액티브 네트워크 응용의 검증
(Verifying Active Network Applications)

박 준 철 †
(Jun-Cheol Park)

요 약 인터넷과 같은 기존의 패킷 교환 네트워크에서 라우터들이 수동적으로 단순히 패킷을 가능한 빠르게 전달하는 역할에 그치는 것에 반해, 액티브 네트워크의 내부 라우터들은 사용자에 의해 정의된 연산을 통과 패킷에 적용하게 된다. 그러므로 인터넷과 달리 액티브 네트워크의 패킷 처리 과정은 네트워크 사용자 또는 응용 프로그램 단위로 특화될 수 있다. 액티브 네트워크는 사용자에게 종단 호스트 뿐 아니라 내부 라우터에서 수행될 프로그램을 기술하거나 제어하는 정보를 네트워크 내부에 투사할 수 있도록 한다. 따라서 액티브 네트워크는 사용자에게 제공되는 프로그래밍 인터페이스를 통해 사용자가 네트워크의 동작을 "프로그램"하므로 실현된다.

본 논문에서는 액티브 네트워크 실현을 위한 네트워크 프로토콜 모델을 제안하고, 이 모델에 의해 표현된 액티브 응용의 정확성을 검증하는 방법을 제안한다. 제안한 검증 방법은 액티브 네트워크의 특정 플랫폼이나 프로그래밍 언어에 의존하지 않으며, 알고리즘으로 표현되어 컴퓨터 프로그램에 의해 자동화될 수 있다는 장점이 있다. 제안한 프로토콜 모델과 검증 방법은 액티브 경매 응용을 통해 그 사용 예를 보였다.

키워드 : 액티브 네트워크, 라우터, 프로토콜 모델, 검증

Abstract The routers in an active network perform customized computations on the messages flowing through them, while the role of routers in the traditional packet network, such as the Internet, is to passively forward packets as fast as possible. In contrast to the Internet, the processing in active networks can be customized on a per user or per application basis. Active networks allow users to inject information into the network, where the information describes or controls a program to be executed for the users by the routers as well as the end hosts. So the network users can realize the active networks by "programming" the network behavior via the programming interface exposed to them.

In this paper, we devise a network protocol model and present a verification technique for reasoning about the correctness of an active application defined using the model. The technique is developed in a platform- and language-independent way, and it is algorithmic and can be automated by computer program. We give an example dealing with network auction to illustrate the use of the model and the verification technique.

Key words : active network, router, protocol model, verification

1. 서 론

액티브 네트워킹은 네트워크를 프로그램할 수 있는 환경을 제공하고자 하는 시도의 연구 분야이다. 인터넷으로 대표되는 현재의 네트워크는 네트워크 내부의 스위치나 라우터의 역할, 즉 수행되는 프로그램이 고정된다. 따라서 이들 노드에서 수행되는 프로그램을 수정, 개선하기 위해서는 오랜 표준화 과정과 그 과정의 결과를 스위치 및 라우터 공급자들이 수용하여 새로운 프로토콜을 새로운 장치에 도입하는 과정이 필요하다. 액티브 네트워크[1,2]는 프로그램 코드나 제어 정보를 네트워크 내부에 침투시켜 수행되게 함으로 새로운 네트워크 서비스 실현을 위한 프로그래머블 플랫폼을 제공한다. 네트워크 사용자는 이러한 액티브 네트워크 기술을 이용하여 자신이 원하는 새로운 또는 개선된 네트워크 서비스를 제공

· 본 논문은 과학기술부 KISTEP 2000(과제번호 00-B-WB-08-A-06) 지원에 의해 수행되었음.
† 정 회 원 : 홍익대학교 컴퓨터공학과 교수
jcpark@cs.hongik.ac.kr
†† 논문접수 : 2001년 6월 7일
심사완료 : 2002년 7월 26일

받을 수 있게 된다. 액티브 네트워크의 장점은 이와 같이 새로운 프로토콜 및 서비스의 구현과 도입이 빨라지고, 여러 사용자의 각기 다른 요구에 대해 특화된 서비스를 제공하는 것이 용이하다는 것이다.

한편 액티브 네트워크는 네트워크의 내부 노드에서 수행되는 프로그램이 고정되지 않는다는 점에서 성능과 보안에 관련해 기존의 IP 기반 인터넷에 비해 더 많은 문제의 소지를 내포하고 있다. 현재 액티브 네트워크 기술에 대한 연구가 활발히 이루어지고 있는데[3,4,5,6,7,8,9,10,11,12,13,14,15,16], 이러한 연구 결과는 감당할 수 있는 정도의 부하로 보안의 문제를 해결할 수 있으며, 성능 측면에서도 기존 IP 라우터의 성능에 크게 뒤지지 않는 액티브 네트워크를 실제로 구성할 수 있다는 가능성을 보여주고 있다. 액티브 네트워크에서 새로운 프로토콜 및 서비스의 실행은 네트워크의 사용자가 액티브 네트워크 노드가 제공하는 프로그래밍 인터페이스를 이용하여 자신이 원하는 기능을 합성함으로 이루어지게 된다. 이에 반해 기존의 IP 기반 인터넷에서는 미리 정의된 네트워크 프로토콜들만이 네트워크 노드에서 수행되고, 네트워크의 사용자는 IP 패킷 헤더의 제한된 인터페이스만을 이용하여 자신이 원하는 서비스를 제공받을 수 있다. 결국 액티브 네트워크상의 서비스는 각 스위치 및 라우터에서 수행될 프로그램이 유동적이라는 점에서 상당히 다양해 질 수 있으며, 실제 각 사용자가 원했던 서비스가 종단 호스트 및 액티브 라우터에서의 특화된 프로그램 수행을 통하여 정확히 이루어지는지 확인하는 것이 주요한 연구 문제가 된다. 서비스를 제공하는 액티브 응용의 정확성을 검증하는 것은 액티브 노드에서 수행되는 프로그램이 액티브 패킷의 전달 내용에 따라 달라질 수 있으므로 기존 네트워크에서의 프로토콜 검증과는 그 성격이 다르다. 구체적으로 액티브 프로토콜의 정확성 검증을 위해서는 액티브 코드가 네트워크 내부 노드에 투입되는 방식, 액티브 노드에서 제공하는 프로그래밍 인터페이스의 성격, 그리고 보안(security) 및 안전성(safety) 보장을 위한 장치 및 소프트웨어가 액티브 코드의 표현 및 수행 능력에 영향을 줄 수 있다는 점을 고려해야 한다.

본 논문은 액티브 패킷 또는 액티브 확장 방식을 채택하는 액티브 서비스 수행 환경에서 모두 적용될 수 있는 포괄적인 액티브 프로토콜의 검증 방법을 제안한다. 이 방법은 특정 수행 환경 - 기존의 ANTS[10], PLAN[5], CANES[13], Smart Packet[9] 등 -에 의존적이지 않으면서 전체 네트워크의 관점에서 서비스가 원하는 동작을 수행하는지를 안전성(safety) 및 활동성(liveness)의 성질로서 검증하게 되며, 알고리즘으로 표현될 수 있어 컴퓨터 프로그램에 의한 자동화가 가능하다는 특징이 있다. 이를 위해 본 논문에서는 우선 여러 액티브 서비스 실현 방법들을 표현할 수 있는 프로토콜 프로그래밍 방식을 서술하고, 이를 바탕으로 액티브 응용 검증 방법을 제안한다. 본 논문의 구성은 다음과 같다. 2장에서는 액티브 네트워크에 대한 기본 지식과 액티브 네트워크에서 프로토콜이 구현되는 방식을 기존의 관련 연구 결과를 중심으로 소개하고, 어떻게 액티브 응용의 동작을 검증할 수 있는가의 문제에 대한 접근 방법을 서술한다. 3장에서는 여러 방식의 액티브 서비스 실현에 적용시킬 수 있는 액티브 프로토콜 모델을 제안하고, 이 모델을 기반으로 만들어진 액티브 응용의 검증 방법을 제안한다. 4장에서는 액티브 경매 서버의 예를 가지고 실제 제안한 검증 방법의 적용 사례를 보이며, 5장에서 결론 및 향후 연구 과제를 제시한다.

2. 액티브 네트워크 기본 및 관련 연구

액티브 네트워크에 대한 연구는 현재 미국을 중심으로 여러 나라에서 활발히 진행되고 있다. 대학이나 연구소, 기업체들은 차세대 네트워크의 패러다임으로 액티브 네트워크의 채택을 심각하게 고려하고 있으며, 다양한 연구 주제 - 네트워크 구조, 보안, 노드 운영체제, 서비스 합성, 응용 서비스 등 -에 대한 연구를 진행하고 있다[2,3,4,5,6,7,16,17,18].

2.1 액티브 노드 구조 및 실현 방식

액티브 네트워크 노드는 어떻게 패킷을 해석하고 처리하는가를 담당하는 요소로서, 액티브 노드 구조[17]는 액티브 패킷을 처리하는 실행환경(Execution Environment)과 이 실행환경에서 요구되는 운영체제 기능을 제공하는 노드 운영체제(Node OS)로 대별되며, 여러 개의 실행환경이 하나의 노드 운영체제상에 공존할 수 있다. 노드의 실행환경은 사용자에 제공되는 API로서 이 인터페이스를 이용하여 사용자는 종단간의 네트워크 서비스를 프로그램 하게 된다. 사용자는 이러한 실행환경을 통해서 노드의 자원(CPU 시간, 메모리, 링크 대역폭)에 접근할 수 있다. 노드 운영체제[18]는 노드의 자원을 관리하고 이러한 자원에 대한 요구를 제어하는 기능을 함으로 실행환경을 자원 관리의 세부적인 사항으로부터 투명할 수 있도록 한다.

액티브 네트워킹을 실현시키기 위한 접근 방식은 액티브 노드에 수행 가능한 프로그램을 적재하고 패킷이 어떤 코드를 수행시킬지를 지정하는 식별자를 운반하도록 하는 액티브 확장(active extension 또는 discrete)

방식과 패킷 자체에 실행시킬 프로그램과 데이타가 탑재되어 전송되는 액티브 패킷(active packet 또는 integrated) 방식이 있다[17]. 이 두 방식을 혼용하여 간단한 코드의 경우 액티브 패킷을 통해 전송하고, 복잡하거나 큰 규모의 코드는 미리 액티브 노드에 적재하여 운용하는 방식도 가능하다[14,21]. 액티브 패킷 방식은 안전성 및 보안을 위한 부담해야 할 부하가 크기 때문에 성능 측면에서 문제가 생길 수 있으나, 이론적으로 액티브 패킷에 자신이 원하는 어떤 프로그램도 실을 수 있기 때문에 사용자에게 유연성을 제공한다. 액티브 확장 방식은 노드에 적재된 코드가 초기에 보안 검증을 마치고 이후에는 보안에 관련한 부하없이 실행할 수 있다는 점에서 액티브 패킷 방식에 비해 성능 측면에서 우월하나, 사용자의 프로그램 표현 능력에 제한을 둔다.

2.2 액티브 프로토콜 검증 및 관련 연구

본 연구에서 네트워크는 다수의 노드와 이들 사이의 패킷 전송 통로가 되는 채널들로 구성된다. 노드와 채널의 연결 형태는 어떤 네트워크 위상(topology)의 제약도 없다고 가정한다. 각 노드는 입력 채널로부터 패킷을 받아, 이 패킷에 포함된 정보를 이용하여 패킷을 처리하고, 그 결과로 노드의 상태가 변경되거나 출력 채널로 패킷을 내보내는 과정을 반복하게 된다. 프로토콜은 관련 노드들이 패킷을 주고 받음으로 각각의 작업을 수행하는 전 과정을 정의한 분산 알고리즘이라 할 수 있다. 액티브 네트워크의 내부 액티브 노드에서 수행되는 코드는 액티브 패킷 방식과 액티브 확장 방식 중 어떤 실현 방식을 택하느냐에 따라 정해지는 절차가 다르기 때문에 프로토콜을 기술하고 액티브 서비스를 합성하는 방식은 다양해지게 된다. 본 논문의 모델은 액티브 프로토콜을 기술함에 있어 각 노드의 동작을 연속된 함수 수행으로 모델링하여 내부 액티브 노드에서 수행되는 프로그램의 출처가 패킷인지 아니면 다른 방법으로 노드에 적재된 것인지 구분을 두지 않는다. 프로토콜 공학 분야에서 널리 쓰이고 있는 모델링 기법으로서 유한상태기계(finite state machine)에 기초한 여러 모델들[22, 23,24,25]이 있다. 본 논문의 제안 모델은 단순한 유한상태기계 모델의 표현 능력을 변수의 도입 등으로 확장시킨 확장유한상태기계(extended finite state machine)[24,25]에 그 기반을 두고 있다.

프로토콜 검증에 대한 관련 연구로 CANES 프로젝트의 액티브 프로토콜 검증 연구[20]가 있다. 이 연구에서는 액티브 확장 방식의 한 형태로 노드 내장 프로그램의 슬롯이라는 특정 위치에 패킷의 코드를 투입하는 방법으로 액티브 프로토콜을 합성하고, 합성 프로토콜의 정확성을 검증하는 방법을 제시했다. 이 연구의 핵심은 실제 패킷에 실려오는 코드가 수행하는 역할은 대부분의 경우 몇 가지로 요약된다는 점에 착안하여 그 수행 위치를 제한하고, 노드 내장 프로그램에서 큰 흐름을 고정하더라도 유연성에 큰 타격을 입지 않는다는 점을 보인 것이다. 그러나 이 연구는 슬롯 프로그래밍 모델 방식에서만 적용되며, 검증 방법을 적용하기 위해 노드 내장 프로그램 및 패킷의 액티브 코드에 제한을 두었다. 또한 증명 방식이 단언(assertion)에 기초한 논리적 추론 방식이기 때문에 알고리즘으로 표현할 수 없다.

액티브 네트워킹을 실현하기 위해 노드의 실행환경을 구현하고 이를 기반으로 액티브 서비스를 제공하고자 하는 많은 연구가 있었다[4,7,8,9,10,11,12,13,14,15]. ANTS (Active Network Transport System)[10]는 MIT에서 개발된 방식으로 사용자에게 최대한의 유연성을 보장하는 API를 제공한다. 캡슐이라 불리는 패킷은 자신을 처리하기 위한 코드의 식별자와 코드에 적용되는 인자를 탑재하고 전송된다. ANTS의 시작품은 Java로 쓰여졌으며, JVM의 바이트코드 검증[19], 그리고 sandboxing에 의존하여 안전성을 보장하고 있다. 그러나 Java로 쓰여진 코드들로 구성된 네트워크 프로토콜의 성질을 추론하고 검증하는 방법은 제시하고 있지 못하다. PAN[8]은 MIT에서 ANTS에 이어 추진한 프로젝트로서 액티브 처리의 성능 향상을 우선 목표로 삼고 코드 캐싱(caching), 패킷 수행 부분의 커널화, 데이타 복사 최소화의 방법을 도입하였다. PAN에서는 Java 외에도 x86의 목적 코드를 사용하였고, ANTS에서와 마찬가지로 어떻게 프로토콜의 성질을 검증할 수 있는가는 명확히 제시되어 있지 않다. UPenn의 SwitchWare 프로젝트[21]에서는 액티브 프로토콜의 프로그래밍을 위해 두 단계의 언어를 사용하는데, 패킷 수준의 스크립트 언어로서 PLAN(Packet Language for Active Networks)[5]이라는 간단한 형태의 언어, 그리고 패킷에 의해 참조되는 노드 상존 서비스(ANTS의 노드 API와 유사) 구현을 위해 별도의 언어를 사용한다. 두 단계의 프로그램 코드를 조합하여 원하는 프로토콜을 완성하는 방식은 CANES의 내장 프로그램 및 투입 코드의 구분과 유사하다고 볼 수 있다. PLAN 언어는 ML 언어에서 그 개념을 가져오고 몇 가지 제한을 가하여 재귀(recursion) 코드가 허락되지 않는 등 노드의 자원을 과다하게 사용하지 못하도록 할 수 있다. SwitchWare에서는 상이한 능력을 가지는 언어로 작성된 코드들이 결합하여 프로토콜의 동작이 기술된다. 일면 CANES 프로젝트의 구조와 유사하나, CANES의 노드 구조에서는 슬롯이라는 특정

위치로만 코드 투입이 가능하다는 점에 더 제한적이다. 따라서 검증에 있어서 PLAN으로 작성된 패킷 코드들이 노드의 상존 서비스와 결합할 때의 종합적인 동작은 CANES 프로젝트와 비교하여 좀 더 일반적인 상황을 고려해야 한다. 한편 같은 UPenn의 ALIEN[5]이라는 액티브 네트워킹 구조는 보안의 측면에 치중하면서 액티브 패킷 및 액티브 확장 방식을 모두 지원한다. SANE (Secure Active Network Environment)[4]은 ALIEN의 개념을 구현한 환경으로서 OCaml이란 언어로 작성되었으며, 암호화 및 API의 module thinning을 제공하고 매 패킷마다 인증(authentication)을 받도록 할 수 있게 하였다. 물론 이런 안전성 및 보안 수준의 달성은 성능의 감소를 필연적으로 가져오게 되었다. ALIEN의 목적은 만족할 수준의 보안 달성에 있으며, 여기서 프로토콜의 동작을 추론하고 검증하는 부분은 관심을 받지 못하였다. BBN의 Smart Packet[9] 프로젝트는 액티브 패킷 방식으로서 네트워크 관리라는 응용 분야에 액티브 네트워크 기술을 적용하는 것이다. 패킷은 하나의 데이타 링크 계층 프레임에 탑재될 수 있을 정도의 소규모 프로그램을 포함하며 Sprocket이라는 언어로 작성되고 Spanner라는 어셈블리 언어로 컴파일 되어 각 노드에서 수행된다. 여기서 제시된 프로토콜의 목적은 네트워크 관리에 집중되어 있으며, 일반적인 프로토콜을 설계하고 그 동작을 추론하는 것은 프로젝트의 관심 분야가 아니었다. 이외에도 여러 액티브 네트워크 서비스 관련 프로젝트가 있으나 대부분 프로젝트의 목적은 액티브 프로토콜을 기술하는 특정 언어를 설계하거나, 이러한 언어로 작성한 프로토콜의 수행시 보안에 관련된 문제점이 없음을 확인하거나 보장하는 시스템을 구현하는데 있었다.

3. 모델링 방법

3.1 네트워크 프로토콜 모델

본 연구에서 네트워크는 노드들과 이들을 연결하는 통신로인 채널로 구성되었다고 가정한다. 네트워크 응용은 각 노드 프로세스(process)의 집합으로 이루어지고, 여기서 프로세스는 각 노드에서 액티브 서비스 제공을 위한 프로토콜 수행의 단위이다. 액티브 응용은 네트워크 노드에서 수행되는 프로세스들의 내부 연산 및 채널을 통한 상호 데이타 송수신에 의해 그 원하는 기능을 달성하게 된다. 구체적으로 각 프로세스는 일련의 함수를 호출하고 그 수행 과정을 제어하는 역할을 한다. 따라서 노드의 수행 상태는 현재 해당 노드의 프로세스 상태를 의미하며, 각 변수의 현재 값으로 나타낼 수 있다. 각 채널은 노드간을 연결하는 통신로이며, 상태는 현재 그 채널을 통해 전송된 메시지 중 수신측 노드의 프로세스에서 읽지 않은 모든 메시지의 순열로 표현된다. 채널은 전송중 메시지를 유실, 훼손할 수 있다고 가정한다. 따라서 네트워크 응용은 다음과 같이 모델링될 수 있다. 응용 P는 tuple $(\langle P_i\rangle, \langle C_{ij}\rangle)$ 로 정의되며, 여기서 $P_i, 1\le i\le n$, 는 프로세스로서 확장유한상태기계로 표현되고, $C_{ij}, 1\le i,j\le n, i\neq j$, 는 P_i로부터 P_j로의 채널로서 그 내용 c_{ij}는 현재 목적지에서 읽어가지 않은 메시지의 순열이다. 각 프로세스 P_i는 $(S_{pi}, V_{pi}, E_{pi}, \delta_{pi}, s^0_{pi})$로 정의되며, S_{pi}는 상태(state)의 유한 집합으로 각 상태는 확장유한상태기계의 한 노드에 대응한다. V_{pi}는 변수(variable)들의 유한 집합으로서 프로세스에서 참조 및 변경하는 변수를 포함한다. 액티브 응용의 어떤 프로세스가 해당 노드의 라우팅 테이블과 같은 자료 구조를 변경하는 것이 필요한 경우는 타 프로토콜의 수행에 영향을 주지 않기 위해 자신의 복사본 자료 구조를 만들어 그 내용을 변경한다고 가정한다. 따라서 프로토콜의 각 노드 프로세스는 그 프로세스의 지역(local) 변수만을 참조하고 변경하게 된다. E_{pi}는 이 프로세스에서 수행하는 사건의 유한 집합으로 확장유한상태기계의 간선(edge)들에 해당하며, $\delta_{pi}: S_{pi}\times E_{pi}\rightarrow S_{pi}$는 부분상태천이함수(partial state transition function)로 한 상태에서 어떤 사건이 수행된 경우 다음 상태를 결정하는 역할을 한다. s^0_{pi}는 프로세스의 초기(initial) 상태이다. 프로세스 P_i의 각 사건 $e\in E_{pi}$는 하나의 함수에 해당하며, 다음 구성요소 $(pre, input, post, body)$로 정의된다. 여기서 $pre(e)$는 사건(함수) e의 수행을 위한 선행조건(precondition)으로 해당 함수를 수행하기 위해서 요구되는 전제 조건이며, 현재의 상태에서 참 또는 거짓으로 판별된다. 이 조건은 해당 프로세스를 수행하는 노드의 실행환경에서 해당 함수가 수행될 수 있는 상태인지를 검사하는 것으로 특정 메시지의 도착, 노드 자원 가용성 여부, 보안 등의 측면을 종합적으로 판단하여 결정하게 된다.[1] $input(e)$는 함수 e의 입력 인자들로서 함수 e의 수행시 필요한 외부 요인(예를 들어, 도착한 패킷 내용) 변수를 포함한다. $post(e)$는 함수의 수행 결과로서 해당 프로세스내 변수의 결과 값, 그리고 해당 프로세스의 현재 상태 정보(CPU 및 메모리, 링크 대역폭 사용량)도 포함할 수 있다. 사건 e가 수행되기 위한 필요 조건은 (1) 현재 프로세스의 상태가 e를 나타내는 상태 천이(간선)의 시작 부분에 해당하는 상태(노드)이며, (2) $pre(e)$가 만족되어

1) 이 부분에 기존 연구 결과의 각종 정책 제어(policy control) 구현 내용을 반영할 수 있다.

```
/*(2) prc: (head(c in(x),x)=<y, x, [something]>) ∧ (state= ready_to_rcv)*/
  function sample_rcv(/* input: */ packet <y,x,[something]>) {
/* body: */
      extract the packet <y,x,[something]> from the channel C in(x),x;
      state ← something_rcvd;
  }
 /*post:   (c^old in(x),x=<y, x, [something]> · c^new in(x),x) ∧ (state= something_rcvd)*/
```

그림 1 프로세스의 함수 명세의 예

야 하는 것이다. 한편, *body(e)*는 함수 *e*의 몸체 부분에 해당하는 내용으로 *pre(e)*가 만족되는 상황에서 *input(e)*이 입력되었을 때 *body(e)*가 수행되면 반드시 종료하고, 종료후 *post(e)*의 결과를 내게 된다. *body(e)*에 속하는 실행문으로 특별히 *send*(<*s*, *d*, [*payload*]>)는 메시지의 송신을 의미하는데 *s*는 송신자(source) 주소, *d*는 수신자(destination) 주소, *payload*는 메시지의 페이로드(payload) 부분을 각각 나타낸다. 기타 *body(e)*의 실행문은 종료가 보장되는 모든 문장이 될 수 있으며,[2] 각 실행문의 결과는 변수의 내용 변경과 함수를 수행하는 프로세스의 CPU 및 메모리, 링크 대역폭 사용량의 변경에 한한다.

프로세스간의 통신을 위한 채널 C_{ij}는 프로세스 P_i에서 프로세스 P_j로 메시지를 송신하는 명령문 *send*(<ad_i, ad_j, [*payload*]>) - 여기서 ad_i, ad_j는 각각 프로세스 P_i와 P_j의 네트워크 주소 - 가 프로세스 P_i의 함수에서 수행되었을 때, 그 내용 c_{ij}가 다음과 같이 변경된다. (1) 정상 전송시 $c_{ij} \leftarrow c_{ij} \cdot <ad_i, ad_j, [payload]>$; (2) 유실시 $c_{ij} \leftarrow c_{ij}$; (3) 훼손시 $c_{ij} \leftarrow c_{ij} \cdot <ad_i, ad_j, [payload^*]>$ (단 *payload**는 *payload* 의 변경된 내용을 의미). 여기서 · 는 연결(concatenation) 연산을 의미한다.

액티브 네트워크에서는 네트워크 내부 노드에서 패킷 처리를 위해 수행될 프로그램이 존재하여야 한다. 패킷 처리를 위한 프로그램 코드는 본 논문의 모델링 방식에서 프로세스의 함수에 대응된다. 따라서 중간 액티브 노드의 프로세스는 실행할 코드가 없는 상황, 즉 해당 함수가 존재하지 않는 상황에 대처하여 동적으로 코드 적재를 할 수 있는 기능을 포함하여야 한다. 본 논문의 각 사건(함수) *e*의 *pre(e)*는 해당 함수가 적재되어 있는지의 여부를 시험하는 부분이 존재하며, 만약 해당 함수의 호출이 요구되는 상황에서 코드가 적재되어 있지 않은 경우 코드 적재 요구를 위한 함수를 호출한다.

프로토콜의 검증을 위하여 각 사건(함수)의 수행은 인터럽트, 오류 발생 등의 이유로 중단되지 않는다고 가정한다. 물론 실제 상황에서 중단 상황이 발생할 가능성은 있으나, 본 연구의 목적은 액티브 노드에서 각 프로세스의 정상적인 수행 흐름에서 원하는 서비스를 달성할 수 있는 것을 보이는데 있으므로 앞으로 중단 상황이 없는 경우만을 고려한다.

3.2 함수 수행 스케쥴링

각 노드의 프로세스는 종료되는 순간까지 해당 노드의 자원(CPU 시간, 메모리, 링크 대역폭)을 사용하여 수행된다. 프로세스는 일련의 연속된 함수 수행으로 구성되며, 함수의 수행 흐름은 프로세스를 표현하는 확장유한상태기계에 명시된다. 만약 둘 이상의 수행 가능한 함수가 존재한다면 임의의 함수 하나가 선택되어 수행된다. 형평성(fairness)에 대한 가정은 약한 형평성(weak fairness) 모델을 상정하여 어떤 함수가 연속해서 수행 가능한 상태로 있는 경우 유한 시간 내에 그 함수는 반드시 수행이 된다고 가정한다.

3.3 액티브 응용의 합성 및 성질 표현

액티브 응용은 각 노드에서 프로세스에 의해 수행될 일련의 함수를 정의하고, 이 함수의 수행 순서 및 흐름을 결정함으로 합성된다. 기존 인터넷의 노드에서 각 계층의 프로토콜이 메시지의 도착 및 송신에 의해 순차적으로 수행되는 것과 같이(예를 들면, 메시지 도착시 이더넷(Ethernet) 수신, IP 처리, TCP/UDP 처리, 응용 프로세스 처리의 순서로 upcall이 일어나는 것) 액티브 노드에서의 메시지 처리도 메시지 전송 및 도착의 사건에 대해 일련의 연속적인 함수가 호출되는 수행 모델을 가정한다. 이 때 수행되는 함수의 흐름은 각 프로세스를 나타내는 확장유한상태기계의 상태 천이에 의해 정해진다. 액티브 노드 프로세스에서 수행되는 함수의 한 예로서 다음 그림은 도착 패킷을 읽고 변수 값을 변경시키는 간단한 함수에 대해 선행조건, 입력 값, 몸체 및 수행 후 조건을 나타내고 있다.

위 그림의 함수 명세에서 *head*($c_{in(x),x}$)는 채널 $C_{in(x),x}$의 헤드(채널의 맨 앞)에 도착한 패킷을 의미하며, 채널 $C_{in(x),x}$는 프로세스 P_x의 인접(neighbor) 노

2) 저장(assignment)문, 선택(if-then-else/switch)문, 횟수가 제한된 반복(for/while)문 및 종료됨을 보장할 수 있는 재귀(recursive)문을 가질 수 있다.

드 $P_{in(x)}$에서 P_x로의 채널을 나타낸다. 또한 $c^{old}_{in(x),x}$ 및 $c^{new}_{in(x),x}$는 각각 해당 함수 수행 전 및 수행 후의 채널 $C_{in(x),x}$의 내용을 표현한다. 이러한 표기 방식은 이 논문을 전개하는데 계속해서 사용할 것이다. 한편 본 연구에서 액티브 네트워크는 액티브 호스트 및 액티브 라우터 만으로 이루어지거나, 또는 좀 더 현실적인 상황으로 다수의 기존 IP 라우터에 약간의 액티브 라우터 및 액티브 호스트가 결합된 상황을 가정한다. 후자의 경우 액티브 패킷은 IP 주소 체계를 그대로 사용하고, 기존 IP 라우터는 자신에게 도착한 액티브 패킷을 IP 패킷으로 취급하여 목적지 주소만을 보고 경로상의 다음 노드로 전송한다. 결국 두 액티브 노드 사이에 일련의 IP 라우터들이 존재하는 경우 액티브 패킷은 중간의 IP 라우터에서 어떤 특별 취급도 받지 못하며, 액티브 노드에 도착한 경우에만 액티브 처리가 이루어진다. 모든 패킷의 경로는 특정한 라우팅 프로토콜(예를 들어, 인터넷 라우팅 프로토콜 또는 응용에 특화된 액티브 라우팅 프로토콜)에 의해 생성된 라우팅 테이블을 참조하여 결정된다고 가정한다.

액티브 응용의 동작은 해당 프로토콜이 안전성 및 활동성 성질을 만족하는지의 여부를 결정하는 방식으로 검증된다. 안전성 성질이란 분산 프로그램의 수행중 원하지 않는 결과에 이르지 않을 것이라는 점을 확인하는 성질로서, 대표적인 예로 교착상태 없음(freedom from deadlock)을 들 수 있다. 활동성 성질은 프로그램 수행이 결국 원하는 상태에 도달할 것이라는 점을 주장한다. 즉, 활동성 성질은 프로그램이 원래의 설계 목적에 부합하는 동작을 하여 궁극적으로 원하는 결과 상태에 이르는가를 검증하는데 사용한다. 본 연구에서 네트워크 응용의 안전성 및 활동성 성질은 각 노드 및 채널 상태의 조합으로 표현되는 전역(global) 상태에서 참 또는 거짓으로 판정할 수 있는 논리식(logical formula)을 사용하여 표현한다. 전역 상태는 각 노드 프로세스들이 현재 어느 함수를 수행한 후의 상태인가에 따라서 결정되며, 안전성 및 활동성 성질은 각 노드 프로세스 함수의 수행 전 및 후의 조건의 조합에 의해 표현된다. 그러므로 본 논문에서는 각 노드 프로세스를 구성하는 개개의 함수 e가 의미 있는 선행조건 $pre(e)$ 및 수행 후 조건 $post(e)$를 가지고 있다는 것을 가정한다. 네트워크 응용에서 수행되는 함수는 대부분 계산보다는 메시지 전송을 위한 것이며, 복잡한 기능을 수행하는 함수는 여러 기본적인 함수로 대치될 수 있다는 점을 고려하면 이러한 제한은 지나치지 않다고 볼 수 있다.

3.4 도달성 분석(Reachability Analysis)

액티브 응용 $P=(\langle P_i\rangle,\langle C_{ij}\rangle)$ 에 대하여, 프로세스 P_i의 상태는 tuple $(u_i,\langle v_i\rangle)$이며, 여기서 $u_i \in S_{pi}$는 P_i의 현재 상태, $\langle v_i\rangle$는 V_{pi}에 속하는 변수의 현재 값을 나타내는 tuple이다. 응용 P의 전역 상태는 tuple $(\langle x_i\rangle,\langle c_{ij}\rangle)$ 로서, x_i는 프로세스 P_i의 현재 상태, c_{ij}는 채널 C_{ij}의 현재 내용이다. 응용 P의 초기 전역 상태 g_0는 tuple $(\langle s^0_{pi},\langle v^0_{pi}\rangle\rangle,\langle\varepsilon\rangle)$ 로 나타낼 수 있는데, $s^0_{pi}\in S_{pi}$는 프로세스 P_i의 초기 상태, $\langle v^0_i\rangle$는 V_{pi}에 속하는 변수의 초기 값을 나타내는 tuple이며, $\langle\varepsilon\rangle$은 초기에 채널들이 공백인 상태에 있음을 의미한다. 임의의 전역 상태 g에 대해 A_g는 g에서 수행 가능한 모든 사건(함수)의 집합, 즉 각 프로세스에서 현재 상태가 e의 시작 부분에 해당하고, $pre(e)$가 참인 모든 사건(함수) e 의 집합이다.

액티브 응용 P의 도달성 그래프(reachability graph) R_P는 초기 전역 상태 g_0에서 시작, 현재의 상태에서 새로운 전역 상태(그래프의 노드에 해당)로 천이를 일으키는 가능한 모든 사건(함수)의 발생을 간선으로 표현하는 방향 그래프(directed graph)이다. P가 주어졌을 때 R_P를 구하는 방법은 아래와 같다.

1. 초기 전역 상태 g_0에 해당하는 노드를 생성하고, 이를 시작 노드로 설정;
2. 모든 생성된 노드 g에 대해 아래 단계를 반복;
   2.1 A_g에 속하는 각 사건 e에 대해 아래 단계를 반복;
      2.1.1 e를 수행한 결과의 상태 g'에 대해 만약 g'에 해당하는 노드가 존재하지 않으면 노드 g'를 생성;
      2.1.2 간선 $\langle g, g'\rangle$를 추가;
      2.1.3 $A_{g'} \leftarrow A_g \cup \{g'$에서 새로이 수행 가능하게 된 사건들$\} - \{A_g$에서 더 이상 수행 가능하지 않은 사건들$\}$

그림 2 P의 도달성 그래프 R_P 생성 알고리즘

프로토콜과 같은 분산 알고리즘의 정확성을 검증하는 대표적인 방법중의 하나가 도달성 분석에 의한 방법이다. 도달성 분석은 프로토콜의 동작을 기술하는 도달성 그래프의 탐색을 통해 해당 프로토콜이 특정 안전성 및 활동성 성질을 만족하는지의 여부를 결정하는 방법으로 알고리즘으로 나타낼 수 있어 컴퓨터 프로그램을 통해 자동화할 수 있다는 장점이 있다.

안정성 성질은 각 전역 상태에서 참 또는 거짓으로 판정할 수 있는 논리식(logical formula)을 이용하여 표현한다. 활동성 성질중 대표적인 것은 $\hookrightarrow$("*leadsto*"라고

읽음)로서 $\alpha \hookrightarrow \beta$의 의미는 도달성 그래프의 초기 전역 상태로부터 도달 가능한 임의의 전역 상태에서 논리식 α가 만족된다면 그 전역 상태에서 시작하는 어떠한 방향 경로에 대해서도 논리식 β가 만족되는 상태(노드)가 경로상에 반드시 존재한다는 것이다. 안전성 및 활동성 성질을 이러한 도달성 그래프의 탐색에서 검증하는 방법은 다음과 같다.

안전성 성질 p의 증명
1. 초기 전역 상태 g_0 노드 및 이로부터 도달 가능한 모든 노드에 대해
   p가 성립하는 경우 YES(성질 p 성립);
   한 노드라도 그렇지 않은 경우 NO(성질 p 성립하지 않음);

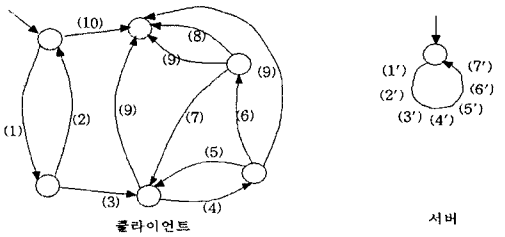
활동성 성질 q: $\alpha \hookrightarrow \beta$ 의 증명
1. 초기 전역 상태 g_0 노드 및 이로부터 도달 가능한 노드 중 α 가 성립하는 각 노드 g' 에 대해 다음을 반복;
   1.1 g'에서 시작하는 각 경로 p: $g' \rightarrow \cdots \rightarrow g^{(k)} \rightarrow \cdots$ (유한 또는 사이클을 포함하는 무한 경로)에 대하여 다음을 반복;
      1.1.1 β를 만족시키는 $g^{(k)}$가 존재한다면 1.1로 돌아감;
      1.1.2 경로 p 상의 어떤 노드도 β를 만족시키지 않는다면 NOK; 빠져 나옴;
   1.2 위의 1.1 반복을 NOK로 빠져 나온 경우 NO(성질 q 성립하지 않음);
   그렇지 않고 반복을 마친 경우 YES(성질 q 성립);

그림 3 도달성 분석을 통한 안정성 및 활동성 성질 증명 방법

단, 여기서 해당 도달성 그래프의 전역 상태의 수는 유한 개라고 가정한다. 만약, 도달성 그래프가 무한개의 전역 상태를 가지고 있다면 - 예로, 채널의 크기가 무한히 커지는 경우가 발생한다면 - 도달성 그래프 탐색에 기반을 둔 위 방법은 결정가능(deterministic)하지 못하다. 그러나 실제 거의 모든 응용은 최대 유한 개수의 메시지를 포함하는 채널을 가정하고, 프로세스 상태에 포함되는 변수도 유한의 영역을 가지게 된다. 본 논문에서도 기술하는 모든 응용은 유한의 노드와 상태 천이를 가지는 도달성 그래프로 그 동작을 표현할 수 있다고 가정한다.

4. 액티브 네트워크 경매의 예

네트워크를 통해 다수의 클라이언트(client)들이 경매에 참여하여 어떤 물건을 구매하는 인터넷 경매의 예를 생각한다. 경매 서버는 세션을 시작하며 각 클라이언트로부터 경매 참가 요구를 받고(rcv_join_rqst) 이를 허락하거나(send_join_ack) 또는 거부(send_join_nack)한다. 경매 참여 허락 응답을 받은(rcv_join_ack) 각 클라이언트는 네트워크를 통해 자신의 입찰가를 경매 서버에 전달하고(send_my_bid) 응답으로 현재 최고의 입찰가 정보를 얻으며(rcv_larger_bid/rcv_accepted), 필요시(rcv_revoked) 다시 자신의 새로운 입찰가를 경매 서버에 제시하는 과정을 반복한다. 최종 낙찰을 받은 클라이언트는 그 확인을 받으며(rcv_confirmed), 타 클라이언트들은 세션을 마감한다는 응답을 받는다(rcv_closed). 경매 서버는 클라이언트들로부터 입찰가를 받아(rcv_my_bid) 현재의 최고가액 정보를 응답으로 제공하며(send_cur_info), 일정 기간 후 또는 입찰가가 정해진 어떤 값에 도달하였을 때 도착한 입찰가중 최고가에 해당 물건을 낙찰하고, 이 사실을 참여한 클라이언트들에 알리면서(send_confirmed/send_closed) 세션을 종료한다. 이러한 경매 과정을 액티브 네트워크상에서 수행한다면 클라이언트와 서버 사이의 액티브 라우터는 도착한 클라이언트 패킷의 입찰가가 만약 서버로부터 얻은(process_server_msg) 현재의 최고 입찰가보다 크지 않은 경우는 패킷을 서버로 전달하지 않고 현재 입찰가 정보를 담아 클라이언트로 반송할 수 있다(process_client_msg). 즉 클라이언트의 입장에서는 이 경우 자신의 입찰가가 서버에까지 전달되었으나 최고가가 되지 못한 경우와 차이가 없이 중간 라우터로부터 반송 패킷을 수신한다. 결과적으로 서버에 몰리는 클라이언트 패킷들은 서버로 가는 경로상에서 반송될 가능성이 있으므로 네트워크의 대역폭을 적게 사용하게 된다. 액티브 네트워크 경매의 핵심 부분은 네트워크의 중간 액티브 라우터에서 수행되는 간단한 비교 프로그램이고, 이 프로그램은 서버로부터 패킷의 페이로드(payload) 부분에 실



(1) send_join_rqst (2) rcv_join_nack (3) rcv_join_ack (4) send_my_bid (5) rcv_larger_bid (6) rcv_accepted (7) rcv_revoked (8) rcv_confirmed (9) rcv_closed (10) giveup
(1') rcv_join_rqst (2') send_join_ack (3') send_join_nack (4') rcv_my_bid (5') send_cur_info (6') send_confirmed (7') send_closed

그림 4 클라이언트 및 서버 프로세스

려서 전달된다(download_codes). 본 예에서는 증명의 편의상 모든 패킷이 유실되지도 분할(fragmentation)되지도 않는다고 가정한다. 그림 4는 각 클라이언트 및 서버에 해당하는 노드의 프로세스를 확장유한상태기계로 나타내고 있다.

여러 클라이언트들과 서버 사이의 경로상에 존재하는 액티브 라우터들은 서버로부터 액티브 패킷의 형태로 수행 프로그램을 받아 이를 저장하고 자신을 통과하는 패킷에 대해 선별적인 처리를 수행한다. 액티브 라우터의 동작을 나타내는 프로세스는 다음 그림과 같다.

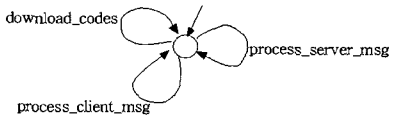


그림 5 액티브 라우터 프로세스

각 클라이언트 프로세스에서 수행되는 함수들, 액티브 서버에서 수행되는 함수들 및 액티브 라우터의 수행 함수들이 각각 의사 코드(pseudo code) 형태로 부록에 제공된다.

이와 같이 정의된 액티브 경매 프로토콜은 액티브 서버가 다수의 액티브 클라이언트들로부터 입찰가를 받아 유효 기간내에 가장 높은 입찰가(동일한 입찰가를 제시한 클라이언트가 둘 이상 있을 때는 먼저 도착한 메시지)를 제시한 클라이언트에게 물건을 낙찰하게 된다. 액티브 라우터는 클라이언트와 서버의 중간 지점에서 서버로 집중되는 트래픽 중 불필요한 것을 차단하는 역할을 함으로 네트워크를 효율적으로 사용할 수 있도록 한다. 이런 시나리오는 다수의 물건 및 동종 물건이 여러 개 있는 경우로 쉽게 확장할 수 있다. 본 논문의 경매 예에서 다음과 같은 대표적인 몇 가지 안전성 및 활동성 성질을 도달성 분석 방법으로 증명할 수 있다. (성질 1)은 한 클라이언트가 입찰가를 제시하여 경매에 참여하면, 최종적으로 자신의 입찰가로 낙찰되든지 아니면 자신의 입찰가가 거부되든지 둘 중의 한 경우가 반드시 일어남을 말하고, (성질2)는 서버가 현재 저장하고 있는 최고가 제시자로부터 메시지를 받은 사실을 있다는 의미이고, (성질3)은 액티브 라우터들이 저장하고 있는 현재 최고가 값들이 서버가 알고 있는 최고가보다 더 클 수 없고, 따라서 액티브 라우터가 클라이언트의 어떤 입찰가를 서버가 알아야하는데도 불구하고 반송하는 경우는 없다는 것을 의미하며, (성질 4)는 서버가 적어도 한 클라이언트가 경매에 참여하면 결국 한 클라이언트를 최종적으로 선정하게 된다는 것을 나타내고, (성질5)와 (성질6)은 각각 최종 낙찰된 가격을 제시한 클라이언트가 확정 메시지를 받게 되고, 확정된 클라이언트는 둘 이상이 될 수 없다는 사실을 보여주고 있다.

(성질1): $any\ client\ c::[my\_state = ready \hookrightarrow (my\_state = confirmed \vee my\_state = failed)]$

(성질2): $(cur\_winner = c \wedge c \neq \text{null}) \Rightarrow \exists x::(ACPT[x].address = c \wedge ACPT[x].bid = cur\_highest\_bid)$

(성질3): $cur\_highest\_bid \geq \max_{r}\ (r's\ saved\_highest\_bid)$

(성질4): $(session = on) \hookrightarrow (final\_winner = c \wedge c \neq \text{null})$

(성질5): $(final\_winner = c \wedge c \neq \text{null}) \hookrightarrow (c's\ my\_state = confirmed)$

(성질6): $|\ no\ of\ clients\ whose\ my\_state = confirmed\ | \leq 1$

5. 결론 및 향후 연구 과제

액티브 네트워크는 프로그램 코드나 제어 정보를 네트워크 내부에 침투시켜 수행할 수 있게 함으로 새로운 네트워크 서비스 실현을 위한 프로그래머블 플랫폼을 제공한다. 네트워크 사용자는 액티브 네트워크 기술을 이용하여 자신의 목적에 맞는 네트워크 서비스를 제공받을 수 있게 된다. 액티브 응용은 네트워크의 사용자가 액티브 네트워크 노드가 제공하는 API를 이용하여 자신이 원하는 기능을 합성하는 방법으로 설계된다. 액티브 네트워크상의 서비스는 호스트에서의 프로그램뿐 아니라 내부 라우터에서 수행될 수 있는 다양한 프로그램의 동작을 통해 이루어지며, 이러한 서비스가 실제 설계자가 의도한 그대로 정확히 이루어지는지 검증하는 것이 필요하다.

본 논문에서는 기존의 액티브 네트워크 실현 방법들에 적용될 수 있는 포괄적인 네트워크 프로토콜 모델과 이를 기반으로 한 액티브 응용의 검증 방법을 제안하였다. 제안한 검증 방식은 활용 범위를 넓히고자 특정한 환경에 의존하지 않는 최소의 가정을 기반으로 전개하였다. 검증 방법은 액티브 응용이 원하는 동작을 수행하는지를 프로토콜의 변수들로 표현되는 안전성 및 활동성의 성질의 측면에서 조사한다. 이 방법은 도달성 분석에 기반을 두었으며 알고리즘으로 표현할 수 있어 컴퓨터 프로그램에 의한 자동화가 가능한 장점이 있다. 본 논문에서 제시한 액티브 프로토콜 모델 및 검증 방법은

액티브 경매의 예를 이용하여 기술함으로 그 활용 가능성을 보였다.

액티브 프로토콜의 검증 분야는 액티브 네트워크가 실제로 차세대 인터넷 실현을 위한 플랫폼 기술의 하나로서 발전하기 위해 불가결한 연구 과제라고 할 수 있으며, 액티브 프로토콜의 합성 기법과 더불어 더욱 일반적인 형태의 검증 기법에 대한 연구가 필요하리라 생각한다. 특히 보안이나 정책 제어가 프로그램 코드에 제한을 주는 경우나 액티브 코드가 라우팅 경로에 영향을 주는 경우에 대한 일반화를 고려하여 현재의 모델링 및 검증 기법을 확장시키는 부분으로 추후 연구가 필요하다고 생각한다.

참 고 문 헌

[1] D. Tennenhouse and D. Wetherall, "Towards an Active Network Architecture," Computer Communication Review 26(2), April 1996.

[2] DARPA AN Architecture Working Group, *Architectural Framework for Active Networks*, http://www.cc.gatech.edu/projects/canes/papers/arch-1-0.pdf, July 1999.

[3] D. Alexander, *ALIEN: A Generalized Computing Model of Active Networks*, PhD thesis, University of Pennsylvania, 1998.

[4] D. Alexander et al., "A Secure Active Network Environment Architecture: Realization in Switch Ware," IEEE Network, 12(3), 1998.

[5] M. Hicks et al., "PLAN: A Packet Language for Active Networks," Proc. of the 1998 ACM SIGPLAN Int'l Conf. on Functional Programming (ICFP'98), September 1998.

[6] M. Hicks and A. Keromytis, "A Secure PLAN," Proc. of the First Int'l Working Conf. on Active Networks, June 1999.

[7] J. Moore, "Safe and Efficient Active Packets," Technical Report MS-CIS-99-24, Dept. of Computer and Information Science, University of Pennsylvania, October 1999.

[8] E. Nygren, S. Garland, and M. Kaashoek, "PAN: A High-performance Active Network Node Supporting Multiple Mobile Code Systems," Proc. of the 1999 IEEE Second Conference on Open Architectures and Network Programming (OPENARCH'99), March 1999.

[9] B. Schwartz et al., "Smart Packets for Active Networks," Proc. of the 1999 IEEE Second Conference on Open Architectures and Network Programming(OPENARCH'99), March 1999.

[10] D. Wetherall, J. Guttag, and D. Tennenhouse, "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols," Proc. of the 1998 IEEE Conference on Open Architectures and Network Programming(OPENARCH'99), April 1999.

[11] I. Wakeman et al., "Designing a Programming Language for Active Networks," http://www.cogs.susx.ac.uk/projects/safetynet, January 1999.

[12] J. Hartman et al., "Liquid Software: A New Paradigm for Networked Systems," Technical Report TR96-11, Department of Computer Science, University of Arizona, 1996.

[13] S. Bhattacharjee, K. Calvert, and E. Zegura, "An Architecture for Active Networking," Proc. of IEEE INFOCOM'97, April 1997.

[14] Y. Yemini and S. da Silva, "Towards Programmable Networks," Proc. IFIP/IEEE Int'l Workshop on Distributed Systems, Operations, and Management, 1996.

[15] D. Decasper and B. Plattner, "DAN: Distributed Code Caching for Active Networks," Proc. of IEEE INFOCOM'98, March/April 1998.

[16] J. Moore, M. Hicks, and S. Nettles, "Practical Programmable Packets," Proc. of IEEE INFOCOM 2001, April 2001.

[17] D. Tennenhouse et al., "A Survey of Active Network Research," IEEE Communications Magazine, January 1997.

[18] DARPA AN Node OS Working Group, NodeOS Interface Specification, http://www.cs.princeton.edu/nsg/papers/nodeos99.ps, January 2000.

[19] J. Gosling and H. McGilton, The Java Language Environment: A White Paper, Sun Microsystems, 1995.

[20] S. Bhattacharjee, K. Calvert, and E. Zegura, "Reasoning About Active Network Protocols," Proc. of IEEE Int'l Conf. on Network Protocols(ICNP'98), 1998.

[21] C. Gunter, S. Nettles, and J. Smith, "The SwitchWare Active Network Architecture," IEEE Network, 12(3), 1998.

[22] H. Lin, "A Methodology for Constructing Communication Protocols with Multiple Concurrent Functions," Distributed Computing, 3, 1988.

[23] G. Singh and Z. Mao, "Structured Design of Communication Protocols," Proc. of IEEE Int'l Conf. on Network Protocols(ICNP'94), 1994.

[24] L. Cacciari and O. Rafiq, "A Temporal Reachability Analysis," Proc. XV IFIP Symp. Protocol Specification, Testing and Verification, 1995.

[25] C.-M. Huang and S.-W. Lee, "Timed Protocol Verification for Estelle-Specified Protocols," ACM SIGCOMM Comput. Commun. Review, 25(3), 1993.

부 록

송신자 a에서 수신자 b로의 패킷은 각 경유 노드의 라우팅 테이블 정보에 따라 $a \rightarrow \text{out}(a) \rightarrow \cdots \rightarrow \text{in}(b) \rightarrow b$의 경로를 따르는데, 여기서 $\text{out}(a)$와 $\text{in}(b)$는 각각 경로상의 a와 b의 이웃 노드이다.

A. 클라이언트 수행 함수들

```
/*(1) pre: my_state=ready_to_join; initially my_state = ready_to_join */
 function send_join_rqst(void) {
    send(<c, s, [join_rqst]);
    my_state ← join_rqst_sent;
 }
/* post: (c^new_{c,out(c)} = c^old_{c,out(c)} · <c, s, [join_rqst]>) ∧ (my_state=join_rqst_sent) */
/*(2) pre: (head(c_{in(c),c})=<s, c, [nack]>) ∧ (my_state=join_rqst_sent) */
 function rcv_join_nack(packet <s,c,[nack]>) {
    extract the packet <s,c,[nack]> from the head of a channel C_{in(c),c};
    my_state ← ready_to_join;
 }
/*post: (c^old_{in(c),c}=<s, c, [nack]> · c^new_{in(c),c}) ∧ (my_state=ready_to_join) */
/*(3)pre:
   (head(c_{in(c),c})=<s, c, [ack;cur_highest_bid]>) ∧ (my_state=join_rqst_sent) */
 function rcv_join_ack(packet <s,c,[ack;cur_highest_bid]>) {
    extract the packet <s,c,[ack;cur_highest_bid]> from the head
          of a channel C_{in(c),c};
    my_state ← ready;
 }
/*post: (c^old_{in(c),c}=<s, c, [ack;cur_highest_bid]> · c^new_{in(c),c}) ∧ (my_state=ready) */
/* (4) pre: my_state=ready */
 function send_my_bid(void) {
       /* initially, my_new_bid = 0 */
    if(my_new_bid = 0) then { /* this will be my first try */
          my_new_bid ← initial_bid; /* initial_bid >0 */
    }
    send(<c, s, [my_new_bid]>);
    my_state ← bid_sent;
 }
/* post: (c^new_{c,out(c)} = c^old_{c,out(c)} · <c, s, [my_new_bid]>) ∧ (my_state=bid_sent) */
/*(5)pre:
   (head(c_{in(c),c})=<s, c, [no;cur_highest_bid]>) ∧ (my_state=bid_sent) */
 function rcv_larger_bid(packet <s,c,[no;cur_highest_bid]>) {
     extract the packet <s,c,[no;cur_highest_bid]> from the head
          of a channel C_{in(c),c};
     my_state ← ready;
     my_new_bid ← cur_highest_bid + α; /* α>0 */
  }
/*post: (c^old_{in(c),c}=<s, c, [no;cur_highest_bid]> · c^new_{in(c),c}) ∧ (my_new_bid^new >
       my_new_bid^old) ∧ (my_state=ready) */
/*(6)pre:
      (head(c_{in(c),c})=<s, c, [yes;cur_highest_bid]>) ∧ (my_state=bid_sent) */
 function rcv_accepted(packet <s,c,[yes;cur_highest_bid]>) {
    extract the packet <s,c,[yes;cur_highest_bid]> from the head
          of a channel C_{in(c),c};
    my_state ← my_bid_accepted;
 }
/*post:
   (c^old_{in(c),c}=<s, c, [yes;cur_highest_bid]> · c^new_{in(c),c}) ∧ (my_state=my_bid_accepted)      */
/*(7)pre:
   (head(c_{in(c),c})=<s, c, [no;cur_highest_bid]>) ∧ (my_state=my_bid_accepted) */
 function rcv_revoked(packet <s,c,[no;cur_highest_bid]>) {
     extract the packet <s,c,[no;cur_highest_bid]> from the head
          of a channel C_{in(c),c};
     my_state ← ready; /* ready to send my new bid */
     my_new_bid ← cur_highest_bid + α; /* α>0 */
  }
/*post: (c^old_{in(c),c}=<s, c, [no;cur_highest_bid]> · c^new_{in(c),c}) ∧ (my_new_bid^new >
       my_new_bid^old) ∧ (my_state=ready) */
/*(8)pre:
(head(c_{in(c),c})=<s, c, [confirmed;cur_highest_bid]>) ∧ (my_state=my_bid_accepted)      */
```

```
  function rcv_confirmed(packet <s,c, [confirmed;cur_highest_bid]>) {
      extract the packet <s,c,[confirmed;cur_highest_bid]> from the head
          of a channel C in(c),c;
      my_state ← confirmed;  /* my bid was confirmed [end of session] */
   }
 /*post:   (c^old_in(c),c = <s, c, [confirmed; cur_highest_bid]> · c^new_in(c),c) ∧ (my_state = confirmed)   */
 /*(9)pre:  head(c in(c),c) = <s, c, [closed]> */
  function rcv_closed(packet <s,c,[closed]>) {
      extract the packet <s,c,[closed]> from the head of a
          channel  C in(c),c;
      my_state ← failed; /* someone else's bid was accepted */
   }
 /* post: (c^old_in(c),c = <s, c, [closed]> · c^new_in(c),c) ∧ (my_state = failed) */
 /*(10)pre: my_state = ready_to_join and some threshold time has passed */
  function giveup(void) {
      my_state ← giveup;
}
 /*post:  my_state = giveup  */
```

B. 서버 수행 함수들

```
 /*(1')pre:  head(c in(s),s) = <c, s, [join_rqst]> */
  function rcv_join_rqst(packet <c,s,[join_rqst]>) {
      extract the packet (<c, s, [join_rqst]>) from the head
          of a channel C in(s),s;
      if (the server decides to accept this join request) then {
          no_acks_to_be_sent ← no_acks_to_be_sent + 1;
              /* initially, no_acks_to_be_sent = 0 */
          ACPT[no_acks_to_be_sent].address ← c;
          ACPT[no_acks_to_be_sent].status ← accepted;
          ACPT[no_acks_to_be_sent].bid ← 0; /* no bid applied yet */
              /* initially, ACPT array was set to null */
      }
      else if(session=CLOSING or the server decides to reject
              this request for some reason){
          no_nacks_to_be_sent ← no_nacks_to_be_sent + 1;
              /* initially, no_nacks_to_be_sent = 0 */
          REJ[no_nacks_to_be_sent].address ← c;
          REJ[no_nacks_to_be_sent].status ← rejected;
              /* initially, REJ array was set to null */
      }
  }
 /*post:  (c^old_in(s),s = <c, s, [join_rqst]> · c^new_in(s),s) ∧ [(no_acks_to_be_sent^new =
          no_acks_to_be_sent^old + 1) ⊕ (no_nacks_to_be_sent^new = no_nacks
          _to_be_sent^old + 1)]   */
 /* (2')pre:  no_acks_to_be_sent > 0 */
   function send_join_ack(void) {
       find a client with index x such that ACPT[x].status = accepted;
       send(<s,ACPT[x].address,[ack;cur_highest_bid;
                   {process_client_msg;process_server_msg}]>);
         /* piggyback the function codes for active routers inbetween */
       ACPT[x].status ← ack_sent;
       no_acks_to_be_sent ← no_acks_to_be_sent - 1;
   }
 /* post:
        (no_acks_to_be_sent^new = no_acks_to_be_sent^old - 1) ∧
        (c^new_s,out(s) = c^old_s,out(s) · <s, c, [ack; cur_highest_bid; {process_client_msg;
        process_server_msg}]>)
  */
  /* (3')pre:  no_nacks_to_be_sent > 0 */
   function send_join_nack(void) {
       find a client with index x such that REJ[x].status = rejected;
       send(<s,REJ[x].address,[nack]>);
       REJ[x].status ← nack_sent;
       no_nacks_to_be_sent ← no_nacks_to_be_sent - 1;
   }
 /* post:
        (no_nacks_to_be_sent^new = no_nacks_to_be_sent^old - 1) ∧
        (c^new_s,out(s) = c^old_s,out(s) · <s, c, [nack]>)
  */
 /*(4')pre: head(c in(s),s) = <c, s, [my_new_bid]> */
```

```
function rcv_my_bid(packet <c,s,[my_new_bid]>) {
    extract the packet (<c, s, [my_new_bid]>) from the head
        of a channel C in(s),s;
    find a client with index x such that ACPT[x].address = c;
    ACPT[x].bid ← my_new_bid;
    if(session=off) then {session ← on;}
        /* at least one client joined; initially, session = off */
    else if(session=closing) then {
        ACPT[x].status ← late_arrived;
        exit; /* exit this function */
      /* just remove any incoming message when closing the session */
    }
    if (my_new_bid > cur_highest_bid) then {
        /* initially, cur_highest_bid = 0 */
        if (cur_winner ≠ null) then {
            /* initially, cur_winner = null */
            find a client with index y such
                that ACPT[y].address = cur_winner;
            ACPT[y].status ← bid_rejected;
            no_clients_to_be_replied ← no_clients_to_be_replied + 1;
              /* initially, no_clients_to_be_replied = 0 */
              /* need to send no(revoke) message to the previous
                  winner because a higher bid was just received */
        }
        cur_winner ← c;
        cur_highest_bid ← my_new_bid;
        ACPT[x].status ← bid_accepted;
    }
    else ACPT[x].status ← bid_rejected;
    no_clients_to_be_replied ← no_clients_to_be_replied + 1;
        /* send either yes or no message to the client c */
}
/* post:
(c^old_in(s),s = <c, s, [my_new_bid]> · c^new_in(s),s) ∧ (session ≠ closing ⇒ cur_highest_bid ≥
my_new_bid) ∧ (∃x:ACPT[x].address = cur_winner ∧ ∀y ≠ x:ACPT[y].address
≠ cur_winner) ∧ {(ACPT[x].address = cur_winner) ⇒ (ACPT[x].status = bid_accepted
∧ ACPT[x].bid = cur_highest_bid)} ∧ {(ACPT[y].address = c ∧ c ≠ cur_winner) ⇒
((ACPT[y].status = bid_rejected ∨ ACPT[y].status = late_arrived) ∧ ACPT[y].bid =
my_new_bid)}                                                              */
/*(5')pre: (no_clients_to_be_replied > 0) ∧ (session = on) */
function send_cur_info(void) {
    find a client with index x such that ACPT[x].status = bid_accepted
        or ACPT[x].status = bid_rejected;
    if (ACPT[x].status = bid_accepted) {
        send(<s,ACPT[x].address,[yes;cur_highest_bid]>);
        ACPT[x].status ← pinfo_replied; /* positive reply */
    }
    else {
        send(<s,ACPT[x].address,[no;cur_highest_bid]>);
        ACPT[x].status ← ninfo_replied; /* negative reply */
    }
    no_clients_to_be_replied ← no_clients_to_be_replied - 1;
}
/* post:
    (no_clients_to_be_replied^new = no_clients_to_be_replied^old - 1) ∧
    (∃x:: c^new_s,out(s) = c^old_s,out(s) · <s, ACPT[x].address, [yes; cur_highest_bid]>
    ⊕ ∃x:: c^new_s,out(s) = c^old_s,out(s) · <s, ACPT[x].address, [no; cur_highest_bid]>)
    ∧ (c^new_s,out(s) = c^old_s,out(s) · <s, ACPT[x].address, [yes; cur_highest_bid]> ⇒
    ACPT[x].status = pinfo_replied) ∧ (c^new_s,out(s) = c^old_s,out(s) · <s, ACPT[x].
    address, [no; cur_highest_bid]> ⇒ ACPT[x].status = ninfo_replied)
*/
/*(6')pre: (a threshold price/time is reached) ∧ (session = on) */
function send_confirmed(void) {
    final_winner ← cur_winner; /* initially, final_winner = null */
    send(<s,final_winner,[confirmed;cur_highest_bid]>);
    find the client with index x such that ACPT[x].address=
                                                    final_winner;
    ACPT[x].status ← confirmed_sent;
    session ← closing;
}
/* post:
   (c^new_s,out(s) = c^old_s,out(s) · <s, final_winner, [confirmed; cur_highest_bid]>)
   ∧ (session = closing) ∧ (ACPT[x].address = final_winner ⇒ ACPT[x].
   status = confirmed_sent)                                          */
/*(7')pre: (session = closing) ∧ (∃x:: ACPT[x].status ≠ closed_sent) */
```

```
function send_closed(void) {
    find a client with index x such that ACPT[x].status ≠ closed_sent
        and ACPT[x].status ≠ confirmed_sent;
    /* notify all participating clients but the final winner
       of the end of the session */
    send(<s,ACPT[x].address,[closed]>);
    ACPT[x].status ← closed_sent;
}
/* post:
  (∃ x:: c^new_{s,out(s)} = c^old_{s,out(s)} · ⟨s, ACPT[x].address, [closed]⟩) ∧ (c^new_{s,out(s)} = c^old_{s,out(s)}
  · ⟨s, ACPT[x].address, [closed]⟩ ⇒ ACPT[x].status = closed_sent);
*/
```

C. 액티브 라우터 수행 함수들

```
/* pre:
   (head(c_{in(r),r}) = ⟨s, c, [ack; cur_highest_bid; {process_server_msg; process_
   client_msg}]⟩)
  and the function codes for "process_client_msg" and
  "process_server_msg" are not resident in this active router */
  function download_codes(packet <s,c,[ack;cur_highest_bid;
                                  {process_client_msg;process_server_msg}]>)
    extract the packet (<s,c,[ack;cur_highest_bid;
                              {process_client_msg;process_server_msg}]>)
        from the head of a channel C_{in(r),r};
    install the function codes for "process_client_msg" and
        "process_server_msg";
  }
/*post:
 c^old_{in(r),r} = ⟨s, c, [ack; cur_highest_bid; {process_client_msg; process_server_msg}]⟩
        · c^new_{in(r),r}
 and the function codes for "process_client_msg" and "process_server
       _msg" exist in this active router" */
/*pre: (head(c_{in(r),r}) = ⟨c, s, [payload]⟩) ∧ this function code exists , where the
  payload matches to any message sent by the clients */
  function process_client_msg(packet <c,s,[payload]>) {
    extract the packet (<c, s, [payload]>) from the head
        of a channel C_{in(r),r};
    case payload of
    payload = [join_rqst]:
        if (router_state = closing)
            /* initially, router_state = on_service */
            /* the server would not accept new join requests any more */
            /* send back join nack to c */
            send(<s,c,[nack]>);
        else forward the packet to the server;
    payload = [my_new_bid]:
        if (my_new_bid ≤ saved_highest_bid) then
            /* initially, saved_highest_bid = 0 */
            /* send no(revoke) to the client */
           send(<s, c, [no;saved_highest_bid]);
        else if (router_state = closing) then ;/* ignore the packet */
        else forward the packet to the server;
    end case;
  }
/*post:
    (c^old_{in(r),r} = ⟨c, s, [payload]⟩ · c^new_{in(r),r}) ∧ {(payload = [join_rqst] ∧ router_state
    ≠ closing) ⇒ (c^new_{r,out(r)} = c^old_{r,out(r)} · ⟨c, s, [join_rqst]⟩)} ∧ {(payload = [join_rqst]
    ∧ router_state = closing) ⇒ (c^new_{r,out(r)} = c^old_{r,out(r)} · ⟨s, c, [nack]⟩)} ∧ {(payload =
    [my_new_bid] ∧ router_state ≠ closing ∧ my_new_bid > saved_highest_bid)
    ⇒ (c^new_{r,out(r)} = c^old_{r,out(r)} · ⟨c, s, [my_new_bid]⟩)} ∧ {(payload = [my_new_bid]
    ∧ router_state ≠ closing ∧ my_new_bid ≤ saved_highest_bid) ⇒ (c^new_{r,out(r)} =
    c^old_{r,out(r)} · ⟨s, c, [no; saved_highest_bid]⟩)}
 */
 /*pre: (head(c_{in(r),r}) = ⟨s, c, [payload]⟩) ∧ this function code exists , where
    the payload matches to any message sent by the server */
  function process_server_msg(packet <s,c,[payload]>) {
    extract the packet (<s,c,[payload]>) from the head
        of a channel C_{in(r),r};
    case payload of
    payload = [yes;cur_highest_bid] or [no;cur_highest_bid] or
              [confirmed;cur_highest_bid]:
        if (cur_highest_bid > saved_highest_bid) then
```

```
                saved_highest_bid ← cur_highest_bid;
                    /* update this router's saved_highest_bid value */
    payload = [ack;cur_highest_bid;{process_client_msg;process_
                    server_msg}]:
          payload ← [ack;cur_highest_bid]; /* truncate the codes */
    payload = [nack]or [closed]: ;
    end case;
    if (payload = [confirmed;cur_highest_bid]) then
          router_state ← closing;
                              /* initialy, router_state = on_service */
    forward the packet to the client;
}
/* post:
    (c^old_in(r),r = <s, c, [payload]> · c^new_in(r),r) ∧ (c^new_r,out(c) = c^old_r,out(c) · <s, c, [payload]>)
    ∧ (saved_highest_bid = cur_highest_bid in [payload], if included)
*/
```

박 준 철

1986년 서울대학교 계산통계학과(학사). 1988년 KAIST 전산학과(석사). 1998년 미국 Maryland 대학교 전산학과(박사). 현재 홍익대학교 컴퓨터공학과 조교수