SYNCHRONIZATION ISSUES IN PROTOCOL TESTING

Behcet SARIKAYA
McGill University
School of Computer Science
Montreal, P.Q. Canada, H3A 2K6

Gregor v. BOCHMANN
Université de Montréal
Département d'I.R.O.
Montréal, P.Q. Canada, H3C 3J7

ABSTRACT

Protocol testing for the purpose of certifying the implementation's adherence to the protocol specification can be done with an architecture which includes a remote Tester and a local Responder processes generating specific input stimuli called test sequences. It is possible to adapt test sequence generation methods of finite state machines, namely transition tour, characterization and checking sequence methods to generate test sequences for protocols specified as incomplete finite state machines. For certain test sequences, the Tester or Responder processes are forced to consider the timing of an interaction in which they have not taken part; these test sequences are called nonsynchronizable. The three test sequence generation algorithms are modified to obtain synchronizable test sequences. Checking the protocol designs for intrinsic synchronization problems is also discussed.

1. INTRODUCTION

Protocol implementation assessment methods are used to determine that a particular protocol implementation (in the following simply called "Implementation " or "I") adheres to the specification of the protocol. There seems to be general agreement on a general architecture to be used for testing one or more layers of the OSI protocol hierarchy[Rayn 82, BoCe 82, INWG 82]: A remote tester(also called "Active Tester", "Tester" or "T" in short) and a supplementary local tester (also called "Test Responder", "Responder" or "R" in short) directly connected to the implementation, and playing the role of a protocol.

This paper addresses the problem of selecting test sequences for protocol implementation assessment. Assuming finite state machine models for protocol specification, methods developed for finite state machines implemented in hardware [Koha 78, NaTs 81] and software [Chow 78] can be applied to the selection of test sequences for protocols, as reported earlier[SaBo 82]. In the context of the proposed test architecture, however, certain problems of synchronization between the tester and the responder may arise.

The paper first gives a short review of the straightforward application [SaBo 82] of three finite state test sequence selection methods[NaTs 81, Chow 78, Koha 78] to protocol implementation assessment, and then explains in section 2 the nature of the possible synchronization problems. Section 3 discusses algorithms for selecting test sequences without synchronization problems, which are called in the following synchronizable. Some protocol examples are also given for which no complete synchronizable test sequence exists.

2. Synchronization Problems of Test Sequences

Test sequences reported in [SaBo 82] were generated with the assumption that Tester and Responder reside on the same machine. In the architecture of figure 1a, Tester and Responder functions are distributed over two sites and the two processes progress asynchronously. It is assumes in this paper that they are synchronized with one another only indirectly through the interactions with the Implementation under test.

This section contains a brief review of test sequence selection methods and theoretical considerations on the detection of synchronization problems.

2.1. Test Sequence Generation for Protocols

2.1.1. Transition Tour Method

An input sequence starting with the initial state and covering all transitions defined in the protocol specification is called a transition tour[NaTs 81] (usually the specification defines an incompletely specified machine as in figure 2). A transition tour for the Transport protocol [ISO TP] is shown in figure 3a.

2.1.2. [W-method]-Characterization Sequences

A characterization set (also called W-set) of a finite state machine consists of input sequences that can distinguish between the behaviors of every pair of states, . A set P consists of input sequences that take the machine to any of its states from the initial state, also it contains all these sequences concatenated with all possible inputs from these states. A characterization sequence is the concatenation of the two sets of sequences, P and W. Each sequence in P. W (. stands for concatenation) is applied starting with the initial state, hence a return to initial state is necessary after each sequence. These transfer sequences are called resets. (The set P defines a testing tree).

2.1.3. [D-Method]-Checking Sequences

A distinguishing sequence (in short DS) is an input sequence for which the output sequence produced by the machine is different for each initial state. A checking sequence consists of two parts: first a state recognition part and the a transition checking part. The state recognition part starts with the initial state and displays the response of the machine to the sequence DS. DS for the purpose of identifying the states both prior and after the application of DS. In the transition checking part the graph defined as:

$$TC = U x_i . \ DS$$

where U stands for set union and the x_i are the machine transitions to be checked, is traversed. Transfer sequences are included in both parts also in between the parts whenever necessary to provide continuity.

2.2. Basic Interaction Model

The Implementation, Tester and Responder, shown in figure 1a, are modelled as processes each represented as a finite-state machine that communicate by exchanging messages through FIFO queues, as shown in figure 1b. The Tester and Responder may send or receive a message to/from the Implementation when they execute a state transition. I can receive a message from T or R when it executes a state transition or it can send a message to one or both of T or R after receiving a message from one of them. The system has a predefined initial state with all queues empty and all three processes in their initial states.

2.2.1. Basic Interaction Sequences (BIS)

A sequence of transitions of the Implementation, Tester and Responder defines a transition sequence for each process. In the following, abstraction is made from the particular transitions, only the information whether an input is received (R) or an output is sent (S) is recorded, and over which FIFO queue. Such an abstracted sequence is called basic transition [RuWe 82]. The individual interactions, i.e. sends and receives, are called basic interactions. For example the basic interaction corresponding to the transition in state 1 under input CR (connect-request PDU) of the protocol of figure 2 is to receive a message through Q^{TI} from T and send a message through Q^{IR} to R and hence is said to correspond to the basic interaction $R^{IT}S^{IR}$. All basic interactions of process I under this model are enumerated in Table 1a. It is noted that some of the entries in Table 1a (namely R^{IR}, $R^{IR}S^{IR}$ and $R^{IR}S^{IR}S^{IT}$) do not occur for the Transport protocol of figure 2, but they may occur in other cases.

Test sequences discussed in this section are composed of transitions of process I each starting in the initial state. Hence a test sequence, such as the one in figure 3a, can be easily converted into a corresponding basic interaction sequence (BIS) by replacing each transition by its corresponding basic interactions.

From a given test sequence, it is possible to obtain BISes for T and R as well. Since there might be some execution steps of I that do not involve any interaction with either T or R, these steps will be shown as ^ in the corresponding basic interaction sequence. The BIS for T corresponding to a given BIS for I can be obtained from the latter by replacing an R^{IT} with S^{TI} , and an S^{IT} with R^{TI} and an execution step not involving "T" by ^. A BIS for R can be obtained in a similar way.

2.3. Synchronization

In the following, a synchronization problem in the test sequence of figure 3a is first discussed as an example, and a theorem with its corollary is stated and proved.

BISes for the three processes of figure 1b corresponding to the first eigth transitions of the test sequence of figure 3a are as follows:

$$\begin{array}{ll}
I: & R^{IT}S^{IR}R^{IR}S^{IT}R^{IT}R^{IT}R^{IT}R^{IT}S^{IR}R^{IT}S^{IT}\overset{*}{R}{}^{IR}S^{IT} \\
T: & S^{TI} \qquad\qquad R^{TI}S^{TI}S^{TI}S^{TI}S^{TI} \quad S^{TI}R^{TI} \quad R^{TI} \\
R: & \quad R^{RI}S^{RI} \quad \hat{} \quad \hat{} \quad \hat{} \qquad R^{RI}\hat{} \qquad S^{RI}
\end{array}$$

A scenario of the execution of these BISes is as follows:Since the BIS for T starts with an S, process T starts the testing. R receives a message from I subsequent to the message sent to I by T. It responds with another message. Then T receives a message and sends four consecutive messages. The first three of these are ignored by I. R receives a message as a response to the last one. T sends and receives a message. The next action, by R, is to send a message (marked by "✗" above). It is difficult to synchronize the processes in this case, since R did not take part in the last step (the ^ before the S in the BIS for R indicates this fact). A possible solution to this synchronization problem would be for R to know the time period of the last step and to wait that much before sending. However, this is undesirable since real time constrains are difficult to realize. An error in the timing might cause the process I to make a different transition than the one expected by the test sequence, resulting in an unexpected response from I. Another solution would be direct synchronization between T and R; but this is not foreseen in the testing architecture of figure 1.

Theorem. Test sequences have synchronization problems if the BISes for T and/or R have any sends (S) preceeded by one or more ^s.

Proof. Assume that the BIS for R has one S preceeded by one or more ^s. From Table 3a it is easily seen that process I should have an interaction with process T whenever there is a ^ in the BIS for R. Since R has no way of knowing the duration of these local exchanges R faces a synchronization problem. The proof can be carried out similarly for T.

Corollary. During the testing with a synchronizable test sequence, the interpretation of the responses of the implementation is unambigious and simple.

This means that the ambigious situation where the Implementation has at least one message in both of its input queues is avoided by synchronizable test sequences because of the fact that the number of messages is always zero in one of the queues Q^{TI} and Q^{RI} which follows from the theorem.

2.4 Synchronizable Pairs of Transitions

Two consecutive basic transitions of I will be called a synchronizable pair of transitions if the second transition can follow the first one without generating a synchronization problem. For example R^{IR} followed by R^{IT} would violate synchronization because in the corresponding BIS for process T, S would be preceeded by ^. Similarly R^{RI} and $R^{IR}S^{IR}$ cannot be followed by R^{IT} or $R^{IT}S^{IT}$ or $R^{IT}S^{IT}S^{IR}$. Also R^{IT} and $R^{IT}S^{IT}$ cannot be followed by R^{IR} or $R^{IR}S^{IR}$ or $R^{IR}S^{IR}S^{IT}$. All nonsynchronizable pairs of interactions are listed in Table 1b.

This concept is useful for checking whether a given test sequence is synchronizable. It is (necessary and) sufficient that any two subsequent transitions of the sequence correspond to a synchronizable pair of basic transitions. The test sequences shown in figures 3a were derived in [SaBo 82] without concern for possible synchronization problems. It is easily seen that they contain synchronization problems, as indicated by "✗"s. The sequence of figure 3a contains two violations, both of the type: $R^{IT}S^{IT}R^{IR}S^{IT}$ which is one of the pairs listed in Table 1b. The characterization sequence given in [SaBo 82] contains four violations of the same kind. Also the checking sequence for the Transport protocol given in [SaBo 82] has three violations of this type.

All the nonsynchronizable pairs of transitions of Table 1b could be encountered in test sequences generated for real protocols unless precautions are taken, as explained in section 3.

2.5. Protocol Specifications with Intrinsic Synchronization Problems

For certain protocol specifications, it is impossible to avoid synchronization problems. Such a situation occurs in the case that a transition p_j from state j to state k is of one of the types

$$\begin{tikzcd}\boxed{\text{State } i} \arrow[r, "p_i"] & \boxed{\text{State } j} \arrow[r, "p_j"] & \boxed{\text{State } k}\end{tikzcd}$$

$[R^{IR}, R^{IR}S^{IT}, R^{IR}S^{IR}, R^{IR}S^{IR}S^{IT}]$, and each transition p_i entering state j is of one of the types $[R^{IT}, R^{IT}S^{IT}]$. Then each pair $p_i j_i$ is a nonsynchronizable pair of transitions. Therefore the execution of the transition p_j implies a synchronization problem. We call such a transition nonsychronizable. A dual situation exists for the case that all p_i are of one of the types $[R^{IR}, R^{IR}S^{IR}]$ and p_j of $[R^{IT}, R^{IT}S^{IT}, R^{IT}S^{IR}, R^{IT}S^{IT}S^{IR}]$. We call a state nonsynchronizable if it can only be reached through nonsynchronizable transitions. For example, the state k above is such a state.

Protocol specifications having nonsynchronizable transitions and/or states are called intrinsically nonsynchronizable. It is clear that any complete test sequence generated for such specifications will carry synchronization problems. By inspection it can be seen that the Transport protocol of figure 2 does not have any nonsynchronizable transitions hence it does not have any intrinsic synchronization problems.

It has been shown that X.25 DTE protocol [CCITT 81] contains some nonsynchronizable transitions.

3. Generation of Synchronizable Test Sequences

Each test sequence generation method discussed in section 2 may give rise, for a given protocol specification, to different test sequences depending on the way the method is implemented. It is clear from the discussion of section 2 that some of these sequences are not applicable in the test architecture of figure 1a because they violate the synchronization rules. The different methods can be adopted to generate only synchronizable test sequences. These adaptations are specific to each method, as explained below. However, the specification should be checked first to see if it is intrinsically nonsynchronizable. If not, one of the algorithms described below may be applied to obtain a synchronizable test sequence. The basic approach for all these algorithms is to check each new transition added to the sequence in order to see whether it is synchronizable with its predecessor. This check is based on Table 1b which lists all nonsynchronizable pairs of transitions.

3.1. Transition Tours

Any graph traversal algorithm such as the one given in [Tarj 72] can be modified to obtain a transition tour. Each transition to be added to the sequence by the algorithm is first checked whether it forms a synchronizable pair together with the last transition of the sequence (using Table 1b). If it is not synchronizable a different transition from the present state is considered. If no suitable transition exists from the present state, the selection algorithm bactracks to the previous state continuing the tour from there in a different way. This process continues until all the transitions of the machine are covered. In general, it may be necessary to deviate from the goal of obtaining minimum length sequences.

Applying such an algorithm to the Transport protocol, the transition tour of figure 3b is obtained. The length of this sequence is 34, as in figure 2; in this case the length is not increased.

3.2. Characterization Sequences

Algorithms to find a W-set and to construct a testing tree (and hence to calculate P. W, without resets) are given in [Chai 81]. Any shortest path finding algorithm, such as the one in [Even 79], can be used for determining the resets. Synchronizable characterization sequences can be obtained in three steps as follows:

In Step 1, all subsequences of P. W (without resets) are checked for synchronization problems using a "subsequence checking algorithm" which checks all pairs of consecutive transitions in a sequence for synchronization problems, using Table 1b. If a subsequence of P. W has synchronization problems, the use of a different W set or testing tree P may be considered, possibly leading to longer sequences.

In Step 2, each subsequence of P. W is completed by appending a synchronizable reset sequence using a backtracking algorithm similar to the one for transition tours explained above.

In Step 3 the subsequences obtained in Step 2 are merged together to obtain a single synchronizable test sequence. Any "concatenation algorithm" could be used which puts the subsequences in such an order that no synchronization problem is generated.

A synchronizable characterization sequence for the Transport protocol is obtained from the same testing tree, containing 69 transitions, three more than the sequence of [SaBo 82]. This is due to longer reset sequences.

3.3. Checking Sequences

Ignoring the problem of synchronization, an algorithm for finding a DS can be found in [Koha 78], and algorithms for state recognition and transition checking parts are reported in [Gone 70]. Shortest path algorithms can be used for finding transfer sequences.

The following measures are proposed to obtain synchronizable test sequences:

(a)A syncronizable DS must be found. In general, it may not be a minimal one.

(b)The state recognition part obtained according to [Gone 70] is checked using the "subsequence checking algorithm" mentioned above. In case of synchronization problems, changing the transfer sequences should first be considered. The use of a different DS may also be considered.

(c)The transition checking step is checked with a two-part procedure. First each subsequence x_i. DS in the set TC as defined in section 2 is checked by the "subsequence checking algorithm". If one of the tests fails a different DS should be generated, if it exists. In the second step, the transition checking part as a whole is checked for synchronization. In the case of synchronization problems, a different order of the subsequences and/or different transfer sequences should be considered.

(d)Finally, the state recognition and transition checking parts are combined using an appropriate transfer sequence.

A synchronizable checking sequence for the Transport protocol has been obtained containing 3 more transitions than the sequence reported in [SaBo 82].

4. SUMMARY and CONCLUSIONS

Test sequence generation methods (transition tours, W- and D- methods) are applicable to protocols specified as incomplete finite state machines. The transition tour has a limited, the other methods have full fault detection capabilities. The transition tour method is generally applicable, the application of the other two methods require certain conditions, i.e. the protocol possessing a W-set or DS, respectively.

With a remote testing architecture, as shown in figure 1, the synchronization between the Tester and Responder modules becomes an issue. The design of these modules is simplified if the selected test sequence does not have any synchronization problems. It is shown in section 3 that test sequences can be checked for synchronization problems if each interaction of a test sequence is associated with Tester or Responder module. Moreover, avoiding the synchronization problems relates to the design of the protocol. A protocol should be first checked for intrinsic synchronization problems and modified if necessary.

Synchronizable test sequences can be generated using modified versions of the algorithms developed for each of the methods. Longer test sequences migth be the price to pay.

More research is needed to apply the test sequence selection methods to protocols specified with extended state transition model [ISO 81b] while taking care of possible parameter values of the interactions and additional state variables.

Acknowledgements

The authors would like to thank E. Cerny for his helpful critisism of the paper.

This work was supported in part by the NSERC of Canada.

5. REFERENCES

[Boch 82] G.v. Bochmann, et al., "Experience with Formal Specifications Using an Extended State Transition Model", IEEE Trans. on Comm., Nov. 1982.

[BoCe 82] G.v. Bochmann, E. Cerny, "Protocol Assessment", Report of DOC of Canada, Feb.1982.

[CCITT 81] "Recommendation X.25", CCITT SGVII/WP2 pp. 100-190, fascicle VIII.2, Sept.1981.

[Chai 81] H. Chaigne, et al., "Un generateur de tests pour systemes modelises par automates d'etats finis", BIGRE (of IRISA, Rennes, France) No.27, Dec.1981.

[Chow 78] T. S. Chow, "Testing Software Design Modeled by Finite State Machines", IEEE Trans.on SE-4, No.3, 1978.

[Even 79] S. Even, "Graph Algorithms", Computer Science Press 1979.

[Gone 70] G. Gonenc, "A Method for the Design of Fault Detection Experiments", IEEE Trans.on Comp. Vol C19 No6, 1970.

[INWG 82] Several papers in the proceedings of 2. Int. Workshop on Protocol Spec., Testing and Verification, North-Holland Publ.1982.

[ISO TP] ISO/CCITT "Draft Transport Protocol Specification", Dec.1981.

[ISO 81b] ISO "A FDT Based on an Extended State Transition Model" working document of Subgroup B, ISO TC97/SC161 WG1 Dec.1981.

[Koha 78] Z. Kohavi, "Switching and Finite Automata Theory", McGraw Hill NewYork, 1978.

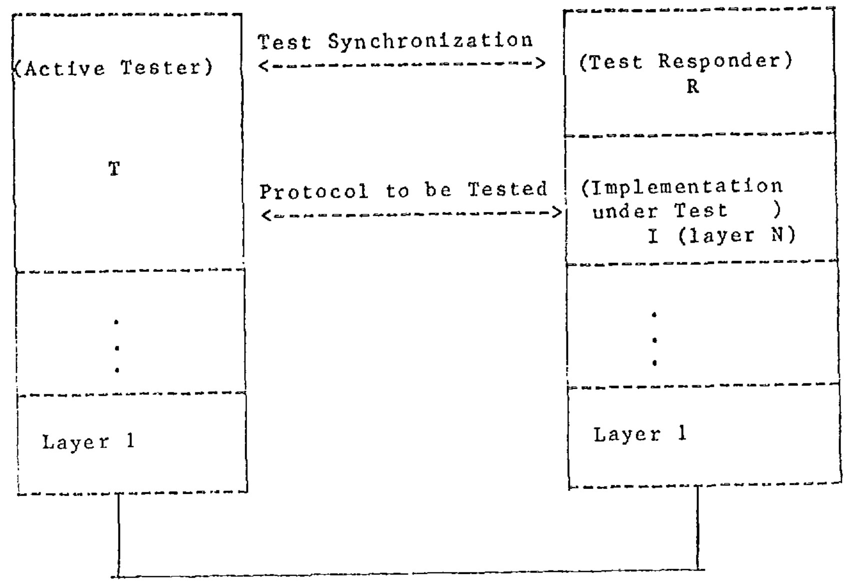
[NaTs 81] S. Naito, M. Tsunoyama, "Fault Detection for Sequential Machines by Transition Tours", Proceedings of IEEE Fault tolerant computing conference 1981.

[Rayn 82] D. Rayner, "A System for Testing Protocol Implementations", in (INWG 82) , also to be published in Computer Networks.

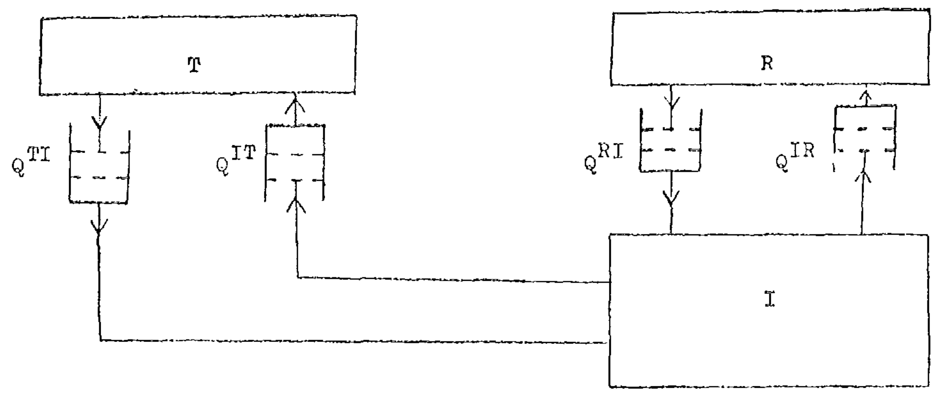
[RuWe 82] R. Rubin, C.H. West, "An Improved Protocol Validation Technique", Computer Networks, May 1982.

[SaBo 82] B. Sarikaya, G.v. Bochmann, "Some Experience with Test Sequence Generation for Protocols", in INWG 82.

[Tarj 72] R. Tarjan, "Dept-First Search and Linear Graph Algorithms", SIAM J. Computing, Vol 1, No 2, 1972.



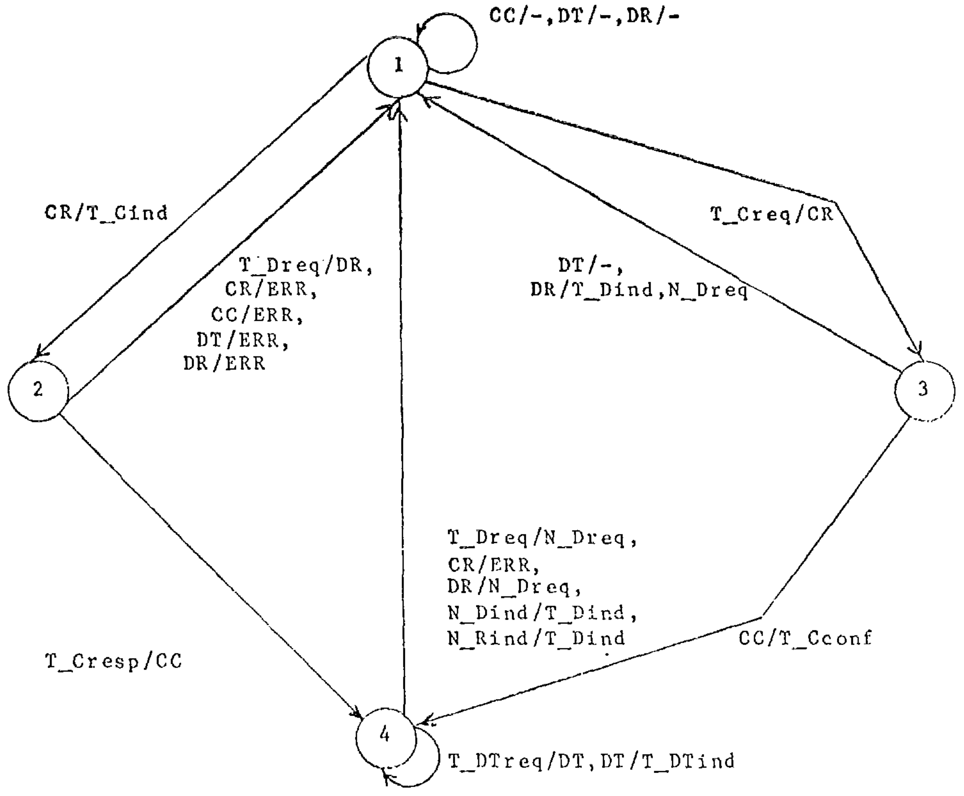
a)An architecture for testing a (N)-layer protocol implementation in the context of the OSI Reference Model.



b)Basic Interaction Model of the test architecture in (a)
Figure 1.Test Architecture and its Interaction Model

```
1 R.T_Creq 3 T.CC      4 R.T_Dreq 1 T.CR      2 R.T_Dreq 1 T.CC 1 T.DT 1
  T.CR        R.T_Cconf  T.N_Dreq   R.T_Cind    T.DR          -      -

T.DR 1 T.CR     2 R.T_Cresp 4 R.T_DTreq 4 T.CR   1 T.CR     2 R.T_Cresp 4
 -      R.T_Cind  T.CC          T.DT        T.ERR    R.T_Cind      T.CC

 T.DT       4 T.N_Rind 1 R.T_Creq 3 T.DT 1 T.CR     2 R.T_Cresp 4
 R.T_DTind    T.T_Dind   T.CR           -    R.T_Cind    T.CC

 T.N_Dind 1 R.T_Creq 3 T.[illegible]                      1 T.CR     2 R.T_Cresp 4
 R.T_Dind   T.CR         R.T_Dind,T.N_Dreq   R.T_Cind  T.CC

 T.DR       1 T.CR     2 T.CR  1 T.CR       2 T.CC  1 T.CR       2 T.DT  1
 T.N_Dreq    R.T_Cind  T.ERR   R.T_Cind     T.ERR   R.T_Cind     T.ERR

 T.CR     2 T.DR  1
 R.T_Cind  T.ERR
```

Figure 3b.Synchronizable transition tour for the transport protocol


CC/-,DT/-,DR/-
1
CR/T_Cind
T_Dreq/DR, CR/ERR, CC/ERR, DT/ERR, DR/ERR
DT/-, DR/T_Dind,N_Dreq
T_Creq/CR
2
3
T_Dreq/N_Dreq, CR/ERR, DR/N_Dreq, N_Dind/T_Dind, N_Rind/T_Dind
T_Cresp/CC
CC/T_Cconf
4
T_DTreq/DT,DT/T_DTind

Notation for Input Events

From Tester		From Responder	
CR	Connect_request PDU	T_Creq	T_Connect_Request
CC	Connect_Confirm PDU	T_Dreq	T_Disconnect_Request
DT	Data_Request PDU	T_Cresp	T_Connect_Response
DR	Disconnect_Request PDU	T_DTreq	T_Data_Request
N_Dind	Network_Disconnect_Indication		
N_Rind	Network_Reset_Indication		

Notation for Output Events

To Tester		To Responder	
ERR	Error PDU	T_Cind	T_Connect_Indication
N_Dreq	Network_Disconnect_Request	T_Dind	T_Disc_Indication
CC	Connect_Confirm PDU	T_Cconf	T_Connect_Confirm
DT	Data_Request PDU	T_DTind	T_Data_Indication
DR	Disconnect_Request PDU		
N_Dind	N_Disconnect_Indication		
N_Rind	N_Reset_Indication		

Figure 2. Finite state machine for the class-0 transport protocol

(a)	(b)
R^{IT}	$R^{IT}R^{IR}$, $R^{IT}R^{IR}S^{IT}$, $R^{IT}R^{IR}R^{IR}$, $R^{IT}R^{IR}S^{IR}S^{IT}$
R^{IR}	$R^{IR}R^{IT}$, $R^{IR}R^{IT}S^{IT}$, $R^{IR}R^{IT}S^{IR}$, $R^{IR}R^{IT}S^{IT}S^{IR}$
$R^{IT}S^{IT}$	$R^{IT}S^{IT}R^{IR}$, $R^{IT}S^{IT}R^{IR}S^{IT}$, $R^{IT}S^{IT}R^{IR}S^{IR}$, $R^{IT}S^{IT}R^{IR}S^{IR}S^{IT}$
$R^{IT}S^{IR}$	-
$R^{IR}S^{IT}$	-
$R^{IR}S^{IR}$	$R^{IR}S^{IR}R^{IT}$, $R^{IR}S^{IR}R^{IT}S^{IT}$, $R^{IR}S^{IR}R^{IT}S^{IR}$, $R^{IR}S^{IR}R^{IT}S^{IT}S^{IR}$
$R^{IT}S^{IT}S^{IR}$	-
$R^{IR}S^{IR}S^{IT}$	-

Table 1 .List of Basic transitions(a) and nonsynchronizable pairs of transitions(b) for an Implementation(I)

```
1 T.CR       2  R.T_Dreq  1  T.CC   1  T.DT   1  T.DR   1  T.CR        2 T.CR  1
  R.T_Cind      T.DR         -         -         -         R.T_Cind      T.ERR
 *
R.T_Creq 3  T.DT    1  T.CR      2  R.T_Cresp  4  R.T_DTreq  4  T.DT       1
T.CR        T.ERR      R.T_Cind     T.CC          T.DT          R.T_DTind

R.T_Dreq 1  T.CR      2  T.CC    1  T.CR      2  T.DT    1  T.CR      2
T.N_Dreq    R.T_Cind     T.ERR      R.T_Cind     T.ERR      R.T_Cind
              *
T.DR       1  R.T_Creq 3  T.DR                1  R.T_Creq 3  T.CC          4
T.ERR         T.CC        R.T_Dind,T.N_Dreq      T.CC        R.T_Cconf

T.CR   1  T.CR      2  R.T_Cresp  4  T.DR      1  T.CR      2  R.T_Cresp 4
T.ERR     R.T_Cind     T.CC          T.N_Dreq     R.T_Cind     T.CC

T.CR      2  R.T_Cresp  4  T.N_Rind  1
R.T_Dind     T.CC          R.T_Dind

 Notation for Transitions:
 Start State   Input Initiating Side.Input Primitive   Final State
               Output Receiving Side.Output Primitive
 "*"s are used to indicate nonsynchronizable transitions.
```

Figure 3a.A transition tour for the transport protocol