다중계층 프로토콜 시험 방법
(Multi-protocol Test Method : MPTM)

이 수 인 † 박 용 범 †† 김 명 철 †††
(Soo-in Lee) (Yongbum Park) (Myungchul Kim)

요 약 하나의 시험 스위트를 가지고 다중 계층 프로토콜 시험 대상 (Implementation Under Test : IUT)을 시험하는 방안이 제안되었다[1]. 기존 방법과 비교하여 이 방법은 상위 프로토콜에 적용하는 단일 계층 시험 방법과 하위 프로토콜에 적용하는 내포 시험 방법을 조합하여 적용함으로써 다중 계층 프로토콜 시험 대상을 시험한다. 그러나 논문[1]은 접근 방법만 제시하였을 뿐, 어떻게 시험 경우를 자동으로 도출할 것인지에 대해서는 고려하지 않고 있다. 본 논문은 논문[1]에 기초하여 다중 계층 프로토콜 시험 경우 자동 생성 알고리즘을 제안한다. 이를 위해 시험대상 프로토콜을 두개의 FSM으로 정의하고, 두 FSM에 대하여 pre-execution과 carried-by로 구성되는 트랜지션의 수행 관계를 정의한다. 제안한 알고리즘을 구현하여 간략화한 TCP/IP와 B-ISDN Signaling/Service Specific Connection Oriented Protocol (SSCOP)에 적용한다. 본 논문의 다중 계층 프로토콜 시험 방법은 프로토콜 사이의 인터페이스가 개방되지 않은 경우에도 시험이 가능하며, 기존 시험 방법에 비해서 적은 시험 경우로 동일한 커버리지(coverage)를 갖는다.

Abstract An approach for testing multi-protocol Implementation Under Test (IUT) with a single test suite has been proposed in[1]. 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 the two FSMs 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. MPTM is able to test the multi-protocol IUT even though the interfaces between the protocol layers are not exposed. It results in that the proposed MPTM allows the same test coverage as conventional test methods even with fewer numbers of test cases.

1. 서 론

프로토콜에 대한 적합성시험을 위한 방법론과 체계는 ISO/IEC JTC1을 통해 국제표준으로 제정되어 사용되고 있다[2]. 표준[2]는 단일 계층을 시험 대상으로 하여 시험하는 것으로서, 여러 계층으로 구성된 다중 계층 프로토콜 (프로토콜의 집합 또는 프로파일)을 시험하기 위한 방안으로 단일 계층 시험을 반복하여 수행하는 방법을 제시하고 있다. 즉, 최상위 프로토콜은 단일 계층 시험 방법을 사용하여 시험하고, 최상위를 제외한 프로토콜은 단일 계층 내포 시험방법을 사용하여 시험한다.

이에 비해 논문[1]에서는 다중 계층 프로토콜 시험대상에서 상위 프로토콜에 적용하는 단일계층 시험방법과 하위 프로토콜에 적용하는 내포 시험방법을 조합하여 적용함으로써, 하나의 시험 스위트에서 두개 이상의 계층에 대한 시험을 동시에 수행하는 방안을 제안하고 있다. 이 방안에서는 최하위 Point of Control and Observation (PCO)의 제어와 관찰을 통해 프로토콜 간의 인터페이스를 간접적으로 제어하고 관찰하여 내포된 프로토콜과 상위 프로토콜을 한꺼번에 시험한다.

현재, 프로토콜은 Multi-protocol Label Switching (MPLS)와 같이 여러 계층이 점차 통합되어 가는 추세

† 비 회 원 : 삼보정보통신 기술연구소 연구원
elsie@icu.ac.kr
†† 비 회 원 : 한국전자통신연구원 네트워크장비시험팀 연구원
ybpark@icu.ac.kr
††† 종신회원 : 한국정보통신대학원대학교 공학부 교수
mckim@icu.ac.kr
논문접수 : 2000년 8월 11일
심사완료 : 2001년 7월 18일

이다. 이러한 상황에서 프로토콜 사이의 인터페이스가 개방되지 않은 다중 계층 프로토콜을 시험할 수 있으며, 두 개 이상의 프로토콜을 하나의 시험 스위트로 시험함으로써 시험규격 작성과 수행에 필요한 오버헤드를 줄일 수 있다는 점 등에서 논문[1]에서 제시한 방안은 가치가 있다.

그러나, 논문[1]에서 시험 경우의 도출은 수작업을 통해 이루어졌고, 간략화한 TCP/IP에만 적용하였다는 한계를 가지고 있다. 따라서, 본 논문에서는 우선 논문[1]에서 제시한 방안을 기초로 다중 계층 프로토콜 시험 경우 자동 생성 알고리즘으로 제시하고, 이를 구현한다. 그리고, 이 구현물을 간략화한 TCP/IP와 B-ISDN Signaling/SSCOP의 시험에 적용하고자 한다. 본 논문에서 제시한 알고리즘의 구현물을 TCP/IP에 적용한 결과, 논문[1]에서 수작업으로 도출한 것과 같은 시험 경우를 얻을 수 있었다. 이를 통해 알고리즘의 정확성을 입증하였다. 또한, 알고리즘의 일반성을 입증하기 위해 구현물을 B-ISDN Signaling/SSCOP에 적용하여 기존의 시험 방법과 동일한 커버리지를 지니는 시험 경우를 자동으로 도출하였다. 다중 계층 프로토콜 시험방안을 구현하는 과정에서, 시험대상 프로토콜을 두개의 FSM으로 나타내는 방법과 두 FSM에 대하여 트랜지션의 수행 관계를 정의하는 방법을 제시하였다.

이 논문의 구성은 다음과 같다. 2장에서 다중 계층 프로토콜의 시험에 관련된 내용에 대해 간략히 살펴본다. 3장에서는 다중 계층 프로토콜의 시험을 위한 알고리즘을 제시한다. 4장에서는 3장에서 제시한 알고리즘의 구현물을 TCP/IP와 B-ISDN Signaling/SSCOP에 적용한 결과에 대해 기술한다. 5장에서는 결론을 맺고, 향후 연구 계획에 대해 기술한다.

2. 관련 연구

다중 계층 프로토콜의 예로는 B-ISDN Signaling/SSCOP과 IETF의 Multiprotocol Label Switching (MPLS)을 들 수 있다[4,5]. B-ISDN Signaling/SSCOP에서 SSCOP의 상위 인터페이스는 개방되어 있지 않고 Signaling 계층을 통해서만 간접적으로 접근이 가능하다. 따라서 SSCOP에 대한 시험을 위해서는 내포 시험방법이 사용되어야 한다. 마찬가지로, MPLS를 구현한 상용 라우터에서 계층 2와 계층 3사이의 인터페이스는 개방되지 않으므로, MPLS를 시험하기 위해서는 역시 내포시험방법이 사용되어야 한다.

내포 시험방법의 시험 범위, 시험 시퀀스 생성 등에 대한 기존 연구들로 논문[6,7,8]이 있었으나 이들은 모두 다중 계층 프로토콜 시험 대상에서 단일 계층에 대한 시험에 초점을 맞추고 있다. 논문[6]에서는 시험 대상을 communicating FSM으로 모델링하고 시험을 생성하는 방안을 제시하였고, 논문[7]에서는 오류 모델(fault model)을 기반으로 하여 내포 시험방법의 시험 커버리지를 평가하는 방법과 도구를 제안하였으며, 논문[8]은 내포 시험방법에 의한 시험 스위트를 최소화하는 방안을 제안하였다. 이러한 연구들은 모두 컨텍스트(context)에는 오류가 없다는 가정 하에서 단일 계층에 해당하는 컴퍼넌트 (component)에 대한 시험에만 초점을 맞추고 있다.

표준[2]에 의하면 다중 계층 프로토콜에 대한 시험은 단일 프로토콜에 대한 시험을 반복하여 수행한다. 이 경우 하나의 프로토콜을 시험할 때 그 이외의 다른 프로토콜에는 오류가 없는 것으로 가정하고 있으나, 실제로는 그렇지 않은 경우가 존재한다. 따라서, 여러 개의 프로토콜을 동시에 시험하여 오류의 원인이 어느 프로토콜에 있는지를 정확히 알아내는 것이 보다 나은 방법이라고 할 수 있다. 이것이 논문[1]의 주요 아이디어로서, 한번의 시험으로 다중 계층 프로토콜의 적합성 여부를 판단하고자 하였다.

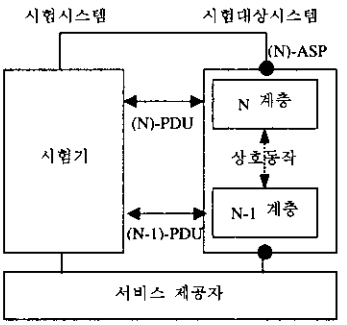


그림 1 다중 계층 프로토콜 시험 대상의 시험 구성

그림 1과 같은 시험 구성에서 N계층 프로토콜과 N-1 계층 프로토콜에 대한 시험을 대상으로 다중 계층 프로토콜 시험 방법을 설명한다. 이 그림에서 N계층의 상위와 N-1계층의 하위 인터페이스는 개방되어 있으나 N계층과 N-1계층 사이의 인터페이스는 직접적으로 제어 및 관찰할 수 없다고 가정한다. 논문[1]에서 제안한 다중 계층 프로토콜 시험방법에서는 상위 계층에 대한 단일 계층 시험방법과 하위 계층에 대한 내포 시험방법을 조합하여 시험 스위트를 표현한다. 따라서 그림 1과 같은 시험 구성에서 시험 이벤트는 (N)-Abstract Service

Primitive (ASP), (N)-Protocol Data Unit (PDU), (N-1)-PDU를 모두 사용하여 표현한다. 특히, (N)-PDU는 (N-1)-PDU의 사용자 데이타 필드에 포함되어 표현된다.

3. 다중계층 프로토콜 시험을 위한 알고리즘

이번 장에서는 논문[1]에서 제시한 방안을 구현하기 위한 알고리즘을 제안한다. 그림 2는 이번 장에서 제시하는 알고리즘의 전체 구조도를 나타낸다. 즉, 시험하고자 하는 다중 계층 프로토콜의 상, 하위 프로토콜의 FSM과 상, 하위 프로토콜 간의 관계를 입력으로 하여 알고리즘을 수행시키면 그에 대한 시험 경우를 출력으로 얻을 수 있다. 이를 위해 시험 대상을 나타내는 FSM을 알고리즘에서 사용할 수 있는 형태로 정의하고, 상위 계층의 FSM과 하위 FSM 사이의 관계를 나타내기 위한 방법을 정의한다.

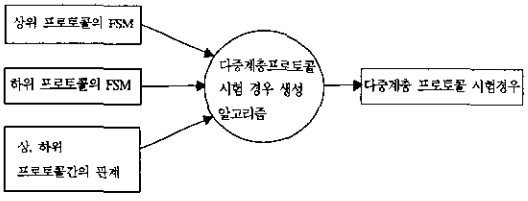
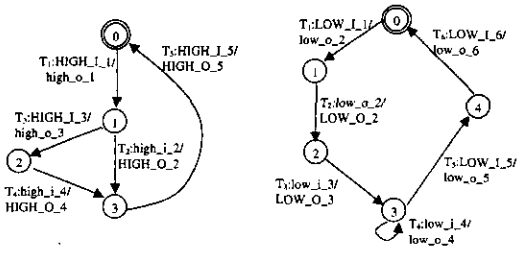


그림 2 알고리즘의 구조도

이 논문에서 제시하는 알고리즘을 설명하기 위한 다중 계층 프로토콜의 예가 그림 3에 나타나 있다. 그림 3의 (a)는 상위 계층 프로토콜을 나타내고, (b)는 하위 계층 프로토콜을 나타낸다. 그림 3에서 트랜지션은 T_i : X/Y의 형태로 표시되어 있다. 여기서, T_i는 트랜지션의 식별자이고, X는 트랜지션의 입력을 나타내고, Y는 트랜지션의 출력을 나타낸다.



(a) 상위 계층 (b) 하위 계층

그림 3 다중 계층 프로토콜의 예

관계 1 그림 3의 (a)와 (b)사이에는 다음과 같은 관계가 존재한다고 가정한다.

- (a)의 T_1을 수행하기 위해서는 (b)의 T_1과 T_2가 먼저 수행되어야 한다.
- (a)의 T_1이 수행되면 (b)의 T_3가 수행된다.
- (a)의 T_2가 수행되면 (b)의 T_4가 수행된다.
- (a)의 T_3가 수행되면 (b)의 T_4가 수행된다.
- (a)의 T_4가 수행되면 (b)의 T_4가 수행된다.
- (a)의 T_5가 수행되면 (b)의 T_4, T_5, T_6이 수행된다.

시험하고자 하는 프로토콜의 FSM은 정의 1과 같이 나타내어진다.

정의 1 하나의 FSM M_i는 (S_i, I_i, O_i, T_i, S_i^0)의 5개 요소로 구성된다. 각 요소들은 다음과 같다.

S_i : 상태 (state)들의 유한집합

I_i : 입력들의 유한 집합

O_i : 출력들의 유한 집합

T_i : S_i x I_i -> S_i x O_i에서 정의되는 결정적 트랜지션 (transition)

S_i^0 : M_i의 초기상태

그림 3의 (a)에 나타낸 FSM M_1을 정의 1의 형태로 표현하면 다음과 같다.

S_1 = {0, 1, 2, 3}

I_1 = {HIGH_I_1, high_i_2, HIGH_I_3, high_i_4, HIGH_I_5}

O_1 = {high_o_1, HIGH_O_2, high_o_3, HIGH_O_4, HIGH_O_5}

T_1 = { T_1 : 0 x HIGH_I_1 -> 1 x high_o_1,
T_2 : 1 x high_i_2 -> 3 x HIGH_O_2,
T_3 : 1 x HIGH_I_3 -> 2 x high_o_3,
T_4 : 2 x high_i_4 -> 3 x HIGH_O_4,
T_5 : 3 x HIGH_I_5 -> 0 x HIGH_O_5}

S_1^0 = 0

또, 상위 FSM과 하위 FSM과의 관계를 정의 2와 같이 나타낸다.

정의 2 상위 FSM M_i와 하위 FSM M_j가 주어졌을 때, FSM M_i와 M_j의 관계 R은 M_i의 트랜지션 $T_{i,k}$에서 M_j의 트랜지션 $T_{j,t}$들로의 사상 (mapping)을 나타내는 $R_{i,k}$의 집합이다. (여기서, $T_{i,k}$는 M_i의 k번째 트랜지션을 나타낸다.)

예를 들어, 그림 3의 (a)에 나타낸 FSM을 M_1, 그림 3의 (b)에 나타낸 FSM을 M_2라고 하면, 정의 2에 의해 M_1과 M_2의 관계 R은 다음과 같이 나타낼 수 있다.

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

정의 2에서 $R_{i,k}$는 정의 3과 같이 세분화하여 나타낼

수 있다.

정의 3 사상 $R_{i,k}$는 pre-execution($T_{j,t}$)와 carried-by ($T_{j,t}$)의 집합이다. 여기서, pre-execution($T_{j,t}$)은 $T_{i,k}$를 수행하기 위해 먼저 수행되어야만 하는 하위 FSM의 트랜지션을 나타내고, carried-by($T_{j,t}$)는 상위 FSM의 $T_{i,k}$가 수행될 때 수행되는 하위 FSM의 트랜지션을 나타낸다.

관계 1과 정의 3에 의해, 위에서 예로 든 M_1과 M_2의 관계 R의 원소 $R_{i,k}$는 다음과 같이 나타낼 수 있다.

$R_{1,1}$ = < 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($T_{2,4}$) >

$R_{1,5}$ = < carried-by($T_{2,4}$), carried-by($T_{2,5}$), carried-by($T_{2,6}$) >

여기서, <>는 각 원소들의 순서가 구별되는 집합을 나타낸다.

이러한 정의를 바탕으로, 다중 계층 프로토콜 시험 방법 알고리즘을 부록 1의 알고리즘 1으로 나타내었다. 알고리즘 1에 대해 보다 자세히 살펴보자. Main_generation 함수는 상위 FSM M_1, 하위 FSM M_2, 관계 R을 입력으로 받아서 시험 경우를 도출해주는 알고리즘의 main함수이다. Main_generation 함수 내부에서 모든 연산이 이루어지며, 필요한 함수가 호출된다. 우선, find_preamble 함수는 시험하고자 하는 트랜지션이 수행될 수 있는 상태에 도달하기 위해 필요한 트랜지션 중 초기 상태로부터의 최단 경로를 찾는 역할을 한다. find_preamble 함수를 통해 각 트랜지션의 preamble을 생성한 후, M_1의 모든 트랜지션에 대해 test_case_generation 함수를 호출한다. test_case_generation 함수는 generate_preamble 함수와 generate_testbody 함수가 수행되도록 한다. generate_preamble 함수는 find_preamble 함수에서 찾아 놓은 최단 경로의 트랜지션을 이용하여, 시험하고자 하는 트랜지션을 수행할 수 있는 상태로 이동시키는 함수이다. generate_testbody 함수는 시험하고자 하는 트랜지션을 방문하여 그에 대한 시험 경우를 도출해준다.

앞에서 정의한 FSM과 FSM 들간의 관계를 이용하여 generate_testbody 함수는 수행된다. 우선, pre-execution관계를 수행한다. 즉, 상위 FSM의 트랜지션의 수행을 위해 선행되어야 하는 하위 FSM의 트랜지션이 있는지 조사하고, 그 트랜지션들을 수행한다. Pre-execution관계의 하위 FSM 트랜지션들이 입력을 시험 대상 프로토콜로 송신하고, 출력을 시험 대상 프로토콜로부터 수신하는 behavior line PCO3! 입력, PCO3? 출력을 생성한다. 두 번째로 carried-by관계를 수행한다. 여기서는 상위 FSM의 트랜지션을 수행하고, 그 트랜지션이 수행될 때 수행되는 하위 FSM의 트랜지션을 수행한다. 이를 위해, 시험하고자 하는 트랜지션의 입력과 출력에 관련된 연산을 순차적으로 수행하게 된다. 상위 FSM 트랜지션의 입력이 ASP이면 PCO1! 입력을 생성하고, 입력이 PDU이면 PCO3! 입력을 생성한다. 입력이 PDU인 경우, behavior line의 '입력' 부분은 하위 FSM 트랜지션 입력의 파라미터로 포함되어진 형태이다. 상위 FSM 트랜지션의 출력이 ASP이면 PCO1? 출력을 생성하고, PDU이면 PCO3? 출력을 생성한다. 입력의 경우와 마찬가지로 출력이 PDU인 경우, behavior line의 '출력' 부분은 하위 FSM 트랜지션 출력의 파라미터로 포함된 형태이다.

상위 FSM의 모든 트랜지션에 대해 generate_testbody이 수행하고 난 후에 시험 경우의 수, behavior lines의 수, 이벤트의 수, 하위 FSM의 시험 커버리지를 출력한다.

4. 다중 계층 프로토콜 시험 방법의 적용

4.1 TCP/IP에 적용

이 절에서는 3장에서 제시한 알고리즘을 프로그램으로 구현하여, TCP/IP에 대해 적용한 결과를 보인다. 이를 통해 3장에서 제시한 알고리즘의 정확성을 입증할 수 있다.

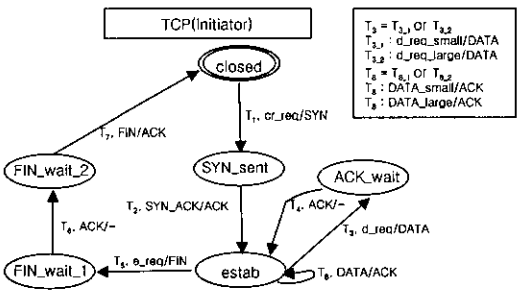


그림 4 간략화 한 TCP 프로토콜의 FSM

그림 4와 그림 5는 각각 TCP 프로토콜과 IP 프로토콜을 나타내는 FSM이다. 그림 4에서 TCP의 트랜지션 T_3과 T_8은 IP에서 fragmentation이 일어나지 않는 경우와 fragmentation이 일어 나는 경우의 2가지로 나누어 각각을 T_{3_1}, T_{3_2}와 T_{8_1}, T_{8_2}로 나타내었다. 3장에

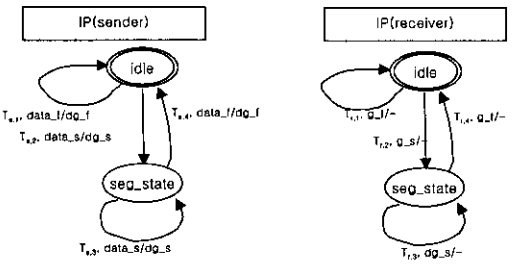


그림 5 IP 프로토콜의 FSM

서 제시한 알고리즘을 그림 4와 그림 5에 나타낸 TCP/IP에 적용하기 위해 FSM을 분석하여 각 트랜지션간의 관계를 찾아낸다. 예를 들어, T_2는 SYN_ACK/ACK인데 SYN_ACK을 IUT가 받게 하기 위해서는 시험기가 SYN_ACK를 포함하는 데이타그램(datagram)을 송신하고 IUT의 IP(receiver)의 데이타그램 수신이 되어야 하므로 $T_{r,1}$이 수행되고, 다음에 ACK을 IUT가 송신하여야 하는데 이를 위해서 IUT의 IP(sender)의 데이타그램 송신이 되어야 하므로 $T_{s,1}$이 수행된다. 그리고 그 데이타그램에 ACK이 포함되어 있는지를 확인하면 T_2가 수행된 것이 확인된다. 이와 같이 하여 상위 FSM (TCP)와 하위 FSM (IP)의 관계를 프로토콜 규격으로부터 다음과 같이 정의한다.

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} }

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($T_{s,1}$) >

R_{8_2} = < carried-by($T_{r,2}$), carried-by($T_{r,3}$), carried-by($T_{r,4}$), carried-by($T_{s,1}$) >

3장에서 제시한 알고리즘의 구현물을 그림 4와 그림 5에 나타낸 간략화 한 TCP/IP에 적용한 결과 전체 10개의 시험 경우가 도출되었다. 그 결과는 부록 2에 나타나 있다. 부록 2에 나타난 결과와 같이 10개의 시험 경우를 가지고 상위 FSM(TCP)의 8개의 트랜지션과 하위 FSM(IP)의 8개의 트랜지션 전체를 시험할 수 있다. 부록 2의 결과를 논문[1]에서 수작업을 통해 구한 결과와 비교해 보면 T_4(부록 2의 시험 경우에서 m_tc_T5)의 경우를 제외하고는 내용이 일치한다. T_4의 경우 분석을 해 본 결과 논문[1]의 오류임을 밝혀낼 수 있었다. 이와 같이 다중 계층 프로토콜 적합성 시험을 위한 알고리즘을 개발하고, 프로그램화하여 정확한 시험 경우를 자동으로 도출하였다는 점에서 본 논문은 의의를 지닌다.

이제, 부록 2의 시험 경우가 어떠한 과정을 통해 도출되었는지를 살펴보자. 그림 4에서 TCP의 3번째 트랜지션 가운데 IP에서 fragmentation이 일어나는 경우(T_{3_2})에 해당하는 시험 경우인 m_tc_T4에 대해 자세히 알아보겠다. 3장에서 제시한 알고리즘과 이번 절에 나타낸 TCP와 IP의 관계가 시험 경우를 도출하는 데 사용된다. 우선, 그림 4의 트랜지션 T_{3_2}를 수행하기 위해서는 T_{3_2}가 수행될 수 있는 상태인 'estab'로 이동을 시켜야 한다. 이 부분이 3장의 알고리즘에서 append-preamble에 해당한다. generation-preamble함수에 의해 T_{3_2}의 preamble로 생성된 T_2를 이용하여 append-preamble 함수는 behavior line +m_tc_T2을 생성한다. T_{3_2}의 관계에서 pre-execution은 존재하지 않으므로 carried-by 관계와 연관된 연산을 수행한다. 우선, T_{3_2}의 입력을 관찰한다. 입력이 ASP인 d_req_large이므로, 이것을 시험 대상 프로토콜로 송신하는 behavior-line PCO1! d_req_large을 생성한다. 다음 과정으로, T_{3_2}의 출력을 관찰한다. 출력이 PDU인 DATA이므로 T_{3_2}의 관계를 관찰한다. 남아 있는 원소는 'carried-by($T_{s,2}$)', 'carried-by($T_{s,3}$)', 'carried-by($T_{s,4}$)'이다. 따라서 시험 대상 프로토콜에서 트랜지션 $T_{s,2}$, $T_{s,3}$, $T_{s,4}$가 수행되도록 dg_s, dg_s, dg_f를 수신하는 behavior line을 생성한다. 이때, 처리중인 원소들이 carried-by이므로 트랜지션 T_{3_2}의 출력인 DATA가 $T_{s,2}$, $T_{s,3}$, $T_{s,4}$의 출력의 파라미터로 포함되어야 한다. 결국 생성되는 behavior line은 각각 PCO3 ? dg_s(DATA), PCO3 ? dg_s(DATA), PCO3? dg_f(DATA)가 된다

이러한 과정을 통해 얻어지는 다중 계층 프로토콜 시험 경우가 기존의 계층 단위 시험 방법과 비교하였을 때, 어느 정도의 효율성을 보이는지 살펴보자. 논문[1]에 의하면, 그림 4의 TCP를 시험하기 위해서는 8개의 시험 경우가 필요하고, 이를 위해 22줄의 behavior line이 사용되며, 총 48개의 이벤트가 필요하다. 또 그림 5의 IP를 시험하기 위해서는 8개의 시험 경우가 필요하고, 이를 위해 23줄의 behavior line이 사용되며, 총 47개의 이벤트가 필요하다. 따라서, 기존의 계층 단위 시험 방법으로 그림 4와 그림 5에 나타난 TCP/IP의 모든 트랜지션을 각각 시험하기 위해서는 16개의 시험 경우가 필요하고, 이를 위해 45줄의 behavior line이 사용되

며, 총 95개의 이벤트가 필요하다. 반면에 본 논문의 구현물은 10개의 시험 경우만을 사용하여 모든 트랜지션을 시험할 수 있었다. 또, 필요한 behavior line의 수와 이벤트 수도 각각 31, 63개로 감소한다. 즉, 다중 계층 프로토콜 시험 방법이 기존 계층 단위 시험 방법보다 적은 시험 경우로 동일한 커버리지를 갖는 것을 알 수 있다.

4.2 B-ISDN Signaling/SSCOP에 적용

앞에서, 본 논문에서 제안한 알고리즘의 구현물을 TCP/IP에 적용하여 논문[1]의 시험경우 생성 결과와 동일한 시험경우가 나오는 것을 보였다. 이 절에서는 제안된 알고리즘을 Q.2931신호계층과 SSCOP에 적용하여 알고리즘의 일반성을 확인하기로 한다.

부록 3의 그림 6은 신호 계층인 Q.2931을 나타내고, 부록 3의 그림 7은 SSCOP을 나타낸다. 앞 절의 TCP/IP와 유사한 방법으로 상위에 해당하는 신호 계층과 하위에 해당하는 SSCOP에 대한 FSM을 정의하고, 두 FSM의 관계를 정의하였다. 두 FSM 간에는 carried-by 관계 만을 가지는 TCP/IP와는 달리 pre-execution 관계도 존재한다. 예를 들어, 그림 6에서 트랜지션 T_1의 경우를 살펴보자. 신호 계층인 Q.2931이 PDU를 보내기 위해서는 하위의 SSCOP에 전달하게 되는데, SSCOP은 이미 만들어 놓은 연결이 있으면 그것을 사용하고 없으면 새로 만들게 된다.

Transition T_1의 경우 초기 상태에서 발생하는 트랜지션이므로 SSCOP에 연결 설정이 되어 있지 않다. 따라서 트랜지션 T_1을 수행하기 위해서는 우선 SSCOP이 연결 설정을 해야 하므로, 그림 7의 T_1과 T_3이 먼저 수행되어야만 한다. 또한, 그림 6의 T_1이 수행되는 과정에서 그림 7의 T_{28}이 수행된다. 결국 Q.2931 FSM의 트랜지션 T_1은 SSCOP FSM과 pre-execution($T_{2,1}$), pre-execution($T_{2,3}$), 그리고 carried-by($T_{2,28}$) 관계를 가지게 된다. Q.2931 FSM과 SSCOP FSM간의 모든 관계를 나타내면 다음과 같다.

R = { R_1, R_2, R_3, ..., R_{20}, R_{21} }

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($T_{2,29}$) >

R_{21} = < carried-by($T_{2,28}$) >

위에서 정의한 관계를 토대로 하여 3장에서 제시한 알고리즘의 구현물을 부록 3의 그림 6과 그림 7에 나타낸 B-ISDN Signaling/SSCOP에 적용한 결과 다중 계층 프로토콜 시험 경우로 21개가 도출되었다. 그 결과는 부록 4에 나타나 있다.

지금부터 부록 4의 시험 경우 중 pre-execution과 carried-by 관계를 모두 갖는 트랜지션 T_8을 예로 들어, 본 논문에서 제안한 알고리즘에 의해 시험 경우가 생성되는 상세한 과정을 설명한다. 트랜지션 T_8에 대한 시험경우는 부록 3에서 m_tc_T8이다. 부록 3의 그림 6에서 T_8은 초기 상태에서 수행되는 트랜지션이므로 preamble이 존재하지 않는다. 앞서 기술한 T_8의 관계에서 첫 번째와 두 번째 원소는 각각 'pre-execution($T_{2,2}$)'과 'pre-execution($T_{2,11}$)'이다. 따라서 시험대상 프로토콜에서 $T_{2,2}$와 $T_{2,11}$이 수행되도록 하는 behavior line PCO3! BGN과 PCO3? BGAK을 생성한다. 더 이상 pre-execution 관계는 존재하지 않으므로, carried-by와 연관된 연산이 수행된다. T_8의 입력이 'SETUP'이므로 하위 FSM인 SSCOP의 PDU 형태로 PCO3를 통해 입력이 된다. 따라서, Q.2931과 SSCOP의 관계 중 T_8에 해당하는 $R_{1,8}$의 'carried-by($T_{2,29}$)'를 해석하여, 해당 트랜지션 $T_{2,29}$가 수행되도록 하는 behavior line을 생성한다. 이때, 처리중인 원소가 carried-by이므로 트랜지션 T_8의 입력에 해당하는 SETUP이 $T_{2,29}$의 입력인 SD의 파라미터로 포함되어야 하며 결국 생성되는 behavior line은 PCO3! SD(SETUP) 이다. 다음 과정으로, T_8의 출력을 관찰한다. 출력이 ASP인 setup_ind이므로, 이것을 시험대상 프로토콜로부터 수신하는 behavior-line PCO1? setup_ind을 생성한다.

표 1은 다중 계층 프로토콜 시험 방법을 통해 얻어진

표 1 다중 계층 프로토콜 시험방법과 계층 단위 시험 방법의 비교

시험방법 / 측정	다중 계층 프로토콜 시험 방법	기존 시험방법		
		Q.2931에 대한 단일 계층 시험 방법	SSCOP에 대한 단일계층내포 시험 방법	두 방법의 합
시험 경우의 수	21	21	6	27
Behavior lines의 수	68(4+3+3+4+3+4+4+4+3+3+3+3+3+3+3+3+3+3+3)	64	12	76
Event의 수	160(4+6+6+7+8+9+9+4+6+6+6+6+8+8+8+8+9+9+11+11+11)	118	16	134
시험 커버리지	Q.2931의 모든 트랜지션 SSCOP의 6개 트랜지션	모든 트랜지션	SSCOP의 6개 트랜지션	Q.2931의 모든 트랜지션 SSCOP의 6개 트랜지션

결과와 기존의 방법을 통해 얻어진 결과를 비교하고 있다. 표 1에 나타난 바와 같이 다중 계층 프로토콜 시험 방법을 위한 21개의 시험 경우를 가지고 상위 FSM (Q.2931)의 21개의 트랜지션 중에서 21개, 하위 FSM (SSCOP)의 29개의 트랜지션 중에서 6개를 시험할 수 있다. 다중 계층 프로토콜 시험 방법은 하위 계층의 오류에 대한 behavior를 시험하는데 한계를 갖고 있기 때문에 하위 계층의 모든 트랜지션을 시험하지는 못한다. 이것은 시험기가 내부 인터페이스를 직접적으로 접근할 수 없고, 상위 계층 프로토콜을 통해 간접적으로 인터페이스를 제어하고 관찰하는 데서 생기는 문제이다. 이것은 내포 시험 방법[6,7,8]에서도 동일하게 존재하는 문제이다.

우리의 방법은 프로토콜 사이의 인터페이스가 개방되지 않은 다중 계층 프로토콜의 모든 계층을 동시에 시험할 수 있으며, 표 1과 같이 기존 방법보다 적은 시험 경우로 동일한 커버리지를 갖는 것을 알 수 있다.

표 1에 의하면 다중 계층 프로토콜 시험 방법이 기존 방법에 비해 오히려 많은 이벤트를 필요로 하는 것처럼 보인다. 그러나, 이것은 기존시험방법을 준비하는데 필요한 하위 계층의 행위를 계산하지 않았기 때문이다. 부록 3의 m_tc_T1과 이에 대응하는 단일 계층 시험 방법에 의한 시험 경우 s_tc_T1을 살펴보자. 표 2에서

표 2 다중 계층 프로토콜 시험 경우와 단일 계층 시험 경우

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

m_tc_T1에 비해 s_tc_T1은 간단하다.

즉, behavior line PCO3 ? BGN과 PCO3 ! BGAK가 m_tc_T1에는 있는 반면 s_tc_T1에는 없다. 이 부분은 시험 경우에서 표현되지 않았지만, s_tc_T1에서도 하위 계층에서는 실제로 수행이 된다. 좀더 상세히 설명하면 다중 계층 프로토콜 시험 방법을 사용하는 m_tc_T1에서는 Q.2931과 SSCOP 모두의 행위가 시험 경우에서 표현되고 또한 이벤트 수로 계산이 되었다. 하지만, 단일 계층 시험 방법을 사용한 s_tc_T1에서는 Q.2931의 행위만 시험 경우에서 표현될 뿐 SSCOP의 행위는 표현되지 않고 또한 이벤트 수로도 계산이 되지 않았다. 실제로는 단일 계층 시험 방법에서도 SSCOP은 시험 환경에서 서비스 제공자로 동작한다.

따라서 이를 계산하면 단일 계층 시험 방법의 이벤트 수는 늘어나게 된다. 결국 시험 경우에는 나타나지 않지만 하위계층에서의 수행을 모두 포함하여 표시한다면 s_tc_T1의 이벤트 수는 4가 된다. 이와 같은 방법으로 단일 계층 시험 방법에 대한 이벤트 수를 다시 계산하면 160개가 된다. 여기에 내포 시험에 필요한 이벤트 수 16을 더하면 기존 방법을 이용하였을 때 소요되는 이벤트 수는 176이 된다. 결국, 하위 계층의 이벤트도 모두 포함하여 실제로 수행되는 이벤트들을 비교하면 다중 계층 프로토콜 시험 방법의 이벤트 수가 적음을 알 수 있다. 이 결과를 정리하면, 표 3과 같다.

5. 결론 및 향후 연구

본 논문에서는 논문[1]에서 제안한 다중 계층 프로토콜 시험방안을 구현하기 위해 시험대상 프로토콜을 두개의 FSM으로 정의하고, 두 FSM에 대하여 트랜지션의 수행 관계를 정의하였다. 본 논문에서 사용된 두 FSM간의 관계는 pre-execution과 carried-by로 정의된다. 정의된

표 3 하위 계층의 이벤트를 포함한 경우에 두 가지 시험 방법의 비교

시험방법 / 측정	다중 계층 프로토콜 시험 방법	기존 시험방법		
		Q.2931에 대한 단일 계층 시험 방법	SSCOP에 대한 단일계층내포 시험 방법	두 방법의 합
시험 경우의 수	21	21	6	27
Behavior lines의 수	68(4+3+3+4+3+4+4+4+3+3+3+3+3+3+3+3+3+3+3+3)	64	12	76
Event의 수	160(4+6+6+7+8+9+9+4+6+6+6+8+8+8+8+9+9+11+11+11)	160	16	176
시험 커버리지	Q.2931의 모든 트랜지션 SSCOP의 6개 트랜지션	모든 트랜지션	SSCOP의 6개 트랜지션	Q.2931의 모든 트랜지션 SSCOP의 6개 트랜지션

FSM과 트랜지션간의 관계를 이용하여 다중 계층 프로토콜 시험 방법에 의한 시험 경우를 자동으로 생성하는 알고리즘을 제안하고 구현하였다. 구현된 알고리즘을 TCP/IP에 적용하여 논문[1]에서 수작업으로 생성된 결과와 동일한 결과가 얻어짐을 보였으며, 수작업으로 생성된 논문[1]의 시험경우에서 오류를 발견할 수 있었다. 또한, 구현된 알고리즘을 Q.2931/SSCOP에 적용함으로써 알고리즘의 일반적인 적용성을 보였다. 우리의 다중 계층 프로토콜 시험 방법은 프로토콜 사이의 인터페이스가 개방되지 않은 경우에도 시험이 가능하며, 기존 계층 단위 시험 방법에 비해서 적은 시험 경우로도 동일한 커버리지를 갖는다는 장점이 있다.

본 논문에서 제시한 FSM의 정의, 각 FSM의 트랜지션간의 관계에 대한 정의와 시험경우 생성 알고리즘은 두 개의 계층을 가지는 다중 계층 프로토콜 시험대상에 대해 시험경우를 도출할 수 있다. 이를 확장하여 세 개 이상의 계층으로 이루어진 다중 계층 프로토콜 시험대상에 대해서도 적용할 수 있도록 알고리즘을 일반화하는 것이 향후의 연구과제가 될 것이다. 또한 본 논문에서 제안된 알고리즘과 생성된 시험 경우를 사용하여 실제로 시험 환경을 구축하고 실제의 프로토콜 구현에 대한 시험에 적용함으로써, 본 논문에서 제안한 방법론의 적용성을 입증하는 것이 필요하다.

참 고 문 헌

[1] Yongbum Park, Myungchul Kim and Sungwon 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. Richard 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] Alexandre Petrenko and Nina Yevtushenko, "Fault detection in embedded components," International Workshop of Testing on Communicating Systems '97, pp.272-287, 1997.

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

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

부록 1. 다중 계층 프로토콜 시험 방법 알고리즘

```
main_generation(M1, M2, R) {
    for (T = each transition in M1) {
        find_preamble(M1, T)
        test_case_generation(M1, M2, R, T)
    }
    write statistics(# of test cases, # of behavior lines, # of events, test coverage)
}

find_preamble (M1, T1,j) {
    for (P = each transition in M1) {
        if (the ending state of P == the starting state of T1,j) choose P
    }
    select P which is the shortest path from the initial state
}

test_case_generation(M1, M2, R, T1,j ) {
    generate_preamble(M1, T1,j)
    generate_testbody(M1, T1,j , R )
}

generate_preamble(M1, T1,j) {
    generate the preamble of T1,j
}

generate_testbody(M1 , T1,j , R) {
    /* pre-execution */
    if (pre-execution of T1,j exists in R1,j) {
        if (an input of T1,j is ASP type) {
            write a behavior line concatenating PCO1! and the input of T1,j
            set pre_execution_flag
            i = 0
            while(pre-execution of T1,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 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 T1,j is PDU type) {
            i = 0
            while (pre-execution of T1,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 T1,j is ASP type and pre_execution_flag is unset)
        write a behavior line concatenating PCO1! and the input of T1,j
    else if (an input of T1,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 T1,j , and )
    }

    for ( each output of T1,j ) {
        if (the output of T1,j is ASP type)
            write a behavior line concatenating PCO1? and the output of T1,j
        else if (the output of T1,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 output of cur_transition, (, the output of T1,j , and )
        }
    }
}
```

알고리즘 1 다중 계층 프로토콜 시험 방법 알고리즘

부록 2. 다중 계층 프로토콜 시험 방법에 의한 TCP/IP의 시험 경우

```
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)
```

부록 3. Q.2931과 SSCOP의 FSM

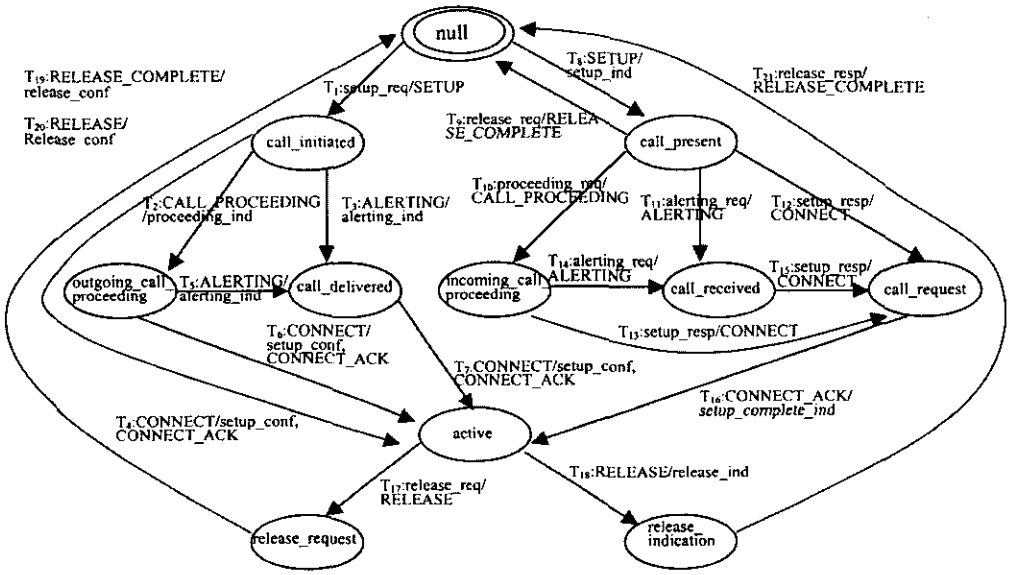


그림 6 Q.2931의 FSM

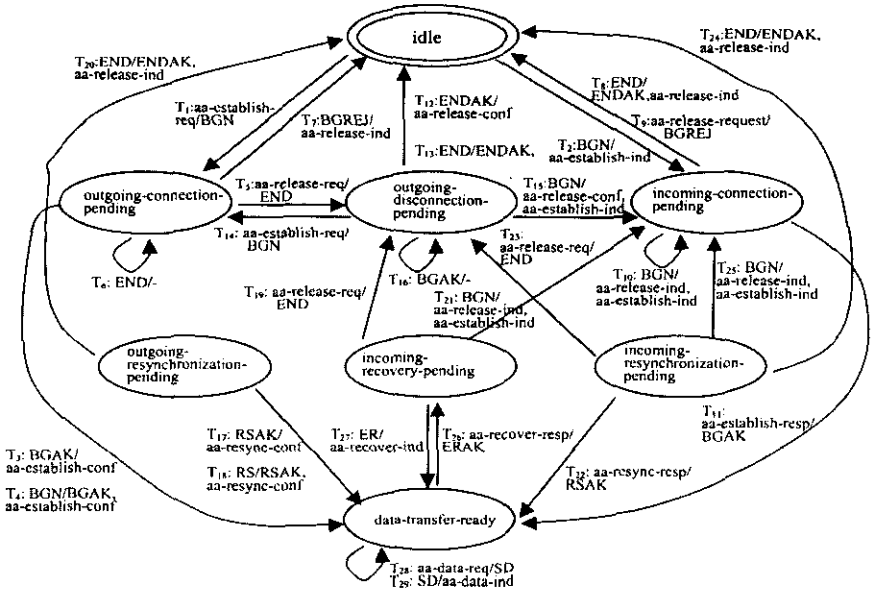


그림 7 SSCOP의 FSM

부록 4. 다중 계층 프로토콜 시험방법에 의한 B-ISDN SIGNALING/SSCOP의 시험 경우

```
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)
```


이 수 인

1999년 연세대학교 컴퓨터과학과 졸업(학사). 2001년 한국정보통신대학원대학교 공학부 졸업(석사). 2000년 한국통신 통신망연구소 인턴연구원. 현재 삼보정보통신 기술연구소 연구원

박 용 범

1986년 경북대학교 전자공학과 졸업(학사). 1989년 한국과학기술원 전산학과 졸업(석사). 1995 ~ 1996 미국 국립표준연구원 객원연구원. 1989년 ~ 현재 한국전자통신연구원 네트워크장비시험팀장. 관심분야는 protocol engineering, performance evaluation, and network testing

김 명 철

정보과학회논문지 : 정보통신
제 28 권 제 2 호 참조