ON THE SPECIFICATION AND VERIFICATION OF COMMUNICATION PROTOCOLS

A. Petrenko

April 1980
WP-80-60

ABSTRACT

The purpose of this paper is to consider the most complicated problem related to computer network design, and especially to the so-called "gateways": the definition and estimation of the logical correctness of protocols. While the simple terminal connection of a computer to a computer system necessitates only the emulation of the chosen terminal, the very complicated interconnection of several computer networks requires the definition and implementation of a whole hierarchy of protocols. Naturally, all the protocols of each level must be rigorously specified and carefully verified before being implemented into soft-, firm-, or hardware. In order to achieve this goal, a technique based on a top-down approach, involving stepwise refinement and verification of the protocol actions in various situations, is proposed in this paper. This technique requires the formalism of a special kind of Petri net: the Petri net with enabling predicates.

CONTENTS

ON THE SPECIFICATION AND VERIFICATION OF COMMUNICATION PROTOCOLS

A. Petrenko

1. INTRODUCTION

The formal specification and verification of communication protocols would seem to be one of the most important and difficult tasks when considering the problem of computer network design. The significance of this task is a consequence of the fact that the correctness and proper implementation of protocols determine the working capacity of the network elements and of the network as a whole. The difficulties which arise in connection with the specification and verification of protocols are due to the necessity of demonstrating both the <u>completeness</u> and the <u>consistency</u> of the protocol. The completeness of a protocol means that all the possible situations which could arise under specific conditions have been foreseen; protocol consistency guarantees simple protocol implementation and determinism of the behavior of the network elements. While the last protocol property is ascertained fairly easily using formal languages (as opposed to natural ones), the proof of completeness involves a thorough analysis of numerous situations, which are supposed to occur in the real system, if all of them can be predicted.

The whole spectrum of techniques: state transition languages, programming languages and hybrid models, have already been tested for protocol specification and verification [1]. Those techniques using the total system state transition graphs have weak representative means due to the "state explosion" effect. Programming languages possess the disadvantage that they burden the protocol description with unnecessary details, from the verification point of view, and a protocol specification of this kind differs slightly from the software implementation of a protocol. Therefore the hybrid models seem to have a wider perspective. Using these models, both the connection control phase and the data transfer phase of protocols can be described. The model used in this paper belongs to this model class and is based on Petri nets (PNs). PNs have already been used by the authors of several previous papers [2,3,4,5,6]. These investigations have shown the necessity of modifying the original PN for protocol specification and verification purposes. At the same time, the choice of this modification must not restrict the modeling power of the PN, which has been carefully and extensively studied and which would appear to be very relevant to our goals.

This paper describes the techniques to be employed in protocol specification and verification based on Petri nets with predicates (PNP) and, unlike other papers (for example [7]), a net for the whole system of communicating processes has been constructed without imposing special restrictions on the transition execution rules; this allows new properties of protocols to be established, due to the full employment of all the PN's means.

The proposed approach in this paper is described with the help of a well-known example [8] in order to have a basis for comparison with various other existing techniques.

2. GOALS OF PROTOCOL SPECIFICATION AND VERIFICATION

The main purpose of protocol verification is to demonstrate that it fulfills the designer's intention, i.e., it does give the service required to the user or higher level protocol, and that it is logically correct. When verifying protocols, the unreliability of the data transmission medium (or lower level protocol) must be borne in mind, namely: errors, messages lost, change in the order of messages, and also possible situations arising in the communicating entities such as wrong timeout expiration, arbitrary initiation of entitites, and so on. A rigorous definition of protocol correctness cannot be obtained without having a formal protocol specification; but the initial requirements for this are usually postulated in a verbal description. Therefore, protocol verification comes down to demonstrating that the protocol has (or does not have) certain properties. Of these properties, the following [1] have been considered:

1. freedom from deadlocks;
2. self-synchronization;
3. correct termination;
4. progressiveness;
5. freedom from overflow.

Some of these can be formulated easily if the formal tools are available, but others involve the interpretation of the protocol specification by the designer. For instance, the progressiveness of the protocol means the absence of cyclic behavior, during which no useful activity takes place. However, the usefulness of the activity can only be estimated after an analysis of a particular situation has been made. Consequently, it is the opinion of the author that a rigorous definition of the protocol properties should be made, only with the features of the specification tools and the protocol itself in mind.

The specification tools to be used must describe concurrent processes, because a verified protocol involves the interaction of at least three subsystems, two communicating entities and a transmission medium. These tools then have to permit the representation of the system in a top-down fashion at various levels of abstraction and detail. It is also desirable that these tools be oriented toward further protocol implementation into soft-, firm-, or hardware. The Petri net formalism largely meets these requirements.

3. PROPERTIES OF PROTOCOLS AND PETRI NETS

Petri nets are widely used tools for representing concurrent systems in a top-down fashion at various degrees of interpretation [8].

A Petri net (PN) can be defined as a bipartite directed graph $N = (T,P,A)$, where

$T = \{t_1,t_2,\dots,t_n\}$ is a set of transitions, represented by bars;

$P = \{p_1,p_2,\dots,p_m\}$ is a set of places, represented by circles;

$A \subseteq \{T \times P\} \cup \{P \times T\}$ is a set of directed arcs.

A marking M of a PN is the mapping $M:P \to \{0,1,2,\dots\}$. M assigns tokens to each place in the net. A marking M is represented by a vector M, where $M(p_i)$ represents the number of tokens assigned to p_i.

Let $I(t) = \{p \mid (p,t) \in A\}$ be a set of input places, and $O(t) = \{p \mid (t,p) \in A\}$ be a set of output places of a transition t.

A transition t is said to be activated under a given marking M of $M(p) > 0$ for all $p \in I(t)$. The activated transition t can fire, changing the marking $M \xrightarrow{t} M'$:

$$M'(p) = \begin{cases} M(p) + 1, & \text{if } p \in O(t),\ p \notin I(t) \\ M(p) - 1, & \text{if } p \in I(t),\ p \notin O(t) \\ M(p), & \text{otherwise} \end{cases}$$

In this case we say that M' is reachable from M. In a system of communicating processes: the transitions of PN refer to certain events or actions, such as arrival of commands and messages, and timeout expiration; the places of PN correspond to certain conditions; and a marking refers to a particular control state of the system. Thus, if R(M) is a set of markings which are reachable from M, then it must also be a set of all the control states of the modeled system.

Marked PN, N = (T,P,A,M) is live if, for all $M' \in R(M)$, there exists an activated transition. A live PN of a protocol shows the absence of protocol deadlock. It is convenient to represent R(M) by means of a marking graph G, the arcs of which are labeled by corresponding transitions.

If graph G has a final node, this node represents a final control state of the system. Therefore it is possible to establish the termination property of the protocol. When analyzing the cycles of graph G, we are also able to study other properties of the protocol, such as progressiveness and self-synchronization.

A marked PN is said to be k-bounded if $M'(P) \leq k$ for all $M' \in R(M)$ and all $p \in P$. PN is safe if k = 1. An unbounded PN implies that the corresponding communication system has an infinite number of states. This property of PN is very useful for analyzing the overflow of the protocol.

Let $M_1 \xrightarrow{t_1} M_2$, and $M_1 \xrightarrow{t_2} M_3 (M_2 \neq M_3)$. It is assumed, however, that t_1 is not activated under M_3; neither is t_2 activated under M_2. In this case we refer to the transitions t_1 and t_2 as being in conflict. How to resolve this conflict depends on the interpretation of PN. Since R(M) of PN represents a set of control states of the system, such a conflict must be resolved by means of data (values of context variables) to be used by the protocol. With the transitions t_i, t_j in conflict, we associate enabling predicates P_i and P_j, where $P_i \wedge P_j$ = false; $P_i \vee P_j$ = true. The introduced predicates depend on certain variables, the values of which can be determined by actions associated with the transitions. Applying the firing rule, it follows that a transition t will in this case be activated, under the additional condition that its enabling predicate is true.

Let the transitions in conflict be $t_1, t_2, \ldots, t_k$, and the enabling predicates which resolve this conflict be $P_1, P_2, \ldots P_k$. For completeness and consistency, the following conditions require to be satisfied:

$$P_1 \vee P_2 \vee \cdots \vee P_k = \text{true}$$

$$P_i \wedge P_j = \text{false}, \quad \text{for all } i \neq j \ (1 \leq i, j \leq k).$$

We call the above defined PN the Petri Net with Predicates (PNP).

Clearly, at a high level of abstraction, it is sufficient to use pure PN to construct a net for the modeled system. Using this kind of PN, the set R(M) represents a set of the total states of the system. If conflicts arise at the lower level specification, then a total state will be determined by a control state and by the values of the variables used in the PNP.

The introduction of predicates into the PN does not restrict its power, but on the contrary permits a large class of protocols to be modeled. Using the PNP it is possible to represent, not only the connection control phases, but also the data transfer phase.

An explanation of our approach to the problem using the PNP is given below, based on a well-known simple protocol [8]. This protocol has been used in a number of papers [4,9,10]; therefore it is possible to compare the technique adopted in this paper with previous ones.

4. SPECIFICATION AND VERIFICATION OF THE ALTERNATING BIT PROTOCOL

4.1 The Alternating Bit Protocol

This protocol is a point-to-point protocol which uses a communication medium alternating in both directions. If we consider the data transfer from the Sender subsystem to the Receiver subsystem, the procedure is as follows: the Sender sends a message containing the user's data and the sequence number seq $\in \{0,1\}$. The Receiver, having obtained the message, compares its number with the expected one exp $\in \{0,1\}$, and then sends an acknowledgement ack $\in \{0,1\}$, which is equal to the received message number. The Sender waits for an acknowledgement before the next piece of data is sent. The system recovers from transmission errors detected by means of a redundancy check, and from lost messages by means of a timeout in the Sender. In both cases retransmissions are involved.

The PN (Figure 1) depicts this protocol from the point of view of the service it provides to the users or to the higher level protocol and is less suitable for an analysis of its properties.

4.2 Normal Operation

If we consider a reliable transmission medium with neither losses nor errors, we can obtain a PN describing the whole system (Figure 2). In this net, the transition t_1 represents a message transfer; t_2 - its reception; t_3 corresponds to the following actions of the Receiver: transference of the data to the user, alteration of the value of the variable $\mathrm{exp} := (\mathrm{exp} + 1)_{\mathrm{mod}2}$, and transference of an acknowledgement to the Sender ack = exp; t_4 corresponds to the arrival of the acknowledgement; t_5 corresponds to the change of the sequence number $\mathrm{seq} := (\mathrm{seq} + 1)_{\mathrm{mod}2}$, and also to the reception of the next piece of data to be transmitted from the Sender's user to the Receiver's user.

Let the control state (p_1,p_4), which is the marking (1001000) be the initial state of the system. The variables have the following values: seq = 1, exp = 0. The marking graph G (Figure 3) depicts all the control states reachable, assuming a reliable transmission medium. This graph is a simple loop; every place has no more than one token, therefore this PN is live and safe. The graph of the control states can easily be transformed into the total state graph (Figure 4), which gives a detailed description of the dynamics of the system. However it is sufficient only to have the marking graph of the PN in order to verify the following properties of the system working on this protocol:

-- absence of deadlocks;
-- proper termination (from every control state, the system arrives at (p_1,p_4));
-- absence of undesirable cycles (the only loop goes through the transitions t_5 and t_3);
-- absence of overflow (PN is safe).

It should be borne in mind that the above properties only hold true under a reliable transmission medium and under proper initialization of both entites.

4.3 Error Recovery

Let us now analyze the system assuming that the medium does not lose messages but can distort the message being transmitted. In this case the protocol is adequately described by the PN of Figure 5, and thus differs from the PN of Figure 2 in that it has two transitions t_6,t_7. Their actions correspond to those of the Sender and Receiver when errors in the data or numbers arise. The transition t_6 stands for keeping the value of the variable seq and ignoring the received acknowledgement; t_7 stands for excluding the received message and sending the acknowledgement ack=exp. The transitions t_5 and t_6 are produced under the same condition p_7 (acknowledgement has been received); hence they are in conflict. To resolve this, we associate the enabling predicates $P_6[\text{Error} \vee (\text{ack} \neq \text{seq})]$ with

t_6 and $P_5 = \neg P_6$ with t_5. In the same way, we associate the enabling predicates $P_7[\text{Error} \vee (\text{seq} \neq (\text{exp} + 1)_{\text{mod}2})]$ with t_7, and finally, the predicate $P_3 = \neg P_7$ with t_3.

The introduced transitions (or events) do not change the structure of the control state graph (Figure 6), and therefore the main protocol properties remain even, if transmission errors arise. In fact, undesirable cycles of operation, i.e., those which do not have the sequence of the main transitions t_3 and t_5, can only exist when the following assertion holds true (this should be compared with the invariant in [11]):

$$P_6[\text{Error} \vee (\text{ack} \neq \text{seq})] \vee P_7[\text{Error} \vee (\text{seq} \neq (\text{exp}+1)_{\text{mod } 2})] = \text{true}.$$

From this it follows that undesirable cycles of operation can repeat themselves as long as transmission errors exist. The only way of terminating such a cycle is to establish a maximum number of retransmissions and to notify the user that correct data transmission is impossible.

This protocol is also self-synchronizing. In fact, if the entitites have not been properly initialized, i.e., (p_1, p_2) is still the initial state, but seq = exp, then according to the graph in Figure 6, the system loses only the first transmitted message, and after that, the action of the entitites is synchronized.

4.4 Message Recovery

We now consider the operation of the system, assuming that the transmission medium can lose the messages transmitted through it. The PNP of Figure 7 is a suitable abstraction of the medium. The transition t* represents the message transmission with or without errors, and leads to the condition p_2 (data sent); t corresponds to message lost and has no output places, because the medium is not able to inform anyone of the data lost. These transitions are in conflict and it is impossible to associate any deterministic enabling predicates with them, since the result of the transmission depends on the medium properties and other factors. For analysis purposes, we assume that the event t is possible if P(loss) = true, and that t* is possible if $P^* = \neg P(\text{loss}) =$ true.

If we incorporate the medium models (transitions t_8, t_9) and the timeout model (t_{10}) into the PNP of Figure 5 and, if for simplification of the net, we combine the transitions t_2, t_4 with t*, the resultant PNP will be as shown in Figure 8. In order to resolve the conflict, we introduce the predicates:

$P_8(\text{loss})$, $P_9(\text{loss})$, $P_2 = \neg P_8(\text{loss})$, $P_{10}(T=0)$, $P_4 = \neg P_9(\text{loss}) \wedge$

$\neg P_{10}(T=0) = \neg [P_9(\text{loss}) \vee P_{10}(T=0)]$

We first assume that the timeout value T can be chosen in such a way that the transition t_{10} will only occur after a transmission loss has occurred (transitions t_8, t_9). It should be noted that in [4] this protocol has only been verified with such a constraint; later on we consider the operation of the system without it.

As can be seen from the marking graph of the PNP in figure 9, the transitions t_8 and t_9 lead the system into the control state (p_3, p_4), which would be a deadlock state, if there were no timeout mechanism. The timeout expiration allows the system to return to the initial state; hence the PNP in Figure 8 is safe and live.

From Figures 6 and 9, it follows that the system has additional undesirable cycles (1,2,6) and (1,2,3,4,6) until the following condition is satisfied:

$$[P_8(\text{loss}) \vee P_9(\text{loss})] \wedge P_{10}(T=0) = \text{true}$$

These cycles, just as those mentioned in Section 4.3, cease after transmission improvement, or after the maximum number of retransmissions has been exceeded.

4.5 Operation with Wrong Timeout Expiration

When the timeout value in the Sender is not properly adjusted, or the arrival of the acknowledgement has been delayed, it is possible that the timeout transition of the Sender will occur in state (p_3,p_5) or (p_3,p_4,p_6), i.e., before the expected response of the Receiver actually arrives. For a clear analysis, we assume that the Sender retransmits the message only once, due to timeout expiration of this kind. The proposed technique does enable the modeling of more complicated situations; however this case is of most practical interest. All the possible control states of the system are represented by the graph in Figure 10. It can be deduced from the graph that the PNP in Figure 8 is live; therefore it will not indicate deadlock situations. The system only returns to the initial state (p_1,p_4) if

$$P_8(\text{loss}) \vee P_9(\text{loss}) = \text{true}$$

In the situation under consideration, the probability of message loss in the transmission medium is much higher than that in the normal operation, since after wrong timeout expiration, the system moves definitely into the state 7 (p_2,p_3,p_4,p_6), which characterizes duplex transmission; however, in accordance with the protocol requirements, the entities should use the transmission medium alternatively. If such usage of the medium leads to loss of the message (the transitions t_8 or t_9), then the system returns to the normal cycle of operation. Here we have the rather exceptional case that the loss of the message actually improves the operation of the system.

If after wrong timeout expiration the medium does not lose the transmitted messages, then the operation of the system can be represented by the simpler marking graph without t_8,t_9 (Figure 11). It follows from this graph that, after the event t_{10}, the system can remain for an infinitely long period in the states 6,7,8,...,16, passing through one of the following cycles:

(6,7,8,9,10),(6,7,8,12,10),(6,7,11,12,13),(7,11,12,13,14), (6,7,11,12,13(,(7,8,12,13,14),(6,7,11,12,10),(7,11,12,13,14), (6,7,8,12,13)

The Sender retransmits every message twice over these cycles, even if transmission errors do not take place. Figure 12 illuminates the situation. This kind of operation is in fact an operation with overflow or with double traffic. The states (p_3,p_4,p_6,p_6) and (p_2,p_2,p_3,p_4) are the other indications of overflow, since the places p_6 and p_2 have two tokens (PNP is not safe). We do not have a more adequate transmission medium model than the one drawn in Figure 7; therefore we may presume that the transmission medium would not be able to manage this traffic, and loss probability would be near to one.

4.6 Self-Synchronization

In Section 4.3 it was mentioned that wrong initial values of the variables exp and seq would lead to normal operation, except for the loss of the first message. With regard to arbitrary initial states other than (p_1,p_4), these can be up to 2^7, assuming that every place has no more than one token. All of these states could, of course, never occur in the real system; only a limited set of states can be initial states. We will presume that the transmission medium is empty at the beginning of the operation (conditions p_2,p_6 are not valid), and will analyze the possible combination of the other conditions. The Sender can stay in one of three initial states: p_1,p_3 or p_7, and the receiver in one of two: p_4 or p_5. Thus the initial state of the system can be one of the following: $(p_1,p_4),(p_1,p_5),(p_3,p_4),(p_3,p_5),(p_4,p_7),(p_5,p_7)$. The states $(p_1,p_4),(p_3,p_5),(p_4,p_7)$ are incorporated into the normal cycle of operation; the system leaves the state (p_3,p_4) due to time-out expiration. If the system has been initiated in the states (p_1,p_5) or (p_5,p_7), then under certain conditions (see Section 4.5) it will operate with double traffic.

5. CONCLUSIONS

In this paper we have analyzed a system designed in accordance with the alternation bit protocol in the following situations:

-- normal operation;
-- error transmission;
-- message loss;
-- wrong timeout expiration;
-- arbitrary initial state.

The technique adopted has allowed us to establish that the protocol shows undesirable properties (operation with double traffic) only in the last two situations, and only if certain conditions are valid. It should be noted that this operation mode has not been rigorously specified in previous papers [4,9,10].

Our approach has been based upon a top-down specification, which refines the protocol step-by-step. At every step in the iterative process of specification and verification, new situations have been taken into consideration. The enabling predicates have allowed us to find out the conditions under which the protocol will show certain properties. Our reachability analysis has dealt only with the control states of the system. This has the advantage over the state machine models that the number of control states in the system is much less than the number of total states. Thus the technique we have proposed combines the advantages of assertion proof methods and state machine languages, it does not burden the description protocol with unnecessary details, and it is subject to a lesser degree to the state explosion effect. The above can be proved when more sophisticated protocols are taken into consideration.

The formalism adopted is useful, not only for proving the logical correctness of the protocol, but also for making a performance analysis of the system designed under the protocol. It is also possible to estimate the timeout value and certain parameters which characterize the protocol performance [12].

REFERENCES

[1] Sunshine, C. (1979) Formal Techniques for Protocol Specification and Verification. Pages 20 - 27, Computer, September.

[2] Merlin, P.M. (1976) A Methodology for the Design and Implementation of Communication Protocols. IEEE Trans. on Comm. Volume COM-24: 614 - 621.

[3] Danthine, A. (1977) Petri Nets for Protocol Modelling and Verification. Pages 663 - 685, Proceedings of the Symposium on Data Communications, Budapest. Budapest: NJSZT.

[4] Bochmann, G.V. and J. Gecsei (1977) A Unified Model for the Specification and Verification of Protocols. Pages 229 - 234, Proceedings of the IFIP Congress.

[5] Symons, F.J.W. (1977) The Application of Numerical Petri Nets to the Analysis of Communication Protocol and Signalling Systems. Technical Report, University of Essex, England, May.

[6] Petrenko, A.F. (1979) On Network Protocol Modelling. Pages 83 - 89, Proceedings of the Conference on Packet Switching Networks, Riga (in Russian). Riga: "Zinante".

[7] Agerwala, T. (1979) Putting Petri Net to Work. Pages 85 - 94, Computer, December.

[8] Bartlett, K.A., Scantlebury, R.A., and P.T. Wilkinson (1969) A Note on Reliable Full-Duplex Transmission over Half-Duplex Links. Pages 260 - 261, C.ACM(12).

[9] Bochman, G.V. (1978) Finite State Description of Communication Protocols. Pages Fe-1 - 11, Proceedings of the Symposium on Computer Communication Protocols, Liege, Belgium, edited by A. Danthine, 13 - 15 February. Liege: Universite de Liege.

[10] Brand, D., Joyner, W.H., Jr. (1978) Verification of Protocols using Symbolic Execution. Pages F2-1 - 7, Proceedings of the Symposium on Computer Communication Protocols, Liege, Belgium, edited by A. Danthine, 35 - 15 February. Liege: Universite de Liege.

[11] Keller, R.M. (1976) Formal Verification of Parallel Programs. Pages 371 - 384, C.ACM (7).

[12] Petrenko, A.F. (1979) Timeouts in Network Protocols. Pages 89 - 96, Proceedings of the Conference on Packet Switching Networks, Riga (in Russian). Riga: "Zinante".

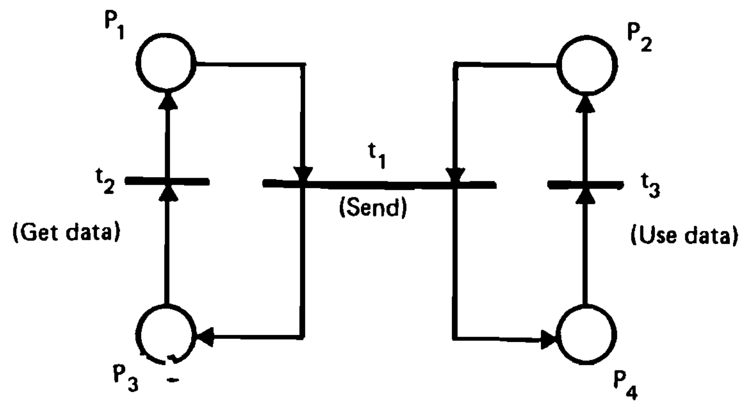

Figure 1. Petri net of the protocol (level I of abstraction)

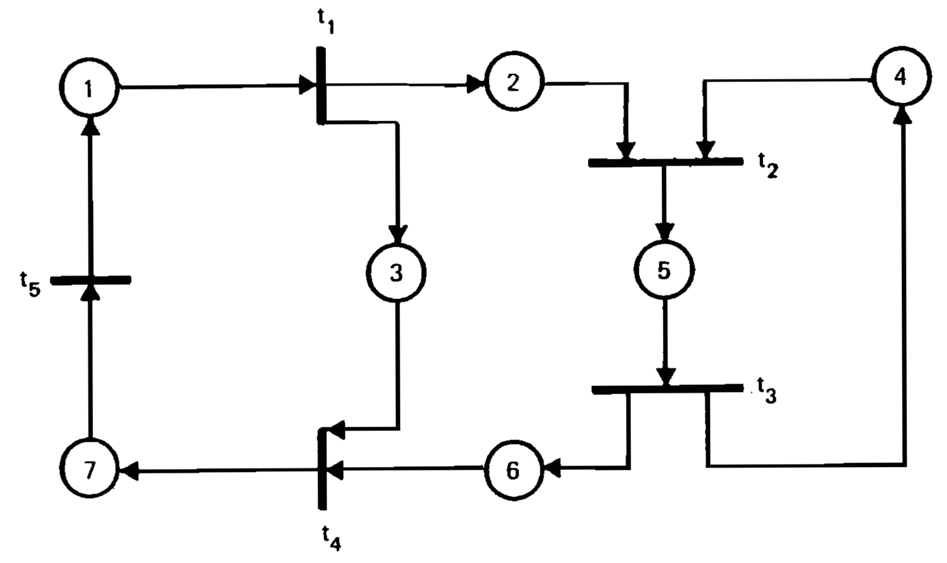

Figure 2. PN of the protocol (level II of abstraction)

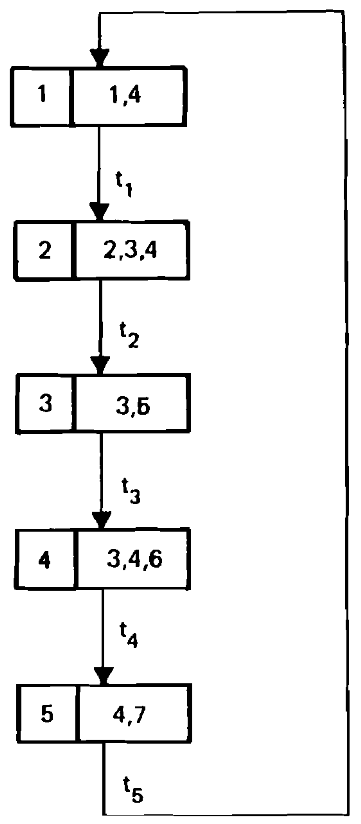

Figure 3. Marking graph of PN of Figure 2

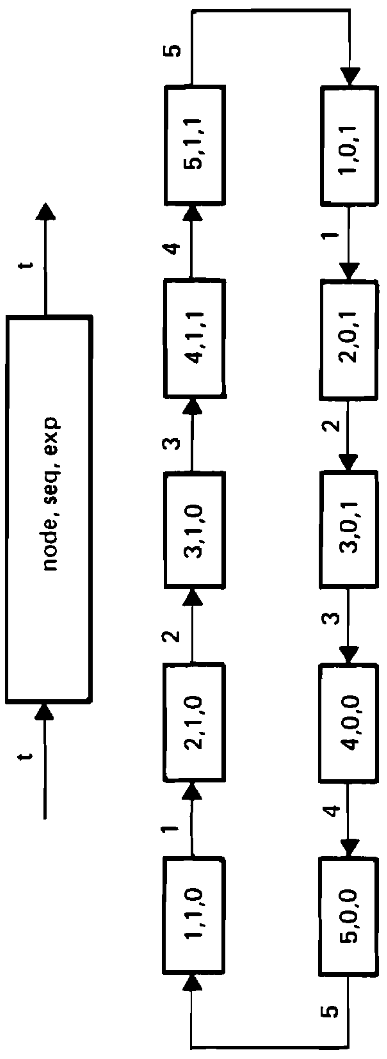

Figure 4. Total state graph

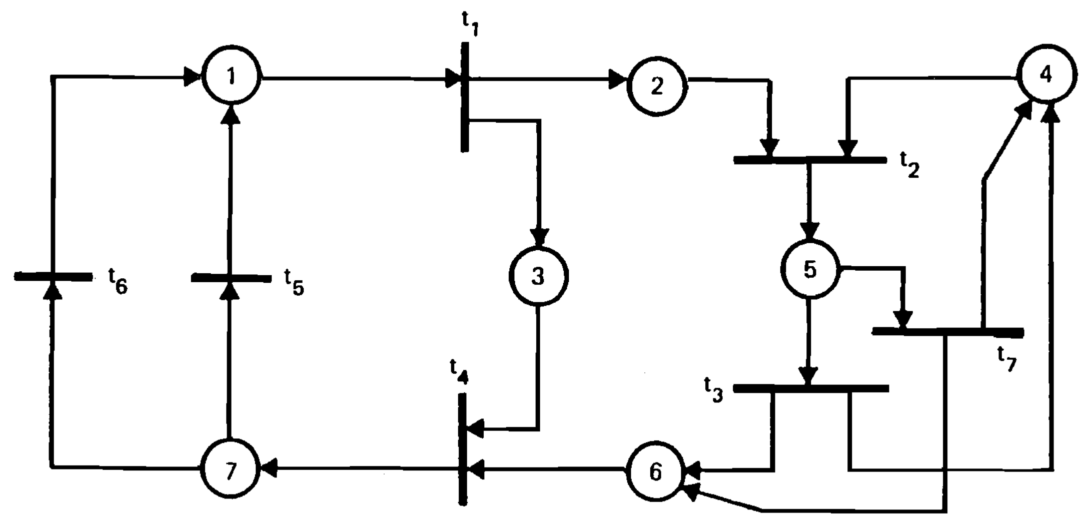

Figure 5. PNP of the protocol (level III of abstraction)

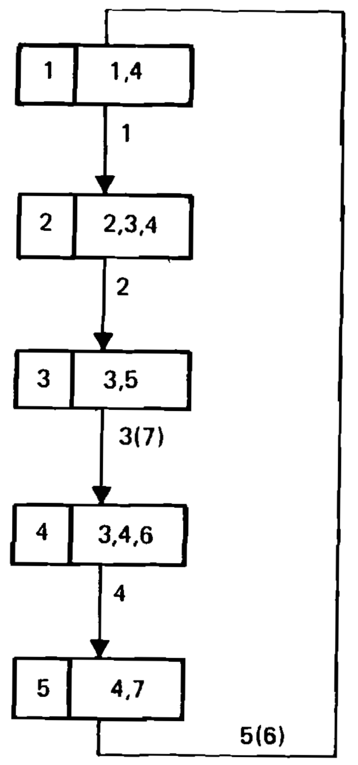

Figure 6. Marking graph of PNP of Figure 5

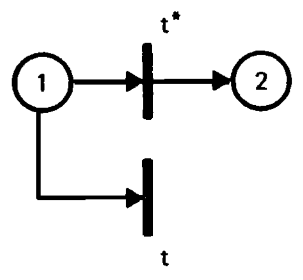

Figure 7. PN of the transmission medium

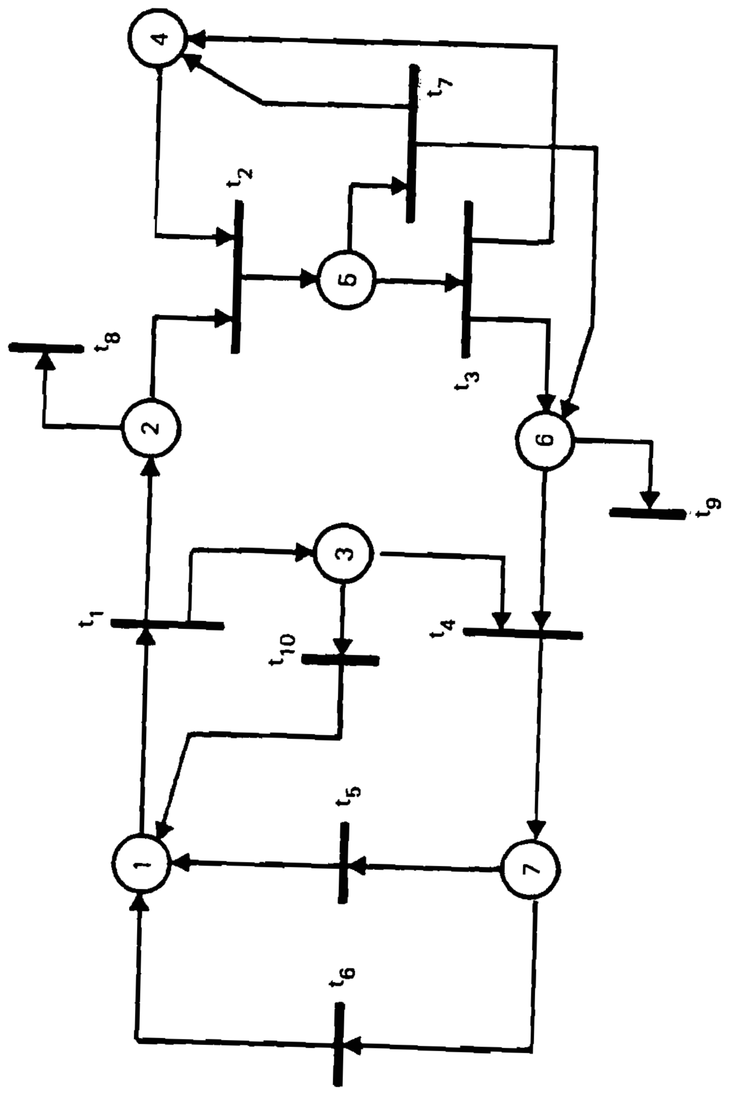

Figure 8. PNP of the protocol (level IV of abstraction)

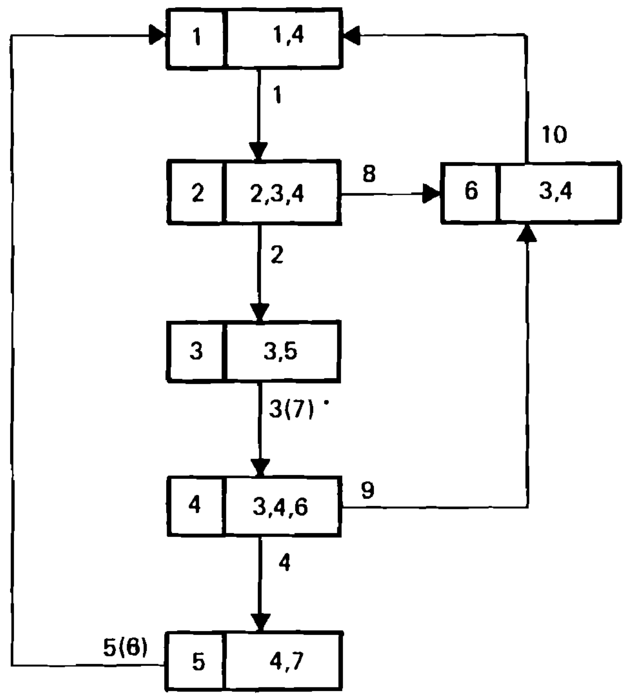

Figure 9. Marking graph of PNP of Figure 8

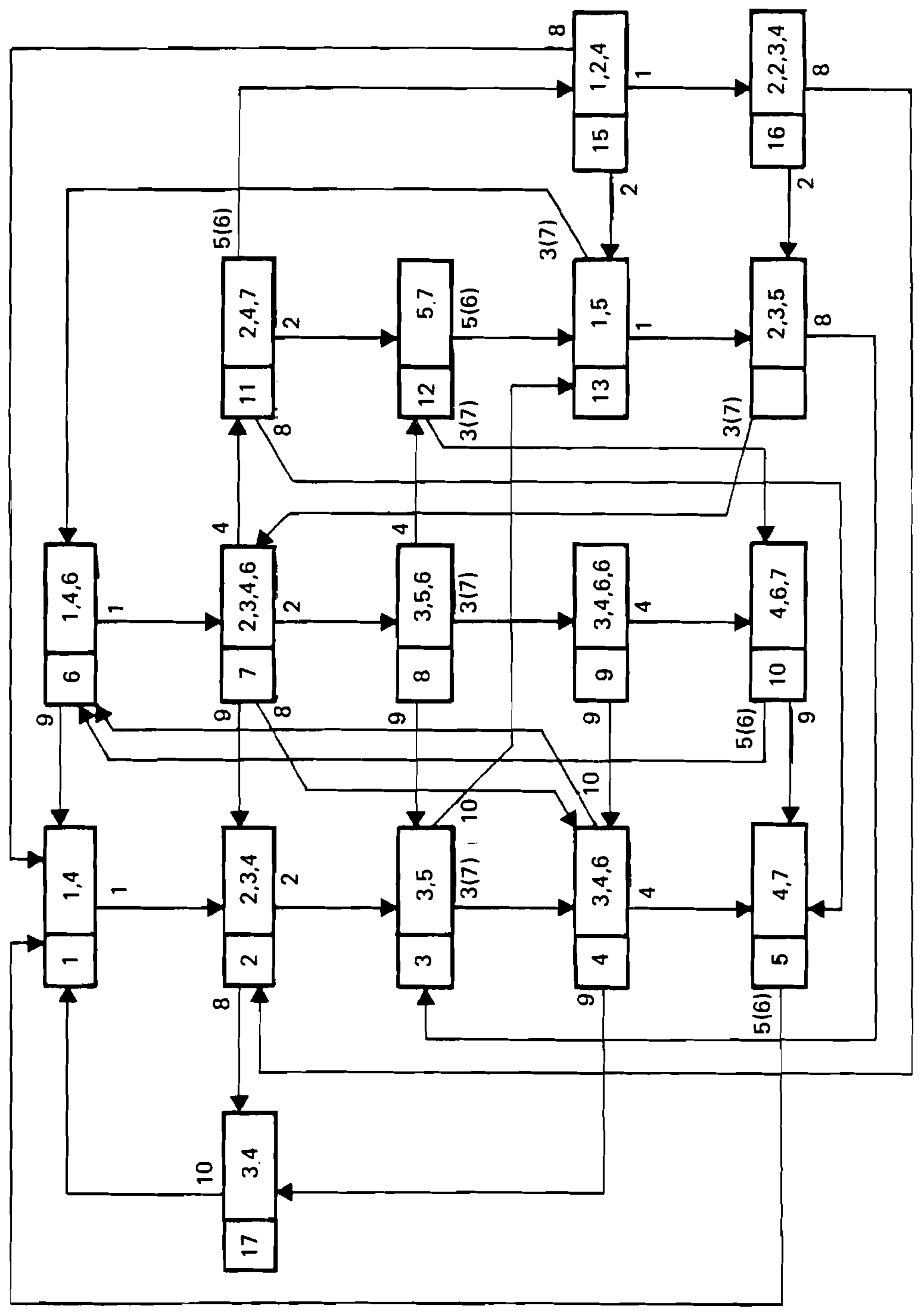

Figure 10. Graph of all the control states

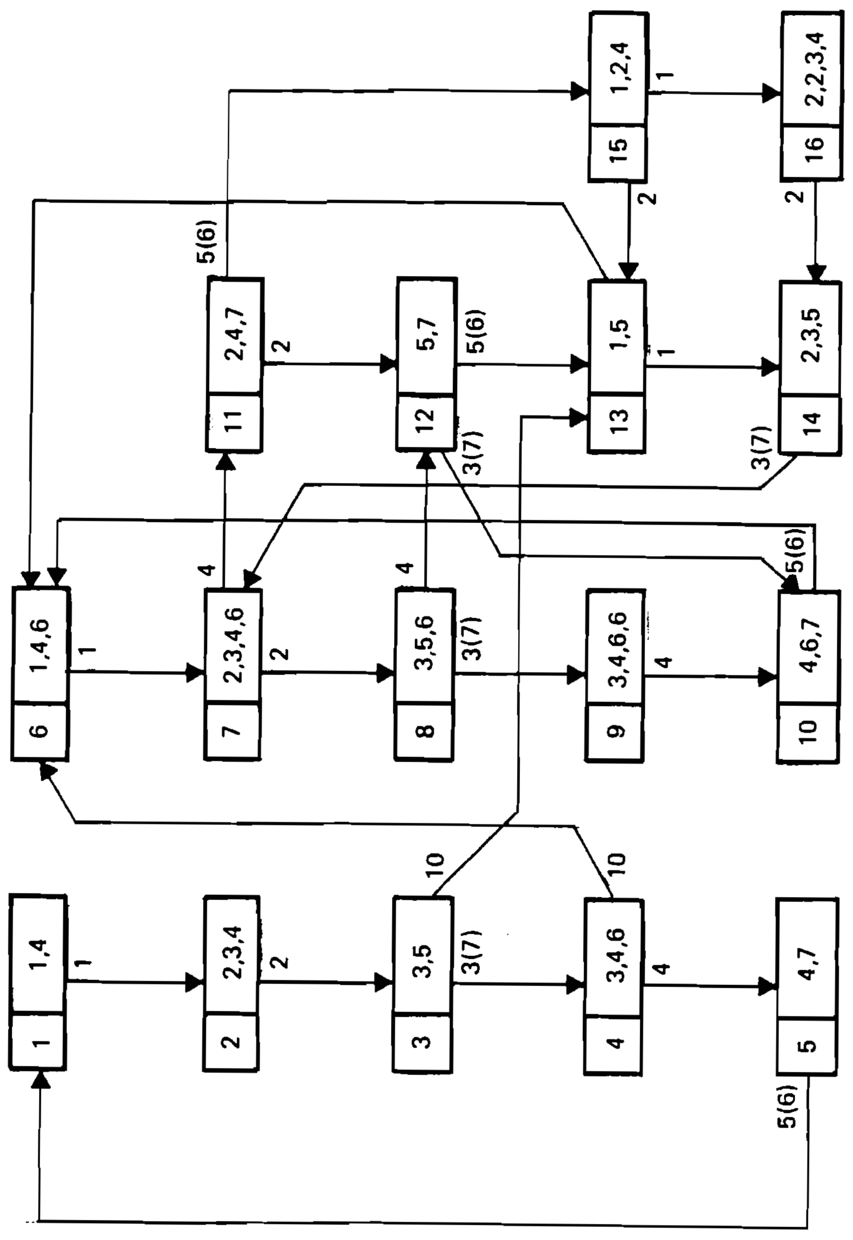

Figure 11. Simplified graph of the control states

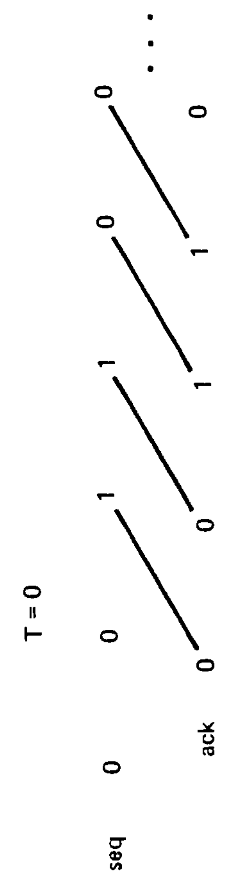

Figure 12. Numbers sequence