Predicting Deadlock in Store-and-Forward Networks

Claudio Arbib
Dipartimento di Elettronica, Università di Roma "Tor Vergata," Rome, Italy

Giuseppe F. Italiano*
Department of Computer Science, Columbia University, New York, New York 10027, and Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza," Rome, Italy

Alessandro Panconesi
Department of Computer Science, Cornell University, Ithaca, New York 14850

We consider the problem of predicting whether a deadlock will necessarily occur in a store-and-forward network. We define two versions of this problem, depending on whether or not the routes to be followed by packets are fixed. For networks with only one buffer per vertex, both versions of this problem are shown to be NP-complete even for simple classes of graphs (among others bipartite graphs, two terminal series-parallel [TTSP] graphs and therefore planar graphs). On the other hand, the same problems are shown to be polynomially solvable for treelike networks. In this case, two efficient algorithms for checking whether a treelike network with n vertices and p packets is bound to deadlock are proposed. The former has an $O(pn)$ time and space complexity, whereas the latter runs in $O(n \log n)^1$ time and requires $O(n)$ space. In the case of multibuffered networks, both versions of the problem are shown to be NP-complete even on treelike networks.

1. Introduction

A *Store-and-Forward Network* (in short *SF-network*) can be represented as an undirected graph $G = (V, E)$, where the vertices stand for processors and the

*Partially supported by an IBM Graduate Fellowship, by NSF Grants CCR-86-05353 and CCR-88-14977, and by the Italian MPI National Project "Algoritmi e Strutture di Calcolo."
[1]In this paper, all the logarithms are assumed to be to the base 2 unless explicitly specified otherwise.

edges for communication links. When necessary, the notation is completed by a function $b : V \rightarrow Z^+$, which associates with each vertex v_i the number $b_i > 0$ of buffers available at v_i [14]. When $b_i = k$ for each vertex v_i, the network is called a *k-network* [2]. If $b_i > 1$ for some vertex v_i, the network is called *multibuffered*. The function b defines the *buffer configuration* of the network. An *acyclic route* $R_p = \{v_i, v_2, \ldots, v_d\}$ is a directed path associated with each packet (message) p. Such routes can be either assumed as fixed *a priori* or not [8]. In the latter case, we are given only the pair (v_1, v_d), which is called the *source-destination* pair of p. A packet in a network is subject to the following *network moves*:

1. Generation. A vertex v creates a packet which is placed in one of its empty buffers.

2. Passing. A vertex v transfers a packet from one of its buffers to an empty buffer of w, where (v, w) is an edge in the network. When the routing is fixed *a priori*, this move is allowed only if (v, w) is an edge of the path specifying the route of p. After p has been moved, the buffer of v holding p is emptied.

3. Consumption. A packet in a buffer of v, such that destination of the packet is v, is removed from that buffer, and the buffer is emptied.

Network moves are performed by the *flow-control procedure* (from now on *fcp*). Several kinds of *fcp*'s can be considered, depending on which moves are allowed [14]: in particular, an *fcp* is *unrestricted* if the only requirement for a vertex v_i to accept a packet p is that v_i has at least one empty buffer.

Each of the previous moves causes a transition from a certain *state* of the network to another. The state of the network at time t_k is a mapping associating with each packet the vertex it occupies at t_k. Let P_k denote the set of packets standing in the network at time t_k, and σ_k: $P_k \rightarrow V$ be such a mapping. Clearly, P_k depends on the time considered. To give a more concise definition, we have to modify slightly the network topology. We introduce extra-nodes s_p and e_p linked, respectively, to the source and destination of each packet p, and working as mailboxes, respectively, for the producer and for the consumer of the message. In the new network, each packet occupies a buffer and the number of packets standing in the network is not time-dependent. Consequently, the only move that can be performed is *passing*. We can set $\sigma_k(p) = s_p$ if at time t_k p is still to be generated and $\sigma_k(p) = e_p$ if it has been consumed. The mapping σ_k completely defines the state of the network at time t_k. A state σ_i *precedes* a state σ_k if $\sigma_i(p) \neq s_p$ precedes $\sigma_k(p) \neq e_p$ along the route R_p for each packet p. A *sequence* $\{\sigma_k\}$ of states is *monotonic* if σ_i precedes σ_j whenever $i < j$. An *fcp* is *monotonic* if the sequence of states it produces is monotonic. In the following, we consider only monotonic *fcp*'s.

The notion of deadlock has been extensively studied in the literature. We recall the definition given in [7]. A packet is said to be *blocked* if it cannot be moved. A state σ_k is a *deadlock state* if there are packets which will remain blocked for ever, no matter what sequence of moves is performed. In this paper, we restrict our attention to the following notions: An SF-network is said to be *bound to deadlock* (or *deadlock-bound*) in a state σ_k if one of the following holds:

(i) σ_k is a deadlock state; or
(ii) for each allowed move, G is bound to deadlock in σ_{k+1}.

In other words, when an SF-network is bound to deadlock, a deadlock will necessarily occur within a finite number of moves, no matter what *fcp* is used. Clearly, when a network is bound to deadlock in a state σ_k, it is also bound to deadlock in a state σ_i succeeding σ_k. SF-networks that are not bound to deadlock in a certain state are called *deadlock safe* in that state. From now on, for sake of simplicity, we will say that a network is bound to deadlock or deadlock safe without making explicit reference to the state.

An SF-network is *exposed to deadlock* (or *deadlock-exposed*) if the *fcp* may perform moves causing a deadlock state [4]. Notice that deadlock-exposed SF-networks are not necessarily bound to deadlock. The deadlock-exposure is a characteristic of the network depending on the *fcp* used and on the buffer configuration. A packet is referred to as *free* if all the vertices in its route have at least one empty buffer. An SF-network is said to be *greedy solvable* if

(i) it contains no packets; or
(ii) there exists at least one free packet p and after its removal the network is again greedy solvable.

In the literature, many efficient flow control procedures for preventing deadlock states in SF-networks [2–6,10,11,13–15] have been proposed. Among others, Toueg and Steiglitz [14] considered the following deadlock-exposure problems:

(P1): *Given an SF-network and a set of routes in G, is the network exposed to deadlock?*

(P2): *Given an SF-networks and a set of source-destination pairs in G, is there a corresponding set of routes in G such that the network is not exposed to deadlock?*

In particular, they characterized the complexity of both problems with respect to the *fcp* adopted. In most cases, these problems turned out to be NP-complete or even NP-hard. In this paper, we deal with the following *deadlock-safety* problems:

(P3): *Given an SF-network G and a set of routes in G, is the network deadlock safe (i.e., not bound to deadlock)?*

(P4): *Given an SF-network G and a set of source-destination pairs in G, is there a corresponding set of routes in G such that the network is not bound to deadlock?*

The main results of this paper are concerned with both 1-networks (i.e., networks with only one buffer per vertex) and multibuffered networks. In the first case, we show that problems (P3) and (P4) are NP-complete for general 1-networks and remain NP-complete even for simple classes of underlying graphs (among others, bipartite 1-networks, two-terminal series-parallel [TTSP] 1-networks and therefore planar networks). On the other hand, (P3) and (P4) are shown to be polynomially solvable for treelike 1-networks. In this case, two algorithms for checking whether a treelike 1-network with n vertices, m edges, and p packets is bound to deadlock are proposed. The first has time and space complexity $O(pn)$, whereas the second requires $O(n \log n)$ time and $O(n)$ space. As a consequence, the first behaves favorably in networks with few packets, whereas the second is competitive when at least $\log n$ packets are in the network.

The case of multibuffered networks is even harder to tackle. In fact, in this case, both problems are shown to be NP-complete even when the underlying network is a tree. Because of the complexity of these problems in the general case, we devote Section 3 to the following two simpler problems, which are still meaningful per se.

(P5): *Given an SF-network G and a set of routes in G, is the network greedy solvable?*

(P6): *Given an SF-network G and a set of source-destination pairs in G, is there a corresponding set of routes such that the network is greedy solvable?*

These problems are shown to be polynomially solvable for 1-networks. In particular, we present algorithms for solving on 1-networks (P5) in $O(pn)$ time and space, and (P6) in $O(n + m \log n)$ time and $O(m + n)$ space, where m is the number of edges, n is the number of vertices, and $p \leq n$ is the number of packets in the network. As a side result, we prove that problems (P3), (P4), (P5), and (P6) are completely equivalent in the case of treelike networks.

2. SOME COMPLEXITY RESULTS

This section deals with NP-completeness results. We show that both versions of the deadlock-safety problem are NP-complete in case of 1-networks even for simple classes of graphs, such as bipartite graphs, grid-graphs (and therefore planar graphs), and two terminal series-parallel (TTSP) graphs. Furthermore, for multibuffered networks, the problems are shown to be NP-complete even on trees. The first NP-completeness proof deals with *grid-graphs*.

Definition 1. A graph $G = (V, E)$ is a *grid-graph* if there is a mapping $f\colon V \to Z^2$ such that $(u, v) \in E \Rightarrow \|f(v) - f(u)\| = 1$. ■

Theorem 1. Problem (P3) is NP-complete on grid 1-networks.

Proof. It is clear that (P3) is in NP; we have to guess a sequence of moves for packets and verify that no deadlock will occur while all packets reach their final destinations. To prove completeness, we reduce 3-SAT to our problem. Let $F = F_0 \wedge F_1 \wedge \cdots \wedge F_m$ be an instance of 3-SAT with variables $x_0, x_1, \ldots, x_n$, where $F_i = (l_{i1} \vee l_{i2} \vee l_{i3})$. We start by associating a subgraph with each clause F_i and to each pair $x_k, \bar{x}_k$. We then show how to embed these subgraphs in a grid-graph; packets and their routes will be specified after the embedding. With each clause F_i, we associate the following graph $G(F_i), 0 \le i \le m$:

$$\begin{aligned} V(F_i) &= \{u_i, a_{ij}, b_{ij}\colon j = 1, 2, 3\} \\ E(F_i) &= \{(a_{ij}, a_{i,j+1}), (b_{ij}, b_{i,j+1})\colon j = 1, 2\} \cup \\ &\quad \{(a_{ij}, b_{ij})\colon j = 1, 2, 3\} \cup \\ &\quad \{(b_{i2}, u_i)\}, \end{aligned}$$

while we associate the graph $G(X_k), 0 \le k \le n$ with each pair $X_k = (x_k, \bar{x}_k)$:

$$\begin{aligned} V(X_k) &= \{x_k, v_k, \bar{x}_k\} \\ E(X_k) &= \{(x_k, v_k), (v_k, \bar{x}_k)\}. \end{aligned}$$

In order to embed the above subgraphs, it is convenient to consider the infinite grid Z^2; the resulting network will occupy only a polynomially large area, thus ensuring the correctness of the transformation. The construction is illustrated in Figures 1 and 2.

We start with the $G(F_i)$'s, embedded as follows:

- a_{ij} has coordinates $(3i + j, 5), 0 \le i \le m, 1 \le j \le 3$;
- b_{ij} has coordinates $(3i + j, 4), 0 \le i \le m, 1 \le j \le 3$;
- u_i has coordinates $(3i + 2, 3), 0 \le i \le m$.

We also give special names to other points in the grid; this will be useful while specifying the route of each packet. Such points are $(3i + 2, 2), 0 \le i \le m$; they will be called bu_i (i.e., "below u_i"). Graph $G(X_k)$ is embedded in the grid as follows:

- x_k has coordinates $(3k + 1, 1), 0 \le k \le n$;
- v_k has coordinates $(3k + 2, 1), 0 \le k \le n$;
- $\bar{x}_k$ has coordinates $(3k + 3, 1), 0 \le k \le n$.

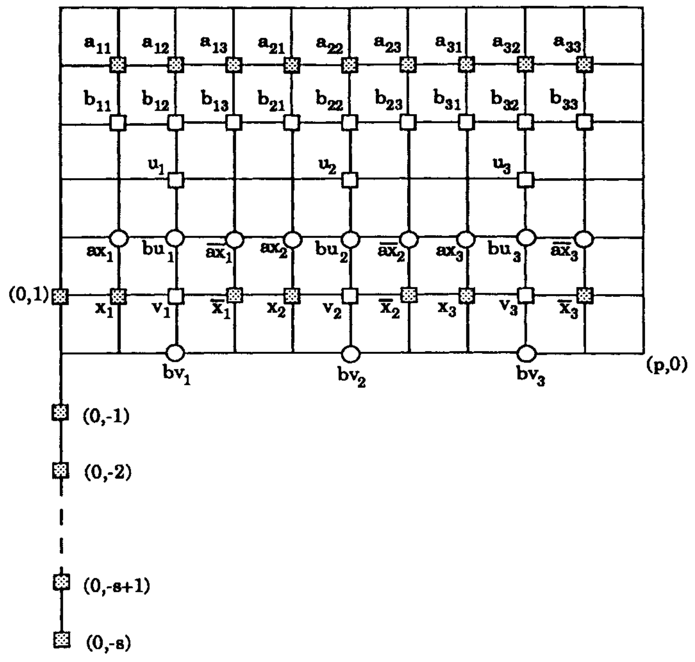


FIG. 1 The gridlike network of Theorem 1. Any intersection of two lines is a vertex, the highlighted vertices are those specified in the construction, and the filled-highlighted ones contain a packet.

Moreover, we introduce the following names:

- the points $(3k + 2, 0)$ will be called bv_k ("below v_k");
- the points $(3k + 1, 2)$ will be called ax_k ("above x_k");
- the points $(3k + 3, 2)$ will be called $\overline{ax}_k$ ("above $\bar{x}_k$).

Let $p = \max\{3m + 1, 3n + 1\} + 1$. Notice that the subgraphs are contained in the rectangle

$$RECTANGLE = \{(0,0), (p,0), (p,6), (0,6)\} ,$$

which is of polynomially large area.

Let us now introduce packets. Even if the network has not been completely defined, its definition will be clear at the end of the construction. The first packets we put on the network are those in the set $T = \{w_1, \ldots, w_s\}$, where s is an integer to be specified later. The initial position of packet w_i is the point $(0, -i)$, $1 \leq i \leq s$. The route $R(w_i)$ of the w_i's is a long tour around the RECTANGLE. The first part of the tour is the path

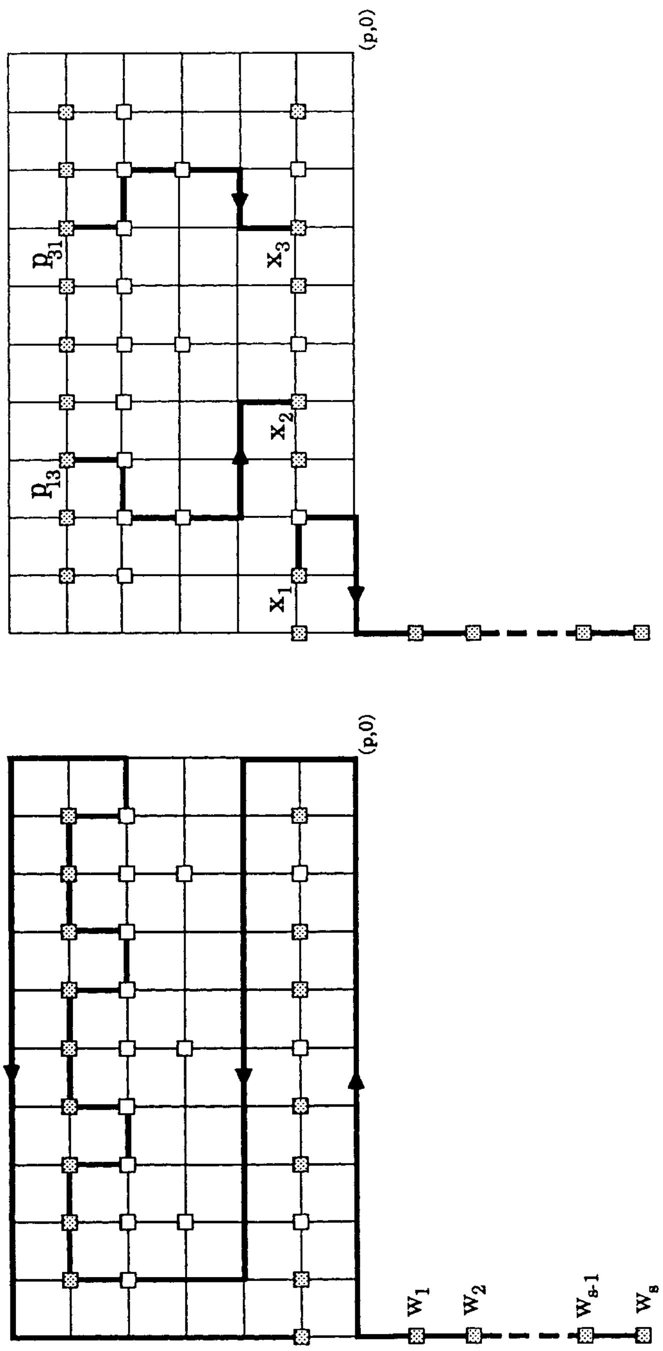


FIG. 2 The routes of packets w_i's (on the left) and the routes of p_{13}, p_{31}, and x_1 (on the right). The network is associated with a boolean formula where $p_{13} = x_2$ and $p_{31} = x_3$.

$$R(w_i)' = (0, -i), (0, 0), (p, 0), (p, 2), (1, 4) ,$$

where only the turning points are specified. From $(1, 4)$ on, packets in T traverse clause subgraphs along the path

$$\begin{aligned} R(w_i)'' = {} & (1, 4), (1, 5), (3, 5), (3, 4), \ldots, \\ & (3i + 1, 4), (3i + 1, 5), (3i + 3, 5), (3i + 3, 4), \ldots, \\ & (3m + 1, 4), (3m + 1,5), (3m + 3, 5), (3m + 3, 4) . \end{aligned}$$

Notice that the w_i's traverse all the vertices of a subgraph $V(F_i)$ except b_{i2} and u_i. The tour is finally completed by

$$R(w_i)''' = (3m + 3, 4), (p, 4), (p, 6), (0, 6), (0, 1) ,$$

where $(0, 1)$ is the final destination of all packets in T. Let s denote the number of points along any path $R(w_i) = R(w_i)' \cup R(w_i)'' \cup R(w_i)'''$ between the two points $(1, 0)$ and $(0, 2)$, extremes included. A packet l (the "locker") with route

$$R(l) = (0, 1), (0, -s)$$

is placed in $(0, 1)$. Note that l cannot reach its final destination unless all the w_i's are out of the y-axis. We end by placing $3m$ packets p_{ij} $(0 \le i \le m, 1 \le j \le 3)$ plus n pairs of packets $q_k, \bar{q}_k$ $(0 \le k \le n)$. The initial positions of a p_{ij} is a_{ij}, whereas the initial positions of q_k and $\bar{q}_k$ are, respectively, x_k and $\bar{x}_k$. Their routes are

- if $l_{i1} = x_k$, then $R(p_{i1}) = a_{i1}, b_{i1}, b_{i2}, bu_i, ax_k, x_k$; otherwise $R(p_{i1}) = a_{i1}, b_{i1}, b_{i2}, bu_i, \overline{ax}_k, \bar{x}_k$;
- if $l_{i2} = x_k$, then $R(p_{i2}) = a_{i2}, bu_i, ax_k, x_k$; otherwise $R(p_{i2}) = a_{i2}, bu_i, \overline{ax}_k, \bar{x}_k$;
- if $l_{i3} = x_k$, then $R(p_{i3}) = a_{i3}, b_{i3}, b_{i2}, bu_i, ax_k, x_k$; otherwise $R(p_{i3}) = a_{i3}, b_{i3}, b_{i2}, bu_i, \overline{ax}_k, \bar{x}_k$;
- $R(q_k) = x_k, v_k, bv_k, (0, 0), (0, -s)$;
- $R(\bar{q}_k) = \bar{x}_k, v_k, bv_k, (0, 0), (0, -s)$.

This completes the construction of the instance of (P3). The underlying network is a grid-graph, and its size is polynomial in $|F|$, since everything takes place in RECTANGLE and the segment $(0, 0), (0, -s)$. We now have to prove that the formula F is satisfiable if and only if the above instance of (P3) is deadlock-safe.

Suppose $F \in$ 3-SAT has a satisfying assignment τ; we show how to deliver all packets in the network. If $\tau(x_k) =$ TRUE $[\tau(x_k) =$ FALSE$]$, we place q_k $[\bar{q}_k]$ in v_k. By construction of the network, and since F is satisfied by τ, we can deliver at least one packet p_{ij} for each subgraph $G(F_i)$. The remaining p_{ij}'s are then placed in the two "parking spots" b_{i2}, u_i. Now we let the w_i's go out of the y-axis and occupy all the vertices of their routes from $(1, 0)$ to $(0, 2)$. This is possible because either the p_{ij}'s are delivered or they occupy the parking spots. But now we can deliver the locker l, which, in turn, makes it possible to deliver all the w_i's. It is now clear that we can also deliver the remaining packets.

Suppose now that the network is deadlock-safe; we derive a satisfying assignment for F. Routes and network topology are such that in order to deliver the locker l we must get the w_i's out of the y-axis. By the choice of s and since l occupies the final destination of the w_i's, the only places where the p_{ij}'s can stay are either their final destinations (either x_k or $\bar{x}_k$) or the parking spots b_{i2}, u_i. Since each $G(F_i)$ provides two parking spots only, we must deliver at least one p_{ij} for each $G(F_i)$. In turn, this implies that for each pair X_k we have to move one of the two packets into v_k. This defines a satisfying assignment for F. ■

An immediate consequence of this theorem is that (P3) is NP-complete for planar graphs.

The next result is the NP-completeness of (P3) on 1-networks isomorphic to *Two-terminal Series Parallel graphs* (in short, TTSP graphs), which are defined as follows [9]:

(i) K_2 is a TTSP graph, in which each vertex is a *terminal*;
if G_1 and G_2 are TTSP graphs with terminals s', t' and s'', t'', then
(ii) the series of G_1 and G_2 obtained by identifying t' with s'' (the new terminals are s' and t''), and
(iii) the parallel of G_1 and G_2 obtained by identifying s' with s'' and t' with t'' (the new terminals are s' and t')

are TTSP graphs.

Theorem 2. Problem (P3) is NP-complete on TTSP 1-networks.

Proof. The proof is similar to that given for the previous theorem. Membership in NP follows from the same argument used in Theorem 1. As before, we reduce 3-SAT to our problem by using subgraphs for truth-assignment and clauses; we also have a "train" of packets ensuring that the allowed moves in the network mirror the structure of a 3-SAT formula. The crucial difference from the previous construction is that now, when the train of packets enters a clause, it cannot exit it. This is due to the particular topology of TTSP graphs. The construction is illustrated in Figures 3 and 4.

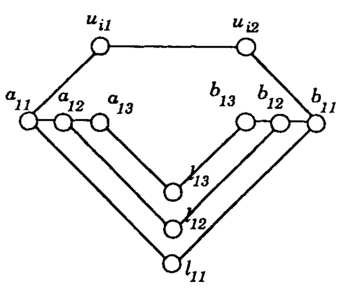
Let the formula be $F = F_1 \wedge \cdots \wedge F_m$ with variables $x_1, \ldots, x_n$. The subgraph $G(F_i)$ for a clause F_i is as follows:

$$V(F_i) = \{l_{ij}, a_{ij}, b_{ij}: j = 1, 2, 3\} \cup \{u_{i1}, u_{i2}\}$$

$$E(F_i) = \{(l_{ij}, a_{ij}), (l_{ij}, b_{ij}): j = 1, 2, 3\} \cup$$

$$\cup \{(a_{ij}, a_{i,j+1}), (b_{ij}, b_{i,j+1}): j = 1, 2\} \cup$$

$$\cup \{(a_{i1}, u_{i1}), (u_{i1}, u_{i2}), (u_{i2}, b_{i1})\},$$



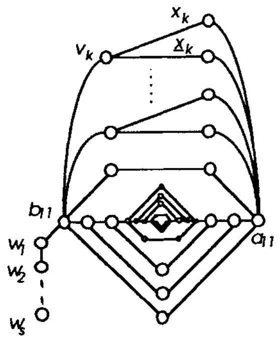


FIG. 3. A subgraph corresponding to a clause and an overall view of the complete construction.

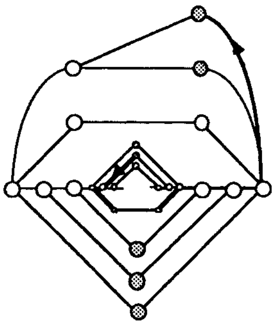
where $1 \leq i \leq m$. The truth-assigner graphs $G(X_k)$ are still defined as in the previous theorem, namely,

$$V(X_k) = \{x_k, \bar{x}_k, v_k\}$$
$$E(X_k) = \{(x_k, v_k), (\bar{x}_k, v_k)\}$$

for $1 \leq k \leq n$. We then have an s-chain $G(C)$

$$V(C) = \{w_1, \ldots, w_s\}$$
$$E(C) = \{(w_1, w_2), \ldots, (w_{s-1}, w_s)\},$$

where $s \geq 9m$. To obtain the SF-network G associated with F, we must connect the graphs $G(F_i)$, $G(X_k)$, and $G(C)$ as follows:



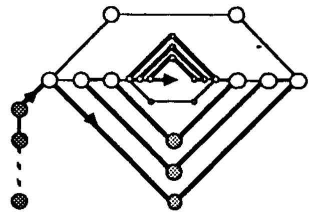


FIG. 4. The routes of a p_{ij} (left) and of the w_i's (right).

$$V(G) = \left(\bigcup_{i=1}^{m} V(F_i)\right) \cup \left(\bigcup_{k=1}^{n} V(X_k)\right) \cup V(C)$$

$$\begin{aligned} E(G) = {} & \left(\bigcup_{i=1}^{m} E(F_i)\right) \cup \left(\bigcup_{k=1}^{n} E(X_k)\right) \cup E(C) \cup \\ & \cup \{(a_{11}, x_k), (a_{11}, \bar{x}_k) : k = 1, \ldots, n\} \cup \\ & \cup \{(v_k, b_{11}) : k = 1, \ldots, n\} \cup \\ & \cup \{(w_1, b_{11}), (b_{13}, b_{21}), \ldots, (b_{m-1,3}, b_{m1}), \\ & \quad (a_{13}, a_{21}), \ldots, (_{m-1,3}, a_{m1})\} . \end{aligned}$$

Notice that the $G(F_i)$'s are nested as Chinese-boxes and that G turns out to be a TTSP graph. We now consider four different types of packets with associated routes:

1. $R_p = \{l_{ij}, b_{ij}, b_{i,j-1}, \ldots, b_{i1}, u_{i2}, u_{i1}, a_{i1}, a_{i-1,3}, a_{i-1,2}, \ldots, a_{11}, d\}$ with $d = x_k$ if $l_{ij} = x_k$, and $d = \bar{x}_k$ if $l_{ij} = \bar{x}_k$ for $p = p_{ij}$;
2. $R_p = \{x_k, v_k, b_{11}, w_1, \ldots, w_s\}$ for $p = q_k$;
3. $R_p = \{\bar{x}_k, v_k, b_{11}, w_1, \ldots, w_s\}$ for $p = \bar{q}_k$;
4. $R_p = \{w_h, \ldots, w_1, b_{11}, l_{11}, a_{11}, a_{12}, l_{12}, b_{12}, b_{13}, a_{13}, a_{21}, l_{21}, b_{21}, \ldots, b_{m3}, l_{m3}, a_{m3}\}$ for $p = r_h$.

Notice that the routes of the packets r_h form a sort of spiral across the edges of the $G(F_i)$'s. Furthermore, if a subgraph $G(F_i)$ is entered by the spiral in a vertex a_{i1}, it is then exited in b_{i3}. Vice versa, if it is entered in b_{i1}, then it is exited in a_{i3}. The remainder of the proof is similar to that given for the previous theorem, and, hence, it has been omitted. ■

As a corollary, the following result can be derived:

Corollary 1. Problem (P3) is NP-complete on bipartite 1-networks.

Proof. The network G constructed to prove Theorem 2 can be modified by duplicating the vertices l_{ij} into l'_{ij} and l''_{ij}, by introducing a corresponding pair of packets for each of the p_{ij}'s and by adding two new places to the "parking areas" u_{i1}, u_{i2}. Consistently, a literal $l_{ij} = x_k(\bar{x}_k)$ is satisfied if and only if both the associated packets p'_{ij} and p''_{ij} are homed at x_k $(\bar{x}_k)$. Also, set $p = 12m$. The resulting graph, shown in Figure 5, is 2-colorable and, hence, bipartite. ■

For multibuffered networks, the following theorem can be proved:

Theorem 3. Problems (P3) and (P4) are NP-complete on treelike multibuffered networks.

Proof. We show that problem (P4) is NP-complete. The NP-completeness

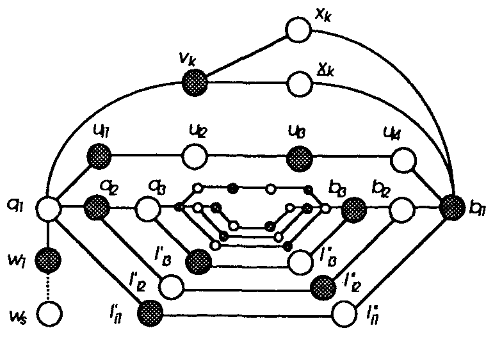


FIG. 5. (P3) is NP-complete on bipartite 1-networks.

of (P3) follows immediately by observing that, on treelike networks, a source-destination pair corresponds to only one route.

NP membership follows from the same argument used in the previous theorems. Again, we reduce 3-SAT by associating a particular network T with a CNF formula F, in such a way that T is deadlock-safe if and only if F is satisfiable, and verify that T is a tree. The topology of T is shown in Figure 6. We have

- a central vertex u (the root) with one buffer, which is linked to
- m 3-buffered vertices $f_1, \ldots, f_m$, one for each clause F_i, plus
- n 2-buffered vertices $v_1, \ldots, v_n$, one for each variable occurring in F, plus
- an $(m + n)$-chain consisting of $(m + n)$ 1-buffered vertices $w_1, \ldots, w_{m+n}$.

Finally, each 2-buffered vertex v_k is linked to a pair $\{x_k, \bar{x}_k\}$ of 1-buffered vertices, for $1 \leq k \leq n$. At time t_k, four different kinds of packets are in the network (corresponding to the shaded vertices in Fig. 6), for a total of $4m + 3n$

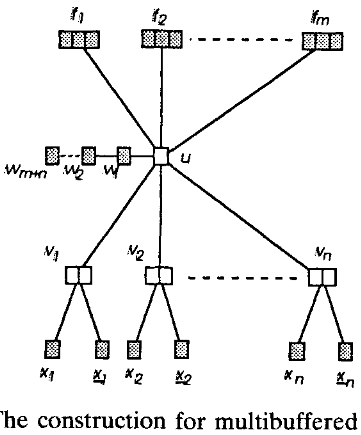


FIG. 6. The construction for multibuffered networks.

packets. A function σ_k defines the state of the network as follows:

- $\sigma_k(p_{ij})$ = the j-th buffer of f_i
- $\sigma_k(r_i) = w_i$
- $\sigma_k(q_k) = x_k$
- $\sigma_k(\bar{q}_k) = \bar{x}_k$.

The following source-destination pairs are associated with each packet:

- $\langle f_i, x_k \rangle$ for packets p_{ij} if $l_{ij} = x_k$
- $\langle f_i, \bar{x}_k \rangle$ for packets p_{ij} if $l_{ij} = \bar{x}_k$
- $\langle w_i, f_i \rangle$ for packets r_i, $i \leq m$ and $\langle w_i, v_{i-m} \rangle$ for packets r_i, $i > m$
- $\langle x_k, w_{m+n} \rangle$ for packets q_k
- $\langle \bar{x}_k, w_{m+n} \rangle$ for packets $\bar{q}_k$.

Suppose F satisfiable. We move q_k [$\bar{q}_k$] into one buffer of v_k if and only if x_k = TRUE [x_k = FALSE] in the satisfying assignment for F. Correspondingly, it is possible to free at least one buffer for each of the f_i's. It is now easy to deliver the remaining packets.

Suppose now that T is deadlock-safe. By construction, in order to deliver all the w_i's, at least one p_{ij} for each f_i must have been delivered. In turn, this implies that it was possible to place for each pair $q_k, \bar{q}_k$ exactly one of these two packets in one of the two buffers of each v_k. This clearly defines a satisfying assignment for F. ■

The following corollary can be derived from Theorem 3.

Corollary 2. Problems (P4) is NP-complete on planar bipartite 1-networks

Proof. The construction is shown in Fig. 7. The graph is clearly planar and

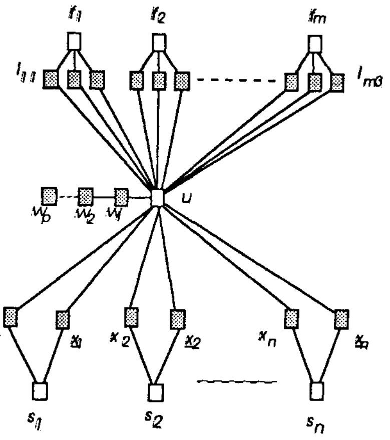


FIG. 7. (P4) is NP-complete on planar bipartite 1-networks.

bipartite. A little technicality is needed; we assume that each variable (with both positive and complemented literals) appears at least twice in the clauses, that is, for each $x_k(\bar{x}_k)$, $1 \leq k \leq n$ there exist p_{ij} and $p_{ij'}$ such that $p_{ij} = p_{ij'} = x_k(\bar{x}_k)$. This variation of 3-SAT is easily seen to be NP-complete.

In the network, we have $p = 2m$ packets w_i (m is the number of clauses in the formula); vertex f_i is the final destination of w_{2i} and w_{2i+1}. As in the previous constructions, packet p_{ij} has destination $x_k(\bar{x}_k)$ if $l_{ij} = x_k(\bar{x}_k)$ in the formula and packet $q_k(\bar{q}_k)$ [initially in $x_k(\bar{x}_k)$] has w_p as final destination.

If the formula is satisfiable, it is clear that the network in Fig. 7 is deadlock-safe.

On the other hand, suppose that the network is deadlock-safe. Because of our choice $p = 2m$, in order to deliver the w_i's it is necessary that for each j at least one packet p_{ij} is out of the subgraph $V(F_i)$ (we do not want p_{ij} to stay in f_i), that is, p_{ij} can be in either x_k or $\bar{x}_k$ and use s_k to reach its final destination. The situation seems symmetric; it seems possible to place either q_k or $\bar{q}_k$ in the "parking spot" s_k. This symmetry would imply that the transformation is not correct. The situation is indeed asymmetric because of the restriction we impose on the formula, namely, that each variable appears at least twice. This implies that for each vertex $x_k(\bar{x}_k)$ there are at least two p_{ij}'s with that final destination. Since routes must be acyclic, a packet p_{ij} can be moved only if its final destination is free. This defines a satisfying truth-assignment for the formula. ■

3. GREEDY-SOLVABILITY OF 1-NETWORKS

In this section, we describe polynomial time algorithms for deciding whether a 1-network is greedy-solvable either in the case of fixed routing or in the case where the routes are not fixed *a priori*. Before stating our results, we need some preliminary definitions. Suppose we are given a network with n vertices, one buffer per vertex, and p packets, each with a given route. If a packet p_i is in the buffer of a vertex belonging to the route of a packet p_j, we say that p_i *blocks* p_j. A *cycle of packets* is a sequence of packets $\{p_0, p_1, \ldots, p_k\}$ such that $p_0 = p_k$ and p_i blocks p_{i-1}, $1 \leq i \leq k$. A cycle of packets is said to be *simple* if it does not contain other cycles of packets. With each packet p_i in a cycle of packets $\{p_0, p_1, \ldots, p_k\}$, we associate an integer, called *rank*(p_i), given by the number of edges in the route of p_i from the source of p_i to the source of $p_{(i+1)}$. The *rank* of a cycle of packets is the sum of the ranks of all the packets in the cycle. The following fact gives necessary and sufficient conditions for a given 1-network to be greedy-solvable.

Fact 1. A 1-network G with a fixed routing procedure is greedy-solvable if and only if it does not contain cycles of packets.

As a straightforward consequence of this fact, one could check whether a 1-network is greedy-solvable by simply testing the presence of a cycle of packets. In order to make this test easier, we associate with each 1-network in a given state a directed graph G_p, referred to as the *packet graph* and defined as follows.

For each packet $p_k, 1 \leq k \leq p$, in the 1-network, there is a chain c_k of length l_k in G_p, where l_k is the length of the route of p_k. These p disjoint chains are completed by introducing n nodes $n_i, 1 \leq i \leq n$, corresponding to the vertices v_i of the 1-network and

(i) one arc from the node n_i to the first node of the chain c_j, if the packet p_j has source in v_i;
(ii) one arc from a node in c_k (corresponding to a vertex v_h in the 1-network) to n_h, if v_h belongs to the route of p_k and is also the source of another packet.

In Figure 8 it is shown a packet graph for a simple network. As a consequence of this definition, there is a one-to-one correspondence between cycles of packets in a 1-network and cycles in the associated packet graph. Hence, one could check whether a 1-network is greedy-solvable by building the corresponding packet graph G_p and then by finding out whether it contains a cycle. G_p can be constructed in $O(pn)$ time and requires $O(pn)$ space, where n is the number of vertices in the network and p is the number of packets in the 1-network. Since the presence of a cycle in G_p can be detected in $O(pn)$ time (see, e.g., [12]), we have the following theorem.

Theorem 4. It is possible to check whether a 1-network with n vertices and p packets (whose routes are fixed *a priori*) is greedy-solvable in $O(pn)$ time and $O(pn)$ space.

This algorithm cannot be applied if the routes are not fixed *a priori*, since no packet graph can be defined. In such a case, the greedy solvability of a network may be checked as follows. Before describing the algorithm, we need some preliminary terminology. For each vertex v in the network $G = (V,E)$, let us introduce the following sets

$$N(v) = \{u \in V | (u, v) \in E\}$$

$$S(v) = \{u \in V \,|\, \text{there exists a packet with source destination pair } (u, v)\}.$$

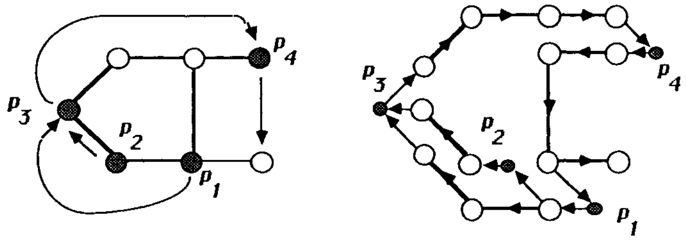


FIG. 8. A packet graph.

In other words, $N(v)$ contains all the neighbors of v in the network, whereas $S(v)$ contains all the vertices that are sources of packets whose destination is v. Furthermore, if a vertex v is the source of any packet p_i, we denote by $d(v)$ the destination of p_i. The following lemma, whose simple proof has been omitted, characterizes the overall size of the sets $N(v)$ and $S(v)$.

Lemma 1. For any network $G = (V,E)$ with n vertices, m edges, and p packets,

$$\sum_{v \in V} |N(v)| = 2m$$

$$\sum_{v \in V} |S(v)| = p\,.$$

The algorithm is based on the idea of removing from the network the vertices with a nonempty buffer. The remaining vertices are organized by maintaining the connected components of the resulting network. By doing so, it follows that if $d(v)$ and any vertex in $N(v)$ are in the same connected component, then there is at least one free route from v to $d(v)$ and, hence, the packet in v may be delivered to its destination. For this reason, for any vertex v with a nonempty buffer, we say that $d(v)$ is a *mate* of any vertex in $N(v)$ and vice versa any vertex in $N(v)$ is a *mate* of $d(v)$. Once the packet with source in v has been delivered, the buffer at v is now empty and thus the new connected components must be recomputed by taking into account the vertex v and all the edges incident on it. We now give a high level description of the algorithm.

Algorithm Solvability

1. Compute $N(v), S(v), d(v)$ for each vertex v in the network.
2. Remove from the network the vertices with a nonempty buffer.
3. Check whether there are two mate vertices in the same connected component (which allow to deliver a packet p_i). If not, go to step 5.
4. Deliver the packet p_i. Insert the source of p_i in the graph together with its incident edges. Go to Step 3.
5. If all packets in the original network have been delivered, then return *yes*. Otherwise, return *no*.

A naive implementation of the algorithm Solvability is easily seen to require $O((m+n)p^2)$ time and $O(m+n)$ space. A more sophisticated and efficient version of this algorithm is based on a different implementation of Steps 3 and 4. In this approach, the connected components are maintained on-line as sets in which unions between sets (when a vertex is reinserted into the graph) and find operations (for testing whether two mates are in the same connected component) are allowed. For this purpose, we make use of a data structure for the set-union problem (see [1]), in which different sets are maintained as rooted trees of height one. In such trees, the leaves are the vertices of the connected component, whereas the root contain a label that identifies the set, referred to as the *name* for the set. Each arc of these trees is directed from a leaf to the root. When a *find* (x) has to be performed, the leaf corresponding to x is first accessed. Then, the pointer to the root is followed and the name of the component contained in the root is returned. With this technique, one is able to check in constant time whether two vertices belong to the same component. The reinsertion of a vertex whose packet has been delivered to its destination entails the merging of two or more different components. When performing a union between sets corresponding to connected components, we make all the leaves in the smaller set children of the root of the larger one, arbitrarily breaking a tie. Also, for each moved leaf v we check whether

(i) for each vertex x in $N(v)$, $d(x)$ is in the new component that v has been moved to;
(ii) for each vertex y in $S(v)$, there is any vertex z in $N(y)$ in the new component that v has been moved to.

The reason for this test is that in case (i) moving v has created a new possibility for a packet in $x \in N(v)$ to be delivered, whereas in case (ii), a packet with destination v may now be delivered. This approach improves the efficiency of the algorithm Solvability.

Theorem 5. It is possible to check whether a 1-network with m edges, n vertices and $p \leq n$ packets (whose routes are not fixed *a priori*) is greedy-solvable in $O(n + m \log n)$ time and $O(n + m)$ space.

Proof. Since the connected components of a graph can be computed in $O(m + n)$ [12], this is also the overall complexity of Steps 1, 2, and 5. If Steps 3 and 4 are implemented as above, at most n unions between disjoint sets (the connected components) are required. Each time that a vertex v is moved from a component to another, at most

$$|N(v)| + \sum_{y \in S(v)} |N(y)|$$

find operations are performed. Furthermore, since a find can be accomplished in constant time, this is also the cost charged to a vertex each time is moved. Each time a vertex is moved from a set, it is moved into a set which is at least twice as large as before. Thus, no vertex v can be moved more than $\log n$ times during the n unions. The total cost charged to v is

$$(|N(v)| + \sum_{Y \in S(v)} |N(y)|) \log n .$$

The total cost of Steps 3 and 4 is obtained by summing up the costs charged to the vertices

$$\sum_{v\in V}\Big(|N(v)| + \sum_{y\in S(v)} |N(y)|\Big)\log n\,.$$

Considering that by Lemma 1

$$\sum_{v\in V}|N(v)| = 2m$$

$$\sum_{v\in V}\sum_{y\in S(v)}|N(y)| \le \sum_{y\in V}|N(y)| = 2m\,,$$

the time complexity of the algorithm is $O(n + m \log n)$. The space required is $O(m + n)$. ■

4. THE DEADLOCK SAFETY PROBLEM IN TREE-LIKE NETWORKS

This section deals with two efficient algorithms for predicting deadlock in treelike networks. Suppose we are given a treelike network T with n vertices, one buffer per vertex and $p \le n$ packets, each with a given source-destination pair. Notice that in case of trees every source-destination pair uniquely denotes a path in T and henceforth a route. As a consequence, problems (P3) and (P5) trivially coincide, respectively, with (P4) and (P6) in case of treelike networks. Before giving a theorem that characterizes bound to deadlock treelike networks, we need the following preliminary result.

Lemma 2. Let $T = (V,E)$ be a treelike 1-network containing a cycle of packets $C = \{p_0, p_1, \ldots, p_s\}$. For each edge (x, y) in E in the cycle C, there are at least two packets in C whose routes traverse (x, y) in opposite directions.

Proof. Let (x, y) be any edge in C. This implies that there exists at least one packet that needs to traverse (x, y). Denote it by p_i, $1 \le i \le s$, and assume without loss of generality that it must traverse (x, y) from x to y. If the edge (x, y) is removed from the tree, this would split the original tree into two resulting trees: T_x (containing x) and T_y (containing y). Since routes are acyclic, the source of p_i is in T_x while its destination is in T_y. By definition of cycle of packets, there must be a packet in C which has to traverse the edge (x, y) from y to x, i.e., in the opposite direction of p_i. ■

We now prove the following theorem.

Theorem 6. A treelike 1-network T is bound to deadlock in a given state if and only if it admits at least one cycle of packets.

Proof. If-part. We proceed by induction on the rank of the cycle of packets.

States with cycles of packets of rank 1 are bound to deadlock since the two packets in the cycle are deadlocked. Suppose now that all the states with cycles of packets of rank at most $l \geq 1$ cause T to be bound to deadlock. Let $C = \{p_0, p_1, \ldots, p_s\}$ with $p_0 = p_s$ be any cycle of packets of rank at most $l \geq 1$ contained in T in a state σ_j. We restrict ourselves to a state σ_h that contains only the packets in the cycle C, and we prove that T is bound to deadlock in σ_h. Since σ_h is preceded by σ_j, if the network is bound to deadlock in σ_h, then it will be bound to deadlock also in σ_j. We distinguish two cases depending on whether C is simple or not.

If C is not simple, then it contains a simple cycle of ranks at most l. For the inductive hypothesis, T will be bound to deadlock in σ_h. If C is simple, then in state σ_k there must exist at least one packet p_i $(1 \leq i \leq s)$ that is allowed to move; otherwise C would contain a cycle of rank 1 and henceforth it would not be simple. Let x and y be, respectively, the second and the third vertex in the route of p_i and let us move p_i to x. The edge (x, y) is in C, and thus by Lemma 2, there exists a packet p_k $(1 \leq k \leq s$ and $k \neq i)$ whose route contains the edge (x, y) traversed from y to x. Since p_k is blocked by p_i in the new situation, a new cycle $\{p_i, p_{i+1}, \ldots, p_k, p_i\}$ $(1 \leq k \leq s$ and $k \neq i)$ of rank at most l is created in σ_{h+1} and therefore T is bound to deadlock in σ_{h+1} for the inductive hypothesis. Since this argument can be repeated for each allowed move in σ_h, T is bound to deadlock in σ_h. As previously noticed, this implies that T is also bound to deadlock in σ_j.

Only-if-part. Assume by contradiction that the network is bound to deadlock but there are no cycles of packets. In this case, at least one packet must be free. If this packet is delivered to its destination and removed from the network, there will not be cycles of packets again. By repeating this argument, we find a sequence of passing and consumption moves that allows to deliver all the packets in the network, clearly contradicting the hypothesis that the network is bound to deadlock. ■

The following corollary is an immediate consequence of both Fact 1 and Theorem 6.

Corollary 3. A treelike 1-network T is bound to deadlock in a given state if and only if it is not greedy-solvable.

Notice that, as a first consequence of Corollary 3, problems (P3), (P4), (P5), and (P6) are completely equivalent in case of treelike networks. Moreover, the algorithms proposed in the previous section for detecting the greedy solvability of a given 1-network could be used for deciding whether a tree-like network is bound to deadlock. This remark plus Corollary 3, Theorem 4, and Theorem 5 gives rise to the following theorems.

Theorem 7. It is possible to check whether a given treelike 1-network with n vertices and $p \leq n$ packets is bound to deadlock in $O(pn)$ time and space.

Theorem 8. It is possible to check whether a given treelike 1-network with n vertices is bound to deadlock in $O(n \log n)$ time and $O(n)$ space.

5. CONCLUSIONS

In this paper, we have studied the problem of predicting deadlocks in store-and-forward networks. We defined two versions of this problem, depending on whether or not the routes to be followed by the packets are fixed.

In the general case of multibuffered networks, both versions of the problem were shown to be NP-complete even on trees.

In case of networks with only one buffer per vertex, we showed that both versions of the problem are NP-complete even for simple graphs (among others bipartite graphs, TTSP graphs, and therefore planar graphs), while polynomially solvable for treelike networks. In particular, two algorithms were given for solving this problem on treelike networks with n vertices and p packets. The former has an $O(pn)$ time and space complexity, whereas the latter runs in $O(n \log n)$ time and requires $O(n)$ space.

In the case of deadlock safe treelike networks, the second algorithm produces as a side effect a feasible sequence of moves that allows all the packets to be delivered. This achieved sequence is serial, as specified in the definition of greedy solvability. It seems worthy of further investigation to study the problem of achieving a sequence of moves that is optimal with some respect

Furthermore, from the complexity analysis carried out in the paper, it seems that the hardness of these problems is related to cycles in the underlying network. It seems promising to consider classes of graphs that enjoy nice properties with respect to cycles, in order to extend the class of 1-networks for which these problems can be solved in polynomial time.

ACKNOWLEDGEMENTS

We would like to thank Daniel P. Bovet for suggesting this problem, Paolo Nobili for useful discussions, and the anonymous referees for their helpful comments.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA (1974).

[2] J. Blazewicz, D. P. Bovet, and G. Gambosi, Deadlock-resistant flow control procedures for store-and-forward networks. *IEEE Trans. Commun. COM*-32 (1984) 884–887.

[3] J. Blazewicz, J. Brzezinski, and G. Gambosi, Time-stamps approach to store-and-forward deadlock prevention. *IEEE-Trans. Comm. COM*-35 (1987) 490–495.

[4] G. Bongiovanni, and D. P. Bovet, Minimal deadlock-free store-and-forward communication networks. *Networks* **17** (1987) 187–200.

[5] D. P. Bovet, G. Gambosi, and D. A. Menasce, Detection and removal of deadlocks in store-and-forward communications networks. *Performance* 84.

[6] G. Campanile, Il problema dello stallo in reti store-and-forward, Tesi di laurea in matematica, Dipartimento di Matematica, Università di Roma "La Sapienza," Rome, Italy (in Italian) (1984).

[7] R. C. Holt, Some deadlock properties of computer systems, *ACM Comp. Surv.* **4** (1972) 178–196.
[8] L. Kleinrock, *Queueing Systems, Vol. II: Computer applications*, John Wiley & Sons, New York (1976).
[9] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York (1976).
[10] P. M. Merlin and P. J. Schweitzer, Deadlock avoidance in store-and-forward networks I: Store-and-forward deadlock. *IEEE Trans. Comm. COM*-28 (1980) 345–354.
[11] P. M. Merlin and P. J. Schweitzer, Deadlock avoidance in store-and-forward networks II: Other deadlock types. *IEEE Trans. Comm. COM*-28 (1980) 355–360.
[12] R. E. Tarjan, Depth-first search and linear graphs algorithms. *SIAM J. Comput.* **2** (1972) 146–160.
[13] S. Toueg, Deadlock- and livelock- free packet switching networks. *Proc. ACM Symp. Theory Comput.* (1980) 94–108.
[14] S. Toueg and K. Steiglitz, Some complexity results in the design of deadlock-free packet switching networks. *SIAM J. Comput.* **10** (1981) 702–712.
[15] S. Toueg and J. D. Ullman, Deadlock-free packet switching networks. *Proc. ACM Symp. Theory Comput.* (1979) 89–98.

Received May 1988
Accepted January 1990