Automatic Test Case Generation Using Multi-protocol Test Method

Soo-in Lee, Yongbum Park, and Myungchul Kim
School of Engineering
Information and Communications University
Taejon, Korea
{elsie, ybpark, mckim}@icu.ac.kr

Hee Yong Youn
Dept. of ECE
SungKyunKwan University
Suwon, Korea
youn@ece.skku.ac.kr

Ben Lee
Dept. of ECE
Oregon State University
Corvallis, OR 97331
benl@ece.orst.edu

Abstract **– A method for testing multi-protocol implementation under test (IUT) with a single test suite has been proposed in the literature. It tests a multi-protocol IUT in an integrated way compared to the conventional method, where single-layer test method and single-layer embedded test method are applied separately to the upper layer protocol and lower layer protocol, respectively. However, it did not consider how to generate the test cases automatically but proposed only an approach for the test method. This paper proposes an algorithm called Multi-protocol Test Method (MPTM) for automatic test case generation based on that approach. With the MPTM, a multi-protocol IUT consisting of two protocol layers is modeled as two Finite State Machines (FSMs), and the relationships between the transitions of them are defined as a set of transition relationships, pre-execution and carried-by. The proposed algorithm is implemented and applied to a simplified TCP/IP and B-ISDN Signaling/SSCOP. The MPTM is able to test the multi-protocol IUT even though the interfaces between the protocol layers are not exposed. It also allows the same test coverage as conventional test methods with much smaller number of test cases and operations.**

1. INTRODUCTION

Conformance Testing Methodology and Framework (CTMF) was published as an international standard by ISO/IEC JTC1 [2]. It focuses on testing single-layer protocols and proposes successive use of single-layer embedded test method for testing a multi-protocol implementation under test (IUT). Here, the single-layer test method is applied to the highest layer of multi-protocol, while the single-layer embedded test method is applied to other layers of multi-protocol IUT except the highest layer.

In light of this, this paper presents a methodology called Multi-protocol Test Method (MPTM), which is able to test a multi-protocol IUT with a single test suite. It combines the single-layer test method used for the highest layer protocol with the single-layer embedded test method used for the lower-layer protocol. With the MPTM, the highest layer and the embedded layers are tested at the same time with a single test suite by directly controlling and observing the lower point of control and observation (PCO) and doing that indirectly for the hidden PCO located between the two protocol layers.

Currently, in the methods such as Multiprotocol Label Switching (MPLS) [5], more than one layer of communication protocol are standardized and implemented as a single merged layer. Considering this trend, the MPTM has the following advantages over conventional methods such as CTMF. Firstly, a multi-protocol IUT is able to be tested even though its interface between the layers is not exposed. Secondly, the overhead for test description and execution is reduced by testing several protocols at the same time with a single test suite. Thirdly, the exact source of failure of a multi-protocol IUT can be identified.

A similar idea to the proposed MPTM was reported in [1]. However, the test cases were manually generated. Therefore, the idea lacks generality, and it was applied only to a simplified TCP/IP. In this paper we propose and implement an automated test case generation algorithm for testing a multi-protocol and apply it not only to a simplified TCP/IP but also to B-ISDN Signaling/SSCOP. With the MPTM, an IUT is implemented as two FSMs representing the protocols and the relationship between the transitions of these FSMs. As the result of applying the proposed algorithm to TCP/IP, the same test cases as the ones appeared in [1] are generated. Moreover, an error in the test cases generated manually [1] is found, which demonstrates the correctness and effectiveness of the proposed algorithm. To show the generality of the proposed MPTM, it is applied to B-ISDN Signaling/SSCOP, and it displays the same test coverage as the conventional test method with much smaller number of test cases and operations.

The rest of the paper is organized as follows. Section 2 gives an overview of existing research on test coverage and test case generation with respect to multi-protocols. Section 3 proposes the MPTM which automatically generating the test cases for multi-protocol IUT. Section 4 presents the application of the MPTM to TCP/IP and B-ISDN Signaling/SSCOP, and the results. Finally, Section 5 provides a brief conclusion and some discussion on future work.

2. RELATED WORK

B-ISDN Signaling/SSCOP [4] and Multiprotocol Label Switching (MPLS) [5] are examples of multi-protocols. In B-ISDN Signaling/SSCOP, the upper interface of SSCOP is not open, and thus it is accessed indirectly only through the Signaling layer. Therefore, embedded test method should be used for testing the SSCOP layer. Similarly, MPLS needs the same test method because the interface between layer 2 and 3 in commercial routers implementing MPLS is not exposed.

Although there have been some related works on test coverage and test sequence generation, most of the works have been focused on testing a single-layer protocol embedded in a multi-protocol IUT [6,7,8]. For example, Petrenko and Yevtushenko [6] modeled the testing environment as a communicating FSM and proposed an embedded test method generating test sequences for IUT. Zhu et al. [7] proposed an approach and developed a tool evaluating the test coverage of the embedded test method based on a fault model. Yevtushenko and Cavali [8] proposed an approach minimizing the test suite for the embedded test method. All these approaches assumed an error-free context and focused on testing a single-layer protocol embedded in a multi-protocol IUT. The assumption, though, seems to be hard to apply in real test environment.

0-7803-6494-5/00/$10.00 © 2000 IEEE

According to CTMF, which is the base of the related work above, test of multi-protocol IUT is performed by successive use of single-layer embedded test method for each protocol of multi-protocol IUT. In this approach it is assumed that all the protocols except the target protocol are error-free, which is rare in reality. Therefore, it is better to test all the protocols constructing the multi-protocol IUT at the same time and point out the exact source of failure. This is the main objective of the method discussed in [1], which tries to check the conformity of the multi-protocol IUT with a single test execution.

Fig. 1 shows the organization of multi-protocol IUT. The upper interface of (N)-th layer and the lower interface of (N-1)-th layer are open, but the interface between (N)-th and (N-1)-th layer is neither directly controllable nor observable by the tester. In the MPTM, the test case has the same effect as simultaneously applying the single-layer test method to the upper layer protocol and single-layer embedded test method to the lower layer protocol. Therefore, the test event for this multi-protocol IUT is described with (N)-Abstract Service Primitive (ASP), (N)-Protocol Data Unit (PDU), and (N-1)-PDU. In this case, (N)-PDU is included in the user data field of (N-1)-PDU. We next present the proposed MPTM.

3. THE PROPOSED ALGORITHM FOR MPTM

Fig. 2 shows the overall structure of the proposed multi-protocol test method (MPTM). Here, given a multi-protocol of two FSMs representing the upper and lower layer protocol respectively and the relationship between the transitions of the two FSMs, the algorithm generates a test suite for the multi-protocol as an output. To accomplish this, the FSMs describing the IUT need to be defined in a form that can be used by the algorithm, and also a method is defined to indicate the relationship between the transitions of the upper and lower layer protocol.

In order to briefly illustrate the proposed MPTM, an example of a multi-protocol described by two FSMs is depicted in Fig. 3. Fig. 3(a) and 3(b) represent the upper layer protocol and lower layer protocol, respectively. Observe from the figure that each transition is marked as T_i : X/Y, where T_i is transition identification, X is input , and Y is output. For example, for T_1 of Fig. 3(a), the input is PDU message and output is ASP message represented as $P_{u\_i,1}/A_{u\_o,1}$.

Definition 1: An operator, ⇨, is defined to represent the precedence condition between two events. For example, A⇨B indicates that event A precedes event B.

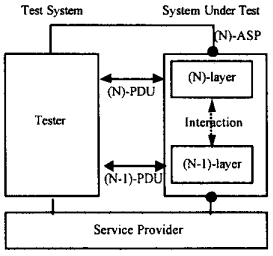


Fig. 1. Organization of multi-protocol IUT.

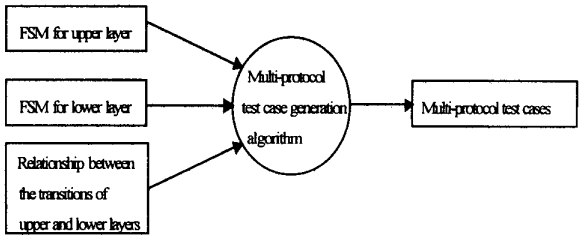
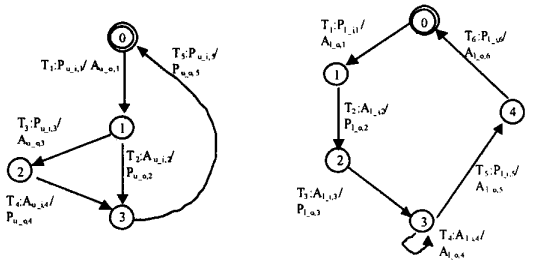


Fig. 2. Overall structure of MPTM.



(a) Upper layer, M_1 (b) Lower layer, M_2

Fig. 3. An example of a multi-protocol.

In addition, the relationships between the two FSMs are assumed as follows. Here, (a) and (b) means Fig. 3(a) and Fig. 3(b), respectively.

Relationship 1 :
- T_1 in (b) ⇨ T_2 in (b) ⇨ T_1 in (a)
- T_1 in (a) ⇨ T_3 in (b)
- T_2 in (a) ⇨ T_4 in (b)
- T_3 in (a) ⇨ T_4 in (b)
- T_4 in (a) ⇨ T_4 in (b)
- T_5 in (a) ⇨ T_4 in (b) ⇨ T_5 in (b) ⇨ T_6 in (b)

Definition 2 formally represents the FSM of a protocol to be tested.

Definition 2: FSM M_i is composed of five elements (S_i, I_i, O_i, T_i, S_i^0), where each element is defined as follows:

S_i : a finite nonempty set of states,
I_i : a finite nonempty set of inputs,
O_i : a finite nonempty set of outputs,
T_i : a finite nonempty set of deterministic transitions defined as $S_i \times I_i \rightarrow S_i \times O_i$, and
S_i^0 : the initial state of M_i.

According to Definition 2, the FSM, M_1, of Fig. 3(a) can be represented as follows.

states $S_1 = \{S_{1,0}, S_{1,1}, S_{1,2}, S_{1,3}\}$,
inputs $I_1 = \{P_{u\_i,1}, A_{u\_i,2}, P_{u\_i,3}, A_{u\_i,4}, P_{u\_i,5}\}$,
outputs $O_1 = \{A_{u\_o,1}, P_{u\_o,2}, A_{u\_o,3}, P_{u\_o,4}, P_{u\_o,5}\}$,
transitions $T_1 = \{ T_{1,1} : S_{1,0} \times P_{u\_i,1} \rightarrow S_{1,1} \times A_{u\_o,1}$,
$T_{1,2} : S_{1,1} \times A_{u\_i,2} \rightarrow S_{1,3} \times P_{u\_o,2}$,
$T_{1,3} : S_{1,1} \times P_{u\_i,3} \rightarrow S_{1,2} \times A_{u\_o,3}$,
$T_{1,4} : S_{1,2} \times A_{u\_i,4} \rightarrow S_{1,3} \times P_{u\_o,4}$,
$T_{1,5} : S_{1,3} \times P_{u\_i,5} \rightarrow S_{1,0} \times P_{u\_o,5}\}$, and
initial state $S_1^0 = S_{1,0}$.

Definition 3 describes the relationship between the transitions of two FSMs.

Definition 3: Given two FSMs describing upper layer, M_i, and lower layer, M_j, the relationship R between the transitions of M_i and M_j is a set of $R_{i,k}$, where $R_{i,k}$ represents the mapping from transition $T_{i,k}$ of M_i to the transitions of M_j that have relationship with $T_{i,k}$. $T_{i,k}$ means the k-th transition of M_i.

The relationship R between the transitions of M_1 and M_2 of Fig. 3 can be represented by Definition 3 as follows.

$R = \{ R_{1,1}, R_{1,2}, R_{1,3}, R_{1,4}, R_{1,5}\}$

$R_{i,k}$ in Definition 3 can be further defined in Definition 4.

Definition 4: Mapping $R_{i,k}$ is an ordered set of pre-execution($T_{j,t}$) and carried-by($T_{j,t}$). Pre-execution($T_{j,t}$) represents a transition in the lower FSM, which has to be executed first before the execution of $T_{i,k}$. Carried-by($T_{j,t}$) represents a transition in the lower FSM, which is executed while $T_{i,k}$ of the upper FSM is executed.

Definition 5: The notation "<>" means that the elements enclosed by it are executed in order from left to right.

According to Definition 4, $R_{1,k}$ for Relationship 1 shown in the previous page can be represented as follows:

$R_{1,1}$=<pre-execution($T_{2,1}$),pre-execution($T_{2,2}$),carried-by($T_{2,3}$)>,
$R_{1,2}$ = <carried-by($T_{2,4}$) >,
$R_{1,3}$ = <carried-by($T_{2,4}$) >,
$R_{1,4}$ = <carried-by($T_{2,4}$) >, and
$R_{1,5}$ = <carried-by($T_{2,4}$),carried-by($T_{2,5}$),carried-by($T_{2,6}$) >.

Based on the aforementioned definitions, the algorithm for the MPTM is shown as Algorithm 1 in Appendix 1. Let us look at it in detail. **Main_generation** is the main function which receives the upper FSM, M_1, the lower FSM, M_2, and the relationship R as inputs, and generates test cases as output. All operations and functions are called by **Main_generation**. The role of **find_preamble** is to find the shortest path from the initial state to the starting state of a transition intended to be tested. After the preamble of each transition is derived by **find_preamble**, **test_case_generation** is called by each transition of M_1. **test_case_generation** calls **generate_preamble**, and then invokes **generate_testbody**. Using the shortest path found by **find_preamble**, **generate_preamble** provides the sequence of transitions up to the starting state where a transition is tested. **generate_testbody** visits a transition to be tested, and then generates a corresponding test case.

Using the FSMs and the relationship between the transitions of them, **generate_testbody** does the followings. First, it processes the operations corresponding to the pre-execution relationship. If there exist some transitions in the lower FSM that must be executed before executing the transition of the upper FSM, they are executed first. Accordingly, the transitions in the lower FSM for pre-execution relationship execute the behavior lines such as "PCO3! Input" and "PCO3? Output". These behavior lines are to send an input and receive an output from the protocol to be tested. Second, the carried-by relationship is executed. Here, when the transition of the upper FSM is executed, the transitions of the lower FSM in the relationship are also executed concurrently. These operations related to the input and output of the transitions to be tested are iteratively executed in order until the input and output are exhausted. If the input of the upper FSM transition is ASP, then "PCO1! Input" is generated. If the input is PDU, then "PCO3! Input" is generated. In the case of PDU, the input in a behavior line is included as an input parameter to the transition in the lower FSM. If the output of the upper FSM transition is ASP, then "PCO1? Output" is generated. If output is PDU, then "PCO3? Output" is generated. In the case of PDU, an output in a behavior line includes the output parameter for a transition of the lower FSM.

After every transition of the upper FSM processes **generate_testbody**, the number of the test cases, behavior lines, and events, and test coverage are printed.

4. APPLICATION OF THE MPTM

In this section the proposed MPTM is applied to TCP/IP and B-ISDN Signaling/SSCOP, and then the results are analyzed in terms of test cases and test case coverage.

4.1 Application to TCP/IP

This subsection describes the implementation of the MPTM proposed in Section 3, and its application to TCP/IP [3] multi-protocol. With the result, the correctness of the MPTM is verified.

Fig. 4 and 5 show the FSMs of simplified TCP and IP protocol, respectively. Transitions in Fig. 4, such as T_3 and T_8, can be divided into two cases: IP with fragmentation and IP without fragmentation. This results in T_{3_1}, T_{3_2}, T_{8_1}, and T_{8_2}. The relationship between the transitions of the two FSMs can be derived by analyzing the FSMs. For example, let us examine T_2 where the input and output are SYN_ACK and ACK, respectively. In order for the IUT to receive SYN_ACK, the tester first sends a datagram containing SYN_ACK and receives an IP datagram from the IUT containing the result of the execution of $T_{r,1}$ in Fig. 5. The IUT then receives ACK. To do this, the IUT sends an IP datagram as a result of the execution of $T_{s,1}$ in Fig. 5. If a datagram containing ACK is confirmed, then the execution of T_2 is confirmed. In this way, the relationship between the upper FSM of TCP and lower FSM of IP is defined from their protocol specifications. The relationships are as follows:

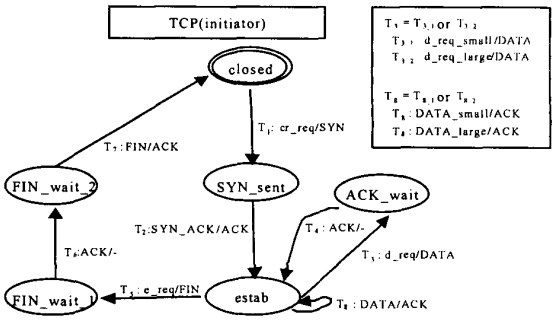


Fig. 4. The FSM of a simplified TCP protocol.

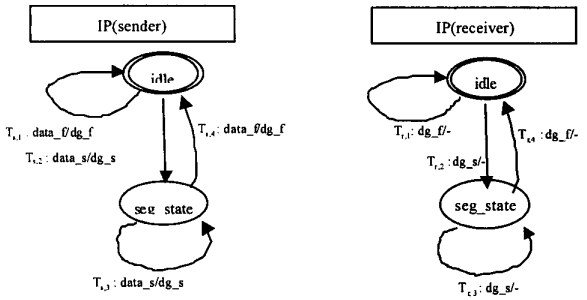


Fig. 5. The FSM of IP protocol.

R = { R_1, R_2, R_{3_1}, R_{3_2}, R_4, R_5, R_6, R_7, R_{8_1}, R_{8_2} } where
R_1 = < carried-by ($T_{s,1}$) >,
R_2 = < carried-by($T_{r,1}$), carried-by($T_{s,1}$) >
R_{3_1} = < carried-by($T_{s,1}$) >,
R_{3_2} = < carried-by($T_{s,2}$), carried-by($T_{s,3}$), carried-by($T_{s,4}$) >,
R_4 = < carried-by($T_{r,1}$) > ,
R_5 = < carried-by($T_{s,1}$) >,
R_6 = < carried-by($T_{r,1}$) > ,
R_7 = < carried-by($T_{r,1}$), carried-by($T_{s,1}$) >,
R_{8_1} = < carried-by($T_{r,1}$), carried-by($T_{s,1}$) >, and
R_{8_2} = < carried-by($T_{r,2}$), carried-by($T_{r,3}$), carried-by($T_{r,4}$), carried-by($T_{s,1}$) >.

Applying the implementation of the MPTM to a simplified TCP/IP results in 10 test cases (see Appendix 2). Using the test cases, 8 transitions from the upper FSM (TCP) and 8 transitions from the lower FSM (IP) can be tested. The test cases are same as the ones generated manually in [1] except for T_4. The test case in [1] contains an error, and the correct test case is shown as "m_tc_T5" in Appendix 2. This demonstrates one of important benefits of generating the test cases for multi-protocol automatically, which is correctness.

Now let us illustrate the process of test case generation using a typical test case shown in Appendix 2. For that, consider "m_tc_T4" corresponding to T_{3_2} of TCP (which is for IP fragmentation). To execute transition T_{3_2}, the IUT has to be set to the starting state of T_{3_2}. Accordingly, the behavior line "+m_tc_T2" is obtained by **generation-preamble**. Since there is no pre-execution for T_{3_2}, the operations associated with the carried-by relationship are executed. First, the input of T_{3_2} is observed. Because the input is ASP "d_req_large", the behavior line "PCO1! d_req_large" is generated for sending an input to the protocol to be tested. Next, the output of T_{3_2} is observed. Since the output is PDU "DATA" and the relationship of T_{3_2} of the remaining elements are "carried-by($T_{s,2}$)", "carried-by($T_{s,3}$)" and "carried-by($T_{s,4}$)", the algorithm generates behavior lines corresponding to receive dg_s, dg_s, and dg_f to execute the transitions $T_{s,2}$, $T_{s,3}$ and $T_{s,4}$, respectively. "DATA", which is the output of transition T_{3_2}, is included as a parameter of the outputs of $T_{s,2}$, $T_{s,3}$ and $T_{s,4}$. As a result, the generated behavior lines are "PCO3 ? dg_s(DATA)", "PCO3 ? dg_s(DATA)", and "PCO3 ? dg_f(DATA)".

We compare the multi-protocol test cases obtained by the process mentioned above and those obtained by the conventional test method. According to [1], the conventional method generates 8 test cases, 22 behavior lines, and 48 events for testing the TCP described in Fig. 4, and 8 test cases, 23 behavior lines, and 47 events for testing the IP in Fig. 5. Consequently, the conventional test method needs 16 test cases, 45 behavior lines, and 95 events to test all the transitions of TCP/IP multi-protocol. On the other hand, the proposed MPTM allows to test all the transitions with only 10 test cases. In addition, the numbers of behavior lines and events are decreased to 31 and 63 from 45 and 95, respectively. This means that the MPTM allows the same test coverage as the conventional test method with much fewer test cases and operations.

4.2 Application to B-ISDN Signaling/SSCOP

In the previous subsection, it was shown that the same test cases presented in [1] can be derived by applying the implementation of the MPTM to TCP/IP. In this subsection, the generality of the proposed algorithm is shown by applying it to a multi-protocol consisting of Q.2931 signaling layer and SSCOP [4].

Fig. 6 and 7 in Appendix 3 show Q.2931 signaling layer and SSCOP, respectively. As in the case for TCP/IP discussed in Section 4.1, two FSMs are defined corresponding to the Q.2931 signaling layer as the upper layer and the SSCOP as the lower layer, and the relationship between the transitions of the two FSMs is extracted. Compared with TCP/IP which has only the carried-by relationship, there is an additional pre-execution relationship in Q.2931 signaling/SSCOP multi-protocol. For instance, consider the transition T_1 in Fig. 6. Q.2931 Signaling layer transmits PDU through SSCOP. If there is a pre-established connection in SSCOP, it is used. Otherwise, a new connection of SSCOP must be established.

Because transition T_1 of Q.2931 starts from the initial state, there is no established connection in SSCOP. To execute transition T_1, thus, SSCOP needs to establish a connection. This means T_1 and T_3 in Fig. 7 must be executed before the transition T_1 in Fig. 6 is executed. After establishing the connection, T_{28} in Fig. 7 and T_1 in Fig. 6 are executed independently (or concurrently). In other words, transition T_1 of Q.2931 FSM has relationships with the transitions T_1, T_3, and T_{28} of SSCOP FSM such as pre-execution($T_{2,1}$), pre-execution($T_{2,3}$), and carried-by($T_{2,28}$). All the relationships between Q.2931 and SSCOP can be described as follows.

R = { R_1, R_2, R_3, ···, R_{20}, R_{21} } where
R_1=<pre-execution($T_{2,1}$),pre-execution($T_{2,3}$),carried-by($T_{2,28}$)>,
R_2=<carried-by($T_{2,29}$)>,
R_3=<carried-by($T_{2,29}$)>,
R_4=<carried-by($T_{2,29}$),carried-by($T_{2,28}$)>,
R_5=<carried-by($T_{2,29}$)>,
R_6=<carried-by($T_{2,29}$),carried-by($T_{2,28}$)>,
R_7=<carried-by($T_{2,29}$),carried-by($T_{2,28}$)>,
R_8=<pre-execution($T_{2,2}$),pre-execution($T_{2,11}$),carried-by($T_{2,29}$)>,
R_9=<carried-by($T_{2,28}$)> ,
R_{10}=<carried-by($T_{2,28}$)>,
R_{11}=<carried-by($T_{2,28}$)>,
R_{12}=<carried-by($T_{2,28}$)>,
R_{13}=<carried-by($T_{2,28}$)>,
R_{14}=<carried-by($T_{2,28}$)>,
R_{15}=<carried-by($T_{2,28}$)>,
R_{16}=<carried-by($T_{2,29}$)>,
R_{17}=<carried-by($T_{2,28}$)>,

R_{18}=<carried-by($T_{2,29}$)>,
R_{19}=<carried-by($T_{2,29}$)>,
R_{20}=<carried-by($T_{2,29}$)>, and
R_{21}=<carried-by($T_{2,28}$)>.

Based on the relationship defined above, 21 multi-protocol test cases are derived as a result of applying the MPTM to the B-ISDN Signaling/SSCOP described in Fig. 6 and 7. The generated test cases are shown in Appendix 4.

Now let us illustrate the generation process of test cases using a typical test case shown in Appendix 4. To illustrate the process, consider "m_tc_T8" for transition T_8 in Appendix 4, which has pre-execution relationship as well as carried-by relationship. Because T_8 is executed at the initial state, it does not have a preamble. With respect to T_8, the first and second relationship elements are "pre-execution($T_{2,2}$)" and "pre-execution($T_{2,11}$)", respectively. The MPTM algorithm generates behavior lines "PCO3! BGN" and "PCO3? BGAK" which execute $T_{2,2}$ and $T_{2,11}$, respectively. Because no more pre-execution relationship exists, the operation related to carried-by is then executed. Since the input of T_8 is "SETUP" PDU, the input is passed to the IUT through PCO3 as a form of SSCOP PDU which is the lower FSM. Accordingly, the algorithm generates a behavior line which executes $T_{2,29}$. Namely, "SETUP", an input of transition T_8, is included as a parameter of "SD", which is the output of $T_{2,29}$. As a result, the generated behavior line becomes "PCO3! SD(SETUP)". As the next step, we observe the output of T_8. Since the output is ASP "setup_ind", the algorithm generates behavior-line "PCO1? setup_ind", which receives "setup_ind" from the protocol to be tested.

The result of the test case generation is summarized in Table I, which compares it with the conventional method. As can be seen from the table, 21 out of the 21 transitions of Q.2931 and 6 out of the 29 transitions of SSCOP from the conventional test method can be tested by 21 test cases obtained by the proposed multi-protocol test method. Having the limit on testing the behavior of the lower layer, the multi-protocol test method cannot test all the transitions of the lower layer. It is due to the fact that the tester cannot access the internal interface directly, but control and observe the interface indirectly through the upper layer protocol. This problem exists in the embedded test methods [6,7,8] as well.

The proposed MPTM can test each layer of a multi-protocol at the same time even though the interfaces between the protocols are not exposed. Also it has the same coverage as the conventional method with fewer test cases as shown in Table I. Note also that the MPTM requires more events than the conventional method. This is because the behavior of the lower layer required for testing is not included in the conventional method. For example, consider "m_tc_T1" in Appendix 4 and "s_tc_T1" from the single-layer test method shown in Table II. Note form the table that there are behavior lines such as "PCO3 ? BGN" and "PCO3 ! BGAK" in "m_tc_T1", but not in "s_tc_T1". They are not described in the single-layer test case, but actually executed in the testing environment. In other words, "m_tc_T1" in the multi-protocol test method includes all the behaviors of Q.2931 and SSCOP. However, with "s_tc_T1" in the single-layer test method, the behaviors of SSCOP is neither presented in the test case nor

TABLE I

COMPARISON OF TEST CASES FOR MPTM AND CONVENTIONAL METHOD

Test methods / Measures	Multi-Protocol Test Method	Conventional test method		
		Single-layer test method for Q.2931	Single-layer embedded test method for SSCOP	Q.2931 + SSCOP
Number of test cases	21	21	6	27
Number of behavior lines	68 (4+3+3+4+3+4+4+4+3+3+3+ 3+3+3+3+3+3+3+3+3)	64	12	76
Number of events	160 (4+6+6+7+8+9+9+4+6+6+6+ 6+8+8+8+8+9+9+11+11+11)	118 (160)	16	134 (176)
Test coverage	all transitions of Q.2931 6 transitions of SSCOP	all transitions	6 transitions of SSCOP	all transitions of Q.2931 and 6 transitions of SSCOP

TABLE II

MULTI-PROTOCOL TEST CASE AND SINGLE-LAYER TEST CASE

m_tc_T1	s_tc_T1
PCO1 ! setup_req	PCO1 ! setup_req
PCO3 ? BGN	PCO2 ? SETUP
PCO3 ! BGAK	
PCO3 ? SD(SETUP)	

included as events. Only the behavior of Q.2931 is counted in the single-layer test method.

Since SSCOP behaves as the service provider in the test environment for the single-layer test method, the number of events in the single-layer test method will increase if these factors are considered. If we count the executions of the lower layer which do not appear in the test case of Table I, the number of events of s_tc_T1 becomes 4. The total number of events of the single-layer test method is recomputed in this way, and then it becomes 160. Adding the number of events for the embedded test method, the total number of events in the existing method becomes 176. Consequently, the total number of events in the MPTM becomes smaller than that with the single-layer test method if the events of the lower layer is included.

5. CONCLUSION AND FUTURE WORK

In this paper an approach for automatic test case generation with the multi-protocol test method proposed in [1] has been developed and implemented. The method called MPTM defines two FSMs which represent the protocols to be tested and the relationship between the transitions of them. Using the two FSMs and the relationship as inputs, the MPTM automatically generate test cases for multi-protocols. The MPTM was applied to TCP/IP, and same result as in [1] was generated. It also found an error in the test cases generated in [1]. We have also demonstrated the generality of the MPTM by applying it to Q.2931/SSCOP. The MPTM is able to test the multi-protocol IUT even though the interfaces between the protocol layers are not exposed. In addition, it has the advantages of providing the same coverage as the conventional test method with fewer number of test cases and identifying the exact source of failure in a multi-protocol IUT.

The MPTM generates test cases for multi-protocol IUT consisting of two protocol layers. As a future work, we plan to extend it to be applicable to multi-protocol IUT consisting of more than two protocols in a stack. Constructing a test environment by using the MPTM and applying the methodology to test real protocols will be done in order to demonstrate the feasibility of the proposed methodology.

REFERENCES

[1] Y. Park, M. Kim and S. Kang, "Conformance Testing of multi-protocol IUTs," International Workshop of Testing on Communicating Systems '99, pp. 267-284, 1999.

[2] ISO 9646, "Information Technology - OSI - Conformance Testing Methodology and Framework," 1992.

[3] W. R. Stevens, TCP/IP Illustrated, Addison-Wesley, 1994.

[4] ITU-T Recommendation Q.2110, "B-ISDN ATM Adaptation Layer - Service Specific Connection-Oriented Protocol (SSCOP)," 1994.

[5] IETF draft-ietf-mpls-framework-04.txt , "A Framework for Multiprotocol Label Switching," 1999.

[6] A. Petrenko and N. Yevtushenko, "Fault detection in embedded components," International Workshop of Testing on Communicating Systems '97, pp. 272-287, 1997.

[7] J. Zhu, S. T. Voung and S. T. Chanson, "Evaluation of test coverage for embedded system testing," International Workshop of Testing on Communicating Systems '98, pp. 111-126, 1998.

[8] N. Yevtushenko and A. Cavali, "Test suite minimization for testing in context," International Workshop of Testing on Communicating Systems '98, pp. 127-145, Tomsk, Russia, 1998.

APPENDIX 1. ALGORITHM FOR MPTM

```
Main_generation(M1, Mk, R) {
  For (T = each transition in M1) {
    find_preamble(M1, T)
    test_case_generation(M1, Mk, R, T)
  }
  write statistics(# of test cases, # of behavior lines, # of events, test coverage)
}

find_preamble (M1, Tij) {
  for (P = each transition in M1) {
    if (the ending state of P == the starting state of Tij) choose P
  }
  select P which is the shortest path from the initial state
}

test_case_generation(M1, Mk, R, Tij) {
  generate_preamble(M1, Tij)
  generate_testbody(M1, Tij, R)
}

generate_preamble(M1, Tij) {
  generate the preamble of Tij
}
```

```
generate_testbody(M1, Tij, R) {
  /* pre-execution */
  if (pre-execution of Tij exists in R1,j) {
    if (an input of Tij is ASP type) {
      write a behavior line concatenating "PCO1!" and the input of Tij
      set pre_execution_flag
      i = 0
      while(pre-execution of Tij exists in R1,j) {
        i ++
        cur_element = read the current element in R1,j
        cur_transition = the transition of the cur_element
        if ( i is an odd number )
          write a behavior line concatenating "PCO3?" and the output of cur_transition
        else if ( i is an even number)
          write a behavior line concatenating "PCO3!" and the input of cur_transition
      }
    }
    else if (an input of Tij is PDU type) {
      i = 0
      while (pre-execution of Ti,j exists in R1,j) {
        i ++
        cur_element = read the current element in R1,j
        cur_transition = the transition of the cur_element
        if ( i is an odd number )
          write a behavior line concatenating "PCO3!" and the input of cur_transition
        else if ( i is an even number)
          write a behavior line concatenating "PCO3?" and the output of cur_transition
      }
    }
  }

  /* carried-by */
  if (an input of Ti,j is ASP type and pre_execution_flag is unset)
    write a behavior line concatenating "PCO1!" and the input of Tij
  else if (an input of Ti,j is PDU type) {
    cur_element = read the current element in R1,j
    cur_transition = the transition of the cur_element
    write a behavior line concatenating
    "PCO3!", the input of cur_transtion, "(", the input of Ti,j, and ")"
  }

  for ( each output of Tij ) {
    if (the output of Tij is ASP type)
      write a behavior line concatenating "PCO1?" and the output of Tij
    else if (the output of Tij is PDU type) {
      cur_element = read the current element in R1,j
      cur_transition = the transition of the cur_element
      write a behavior line concatenating
      "PCO3?", the output of cur_transition, "(", the output of Tij, and ")"
    }
  }
}
```

Algorithm 1. Algorithm for multi-protocol testing

APPENDIX 2. TEST CASE FOR TCP/IP BY MPTM

```
m_tc_T1
 PCO1 ! cr_req
  PCO3 ? dg_f(SYN)

m_tc_T2
+m_tc_T1
 PCO3 ! dg_f(SYN_ACK)
  PCO3 ? dg_f(ACK)

m_tc_T3
+m_tc_T2
 PCO1 ! d_req_small
  PCO3 ? dg_f(DATA)

m_tc_T4
+m_tc_T2
 PCO1 ! d_req_large
  PCO3 ? dg_s(DATA)
   PCO3 ? dg_s(DATA)
    PCO3 ? dg_f(DATA)

m_tc_T5
+m_tc_T3
   PCO3 ! dg_f(ACK)

m_tc_T6
+m_tc_T2
 PCO1 ! e_req
  PCO3 ? dg_f(FIN)

m_tc_T7
+m_tc_T6
 PCO3 ! dg_f(ACK)

m_tc_T8
+m_tc_T7
 PCO3 ! dg_f(FIN)
  PCO3 ? dg_f(ACK)

m_tc_T9
+m_tc_T2
 PCO3 ! dg_f(DATA_small)
  PCO3 ? dg_f(ACK)

m_tc_T10
+m_tc_T2
 PCO3 ! dg_s(DATA_large)
  PCO3 ! dg_s(DATA_large)
   PCO3 ! dg_f(DATA_large)
    PCO3 ? dg_f(ACK)
```

APPENDIX 3. ALGORITHM FOR MPTM

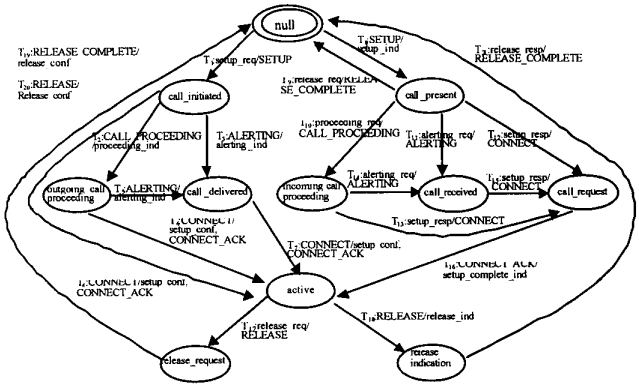


Fig. 6. FSM of Q.2931.

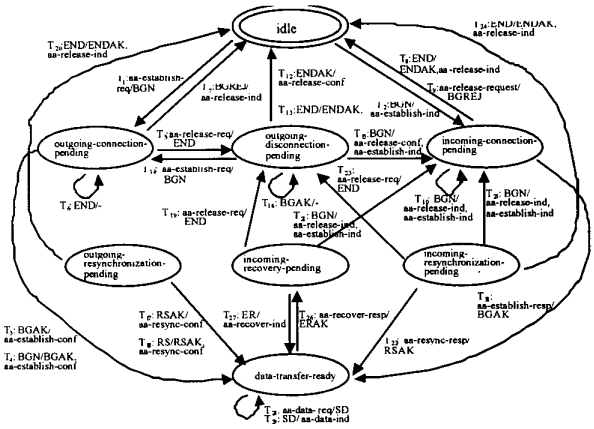


Fig. 7. FSM of SSCOP.

APPENDIX 4. TEST CASE FOR B-ISDN SIGNALING/SSCOP BY MPTM

```
m_tc_T1
 PCO1 ! setup_req
  PCO 3 ? BGN
   PCO 3 ! BGAK
    PCO3 ? SD(SETUP)

m_tc_T2
+m_tc_T1
 PCO3 ! SD(CALL_PROCEEDING)
  PCO1 ? proceeding_ind

m_tc_T3
+m_tc_T1
 PCO3 ! SD(ALERTING)
  PCO1 ? alerting_ind

m_tc_T4
+m_tc_T1
 PCO3 ! SD(CONNECT)
  PCO1 ? setup_conf
   PCO3 ? SD(CONNECT_ACK)

m_tc_T5
+m_tc_T2
 PCO3 ! SD(ALERTING)
  PCO1 ? alerting_ind

m_tc_T6
+m_tc_T2
 PCO3 ! SD(CONNECT)
  PCO1 ? setup_conf
   PCO3 ? SD(CONNECT_ACK)

m_tc_T7
+m_tc_T3
 PCO3 ! SD(CONNECT)
  PCO1 ? setup_conf
   PCO3 ? SD(CONNECT_ACK)

m_tc_T8
 PCO 3 ! BGN
  PCO 3 ? BGAK
   PCO3 ! SD(SETUP)
    PCO1 ? setup_ind

m_tc_T9
+m_tc_T8
 PCO1 ! release_req
  PCO3 ? SD(RELEASE_COMPLETE)

m_tc_T10
+m_tc_T8
 PCO1 ! proceeding_req
  PCO3 ? SD(CALL_PROCEEDING)

m_tc_T11
+m_tc_T8
 PCO1 ! alerting_req
  PCO3 ? SD(ALERTING)

m_tc_T12
+m_tc_T8
 PCO1 ! setup_resp
  PCO3 ? SD(CONNECT)

m_tc_T13
+m_tc_T10
 PCO1 ! setup_resp
  PCO3 ? SD(CONNECT)

m_tc_T14
+m_tc_T10
 PCO1 ! alerting_req
  PCO3 ? SD(ALERTING)

m_tc_T15
+m_tc_T11
 PCO1 ! setup_resp
  PCO3 ? SD(CONNECT)

m_tc_T16
+m_tc_T12
 PCO3 ! SD(CONNECT_ACK)
  PCO1 ? setup_complete_ind

m_tc_T17
+m_tc_T4
 PCO1 ! release_req
  PCO3 ? SD(RELEASE)

m_tc_T18
+m_tc_T4
 PCO3 ! SD(RELEASE)
  PCO1 ? released_ind

m_tc_T19
+m_tc_T17
 PCO3 ! SD(RELEASE_COMPLETE)
  PCO1 ? release_conf

m_tc_T20
+m_tc_T17
 PCO3 ! SD(RELEASE)
  PCO1 ? release_conf

m_tc_T21
+m_tc_T18
 PCO1 ! release_resp
  PCO3 ? SD(RELEASE_COMPLETE)
```