MODELLING AND VALIDATION OF PROTOCOLS

P.M. Merlin

Abstract

Complex protocols are used to coordinate remote activities in computer networks. To insure proper operation, formal techniques of protocol definition and validation have been proposed, and developed to the point that they can be applied to actual protocols. However, much work remains to be done in order to cope with protcols of ever-increasing complexity: in particular, those coordinating the activities of many interacting entities.

The characteristics that determine the applicability of a modelling and validation technique to a protocol will be treated. Then a comparative description of techniques for protocol formal modelling and validation will be presented, including new techniques that are applicable to protocols which conventional techniques are incapable of handling.

1. Introduction

Given a system or cooperating processes such that the cooperation is done through the exchange of messages, a protocol is the set of rules which govern this exchange [DANT, GOUD]. By limiting the interaction to message exchange, we mean that information about the state of a process may be known to others only if this information is explicity released (i.e. a message is sent) by the process. Distributed Systems naturally employ protocols because if the interacting entities are physically remote to each other, message exchange is the only possible way of coordinating their activities. However, the use of protocols is not limited to physically distributed systems, but to any system in which the interaction between entities is done by message exchange. In this paper, we assume a quite general definition of a "message": a letter, a finite sequence of bits, a signal, a pulse, are all considered as messages.

The exchange of messages between the processes have some purpose and the role of the protocol is to insure that the purpose is indeed achieved. There are many different purposes that protocols can be designed for. The most common purpose is that of data transfer between processes. For illustration, Figure 1(a) shows a card reader connected to a computer via a communication channel. Typically, a protocol will take care of the transfer of data from the card reader to the computer, and will handle also exceptional situations such as transmission errors and lost messages. Examples of such point-to-point data transfer protocols appear in [X.25, STEN, SDLC], and a simple one is also described in section 2. Protocols may also take care of transmitting data between processes via a shared medium such as a bus, a satellite link, or a radio channel (see Figure 1b). If in shared medium data transfer the processes transmit messages asynchronously, it may occur that more than one process could attempt to transmit simultaneously. The occurence of such a situation is called a collision. If a colision occurs, the transmitted messages will be garbled. In shared medium data transfer, the main role of the protocol is to resolve contention for the use of the medium by either preventing collisions (e.g. synchronising the use of the medium), or recovering from them (e.g. using collision detection mechanisms and retransmitting after collision). These protocols are called "contention protocols" and examples of them can be found in the DEC-UNIBUS [PDP 11], ALOHA NETWORK [ABRA], ETHERNET [METC], etc.

Another purpose which is quite common for protocols is that of synchronisation and initialisation. For illustration, a protocol may take care of synchronising the time clocks of the different computers of a computer network [FINN], or protocols may initialise variables or counters in remote computes. This is done, for instance, during the establishment of a session between a user and a time sharing system, or a session (called also "connection" or "virtual call") between two users of a computer network. In distributed data bases, protocols take care of synchronising the update of multiple copies of the same datum (e.g. A file or a variable) to guarantee a consistent view of the data base [MULL, ELLI]. Other purposes of protocols are flow control of the traffic in a store-and-forward network, location of a mobile unit in a communication network (search), etc.

Modern distributed systems may require extremely complex protocols. Protocols can be so subtle that a formal treatment is necessary in order to guarantee that its definition is complete and unambiguous, and that the purpose of the protocol will be correctly achieved. Formal modelling techniques are used to define protocols, and validation techniques are used to insure their correctness and proper operation. These techniques are the main topic of this paper.

As shown in subsequent sections, different classes of protocols require different modelling and validation techniques. There is no single method that can be conveniently used to model and validate all protocols, much the same as there is no single universal tool in the mechanic shop. For bolts we can use pliers which cannot be used for screws; for screws we use screw-drivers. However, while theoretically there is a correspondence between bolts and pliers, clearly pliers are not suitable for large railway bolts, because there is a practical mismatch. Similarly, there are matches between classes of protocols and modelling and validation techniques. Sometimes these matches are practical, other times, although theoretically a technique can be applied to a protocol, in practice, this could be impossible.

In the next section we show an example of the usage of a simple modelling and validation technique, illustrate its advantages and demonstrate the shortcomings of this technique to handle protocols having certain characteristics. Then, the characteristics that determine the applicability of a technique to a protocol are discussed in general, and different techniques are presented and evaluated according to the types of characteristics that they can handle. Last, we present an example using a technique which is applicable to a type of protocol characteristics that most established and known protocol validation techniques are not applicable to. Space limitations force us to omit detail discussion of many meritorious works. We regret these inevitable omissions and refer the reader to the surveys of [SUNS1, SUNS2] and the bibliography of [DAY].

2. <u>An Introductory Example: The Alternating Bit Protocol</u>

Using Petri Nets [PETR, HOLT, PETE, MILL], Figure 2 shows a model of a simplified version (the recovery mechanism is omitted) of the alternating bit protocol. A more complete description of this technique and the example appearing in [MERL1, MERL2]; [EOCH1] present a theoretically equivalent technique for this protocol, which involves two parties - a sender and a receiver - connected through a medium. The sender sends messages to the receiver, and the receiver responds with acknowledgements. Each message carries a control bit (0 or 1) whose value alternates for consecutive messages, and each acknowledgment carries a bit equal to the one carried by the message it acknowledges.

The places of Figure 2 represent the following conditions:

A1 = ready to send message with control bit 0
B1 = ready to receive message with control bit 0
M1 = message 0 in transit
K1 = acknowledgement 0 in transit
W1 = waiting for acknowledgment to message 0
C1 = message 0 was received
E1 = acknowledgment to message 0 was received
D1 = message 0 is being consumed; and A2, B2, M2, K2, W2, C2, E2, D2 have the same respective meanings but for messages or acknowledgements carrying control bit 1.

The bars perform the following events:

11 sends message 0
12 receives message 0
13 sends acknowledgment to message 0
14 receives acknowledgment to message 0
15 consumes message 0

16 produces message 0; and 21, 22, 23, 24, 25, 26 perform the same respective events but for messages or acknowledgments carrying control bit 1.

The initial state is one token in A1 and one token in B1. By generating all possible global states and transitions between them, the reader may familiarise himself with the behaviour of the protocol. The resulting graph of such global state generation is a state machine called <u>Token Machine</u> and it is given in Figure 3. In the Token Machine, states represent global states of the Petri Net model of the protocol, and each transition is labeled by the number of the bar that effects it. Since in this case the Token Machine is finite, we can easily see (i.e. validate) that after a message is sent it will be received, that consecutive messages are sent carrying alternating control bit, that there is no deadlock, etc. [MERL 1,2] shows the null model and TM of the protocol including the recovery mechanisms and accounting for possible failures. Several validation techniques were proposed based on a finite state description of each party of the protocol and generating all possible global states and transitions [BOCH 1, MERL3, RUDI, WEST1, ZAFI, POST], in a way similar to the one demonstrated above. These techniques are theoretically equivalent.

In this paper, each part of a protocol residing at a single process is called a party of the protocol, i.e. a party is a portion of a process which is relevant to a protocol. The first cause which can preclude the applicability of the technique shown above is high party complexity. For illustration, if in the example instead of an alternating bit the sender will label the messages with a sequence number of, say, 32 bits, then it will be clearly impractical to generate the Token Machine, because it will include more than 2 to the 32 states. In that case, it may be even quite complex just to describe the protocol itself with a finite state model as the one of Figure 2. Moreover, one may even want to model and validate a protocol whose entities include unbounded variables, for example if the messages would carry an acyclic increasing sequence number. In this case, it is even theoretically impossible to apply a finite state technique.

The topology or a protocol is the graph whose nodes are the parties of the protocols and arcs denote possible interactions between the parties (see Figures 1 and 2). Also increase in topology complexity can preclude the applicability of a technique. For example, a protocol with even simple parties may not be validated by exhaustive global state generation if it includes too many parties.

3. Protocol Classification

The amenability of a protocol to the application of a technique is affected by two relatively independent protocol characteristics: party characteristic and topology characteristic. The characteristic of a party is given by the (possibly infinite) set of all possible pairs of incoming-outgoing message sequences, i.e. the characteristics describe all possible behaviours of the types of topologies the protocol can work on. A protocol may be designed to correctly work in any topology of a given set of topologies. For example, the same protocol of Figure 1(b) could be used for any number (possibly up to a certain limit) of parties connected to the shared medium. Table 1 shows a list of types of topology sets. The entries in the table are ordered by increasing generality, i.e. each entry is a special case of entries appearing later.

A protocol may also be designed to work on an evolving topology. For example, a routing protocol may work on any computer network topology where operating nodes and links may fail, and new nodes and links may become operational [FINN, MERL4]. This can be considered as if the topology would be evolving during operation. Another example of topology evolution is given by the progress of a multiparty phone conversation in an advanced telephone exchange [FOOX]. The topology characteristic of a protocol is defined as the set of permitted topologies and their possible evolutions.

A modelling technique is theoretically applicable to a given protocol if and only if it is powerful enough to be able to represent each of the characteristics of the parties and the characteristic of the topology. For illustration, the party characteristic of the sender (or of the receiver) in Figure 2 can be expressed as repetitions of the string: (M0 K0 M1 K1). But any such expression can be represented by a Petri net, in fact, even a Finite State Machine could be used to represent that characteristic. The simple topology of the alternating bit protocol is also represented by the connectivity of the Petri net of Figure 2.

Theoretical or practical applicability of a validation technique depends on the protocol characteristics (parties and topology), on the modelling technique used to describe the protocol (i.e. the description to which the validation is applied), and finally, the properties to be validated. When surveying techniques in subsequent sections these dependencies will be further discussed.

The practical applicability of a modelling technique or of a validation technique is difficult to formalise. Practical applicability implies theoretical applicability, but also "conciseness", "ease of understanding", "convenience to use", etc., which are difficult to quantify and sometimes depend on personal experience and taste.

Initially, most work on protocol modelling and validation was done for protocols of simple party characteristic, i.e. those which can be described by a Finite State Machine, but later there were several developements applicable to more complex parties. However, from the topology point of view, with a few exceptions [FIN, MERL4, MERL5, DIJK, FOOX, ELLI, etc.] most of the formal work done to date is applicable only to extremely simple topologies -- usually a pair of entities.

4. Abstraction

It is important to remember that a (formal) description is only a model of a system and not the real world. When modelling a protocol, we don't take into account each electron in the system; we even don't consider the logical dates which make the computer that performs that protocol. We limit our view to certain aspects because otherwise the complexity of the model will be intractable and the effort necessary to build the model will be as large as that of building the real system or even greater. A limitation to deal with certain aspects while ignoring those details of the protocol and its environment which are irrelevant to these aspects is called an abstraction. Abstraction implies assumptions on the behaviour or properties of the protocol environment which are explicity or implicitly made by ignoring details. Hence, the validated properties of the protocol will indeed be true only if the assumptions implied by the modelled abstraction hold.

Similarly to operating systems, protocols are typically built in hierarchical levels of abstraction, where the protocols at a level use the functions provided by the levels below without concern for how those functions are actually implemented. An example of this is shown in Figure 4. The shown protocol structure is typical of computer networks. In this example, the user protocols of the highest level take care of communication between "processes". At this level of abstraction it appears as if processes directly communicate with each other by exchanging "letters". However, in practice, the lower level protocol, i.e. the protocol communicating between the hosts, is responsible for delivering the letters and implements this by exchanging what is abstracted as "messages". These messages may carry parts of letters as well as necessary control and synchronisation information for the host-host protocol. The protocols at the user level of abstraction is not concerned with the details of the host-host protocol, and assumes that the letters it sends wil be delivered and that certain types of failures may occur. Similarly, the message exchange is implemented by the end-to-end network-node protocol which communicates by packets, and so on for the lower levels of abstractions. All levels, except the store-and-forward include only two parties per protocol. The store-and forward protocol may include more than two parties connected on any mesh topology and Figure 4 shows only one of the paths through that topology.

The abstraction made determines also what we consider to be parties and what we consider to be communication links between them. A common abstraction is to consider complex links as being parties that connect between other parties. For example, the shared medium of Figure 1(b) can be seen as a party that represents the possible behaviours of the medium, and interconnect between A,B,C,D,E.

5 Models

In this section we discuss a few of the existing protocol models and modelling techniques.

Finite State Machines

Finite state machines were proposed quite early to model protocols [KAWA, BIRK, KNOB, BOCH2]. A single finite state machine can be used to describe the global state of the protocol or, alternatively, one machine can be used for each party, as described by the simple example of a sender and a single buffer receiver of Figure 5. The sender sends a "data" message and waits for the "done" message sent by the receiver when the buffer is empty again. In the multi-machine approach, a transition marked with a SEND and a transition on a different machine marked with a RCV having the same parameter (i.e. the same message) are performed simultaneously and the machines are said to be coupled. If the message transmission delay is not important, the sender and receiver machines can be directly coupled and the machine for the medium omitted. The single machine and the coupled machines models are theoretically equivalent and they are theoretically applicable to any protocol having finite entities (i.e. characterizable by regular expressions) and a bounded number of topologies. Infinite state entities are not representable, and the generalizations necessary to represent unbounded number of topologies will be discussed later.

In practics, both approaches are applicable to simple topologies (usually a pair of entities) and to entities having no more than a few dozens of states. A practical advantage of the single machine approach is that global properties can be directly checked (or designed) on the model. An advantage of the coupled-machines approach is that it can be directly implemented in each party without the problems of decomposing the single machine description amongst the entities. Such decomposition may be done in different ways leading to possibly non-compatible implementations of the same protocol [WEST2]. The single machine approach is used to describe several adopted standards [e.g. X.21, X.25]. More details and examples of the finite state machine approach can be found in [BOCH1,GOUD, RUDI, WEST].

Several extensions to the finite state machine model were proposed [GOUD, BOCH3, MERL4, VISS] one of which is demonstrated in section 7. These extensions broaden the theoretical and practical applicability of the model.

Petri Nets and Related Models

The use of the Petri net model was already shown in Section 2. The theoretical applicability of Petri nets is broader than Finite State Machines - any Finite State Machine as well as some types of protocols having an infinite number of states can be represented by Petri nets. However, the Petri net model is not universal because certain party characteristics are not representable in this model. Also protocols with unbounded number of permitted topologies cannot be represented and the necessary generalisations will be discussed later.

The practical applicability of Petri nets is close to that of finite state machines, but in many cases broader. For example, the protocol of Figure 6 is represented in practice by a Petri net, but cannot be represented by a finite state machine even theoretically. This protocol has the property that permits any number of outstanding messages which can be sent and received out of order. However, if we require an arbitrary number of outstanding messages that will be received in the same order that they are sent, this will not be representable even theoretically by a Petri net. Petri nets are convenient for representing protocols which can operate with various amounts of some resouces (e.g. number of buffers, etc.). In this case, a single Petri net will suffice, and the actual amount of resources will be represented by the initial number of tokens placed. Petri nets will also be convenient for representing parties in which several events may occur in arbitrary order. For example, the Petri net of Figure 7 will be quite complex to represent by a finite state machine, but the Petri net representation is relatively compact.

A typical failure handled by protocols is the loss of a message in the medium. In the Petri net model, this can be described by arbitrarily removing the token from the place that represents the medium (e.g. M1 or M2 of Figure 2). Using such a representation of failures [MERL1, MERL2] studied the recoverability of distributed computing systems and its applications to protocols. Unfortunately, [MERL1] shows that the necessary and sufficient conditions a Petri net must satisfy in order to be recoverable from lost messages imply some properties that are usually unacceptable in practical systems. This is due to the fact that Petri nets do not include any knowledge (or limitations) of the execution time of the events, and relations among these times play a central role in all practical recoverable protocols. The practical implication of this is that Petri nets cannot faithfully represent the entire meaning of time-outs, and a similar situation exists with respect to the coupled finite state machines model.

In order to allow the representation of timing knowledge by a Petri net-like model the Time Petri Net was created. A Time Petri net is defined by a Petri net where each bar has two times specified. The first denotes the minimal time that must elapse from the time that all the input conditions of the bar are enabled until this bar can fire. The other time denotes the maximal time that the input conditions can be enabled and the bar does not fire. After this time, the bar must fire. In general, these two times give some measure of minimal and maximal execution times of the bars, while maintaining the basic characteristics of the Petri nets. This model is useful in describing practical recoverable protocols [MERL 1,2,3] and allows the exact representation of time-cuts which is impossible in most other models.

The principal practical shortcoming of Petri nets (as well as state machines) is the rapid growth of the graph with the complexity of the protocol. To alleviate this, in addition to the Time Petri net, other enhancements and variations of the basic model were proposed and used to represent protocols [POST, KELL, MERL3, ELLI, YOEI, SYMC, FOOX]. These enhancements result in a more compact notation, but they also increase the theoretical applicability of the model.

High Level Programming Lanaugages

High level programming languages were also proposed and used to model protocols [BOCH4, STEN, DANT, KROG]. In this model, each party is represented by a formal description similar to a high level program. Since these languages are universal, they permit the representation of any entity characteristic. However, in the standard way in which these languages are used only simple topologies can be represented, and the extensions necessary to represent unbounded numbers of topologies or evolving topologies are described below.

In practice, standard high level programming languages are convenient to represent numbers, data, variables, counters, etc., but not complex control structures. Therefore, this model was mainly used to represent the data transfer aspects or protocols while the graph models (State Machines and Petri nets) were mainly used to represent the control aspects (asynchronisation, initialisation, etc.) for which they are more convenient. Hence, there were also protocol models proposed [BOCH3, DANT, MERL4] which combine high level languages with graph models.

A protocol model based on formal grammars is proposed in [HARA1, HARA2].

Representation of Unbounded Number of Topologies and Evolution

If we assume that each party and each link is individually represented, as implicity done before, then unbounded numbers of topologies cannot be represented because this will require an infinite expression. Hence, we must find finite ways of expressing such protocols. This can be done by giving a bounded number of basic parties and a rule of connecting replications of the basic parties into permitted topologies. For example, a loop of any arbitrary number of identical parties can be represented by representing one copy of the parties, showing its connections to the neighbours, and assuming a finite number of entites as shown in the example of Figure 8. Any of the models can be used to describe the basic entities provided that indices or other ways of expressing the rule for connecting the entities are added. Sometimes, as in Figure 8, the indices i, j, k are used only as linkages to show the connectivity of the topology and different values of indices point only to different locations in the network but not to different properties of the parties. Other times, the values of the indices are used to denote differences between entities, e.g. higher index may denote higher priority.

An unbounded number of topologies may include not only an unbounded number of parties but also an unbounded number of links to other parties. In such a case, each party may have an arbitrary number of neighbours, which can be represented, as shown in the example of section 7, by allowing the basic party to have a list of neighbours of arbitrary length.

Clearly, these modelling techniques are applicable not only to an unbounded number of topologies, but they can also become of great practical value in modelling protocols with bounded but large numbers of topologies. For illustration, if the loop generated by the basic party of Figure 8 is limited to 1024 elements, it can be best represented as for the unbounded case in which 0<i,j,k< infinity is changed by 0<i,j,k,<1023. The use of this technique to represent large or unbounded topologies appears in [FINN, ELLI, MERL4, MERL5, DIJK].

If each party has a list (or table) of neighbours, a topology evolution can be represented as a change in these tables (possibly including creation or destruction of tables), which can occur during the protocol operation. An example of this appears in [MERL4]. A similar technique, but with a central relation describing the connectivity, was used in [FOOX] to model and validate the protocols of a quite complex telephone exchange. The description of protocols involving topology evolution requires operations that cause changes in the party interconnections, and the model used to describe such protocols should include these operations.

6. Validation Techniques

In [BOCH1, (also GOUD)] there appears a list of the properties which are usually validated in protocols; a summary of them is given below including slight changes and additions. The list is quite non-committing in the sense that there exist many protocols from which some of the properties are not required, or for which some of the properties have a slightly different meaning than the one described:

1. DEADLOCK FREENESS: "No terminal state".

2. LIVENESS: "From each reachable state any other state is reachable" or "for each reachable state and event there exists a reachable state from which this event can occur" (in a sense, this represents the concept of no degradation).

3. TEMPO-BLOCKING FREENESS: "There is no non-productive infinite looping".

4. STARVATION FREENESS: "If several processes contend for resources which become available infinitly many times, no process will be prevented forever from acquiring the resources that it needs".

5. RECOVERY FROM FAILURES: "After a failure the protocol will return to normal execution within a finite number of steps (or within finite time)".

6. SELF SYNCHRONISATION: "From any abnormal state, the protocol will return to a normal state within a finite number of steps (or within finite time). This property and recovery are closely related".

7. CORRECT PURPOSE EXECUTION: This is very protocol dependent, e.g. "correct data delivery" for a data transmission protocol, and "only one user at a time on bus" for a bus contention-resolution protocol.

Many validation methods have been proposed, but most of them make use of one of four basic validation techniques. Although two methods using the same basic validation technique may display some practical differences, they are usually theoretically equivalent for every protocol which is representable by the two modelling techniques on which the validation methods are respectively applied. The basic validation techniques are described below.

One of the most common validation techniques is exhaustive global state generation as demonstrated by the Token Machine of Figure 3. Uses of this technique appear in [BOCH1, BOCH2, BOCH3, MERL1, MERL2, MERL3, ZAFI, RUDI, WEST1, WEST2, etc.]. The theoretical applicability of this technique is limited to protocols with bounded number of topologies and finite state parties. The practical applicability is limited to very simple topologies (say up to half a dozen parties). Several actual protocols (or parts of them) were validated using this technique (e.g. alternating bit protocol [BOCH1, MERL2], X.21 [WEST2], X.25 [BOCH117, a telephone exchange having evolving simple topologies [FOOX]). An advantage of this technique is that the state generation can be easily mechanised, and several properties can be automatically tested [RUDI, WEST1, WEST2]. However, since there exist protocols where some properties should "usually" hold but exceptional cases are permitted, the failure to pass an automatic test does not necessarily imply that the protocol is not correct, and therefore, human interpretation of the results is nevertheless needed [MERL3, WEST2, FOOX]. Sometimes, (e.g. [DANT, FOOX]) properties of the

Total state space can be validated by generating a small subset of the states. This can greatly increase the applicability of the technique.

Another common protocol validation technique is assertion proving [KELL, EOCH3, BOCH4, STEN, KROG] which is applied to the protocol description, as if the description were a parallel program. This technique was usually applied on protocols modelled in high level programming languages, but theoretically it can be applied to any other model. The usual way of applying this technique is by attaching a predicate of the variables' values to certain points in a program and proving that whenever the program reaches these points the predicate is true. This can be generalised to a given collection of cooperating programs by attaching the predicate to sets of points such that in each set there is at most one point from each program. Then it is proved that whenever the programs reach the points of any such set the predicate holds. As described above, this method is limited to protocols with a bounded number of topologies. However, the method can be generalised to protocols with an unbounded number of topologies provided that the desired predicate and the (possibly unbounded) sets of points can be expressed by bounded expressions, as to be demonstrated by the example of section 7.

In practice, assertion proofs were mainly applied to simple topologies, but sometimes, having quite complex parties. Examples of actual protocols validated using this method are: alternating bit [BOCH 317, HDLC [BCCH4], Data Transfer Protocol [STEN]. Since the construction of proofs may require an act of creativity, this technique cannot be fully automated. However, quite powerful theorem provers have been constructed which are capable of automatically proving many of the required properties. An application of such a prover to the validation of protocols is reported in [BFAN].

While global state generation is more convenient in proving control properties (e.g. that certain events will or will not occur), assertion proving is mainly used in proving data transfer properties, in particular, in protocols involving parties with large or infinite state space. The two techniques can be also combined [ECCH3] in order to capitalise on the advantages of each.

A third validation technique is induction over the topology. By this technique, the holding of a property or the occurrence of an event is proven by showing that certain conditions will propagate throughout the topology. The use of induction over the topology is theoretically applicable to protocols of any characteristic, and is particularly useful in cases of protocols having large or an unbounded number of topologies, and evolving topologies. The technique was successfully applied in practice to several such protocols [FINN, MERL4, MERL5, DIJK]. These examples involve relatively simple parties, and from the limited experience with this technique it seems that formal proofs for such topologies but more complex parties could be difficult. However, we clearly cannot expect simple validations for very complex systems (i.e. complex topologies together with complex parties) and still this could be the best technique for those cases. An example of the use of this technique is given in the next section.

The last technique is adherence to sufficient conditions. In this technique, the protocol is designed in such a way that each design step is done satisfying conditions which are sufficient to guarantee the required properties. That is, instead of designing a protocol and later proving its correctness, this technique is aimed at directly designing one which is correct by construction. (Notice that similar concepts exist for software development.) This technique can be used in any topology and entity characteristic, and its main advantage is that it is easy to apply and that correctness is directly guaranteed. Its main shortcoming is that sufficient conditions could be too strong, i.e. there may be many correct protocols that will be rejected because they do not satisfy the sufficient conditions. On the other hand, tight sufficient conditions (or preferable necessary and sufficient conditions) are usually complex and difficult to find. Examples of this technique appear in [GUNT, MERL6].

7. A Concluding Example: A Routing Protocol

The modelling and validation example described below is a simplified version of the routing protocol proposed and validated in [MERL4]. The protocol is executed by the nodes of an arbitrary computer network of nodes N and links $L \leq NxN$ (since the arcs are non-directed, if i,j belong to L so also do i,j). To each link i,j of L a positive constant dij (called "distance") is assigned. The distance represents the cost of using the link. One of the nodes of N is called the SINK. Each node I of N has a variable di which stores the "estimated distance" from node I to SINK and a variable pi called "preferred neighbour" that points to one of the neighbours of i. The set of all pointers pi is denoted as P. Initially, the pointers P form over the entire network a directed tree routed at SINK. An example of such a network is shown in Figure 9. The pointers P provide a loop free path from every node to SINK and the purpose of the protocol is to update the pointers in such a way that:

1. At any time they form a tree routed at SINK, and
2. Minimise the path length (i.e. the sum of dij through the path) from each node to SINK.

In situations defined below, a node I will send to its neighbours a message communicating its estimated distance di to SINK. Such a message is denoted MSG(di), and it will arrive at the receiving node within an arbitrary finite time. In this section "message" refers to the control message MSG(di) - we don't refer to the ordinary data messages which are transmitted through the network.

The protocol operates in update cycles which are triggered by the SINK. Each update cycle improves the paths to SINK, and after such a cycle ends the SINK may start a new one. After a finite number of cycles the paths converge to the minimal path from each node to SINK. As elaborated below, each update cycle proceeds in two phases:

(1) Control messages are sent uptree from SINK to the leaves of the current P tree. During this phase, distance estimates di are updated.

(2) Control messages are sent down tree to the SINK. During this phase, new preferred neighbours P are selected.

In this example, the topology of the protocol corresponds with the topology of the network, and each party corresponds with a network node. This protocol works on any topology, but since, except the SINK, all parties perform the same algorithm, the description of only one party and the SINK are necessary. The details of the protocol are formally described in Figure 10 where the algorithm performed by an arbitrary node I is shown. Each party has two states: S1 ("ready for next cycle") and S2 ("phase 1 was performed, waiting for phase 2"). Each party has two variables, pi, di. The network connectivity is represented by an arbitrary list of neighbours L. For each neighbour k of cf L there is:

(1) a link distance dik (as already discussed),
(2) a flag Ni(k) initialised to NIL, set to RCVD when party I receives a MSG from k, and reset to NIL when phase 2 is completed.
(3) a variable Di(k) that stores the last estimated distance received from k.

Whenever a MSG with any parameter d is received from k the "FOR" statement is performed, i.e. new values are stored in Ni(k) and Di(k). Then, transitions of the finite state machine are performed if the party is in the corresponding state and the "CONDITION" associated with the transition is true. When the transition is performed, the state is changed and the "ACTION" associated with the transition is performed. T12 is performed by party I (i.e.,phase 1 for node i) when it receives a MSG from its pi, then a new di is calculated and MSG (di) is sent to each neighbour except pi. As proven below, the tree structure of P guarantees that after a cycle is triggered, T12 will be performed by every node. T21 is performed by party I (i.e. phase 2 for node i) when it has received a MSG from each of its neighbours. Then I sends MSG(di) to its pi (allowing phase 2 to propagate down tree), updates pi, and resets the flags Ni. The cycle ends when the SINK performs T21. The algorithm for the SINK (see Figure 11) is slightly different because pSINK does not exist, dSINK=C always, and the SINK can spontaneously start new cycles while at S1. When the SINK performs T12 a cycle begins and when T21 the cycle is completed.

The protocol is said to be in idle state if no message is in transit, all parties are at S1, and for all I and k Ni(k)=NIL. It is easy to show that in idle state, the only event that can occur is transition T12 at SINK. Initially, the protocol is in idle state and P forms a tree rooted at SINK. Then:

Theorem 1 : (No deadlock). Within finite time the SINK will always be able to start a new cycle, and any cycle started will be completed in finite time.

Theorem 2 : The pointers of P always form a tree rooted at SINK.

Theorem 3 : Within a bounded number of cycles, P corresponds with the directed graph given by the shortest paths from the nodes to SINK. The bound equals the longest path in terms of number of hops in this graph.

The proofs of Theorems 1 and 2 follow directly from the lemma below. The proof of part 1 or the lemma demonstrates the use of induction over the topology. The combined proof of parts 2 and 3 demonstrate the use of assertion proofs: The predicate is given by conditions (a)-(f) of the proof, they should hold at any time (i.e. such conditions are said to be "invariant") and the proof is done by induction over all possible events that can occur.

Lemma: If in idle state and while P forms a tree the SINK performs T12 then:

1. every party (including SINK) will perform T21 within finite time:
2. when T21 is performed by SINK the protocol enters idle state:
3. from the time that T12 is performed by SINK, P remains a tree at least until after T21 is performed by SINK.

Proof of part 1 : Let P1 denote the initial tree. Initially all parties are at S1, hence a party I can change its pi only when performing T21 and this after performing T12. Suppose a party I laying s ≥ 0 steps over P1 from SINK performs T12. When performing T12 it sends MSG(di) to all its neighbours except pi. This guarantees that every party j such that pj=i will perform T12. Hence, every party j that lays s+1 steps over P1 from SINK will perform T12, and by induction, every party will perform T12. This implies that each party in the leaves of P1 will receive a MSG from each of its neighbours and perform T21 at which time it sends MSG to its old best neighbour. Hence, by induction in a similar way as above but down tree every party will perform T21.

Proof of parts 2 and 3 : Let us assume that up to a time t in the interval between T12 and T21 performed by the SINK, the following conditions hold:

(a) P forms only trees;

(b) since SINK performed T12, each party has performed T12 and T21 at most once each;

(c) for every party i, if i performed T21 then for all k, Ni(k)=NIL, and there is no message in transit to i;

(d) each party has sent at most one message on each link.

(e) for every party i, if i is in S1 then every party j having a path in k to i is also in S1;

(f) for every party i that is in S1, if pi=j then either j is at S2 or j is at S1 and di>dj.

The only events where their occurrence may invalidate the conditions are that some party r performs T12 or T21. We check that such occurrences maintain the conditions. If at time t party r performed T12 (i.e. entered S2):

(a) is maintained because T12 does not change P;

(b) if r already had performed T12, it also performed T21 and party pr violates (d) before t;

(c) no additional party performed T21;

(d) from (b), this is the first time that r performs T12, hence the first time that sends MSGs;

(e) pr has sent MSG to r, but r has not sent to pr, therefore, pr is in S2;

(f) if a party enters S2 (f) is not changed.

If at time t party r performed T21 (i.e. entered back S1):

(b) since r performed T12 only once, then this is the first time it performed T21;

(c)(d) since r performed T12 and then T21, r sent one MSG on each of its links;

(e) if pj=r and since j performs T12 and T21 at most once, then if j is in S2 it has not sent MSG to r and r could not perform T21.

(f) since pr performed T12 only once, at t Dr(pr)=d(r,pr)+dpr and dr $\geq$ Dr(pr), hence dr dpr. If there exists a node j at state S1 such that pj=r, because r performed T21 we know that j performed T12 and T21 and the same previous argument holds for dr and dj.

(g) every j having a path in P to r is in S1 (by (e)); if after r performs T21 pr is in S2 there is no loop formed; if it is in S1 (f) guarantees no loop.

Hence, (a)-(f) hold until after SINK performs T21; (a) implies (3) or lemma and, since initially all nodes are at S1, (1), (b), (c) imply (2). Q.E.D.

The protocol in [MERL4] allows the distances dij to vary with time and generalise the protocol to handle topological changes, i.e. link or node failures and new links or nodes becoming operational. That paper also shows how the data messages can be routed using the paths p maintained by the protocol.

8. Concluding Remarks

We expect in the future the need for coping with more complex protocols. In particular, work is needed in protocols which operate over complex topologies - an area where not much work has been done. The development of automatic or semiautomatic tools may help in this task, however, human ingenuity will always be required. The discovery of better sufficiency conditions may also facilitate the design of correct protocols. Better understanding is also needed in ways of insuring that errors are not introduced when the validated model is translated into an implementation.

REFERENCES

[ABRA] Abramson, N., "The ALOHA System - Another Alternative for Computer Communications", Fall Joint Computer Conf., AFIPS Conf. Proc. Vol. 37, 1970, pp. 281-285.

[BIRK] Birke, D.M., "State Transition Programming Techniques and their use in Producing Teleprocessing Device Control Programs", IEEE Trans. Comm., Vol. COM-20, No. 3, pp. 568-570 (June 1972).

[BOCH1] Bochmann, G.V., "Finite State Description of Communication Protocols", Proc. of the Computer Network Protocols Symposium, Liege, Belgium, 13-15 Feb. 1978. (Also Pub. No. 236, Dept. d'Informatique, University of Montreal, July 1976.

[BOCH2] Bochmann, G.V., "Communication Protocols and Error Recovery Procedures", ACM Operating Systems Review, Vol. 9, No. 3, July, 1975.

[BOCH3] Bochmann, G.V. and Gecsei, J., "A Unified Method for Specification and Verification of Protocols", Proc. of IFIP Congress, Toronto, Canada, Aug. 1977, pp. 229-234. (Also: Pub. No. 247, Dept. d'Informatique, Univ. of Montreal, Canada).

[BOCH4] Bochmann, G.V., "Logical Verification and Implementation of Protocols", Proc. of 4th Data Comm. Symp., Quebec City, Canada, Oct. 1975, (IEEE). (Also:Pub. No.190, Dept. d'Informatique, University of Montreal, Canada).

[BRAN] Bran, D. and Joyner, W.Jr., "Verification of Protocols Using Symbolic Execution", Proc. Computer Network Protocols Symp., 13-15 Feb. 1978, Leige, Belgium.

[DANT] Dantine, A. and Bremer, J., "Modelling and Verification of End-to-End Transport Protocols", Proceedings of the Computer Network Protocols Symposium : Liege, Belgium, 13-15 Feb. 1978.

[DAY] Day, J.C., "A Bibliography on the Formal Specification and Verification of Computer Network Protocols", Proc. of the Computer Network Protocols Symp., Liege, Belgium, 13-15 Feb. 1978.

[DIJK] Dijkstra, E.W., "Two Starvation-free Solutions of a General Exclusion Problem", EWD 625, Prof. Dr. E.W. Dijkstra, Burroughs Research Fellow, Plataanstraat 5, 5671 Al Nuenens, The Netherlands. (See also Proceedings of IBM/University of Newcastle Seminar, Newcastle, England, 5-8 Sept. 1978.)

[ELLI] Ellis, C.A., "Consistency and Correctness of Duplicate Database Systems", Proc. Of Sixth ACM Symp. on Operating Systems Principles (Nov. 1977) 67-84.

[FINN] Finn, S.G., "Resynch Network Protocols", Proc. of 1977 Intern. Conf. on Comm., Chicago 1977 (IEEE).

[FOOX] Foox, E. and Merlin, P.M., "Modelling Validation and Implementation of a Computer Controlled Exchange using Protocols". In preparation.

[GOUD] Gouda, M.G., and Manning, E.G., "On the Modelling, Analysis and Design of Protocols - A Special Class of Software Structures: Proc. 2nd Internl. Conf. on Soft. Eng., San Francisco, CA., Oct. 1976, pp. 256-262. (IEEE Catalog No. 76CH1125-4C.) See also: Gouda and Manning, "Protocol Machines: A consise formal Model and its Automatic Implementation, Proc. of the Third Int. Conf. Comp. Comm., Toronto, 3-6 Aug. 1976.

[GUNT] Guntner, K.D., "Prevention of Buffer Deadlocks in Packet-switching Networks", Report presented at IFIP-IIASA Workshop on Data Comm., Laxenburg, Austria, 15-19 Sept. 1975.

[HARA1] Harangozo, J., "Protocol Definition with Formal Grammars", Proc. of the Computer Network Protocols Symposium, Liege, Belgium, 13-15 Feb. 1978.

[HARA2] Harangozo, J., "Formal Language Description of a Communication Protocol, KFKI-1977-92, Computer Development, Central Research Inst. for Physics, H1525, Budapest, P.O. Box 49, Hungary (HU ISSN 0368 5330) (ISBN 963 371 337 4).

[HOLT] Holt, A.W. and Commoner, F., "Events and Conditions", Proc. Project MAC Conf. on Concurrent Systems and Parallel Computation, MIT, June 1970.

[KAWA] Kawashima, H., Futami, K. and Kand, S., "Functional Specification of Call Processing by State Transition Diagrams", IEEE Trans. Comm. Tech., Vol. COMM-19, Oct. 1971.

[KELL] Keller, R.M., "Formal Verification of Parallel Programs", CACM, 7 (1976), pp. 371-384.

[KNOB] Knoblock, D.E., Loughry, D.C. and Vissers, C.A., "Insight into Interfacing", Spectrum, Vol. 12, No, 5, May 1975.

[KROG] Krogdahl, S., "Verification of Some link-Level Protocols", Computer Science Reports, Comp. Sci. Dept., University of Tromso, N-9001, TROMSO, Norway, June 1977.

[MERL1] Merlin, P.M., "A Study of the Recoverability of Computing Systems", Ph.D. Dissertation, Dept. of Info. and Comp. Sci., University of California, Irvine, Calif., Dec. 1974. See also: Merlin and Farber, "A Note on Recoverability on Modular Systems", Proc.AFIPS National Computer Conf., Vol. 44, pp. 695-699, May 1975.

[MERL2] Merlin, P.M. and Farber, D.J., "Recoverability or Communication Protocols - Implications of a Theoretical Study", IEEE Trans. Commun., Vol. COM-24, No. 9, Sept. 1976, pp. 1036-1043. See also: Merlin and Farber, 1976 International Conf. on Com., 14-16 June 1976, Philadelphia (IEEE): and ACM Operating Systems Review, Vol. 9, No. 3, July 1975

[MERL3] Merlin, P.M., "A Methodology for the Design and Implementation of Communication Protocols", IEEE Trans. on Comm., Vol. COM-24, No. 6, June 1976, pp. 614-621.

[MERL4] Merlin, P.M. and Segall, A., "A Failsafe Distributed Routing Protocol", E.E. Pub. No. 313 1978, Dept. of EE, Technion, Haifa, Israel, submitted for publication, (see also Segall, Merlin and Gallager, 1978 ICC, Toronto, Canada June 1978 (IEEE)).

[MERL5] Merlin, P.M. and Randell, B., "Consistent State Restoration in Distributed Systems", EE Pub. No. 315, Dec. 1977, Dept. of EE, Technion, Haifa, Israel, submitted for publication, (see also FTCS-8, Toulouse, France June 21-23, 1978 (IEEE)).

[MERL6] Merlin, P.M. and Schweitzer, P.J., "Deadlock Avoidance in Store-and-Forward Networks, I: Store-and-Forward Deadlock, II: Other Deadlock Types: to appear IEEE Trans. on Comm. (Draft version: RC-6624 and RC-6625, IBM Research, P.O. Box 218, Yorktown Heights, N.Y. 10598.) See also: Proc. 3rd Jerusalem Conf. on Information Technology (JCIT) 6-9 Aug. 1978 (NORTH-HOLLAND Pub. Co.).

[METC] Metcalfe, R.M. and Boggs, D.R., "Ethernet: Distributed Packet Switching for Local Computer Networks", CACM19, No. 7, 1976.

[MILL] Miller, R.E., "Graph Theoretic Models of Parallel Computation", Proceedings of IBM/University of Newcastle Seminar, Newcastle, England, 5-8, Sept. 1978.

[MULL] Mullery, A.P., "The Distributed Control of Multiple Copies of Data", RC5782, IBM-T.J. Watson Research Center, Yorktown Heights, N.Y. 10598, Dec. 1975.

[PDP11] PDP11 Peripherals Handbook, DEC, Maynard, Mass. 01754.

[PETE] Peterson, J.L., "Petri Nets", ACM Computing Surveys 9, 3 (Sept. 1977), pp. 223-252.

[PETR] Petri, C.A., "Kommunikation mit Automaten", Schriften des Rheinisch-Westfalischen Institutes fur Instrumentelle Mathematik an der Universitat Bonn, Heft 2, Bonn, W. Germany 1962: Tech. Rep. RADC-TR-65-337, Vol. 1, Rome Air Development Center, Griffis Air Force Base, N.Y., 1965, 89 pp.

[POST] Postel, J.B., "A Graph Model Analysis of Computer Communications Protocols", Ph.D. Dissertation, Dept. of Computer Science, UCLA, Los Angeles, CA., 1974.

[RUDI] Rudin, H., West, C.H. and Zafiropulo, P., "Automated Protocol Validation: One Chain of Development", Proc. of the Computer Network Protocols Symp., Liege, Belgium, 13-15 Feb. 1978.

[SDLC] IBM Synchroneous Data Link Control, General Information, GA27-3093-1, FILE No. GENL-09, IBM 1975.

[STEN] Stenning, N.V., "A Data Transfer Protocol", Computer Networks, 1 (1976), pp. 99-110.

[SUNS1] Sunshine, C.A., "Survey of Protocol Definition and Verification Techniques". Proc. of the Computer Network Protocols Symp., Liege, Belgium, 13-15 Feb. 1978.

[SUNS2] Sunshine, C.A., "Survey of Communication Protocol Verification Techniques", Proc. Symp. Computer Networks, NBS, Gaithersburg, Maryland, (Nov. 1976), pp. 24-26 (IEEE).

[SYMO] Symons, F.J.W., "Modelling and Analysis of Communication Protocols Using Petri Nets", Report No. 140 Telecomm. Systems Group, Dept. of Elect. Eng., Univ. of Essex, Sept. 1976. Also "A General Graphical Model of Processing Systems Using NEMS, A Generalization of Petri Nets", Report No. 141, ibid, Oct. 1976.

[VISS] Vissers, C.A., "Interface, A Dispersed Architecture", 3rd Annual Symp. on Computer Architecture, Computer Architecture News (ACM-SIGARCH), Vol. 4, Jan. 1976.

[WEST1] West, C.M., "General Technique for Communication Protocol Validation", IBM J. Res. and Dev., Vol. 22, No. 4, July 1978.

[WEST2] West, C.H. and Zafiropulo, "Automated Validation of a Communication Protocol: the CCITT X.21 Recommendation", IBM J. Res. and Dev., Vcl. 22, No. 1, Jan. 1978.

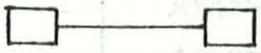
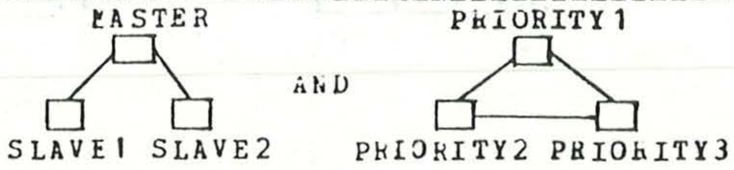
[X.21] "Recommendation X.21 (Revised) AP VI-No. 55E", Published by the CCITT (Internl. Telegraph and Telephone Consulting Committee), Geneva, Switzerland, March 1976.

[X.25] "Recommendation X.25", Vol. VIII.2, CCITT, Geneva 1976.

[YOEL] Yoeli, M. and Barzilai, Z., "Behavioral Descriptions of Communication Switching Systems Using Extended Petri Nets", Digital Processes, 3 (1977), pp. 307-320.

[ZAFI] Zafiropulo, P., "Protocol Validation by Dialogue Matrix Analysis", RZ816, IBM Zurich Research Laboratories, 8803 Ruschlikon, Switzerland, 1977.

TABLE 1: TYPES OF TOPOLOGY SETS

TYPE OF SET	A SET EXAMPLE
PAIR OF PARTIES	
SMALL SET OF GIVEN SMALL TOPOLOGIES	 MASTER SLAVE1 SLAVE2 AND PRIORITY1 PRIORITY2 PRIORITY3
A BOUNDED CLASS OF FINITE TOPOLOGIES (i.e. A SET OF A BOUNDED NUMBER OF TOPOLOGIES)	ANY LOOP OF UP TO 64 PARTIES
AN UNBOUNDED CLASS OF FINITE TOPOLOGIES (I.E. A SET OF AN UNBOUNDED NUMBER OF TOPOLOGIES)	-ANY LOOP OF A FINITE NUMBER OF PARTIES. ANOTHER EXAMPLE: -ANY TOPOLOGY OF A FINITE NUMBER OF PARTIES

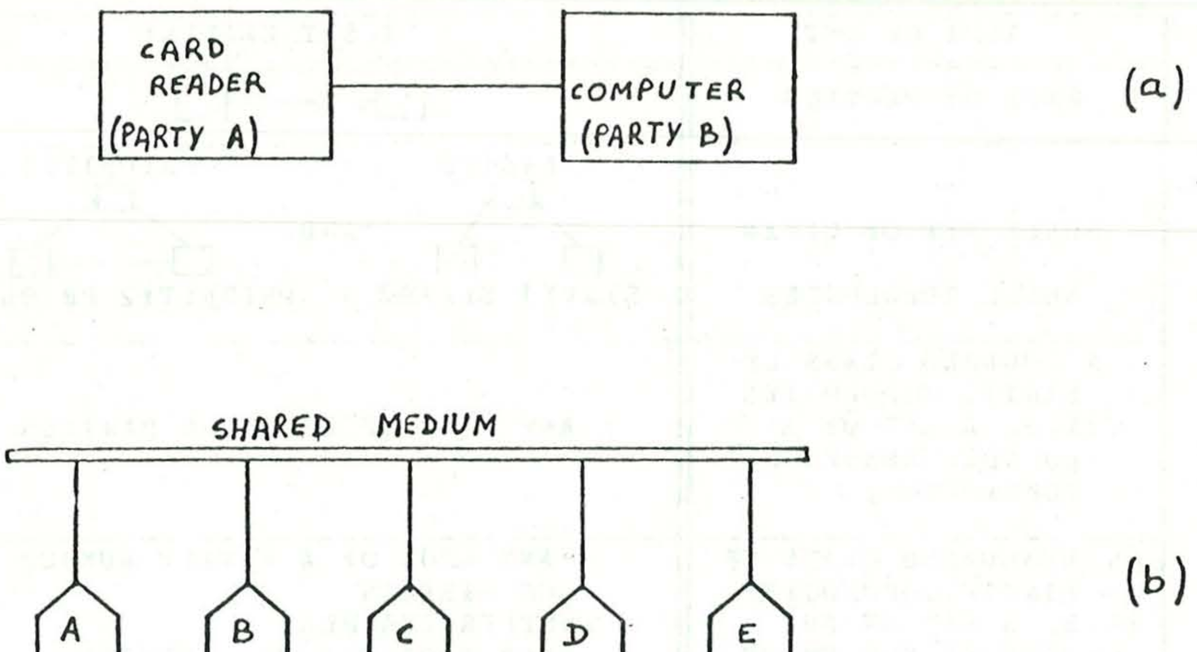


Figure 1 Configurations of Data Transfer Protocols

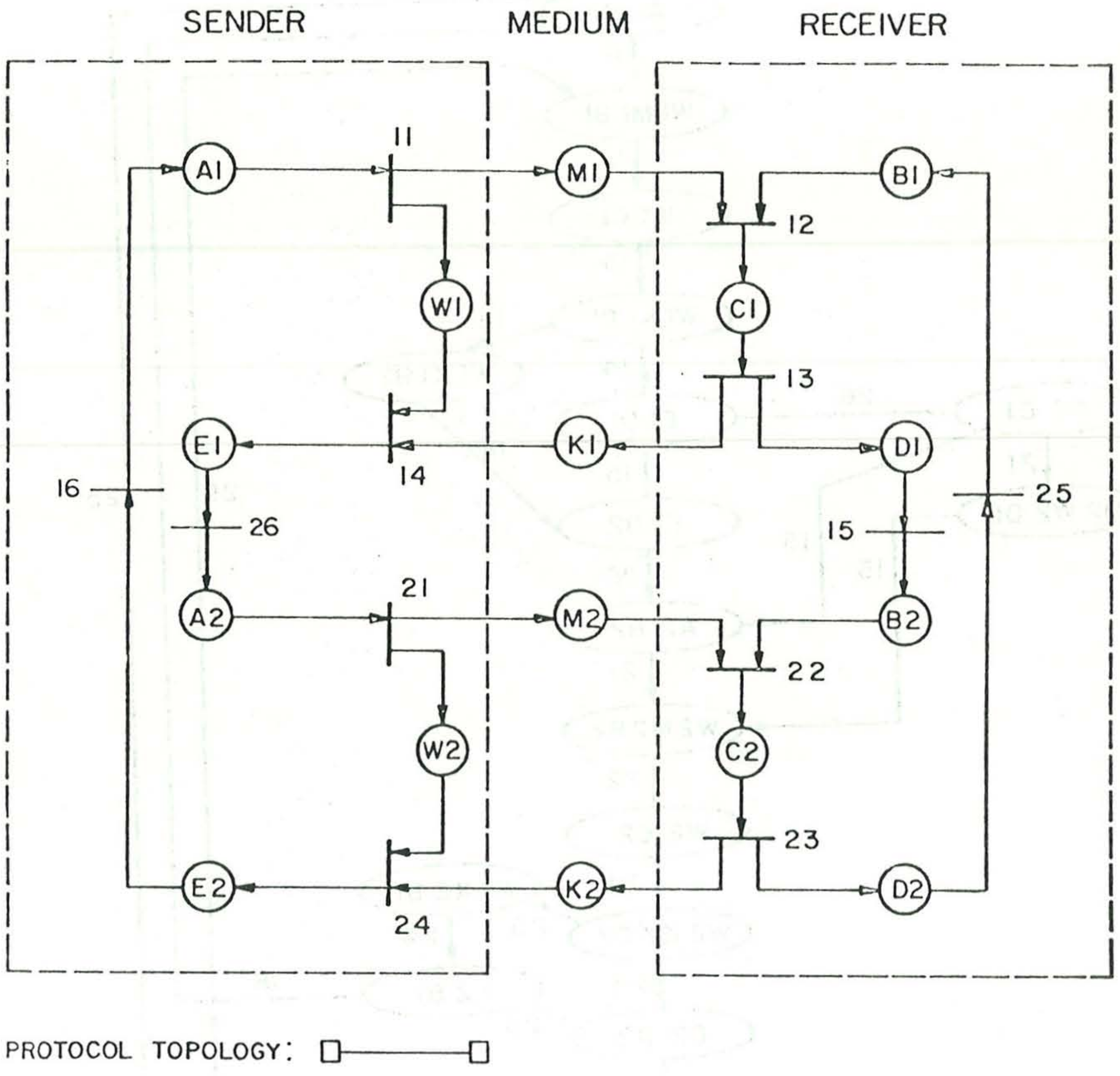


Figure 2 Petri Net Representation of the Alternating Bit Protocol Without Recovery

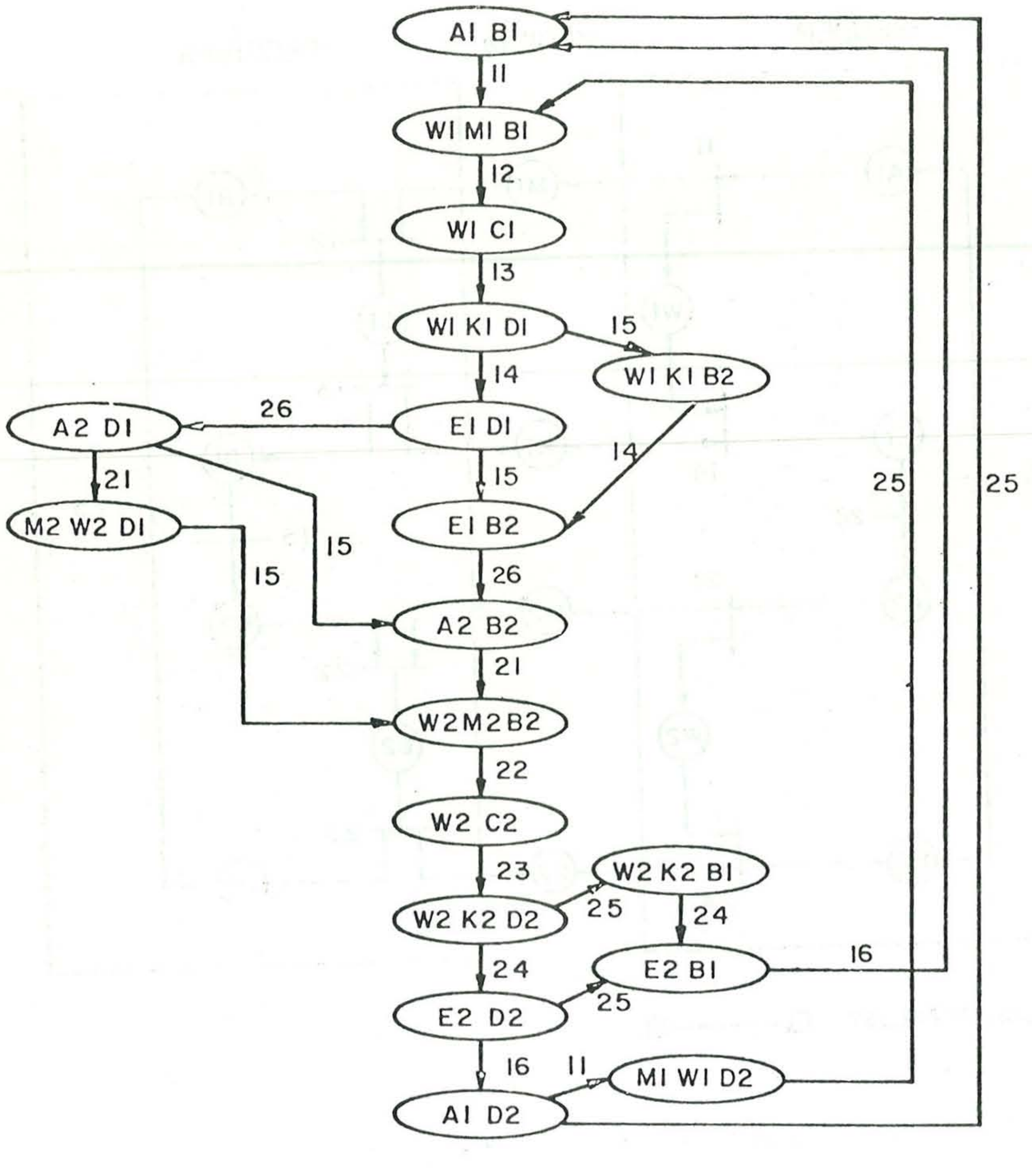


Figure 3 Token Machine for the Protocol of Figure 2

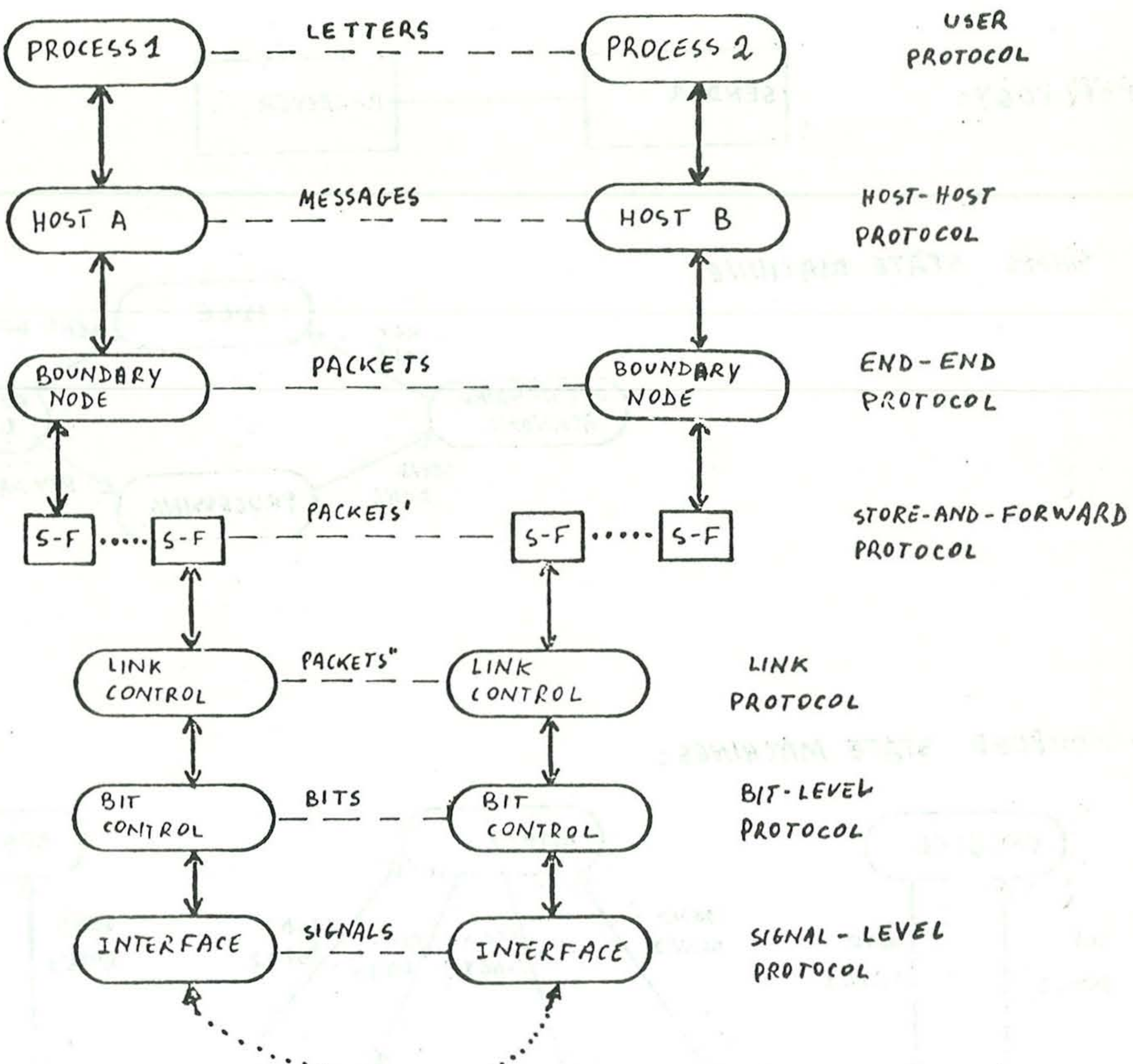


Figure 4 Typical Hierarchy of Protocols of a Computer Network

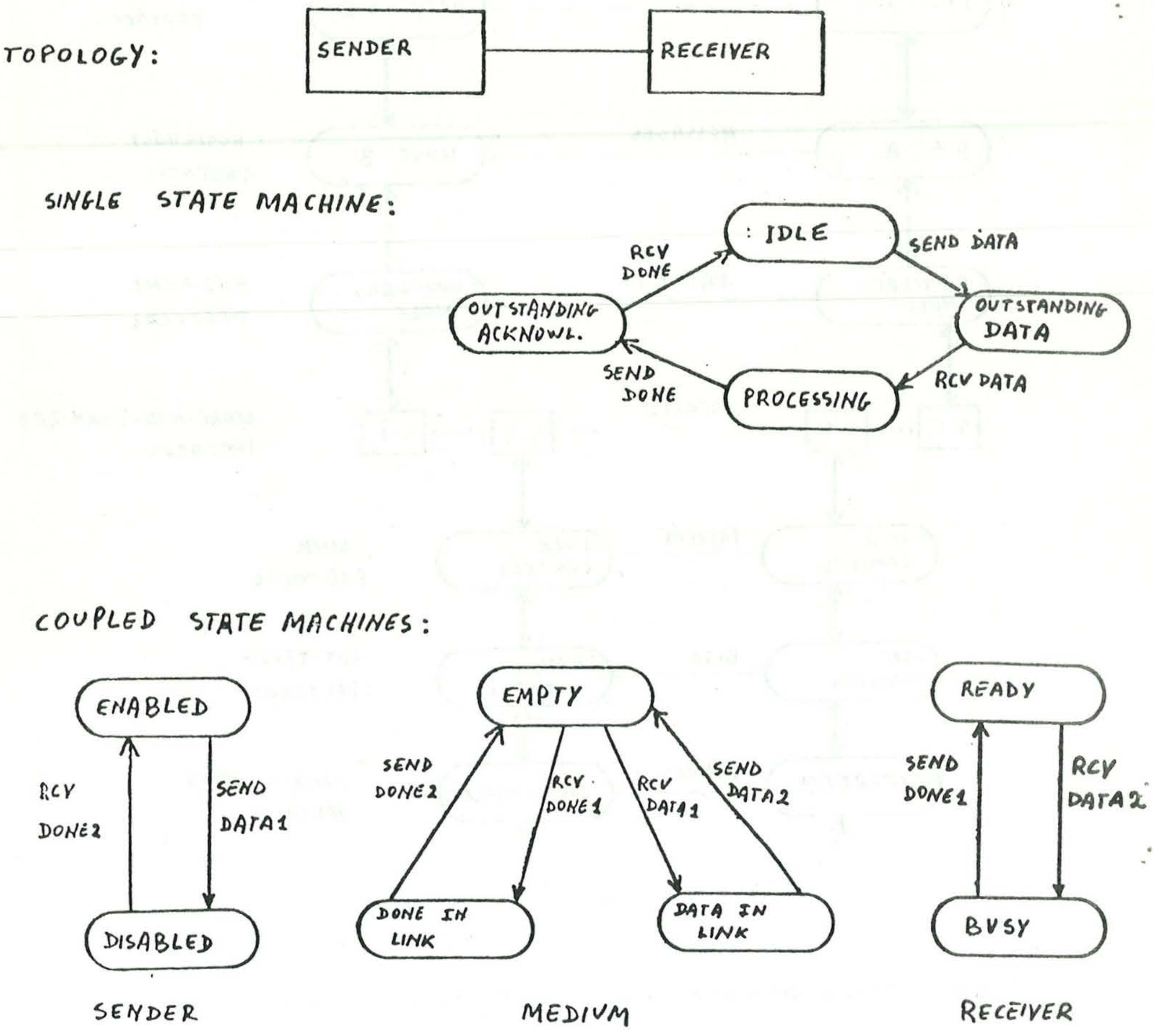


Figure 5 A State Machine Representation of a Simple Protocol

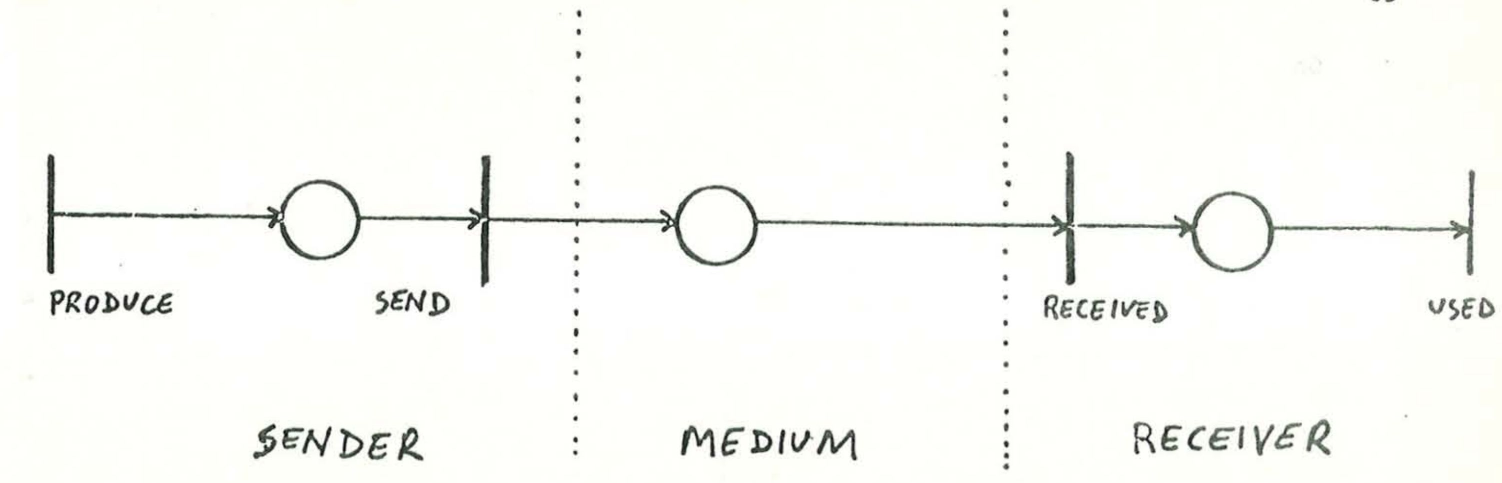


Figure 6 A Petri Net Representation of a Protocol which is not Representable by a Finite State Machine

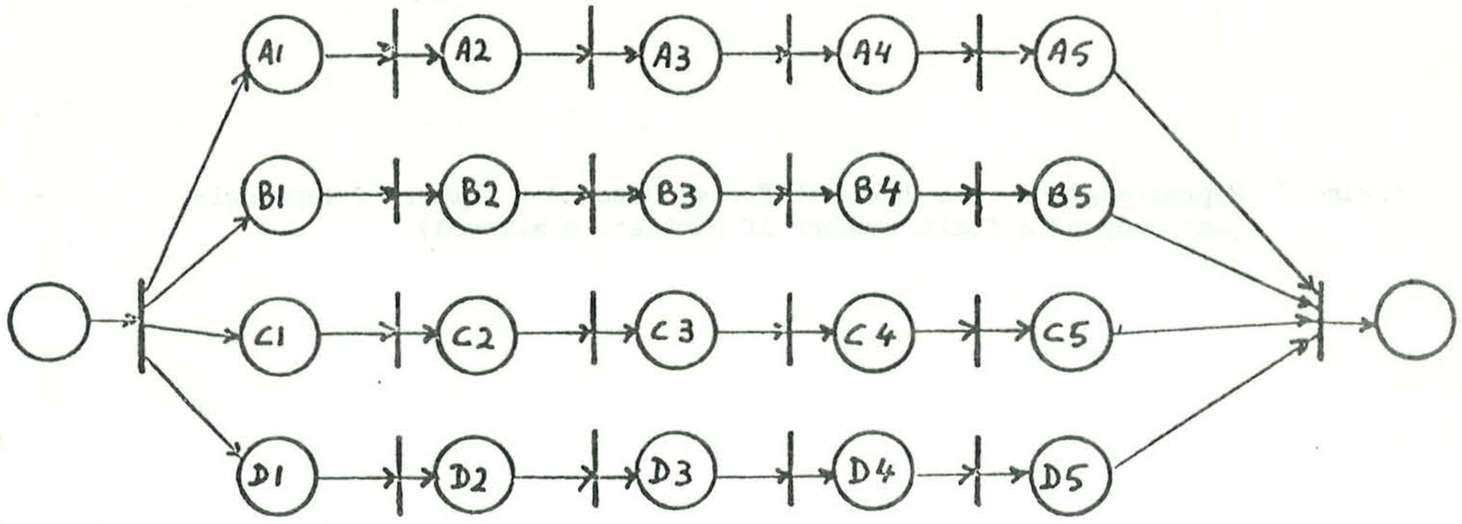


Figure 7 An example of a Petri Net which is difficult to represent by a Finite State Machine

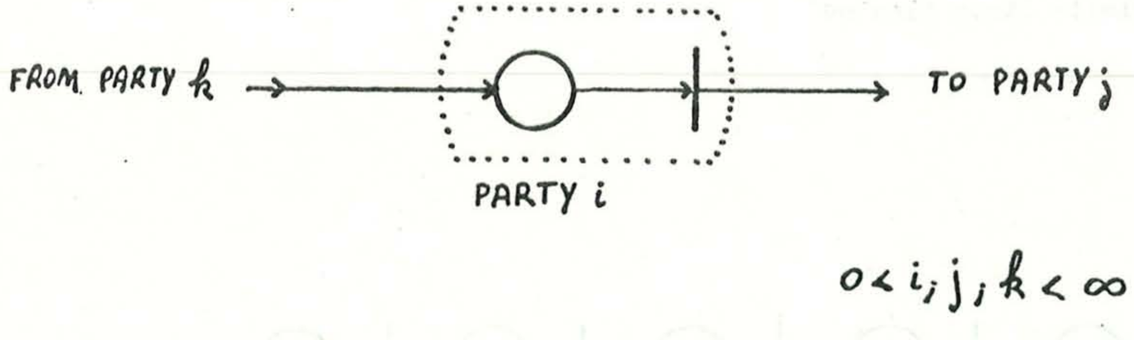


Figure 8 Representation of a Protocol for an Unbounded Number of Topologies (any loop of a finite number of parties is allowed)

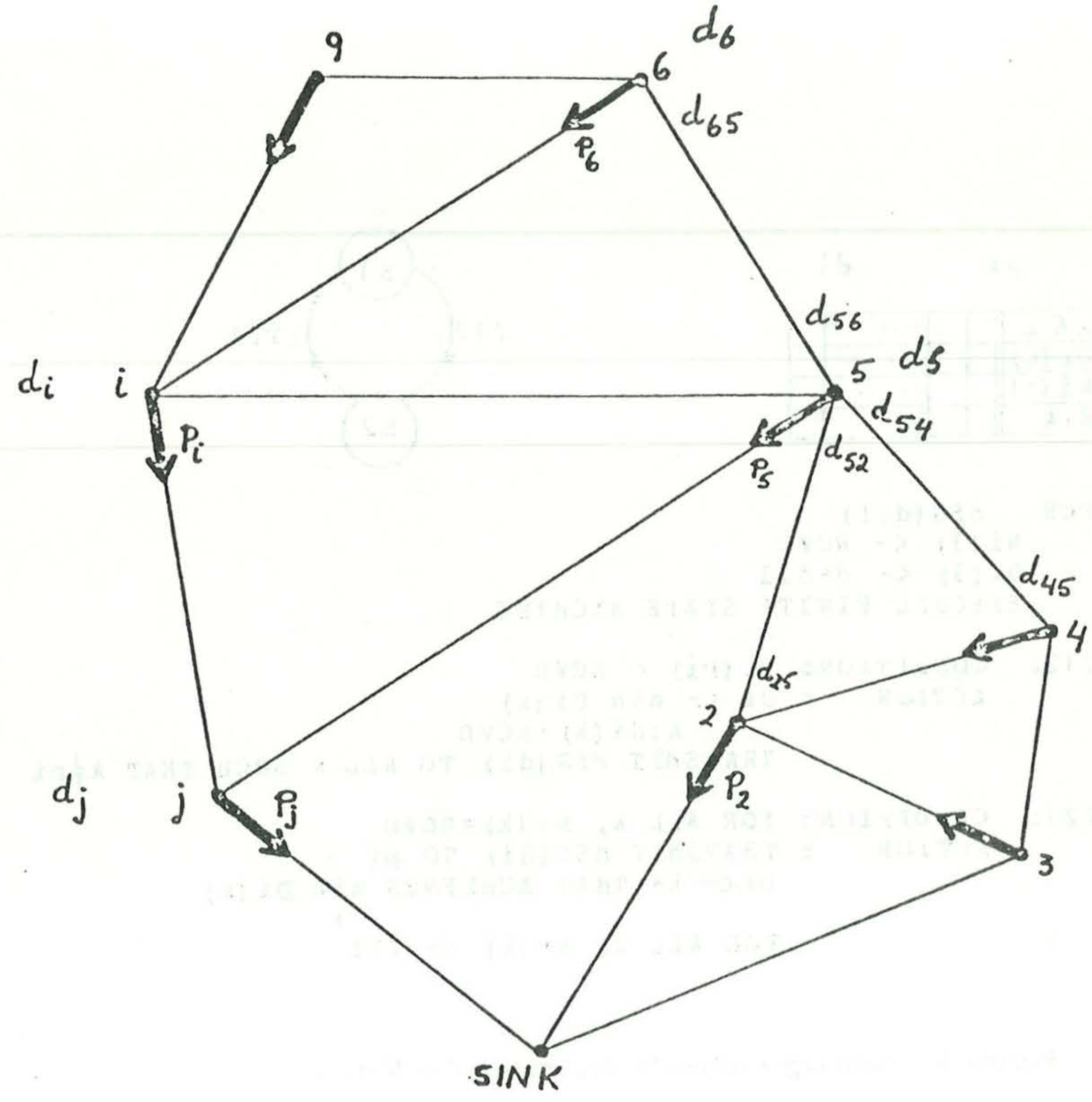


Figure 9 Example of a Network Using the Routing Protocol of Section 7

pi   di

k ∈ L			.....	
Di (k)			.....	
Ni (k)			.....	
dik			.....	

$$\begin{tikzcd} S1 \arrow[d, bend right=60, "T12"'] \\ S2 \arrow[u, bend right=60, "T21"'] \end{tikzcd}$$

```
FOR    MSG(d,l)
     Ni(l) <- RCVD
     Di(l) <- d+dil
     EXECUTE FINITE STATE MACHINE

T12:   CONDITION: Ni(Pi) = RCVD
       ACTION   : di <- min Di(k)
                        k:Ni(k)=RCVD
                  TRANSMIT MSG(di) TO ALL k SUCH THAT k≠pi

T21:   CONDITION: FOR ALL k, Ni(k)=RCVD
       ACTION   : TRANSMIT MSG(di) TO pi
                  pi <— k* THAT ACHIEVES min Di(k)
                                          k
                  FOR ALL k, Ni(k) <- NIL
```

Figure 10 Routing Protocol: Algorithm for Node i

K ∈ L			·······	
Ni(K)			-- ---	

$$\begin{tikzcd} S1 \arrow[d, bend left=60, "T12"] \\ S2 \arrow[u, bend left=60, "T21"] \end{tikzcd}$$

```
FOR  MSG(d,l)
     Ni(l) <- RCVD
     EXECUTE FINITE STATE MACHINE

FOR NEW_CYCLE
    EXECUTE FINITE STATE MACHINE

T12: CONDITION: NEW_CYCLE
     ACTION    : TRANSMIT MSG(di) TO ALL K SUCH THAT k≠pi

T21: CONDITION: FOR ALL k, Ni(K)=RCVD
     ACTION    : FOR ALL k, Ni(k) <- NIL
```

Figure 11 Routing Protocol: Algorithm for SINK