no free buffer and $t_a$ is in $T''$.

Repeating the above argument for $t_a, t_b, \cdots, t_n$ implies that $A(t_2)$ has no free buffer.

This contradicts the assumption. Hence the task $t_1$ is not in the set $T''$ of tasks.

## Algorithm
Our algorithm is based on the above results and is described next. Let $N = (X, A, B)$ be the network, as defined earlier, and $T'$ be its state to be checked. The nodes in $X$ are referred to as $x(i)$, $i = 1$ to $n$, where $n$ is the number of nodes in $X$.

Step 1. Define a $n \times n$ matrix **M** and a vector **V** of dimension $n$. Variables $i, j, k, l, p$ will be initialized as they are used.

Step 2. For each node $x(i)$, initialize **V** as follows:

$\mathbf{V}(i) = 0$ if $x(i)$ has no free buffer,
$= 1$ if $x(i)$ has at least one free buffer.

Now check the vector **V** for elements with a zero. If there are fewer than two elements with a zero, then $T'$ cannot be a deadlock state and the algorithm terminates. (The requirement for two elements with a zero results from the fact that a deadlock must involve at least two nodes with no free buffers.)

Step 3. For each node $x(i)$, initialize **M** so that: $\mathbf{M}(i, j) = 1$ if there is a task in node $x(i)$ that has requested a buffer in node $x(j)$; otherwise, $\mathbf{M}(i, j) = 0$.

Step 4.

a. Create a vector **Y** that contains indices of all those nodes, $x(i)$, for which $\mathbf{V}(i)$ is 1. (If none, proceed to Step 5.) Proceed to Step 4(b) with $k$ set to the value of the first element in **Y**.

b. For each row $p$ such that $\mathbf{M}(p, k)$ is 1 and $\mathbf{V}(p)$ is 0, set $\mathbf{V}(p)$ to 1 and add $p$ to the vector **Y**.

c. Set $k$ to the next element of **Y** and proceed to Step 4(b). Continue until all entries in **Y** have been checked.

Step 5. Check the vector **V** for an element that equals zero. If there is at least one element $l$ such that $\mathbf{V}(l)$ is zero, then and only then the state is a deadlock.

To demonstrate application of the algorithm, we apply it to the network of Fig. 1(a). Let $T'$ be the network state to be examined. Tasks in $T'$ have been specified earlier in (1).

Step 1. See Fig. 2(a).
$\mathbf{V} = (0, 0, 0, 0, 0, 0)$.

Step 2. $\mathbf{V} = (0, 0, 0, 0, 0, 1)$.
Since there are more than two elements with a zero, proceed to Step 3.

Step 3. Using the tasks specified in (1), elements of the matrix **M** are initialized as in Fig. 2(b).

Step 4.

a. $\mathbf{Y} = \{6\}$,
$k = 6$.

b. $p =$ None.

c. There are no more entries in **Y**.

Step 5. The state is a deadlock since **V** has five elements with a zero.

Next, we provide an estimate of the computational complexity for the above algorithm, based on the comparison operations.

Step 1 defines a matrix **M** and a vector **V**. The initialization of **M** will require $O[n^2]$ assignment operations, but no comparisons.

In Step 2 we check to determine if there are at least two nodes that have no free buffers. If not, then as argued earlier the state is not a deadlock. This would require $O[n^2]$ comparisons.

In Step 3, we set to 1 those elements $\mathbf{M}(i, j)$ for which a task in node $x(i)$ has requested a buffer in node $x(j)$. This step would require $O[m]$ comparisons to check the tasks in all the nodes and then to initialize at most $m$ elements in matrix $\mathbf{M}$, since there are only $m$ tasks in the state. (If each node is checked for tasks for other nodes, it will require $O[n]$ comparisons.)

In Step 4, we insert a value of 1 for those node indices in $\mathbf{V}$ that need not be checked for *mutual* wait (Theorem 2). To vector $\mathbf{Y}$ we add those nodes (indices) that have no free buffers and also have some tasks in them that are waiting for a node that either has a free buffer or is, in turn, waiting for a node that can free a buffer. Since an index is added to $\mathbf{Y}$ if $\mathbf{V}(i)$ is also zero, therefore, by construction, the vector $\mathbf{Y}$ cannot have more than $n$ elements. For each entry in $\mathbf{Y}$, the algorithm checks at most $n$ elements of the corresponding column in $\mathbf{M}$. This would require $O[n^2]$ comparisons.

In Step 5, we check the vector $\mathbf{V}$ for any element with a zero. If so, such elements represent nodes that have no free buffers and also have tasks in them that cannot be removed from deadlock consideration. Using Theorem 3, tasks in such nodes are in a mutual wait. So, the state is a deadlock if and only if there exists a node $x(i)$ such that $\mathbf{V}(i)$ is zero. This would require $O[n]$ comparisons.

So, the complexity of this algorithm is of $O[m + n^2]$.

Although the algorithm requires $O[m + n^2]$ comparisons for a state, there may be several states to be checked for a deadlock. The maximum number of states of a network is given approximately by (see [7] for its derivation)

$$\left(\sum_{i=1}^{h-1} \binom{c-1}{i-1} \binom{h-1}{i}\right)^p,$$

where $p$ is the number of routes, $h$ is the average number of nodes on each route (directed paths of the network), and $c$ is the maximum number of tasks concurrently active on a route. So $m$ is not greater than $p \times c$. Also, $h$ is related to $n$ since, in general, $h$ increases as the number of nodes ($n$) increases. Now consider the states for which there is at most one node that has no free buffer. For such states, only the first two steps of the algorithm are executed. This requires $O(n)$ comparison operations, as compared to $O(m + n^2)$, thereby reducing the total computation cost of the algorithm.

## Summary

An algorithm was developed that can check a network state for a deadlock in $O[m + n^2]$ comparison operations, where $n$ is the number of nodes in the network and $m$ is the number of tasks in the state. This is better than the existing more general algorithms that require $O[mn]$ comparisons, since, in general, there are more tasks ($m$) than the number of nodes ($n$) in a network. Our algorithm derives its efficiency from the fact that it eliminates nodes whose tasks are not involved in a deadlock. This approach, which is based on Theorem 3 and is computationally inexpensive, removes chains of nodes by checking the tasks that can have their buffer requests satisfied. Existence of any remaining tasks necessarily implies that the state is a deadlock.

The algorithm also provides information that can be used to prevent occurrence of corresponding deadlocks. After execution of Step 5, elements in vector $\mathbf{V}$ with a zero represent the nodes that have their tasks in a mutual wait. So the network designer must prevent all such tasks from simultaneously occupying buffers in these nodes. This may be achieved by reducing the number of tasks in one of the nodes that have tasks in a mutual wait. Alternatively, the network designer may allow one or more extra buffers in one of the nodes in deadlock.

This algorithm cannot be used for systems in which a task can have more than one resource unit allocated to it while requesting another one. Further work is needed to extend it to the general class of operating systems.

## References

1. E. G. Coffman, Jr., and P. J. Denning, *Operating Systems Theory*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1973.
2. V. Ahuja, "On Congestion Problems in Communication Networks," *Trends and Applications: 1978 Distributed Processing*, National Bureau of Standards, Gaithersburg, MD, May 18, 1978.
3. D. W. Davies, "The Control of Congestion in Packet-Switching Networks," *IEEE Trans. Commun.* **COM-20,** 546 (1972).
4. R. E. Kahn and W. R. Crowther, "Flow Control in a Resource-Sharing Computer Network," *IEEE Trans. Commun.* **COM-20,** 539 (1972).
5. L. Kleinrock, *Queuing Systems Volume 2: Computer Applications*, John Wiley & Sons, Inc., New York, 1976.
6. E. Raubold and J. Haenle, "A Method of Deadlock-Free Resource Allocation and Flow Control in Packet Networks," *Proceedings of the Third International Conference on Computer Communications*, International Council for Computer Communications, August 1976, p. 483.
7. V. Ahuja, "Exposure of Routed Networks to Deadlock," Ph.D. Thesis, Department of Computer Science, University of North Carolina at Chapel Hill, 1976 (*Technical Report 29-160*, IBM System Communications Division laboratory, Research Triangle Park, NC, July 1976).
8. V. Ahuja, "Determining Deadlock Exposure for a Class of Store and Forward Communication Networks," to be published.

*Received December 7, 1977; revised August 10, 1978*

*The author is located at the IBM System Communications Division laboratory, Research Triangle Park, North Carolina 27709.*