A Review on Experimental Facilities for Future Internet Testbed

FIF Testbed WG (최종수정일: 2011-12-23)

김종원, 차병래, 한상우, 김남곤 (GIST), 김영화, 남기혁 (ETRI), 이성원 (경희대), 문수복 (KAIST)

요약

미래인터넷 테스트베드는 연구자들이 창의적인 네트워킹 개념들을 실험적으로 실증할 수 있도록 연결용 네트워크와 이에 연동된 장비들, 그리고 이를 지원하는 각종 실험용 소프트웨어 일체를 지칭하는 일종의 가상적인 실험실이 되어야 한다. 즉 기존의 네트워크 연결 위주의 실험에서 벗어나 새로운 서비스를 지향하는 다양한 실험자들의 요구사항들을 수용하는 것이 매우 중요하다. 이를 위해서 활용되는 컴퓨팅/네트워킹 자원들이 여러 네트워크 계층에서 프로그램이 가능해야 하며, 다수의 실험자들이 자신들의 실험들을 동시에 수행하도록 컴퓨팅/네트워킹 자원의 가상화도 필요하다. 또한 확장성을 검증하기 위해서 대규모 환경에서 지속적으로 실험하면서 검증할 수 있어야 한다. 이러한 요구를 감안하여, 본 문서에서는 미래인터넷 테스트베드에서 프로그램화 및 가상화 가능한 실험 장비들에 관한 요구사항과 기술개발 동향에 대해 살펴본다. 먼저 안정적인 실험자원 공급을 위한 요구사항을 도출하고, 슬라이스 기반의 실험자원 관리에 관한 동향과 이슈들을 분석한다. 다음으로, 클라우드 기반의 미래인터넷 컴퓨팅 자원에 대한 기본 개념과 트랜드를 살펴본 뒤, 미래인터넷과 클라우드 컴퓨팅의 연계 방안을 모색한다. 이후 프로그램화와 가상화를 지원하는 네트워킹 실험자원에 관한 개념 소개와 개발 현황에 대해 논의한다.

목차

그림 목차

표 목차

1 Introduction

첨단 디지털 저작기술의 발전과 스마트 네트워크 컴퓨팅 장치들의 보급은 고성능 미디어 응용서비스의 적용 범위를 실생활의 모든 분야로 광범위하게 확대시키고 있다. 미래의 인터넷을 통해 제공될 응용서비스들은 다양한 그리고 복합적인 형태의 디지털 정보, 매체, 콘텐츠들을 창출하고, 이들은 사용자의 요구에 따라 상황별로 가공, 조립되어 안전하고 빠르게 제공될 것이다 [1]. 그러나 현재의 인터넷은 실감 콘텐츠의 보급으로 인한 폭발적인 트래픽 증가와 수준 높은 서비스 품질에 대응할 근본적인 해결책을 제시하지 못하고 있는 실정이다 [2]. 이런 문제를 근본적으로 해결하기 위해, 인터넷을 처음부터 재설계하자는 clean-slate 개념에 따라, 혁신적이고 창의성이 있는 새로운 미래를 위한 네트워크 구조를 수용하기 위한 미래인터넷 관련 연구들이 전 세계적으로 활발히 진행되고 있다. 이러한 연구들은 확장성, 보안성, 이동성, 이질성, 서비스 품질, 자동설정, 상황인지, 관리성, 데이터 중심, 경제성이라는 주요 요구사항들을 세우고 여기에 맞는 새로운 네트워크 구조를 그리면서 한편으로는 이를 실증할 테스트베드 프로토타입의 개발을 병행하고 있다 [3]. 예를 들면 미래인터넷을 위한 새로운 아이디어들을 다양하게 살펴보았던 FIND (Future INternet Design) [4] 프로젝트에 이어서 4 가지 가능성 있는 아키텍처를 연구하는 FIA (Future Internet Architecture) 프로젝트가 한편에 있다. 그리고 이들 프로젝트들에서 제안될 새로운 미래인터넷 아키텍처들을 대규모 사용자들을 대상으로 하여 검증해 보는 대규모의 실증적 테스트베드를 나선형 (spiral) 개발 방법론에 따라 구축하는 GENI (Global Environment for Network Innovations) [5] 활동이 다른 한편의 대표적인 사례이다. 또한 유럽의 FP7(Future Internet in Framework Programme 7)이 후원하는 FIRE (Future Internet Research and Experimentation)[6], 일본의 NWGN (New Generation Network) [7]도 테스트베드 부분에서도 비슷한 구도를 나름대로 그리면서 테스트베드에 관한 차분하고 의미가 있는 진전을 위해 노력하고 있다.

미래인터넷을 연구하기 위한 테스트베드는 연구자들이 창의적인 네트워킹 개념들을 실험적으로 실증할 수 있도록 연결용 네트워크와 이에 연동된 장비들, 그리고 이를 지원하는

각종 실험용 소프트웨어 일체를 지칭하는 일종의 가상적인 실험실이 되어야 한다. 즉 기존의 네트워크 연결 위주의 실험에서 벗어나 새로운 서비스를 지향하는 다양한 실험자들의 요구사항들을 수용하는 것이 매우 중요하다. 이를 위해서 활용되는 컴퓨팅/네트워킹 자원들이 여러 네트워크 계층에서 프로그램이 가능해야 하며, 다수의 실험자들이 자신들의 실험들을 동시에 수행하도록 컴퓨팅/네트워킹 자원의 가상화도 필요하다. 이와 같은 요구사항들을 아래와 같이 세부적으로 설명한다.

- **안정적인 실험자원 공급:** 장기간 지속적으로 미래인터넷 관련 실험을 수행하려면 안정적인 실험자원의 확보가 우선적으로 보장되어야 한다. 이를 위해서는 실험자가 필요한 자원을 정량적으로 기술하게 하여 그에 맞는 자원을 확보하여 안정적으로 약속된 대로 자원이 제공되고 있는지를 감시하면서 문제가 발생하면 이를 해결하여 자원을 지속적으로 공급하는 테스트베드 관리자를 위한 기술 지원이 중요하다.
- **컴퓨팅 중심적인 자원들:** 필요한 컴퓨팅 자원을 쉽게 액세스하고 확장할 수 있도록 클라우드 기반의 컴퓨팅 자원을 도입하여야 한다. 클라우드 컴퓨팅은 실험자들이 확장 가능한 컴퓨팅 자원을 사용한 양에 따라 비용을 지불하고 사용하는 것이다. 따라서 실험자는 사용한 컴퓨팅 자원에 대한 비용만을 지불하며 클라우드 환경에 있는 모든 컴퓨팅 자원을 언제 어디서나 인터넷을 통해 접근할 수 있다. 실험자는 더 이상 실험용 컴퓨팅 인프라를 관리할 필요가 없으며 이러한 관리는 테스트베드 관리자들이 담당한다.

- **네트워킹 중심적인 자원들**: 네트워킹 자원들이 사용자가 원하는 대로 활용하려면 새로운 패러다임에 따라서 네트워킹 장비를 개발하는 혁신이 필요하다. 호환성 검증이라는 큰 장애를 가진 "프로토콜" 중심의 네트워킹 패러다임에서 소프트웨어 정의 네트워킹 (Software Defined Networking: SDN)으로 불리는 프로그래밍이 쉬운 네트워킹 패러다임으로 전환되어야 한다. 즉 논리적으로 중앙집중된 콘트롤러를 통해서 네트워킹 장치들을 오픈플로우(OpenFlow)와 같은 표준적인 인터페이스를 활용하여 직접 프로그래밍하면서 제어하는 네트워킹 중심적인 자원이 필요하다.

따라서 본 백서에서는 미래인터넷 테스트베드에서 사용될 실험적인 설비에 대한 준비 상황을 이해하는 차원에서 전체적인 요구사항, 기술적인 세부 내용, 그리고 향후 전망을 살펴보고자 한다. 먼저 2 절에서 미래인터넷 테스트베드를 위한 안정적인 실험자원 공급이라는 주제를 논의한다. 이어서 3 절에서는 클라우드 컴퓨팅을 개념을 수용하는 미래인터넷 컴퓨팅 실험 자원에 관련된 트랜드와 기술 이슈들을 소개하도록 한다. 그리고 4 절에서 소프트웨어 정의 네트워킹을 위한 네트워킹 실험 자원들의 개발 동향을 설명한다. 설명한 컴퓨팅/네트워킹 실험 자원들을 활용하는 여러 사례들을 5 절에서 살펴본 뒤, 6 절에서 본 문서를 맺는다.

2 미래인터넷 테스트베드를 위한 안정적인 실험자원의 공급이란?

2.1 효율적인 실험 지원을 위해 필요한 기능 요구들

미래인터넷 테스트베드는 컴퓨팅/네트워킹 자원들의 집합을 넘어, 실험자들이 수행할 미래지향적인 네트워킹 실험들을 편리하게 수행할 수 있도록 여러 가지 유용한 기능들을 제공해야 한다. 첫째로 실험자들은 테스트베드를 통해 실험에 필요한 자원들을 추상화된 형태로 선택/이용할 수 있어야 한다. 각 실험자는 개방된 인터페이스들을 통해 주어진 제어 권한에 따라 실험 과정에서 소요되는 컴퓨팅과 네트워킹 자원을 원하는 형태와 양, 그리고 이 자원들의 연결 관계를 명시적으로 기술하여 테스트베드에게 요청해야 한다. 여기에 대응하여 테스트베드는 계산 자원(CPU, GPU), 저장 자원(메모리, HDD), 네트워크 자원(대역폭, 버퍼)의 능력을 가상화하여 단위 자원으로 준비하게 된다. 둘째로, 실험자는 자원들의 능력을 정량화된 수치로 이해할 수 있어야 한다. 테스트베드들은 이종 노드들로 구성되는 경우가 일반적이며, 이로 인해 실험자는 이종 노드들이 제공하는 계산, 저장, 네트워킹 능력을 일관된 기준으로 가늠하기 어렵다. 따라서 특정 기능을 공급하기 위한 소요 자원의 형태와 양을 정량화하여, 사용자가 실험에 필요한 자원을 주의 깊게 선택할 수 있도록 도와주는 것이 필요하다. 셋째로 공급하는 자원의 성능 변화를 주어진 (또는 상황에 따라 변화된) 시간 간격 마다 측정하여 실험과정에서 계산, 저장, 네트워크 자원의 양이 부족하지 않은지 점검해야 한다. 실험자가 요청한 경우, 테스트베드는 실험자에게 사용 중인 자원의 상태와 변동 추이를 제때에 알려, 실험자가 이에 선제적으로 대처할 수 있도록 도와주어야 한다. 마지막으로 테스트베드 위에서 콘텐츠와 서비스들을 쉽게 설정하고 제어하기 위한 상위 수준의 소프트웨어 중심의 실험제어 기능들이 포함되어야 한다.

미래인터넷 테스트베드에서 네트워킹은 실험에 직접적인 영향을 미치는 중요한 부분이지만, 미래인터넷의 유용성은 콘텐츠와 이를 활용하는 서비스들에 의해서 검증된다는 점에 유의해야 한다. 이를 위해 테스트베드는 사용자 요구에 대응하여 콘텐츠의 지속적인 공급과 소비를 위해, 네트워킹/컴퓨팅 자원들을 이용하는 서비스들의 연결 관계와 제어방법을 쉽게 기술하고, 그에 맞게 효율적인 방법으로 적시에 서비스들을 합성해야 하며, 서비스 합성이 적절한 품질로 계속 유지되는지 정량적으로 측정할 수 있는 도구를 제공해야 한다.

이러한 요구사항들과 비교한다면, 현재 미래인터넷 테스트베드들은 자원(resource)의 가상화와 프로그램화에 주력하면서 실험자가 요청한 자원을 확보해줄 수 있는 편리한 도구를 제공하는데 초점을 두고 있다. 특히 실험에 필요한 자원을 요구하는 부분에서 많은 진도를 보이고 있는데, GENI 에서는 자원 요구사항을 정형화된 틀에 맞게 기술하는 RSpec 을 현 단계에서 논의하고 있으며, ORCA 와 SURFnet 에서는 테스트베드의 다양한 측면을 지식으로 표현하기 위해 RDF (resource description framework)과 OWL (web ontology language)와 같은 확장 가능한 시맨틱 웹 언어들을 이용하여 컴퓨팅/네트워킹 자원들, 네트워킹 경로들, 서비스들, 서비스 합성을 위한 제약 조건들, 개체들의 관계들을 의미론적으로 정의하고 있다. 리소스 능력을 정량화하는 부분은 미진한 진도를 보이고 있으며, 테스트베드의 성능 모니터링 부분에서는 GENI I&M (instrumentation and measurement) 논의를 중심으로 하여 Instools 과 OML 를 비롯한 여러 도구들이 활용되고 있다 [8]. 마지막으로 서비스 운용 및 제어와 관련해서는 네트워킹 프로토콜 스택을 세분화/추상화한 뒤 서비스 지향 구조에 따라 이들을 조합하고 세밀한 조절하는 SILO 등이 대표적이나, 아직 대부분의 테스트베드 구축 및 운용에는 실제로 적용되지는 않은 상태이다 [9].

2.2 슬라이스 기반 실험관리를 활용하는 관련 테스트베드 동향

GENI 에서 정의하고 있는 슬라이스는 테스트베드에서 제공되는 컴퓨팅 및 네트워킹 자원으로 구성된 하나의 실험용 네트워크로써, 물리적인 자원에서 할당된 가상 자원들의 집합을 제공한다.

사용자 요청에 의한 실험자원 공급을 위해 슬라이스의 구조 설계에 관한 일들이 진행되고 있으며, 안정적으로 슬라이스를 유지하기 위한 노력들도 병행되고 있다. 본 절에서는 몇 가지 사례를 소개하면서 슬라이스 관리에 관한 동향을 살펴본다.

2.2.1 Slice Federation Architecture

SFA(slice federation architecture)는 슬라이스 기반의 테스트베드 연합에 관한 GENI 의 제어 프레임워크 아키텍처로서, 슬라이스를 구성하기 위한 네트워크 재료(substrate)들의 인터페이스들의 집합을 정의한다 [10]. 여기서 슬라이스는 여러 substrate 들에 걸쳐서 네트워킹 실험을 지원하기 위한 컴퓨팅과 네트워킹 자원의 네트워크로 정의된다. 슬라이스는 물리적인 자원들(예, CPU, 메모리, 디스크, 대역폭)과 논리적인 자원(예, 파일 기술자, 포트 번호, 패킷 포워딩 경로)들을 포함한다. 슬라이스는 3 가지 상태를 가지게 되는데, 이는 사용자에게 리소스를 할당하는 단계, 할당된 자원들을 활성화하는 단계, 서비스 코드를 내리고 실행하는 상태로 이루어진다.

SFA 는 소유자, 운영자, 연구자로 구성되는 행위자들을 정의하고, 이 행위자들의 관계를 설정하기 위해 관리 권한 (management authority), 슬라이스 권한 (slice authority)를 설명한다. 관리 권한은 substrate 를 안정적으로 운영하고, 사용자에게 (소유자가 선언한 정책에 따라서) 컴포넌트의 접근 권한을 부여한다. 슬라이스 권한은 사용자들에게 슬라이스들에 대한 접근과 제어 권한을 부여한다.

SFA 는 RSpec, Ticket, Credentials 을 정의하는데, 이것들은 행위자들 사이에 권리에 대한 허가, 위임, 철회를 하는데 사용되는 데이터 형태이다.

- RSpec: RSpec 은 AM(aggregate manager)에서 제공하는 자원 현황을 표현하고 이러한 자원을 요청할 때 사용되는 형식이다. 이용가능한 자원들을 명세표를 전달하거나, 필요한 자원들을 요구할때 사용한다. 그러나 SFA 는 RSpec 에 대한 통일된 형식은 요구하지 않는다. 참고로 PlanetLab 과 ProtoGENI 에서는 리소스 요구사항을 정형화된

틀에 맞게 기술하기 위해 XML 을 이용한 RSpec 을 설계하고 있으며, ORCA 와 SURFnet 에서는 테스트베드의 다양한 측면을 지식으로 표현하기 위해 RDF (resource description framework)과 OWL (web ontology language)와 같은 확장 가능한 시맨틱 웹 언어들을 이용하고 있다.

- Ticket: Ticket 은 AM 이 서명한 RSpec 으로써 사용자가 요청한 자원을 주어진 시간동안 할당하겠다는 증서이다. 발급한 티켓은 요청한 자원이 할당된 후에 회수된다.
- Credential: 사용자가 슬라이스를 발급받고 AM 으로부터 자원 사용을 인가받으로면, 사용자가 합법적인 권한이 가지고 있다는 것을 증명하기 위해 credential 을 제공해야 한다. Credential 의 구체적인 형식은 제어 프레임워크에 따라 달리 결정된다.

2.2.2 GENI Plastic Slice Project

GENI Plastic Slice Project 는 장기간 운용되고 많은 실험들에 의해서 공유될 수 있는 실용적인 슬라이스를 제공하는 것을 목표로 한다 [11]. 이를 위해 한 슬라이스에 여러 RA 들의 자원을 포함시키면서, 한 슬라이스에 속한 자원을 여러 실험자들이 공유하는 것을 허용한다. 이를 통해 실험자들에게 한 슬라이스 안에서 종단간 데이터 평면을 제어할 수 있게 해준다.

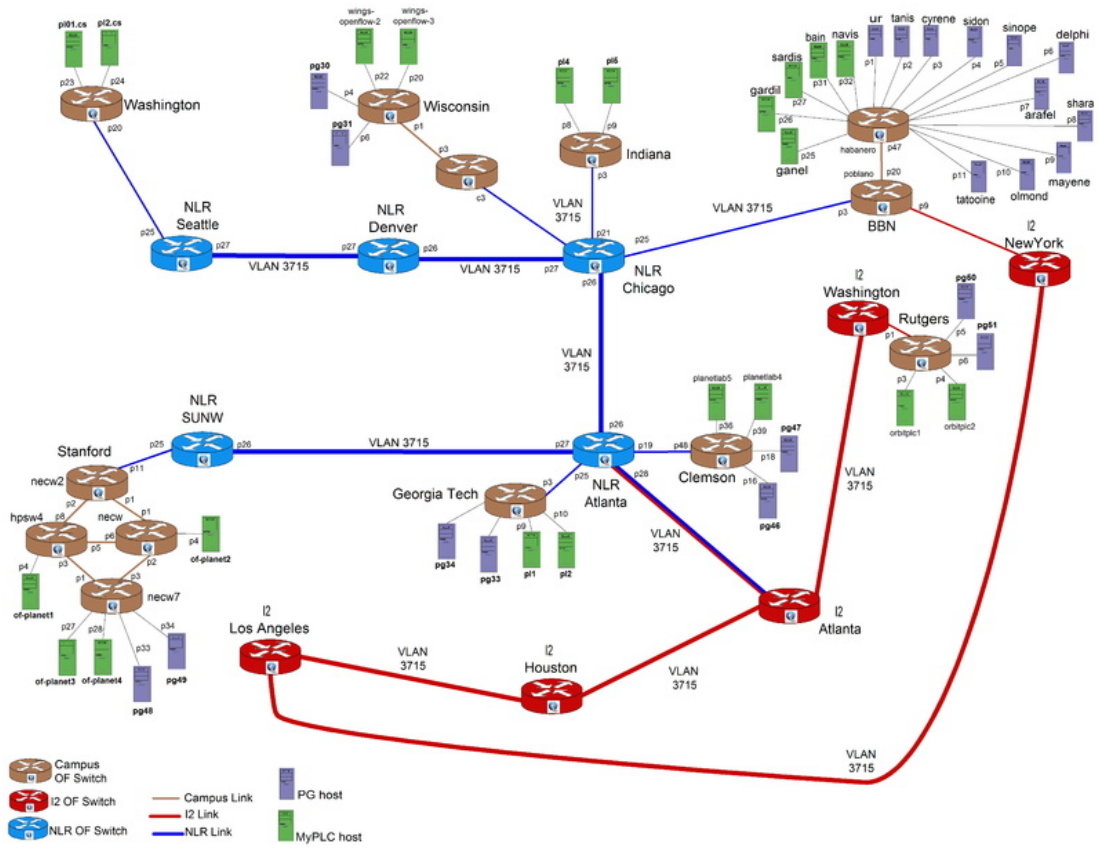

그림 1. GENI Plastic Slice Project 의 주요 VLAN 구성도.

이 프로젝트는 실험자에게 제공되는 GENI 자원의 품질을 향상시키기 위해, 장기간 연속적으로 중간 규모의 실험(8 개의 캠퍼스를 연결한 전국 규모의 인프라에서 3 개월 동안 10 개의 GENI 슬라이스를 실행하는 실험)을 진행하면서, 이를 통해 슬라이스를 관리하는 경험과 노하우를 축적하는 것을 중요한 목표로 삼고있다. 그림 1 은 구축된 VLAN 의 구성을 보여준다. 이 프로젝트에서는 슬라이스들을 이용하여 ping 실험, 비암호화된 TCP 스트림을 분석하는 netcat, HTTPS (Secure Hypertext Transfer Protocol)를 통해 웹 콘텐츠를 내려받는 wget 실험, 그리고 TCP 와 UDP 의 iperf 의 5 가지 실험을 수행하였다. 향후에는 보다 복잡한 프로토콜을 실험하기 위해, 비 IP 프로토콜을 실험할 수 있도록 테스트베드를 개선하고, IP 와 비 IP 라우팅 성능을 비교하면서, 비디오와 오디오 응용 지원을 위한 슬라이스 요구사항을 도출할 예정이다.

2.2.3 슬라이스 기반의 GENI Cloud

GENI Cloud 에서는 유칼립투스(Eucalyptus)를 이용하여 네트워크 단말 노드에 클라우드 컴퓨팅 자원을 제공한다 [12]. GENI Cloud 는 독립적으로 관리되는 클라우드들을 서로 연결한다는 Inter-Cloud 개념에서 출발하였다. 여러 지역에 흩어져 있는 클라우드 컴퓨팅 자원들을 미래 인터넷 테스트베드에 연동시키고, 실험자가 원하는 만큼 가용 계산 또는 저장 자원들을 가상화하여 제공하게 된다.

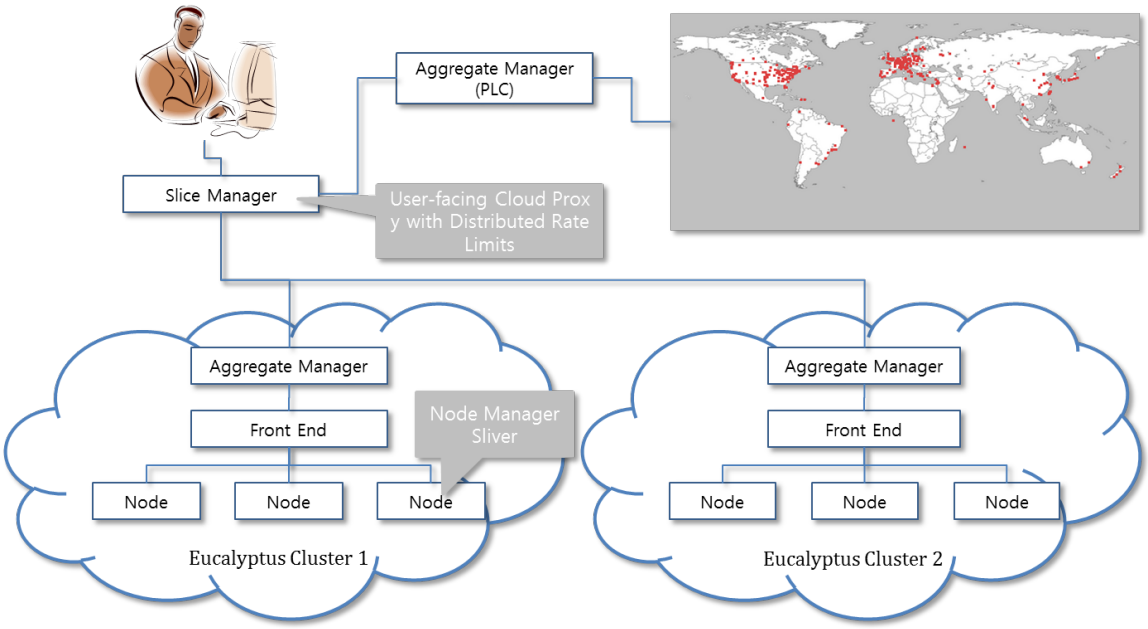

그림 2. GENI Cloud 의 구조.

GENI Cloud 는 SFA 에 의해서 슬라이스의 생성 (슬라이스의 생성/인스턴스화) 및 할당 (가상 머신의 셋업), 인증, 위임 기능을 구현하고 있다. 또한 GENI Cloud 는 유칼립투스에 GENI 연동 인터페이스를 제공한다. 이 인터페이스는 유칼립투스 클러스터들을 SFA 를 통해 슬라이스로 만들기 때문에, 실험자들이 유칼립투스 클러스터를 네트워킹 실험에 사용할 수 있게 해준다. GENI Cloud 는 그림 2 와 같이 PlanetLab 의 단말 노드의 위치에 클라우드 클러스터들을 설치하며, 이 클라우드 컴퓨팅 자원을 슬라이스 형태로 이용하기 위해 클라우드 콘트롤러와 클러스터 콘트롤러를 통해 클라우드 컴퓨팅 자원을 획득한다.

2.3 탄력적이고 안정적인 실험자 지원을 위한 슬라이스의 관리

각각의 실험자에게 부여된 슬라이스는 실험에 사용되는 컴퓨팅과 네트워킹을 위한 각종 자원들의 집합에 대한 실험자의 권한을 연결하는 고리이다. 각 실험자는 슬라이스를 이용하여 확보된 자원들에 자신이 원하는 서비스 기능을 제공할 수 있는 응용 코드들을 수행하고 이에 대한 필요한 조정 및 관찰을 지속적으로 진행해야 한다. 서비스 코드들이 안정적으로 동작하기 위해서는 2.1 절에서 설명한 것과 같이 자원의 상태를 계속적으로 관찰해서 파악하고 있으면서 문제가 발생하면 적절히 대응하는 조정 기능이 필요하다. 이러한 맥락에서 슬라이스에는 실험을 보조하기 위한 특별한 기능들이 추가되어야 하는데, 이에 대한 상세한 내용을 본 절에서 설명한다.

2.3.1 슬라이스의 구성

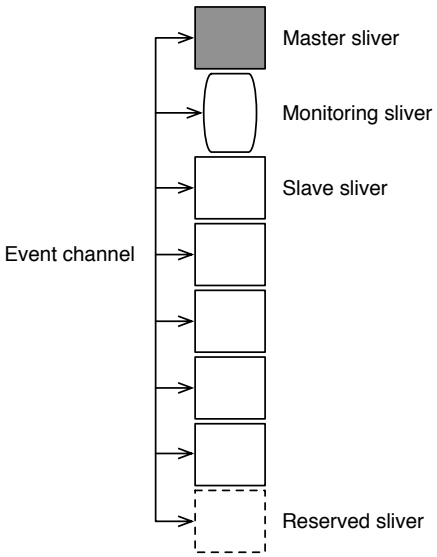

그림 3. 슬라이스의 구성.

하나의 슬라이스는 컴퓨팅과 네트워킹 자원들을 제공하는 슬리버들뿐만 아니라, 특별한 기능들을 수행하는 슬리버들도 함께 포함하며, 이는 그림 3 과 같다. 슬라이스는 마스터 슬리버 (master sliver), 슬레이브 슬리버 (slave sliver), 감시 슬리버 (monitoring sliver), 예비

슬리버(reserved sliver)로 구성된다. 또한 슬리버들 간에 정보 전달을 위한 이벤트 채널(event channel)이 포함된다. 개별적인 슬리버들의 역할은 아래와 같다.

- 마스터 슬리버: 마스터 슬리버는 모니터링 슬리버를 통해 각각의 슬레이브 슬리버의 상태을 파악하면서 문제가 발생한 슬레이브 슬리버들을 예비 슬리버로 교체한다.
- 슬레이브 슬리버: 슬라이스를 구성하는 컴퓨팅 자원과 네트워킹 자원을 제공한다.
- 감시 슬리버: 모니터링 슬리버은 슬레이브 슬리버들의 성능을 감시하면서 슬라이스가 안정적으로 유지되고 있는지 판단한다. 만약 문제가 발생하면 마스터 슬리버에게 통지한다.
- 예비 슬리버: 운용중인 슬레이브 슬리버가 비정상적으로 동작할 경우에 대비해, 예비 슬리버를 운용한다. 슬라이스 안에서 예비 슬리버는 문제가 발생한 슬레이브 슬리버를 대신할 수 있다.
- 이벤트 채널: 슬라이스 안에 있는 모든 슬리버들이 이벤트 정보를 주고받을 수 있는 통로를 제공한다. 이벤트 채널을 통해, 마스터 슬리버, 슬레이브 슬리버들, 감시 슬리버들은 제어 명령과 이벤트, 상태에 관한 정보들이 교환한다.

2.3.2 슬라이스들의 연결

주어진 서비스를 실행하는데 필요한 자원들이 기존 슬라이스에 없다면, 필요한 능력을 제공할 수 있는 다중 슬라이스들을 연결하여 사용할 수 있다. 예를 들면, 컴퓨팅 자원을 공급하는 RA(들)에서 만들어진 슬라이스와 네트워킹 자원을 공급하는 RA(들)에서 만들어진 슬라이스가 합쳐질 수 있다. 이와 같은 다중 슬라이스들의 연동을 통한 서비스 운용을 하기 위해서는 슬라이스간에 연결 관계가 명확히 정리되어야 한다. 그림 4 와 같이 슬라이스들을 연결하기 위해 주 슬라이스(primary slice)와 하나 이상의 보조 슬라이스(secondary slice)들이 연결된다. 이때주어진 연동 프로토콜에 따라 다중 슬라이스의 슬리버들을에 대한 접근 권한을 부여하고, 연결시킨다. 연결되는 슬라이스들은 이벤트 채널을 통해 이벤트 정보를 주고받는다. 주

슬라이스는 연결되는 슬라이스들이 주어진 서비스를 잘 지원할 수 있도록 관리하면서, 서비스 코드의 실행을 위해 할당된 자원들로부터 문제가 발생하면 주어진 규칙(예, 예비 슬리버 운용)에 따라 해결한다. 주 슬라이스는 마스터 슬리버를 가지고 있는데, 이 마스터 슬리버는 주 슬라이스의 슬레이브 슬리버들뿐만 아니라 보조 슬라이스의 슬리버들도 함께 관리한다. 주 슬라이스와 보조 슬라이스가 가지고 있는 감시 슬리버는 슬레이브 슬리버들로부터 수집한 상태 정보를 주 슬라이스의 마스터 슬리버에게 전달한다.

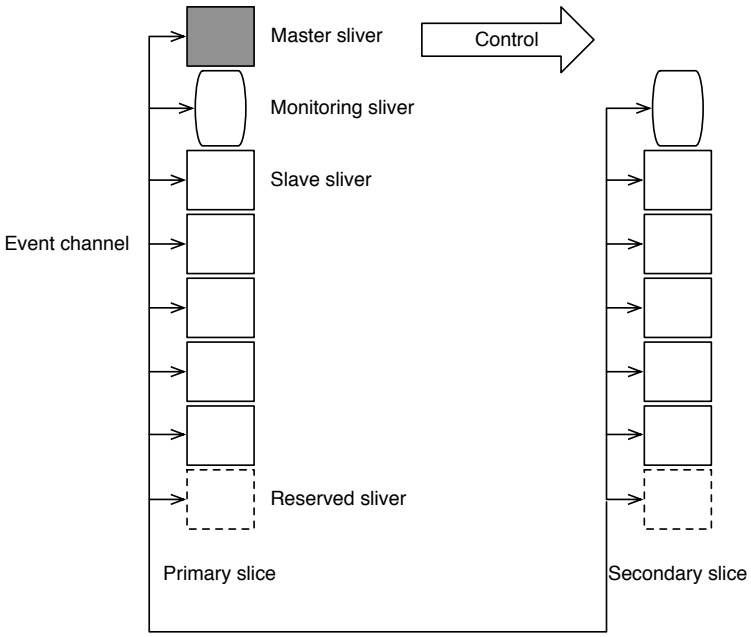

그림 4. 슬라이스의 연결.

2.3.3 슬라이스의 유지

안정적인 실험자원 공급을 위해, 일부 슬리버에서 오류가 발생하면 그에 대한 대처방안으로 마스터 슬리버는 문제의 슬레이브 슬리버를 예비 슬리버로 교체한다. 슬리버 교체는 그림 5 와 같이 진행되는데, 사용중인 슬레이브 슬리버의 작업 내용을 예비 슬리버에게 온전히 이전시킨 다음, 이 예비 슬리버를 슬레이브 슬리버로 사용하면서 오류가 발생한 슬레이브 슬리버의 사용을 중단시킨다. 슬레이브 슬리버에 문제가 발생하게 되면, 감시 슬리버는 이를 감지하고 문제의 이벤트를 마스터 슬리버에게 통보한다. 마스터 슬리버는 대체할 예비 슬리버를 결정한다. 작업 내용을 안전하게 이전시키기 위해, 오류가 발생한 슬리버에서 작업 중인 모든 콘텍스트를 예비

슬리버에게 전달하고, 이 예비 슬리버를 다른 슬레이브 슬리버들과 연결시켜서 슬라이스의 형태를 지속적으로 유지한다.

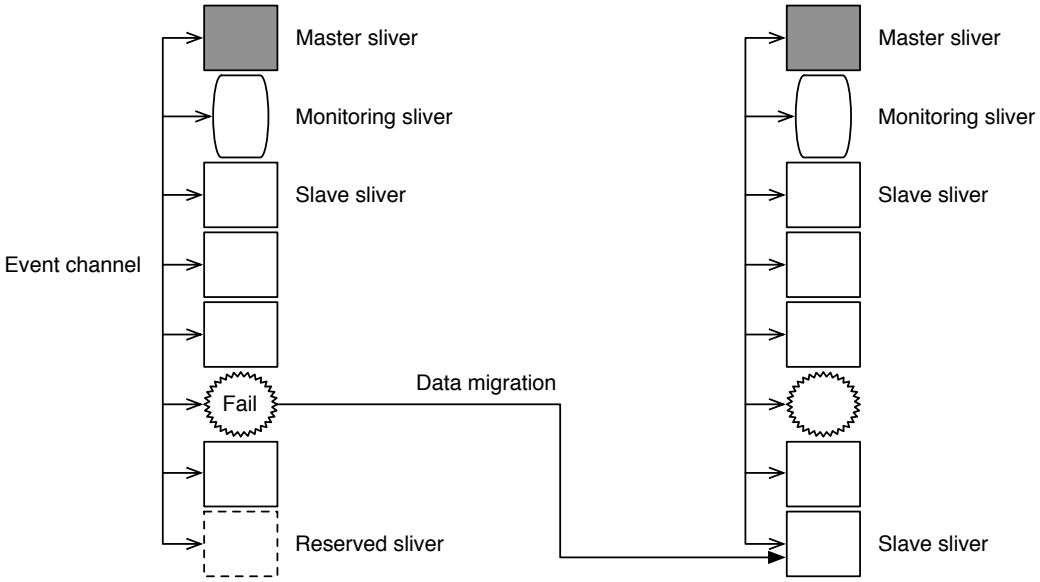

그림 5. 슬리버 오류발생시 예비 슬리버의 활용을 통한 서비스 연속성 보장.

2.3.4 슬라이스 상에서의 서비스 운용

주어진 슬라이스를 이용하여 서비스를 운용하기 위한 상위 수준의 실험 제어가 필요하다. 실험 제어는 서비스가 정의하는 서비스의 단위 작업들을 이해하고, 이 단위 작업들을 주어진 종류와 양의 자원들을 공급하는 여러 슬레이브 슬리버들에게 대응시킨다음, 서비스 품질이 적정한 수준에서 유지되는지를 정량적으로 측정한다. 세부적인 과정은 다음과 같다. 세부적인 순서는 다음과 같다.

- 자원 정합 (resource matchmaking): 어떤 슬레이브 슬리버 위에 어떤 서비스 단위 작업들을 대응시킬지 결정한다. 슬리버들은 서로 다른 자원 능력(resource capability)를 가지고 있기 때문에, 서비스 단위 작업들의 요청사항과 슬리버들의 자원 능력을 조화시키는 것이 중요하다.
- 자원 할당 (resource allocation): 서비스 단위 작업들에게 슬레이브 슬리버를 할당하고 서비스 코드를 설치한다.

- 자원 감시 (resource monitoring): 슬라이스의 성능을 감시한다. 슬라이스의 모니터링 슬리버들은 관련된 슬리버들의 성능 변화를 주어진 (또는 상황에 따라 변화된) 시간 간격 마다 측정하여 실험과정에서 계산, 저장, 네트워크 자원의 양이 부족하지 않은지 점검한다.
- 자원 조절 (resource tuning): 슬라이스 또는 슬리버의 자원 종류와 양을 조절한다.
- 자원 회수 (Cleanup): 슬라이스의 사용 기한이 만료되면 서비스 코드를 제거하고 할당된 슬리버들을 회수하고 슬라이스를 종료한다.

그림 6 에서는 준비된 슬라이스 위에서 서비스를 운용하는 사례를 보여준다. 서비스의 시나리오는 다음과 같다. 미디어 서버로부터 획득한 비압축 영상을 단계적으로 DXT (DirectX Texture) 압축을 하면서 다양한 프레임율을 가진 영상 스트리밍 서비스를 제공한다. 이렇게 하는 이유는 한 호스트에서 모든 작업을 병행하기에는 컴퓨팅 비용이 많이 들기 때문이다. 여러 호스트들 위에서 운용되는 슬레이브 슬리버마다 DXT 압축을 위한 서비스 코드를 설치하고, 일부 슬레이브 슬리버에는 캐싱(caching) 코드를 내려서 영상 공급을 하면, 컴퓨팅 부하를 분산시키면서 다양한 품질의 영상 스트리밍 서비스를 할 수 있다.

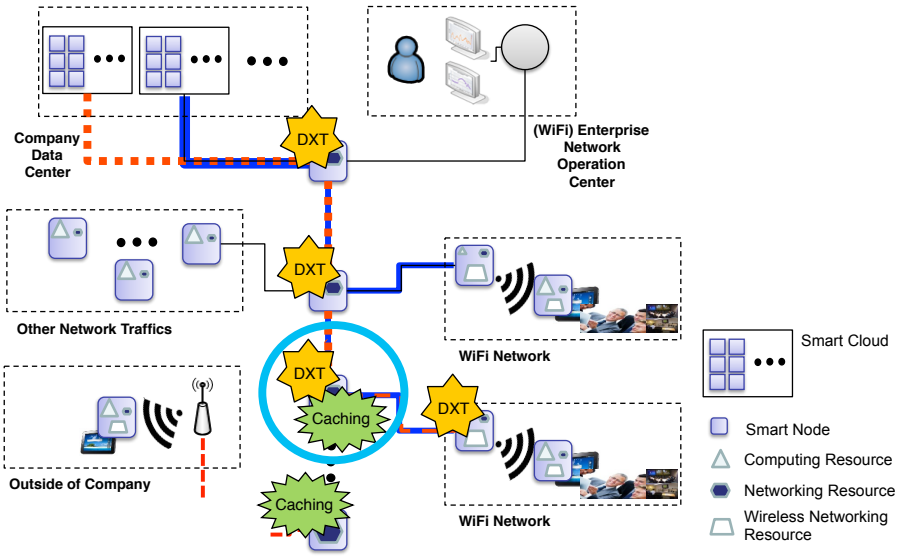

그림 6. 슬라이스 상에서의 서비스 운용 사례.

2.3.5 보안 문제들

고려되어야 할 보안 이슈들을 다음과 같다. (1) 먼저 실험자를 인증을 하고, (2) 실험자에게 슬라이스에 대한 접근 권한을 인가하여 필요한 실험 자원들을 추가/삭제한다. 모든 절차는 인증이 먼저 이루어지며, 인증이 완료됨과 동시에 자원들과 서비스들에 대한 접근 권한을 획득하게 된다. 모든 자원과 서비스에 대한 활동들은 주어진 권한에 의해서 접근 제어가 이루어지게 된다. 기존에는 인증과 인가를 위해 IBAC(Identification-Based Acccess Control)을 사용하였다. IBAC 은 ID 를 인증하는 방식인데, 중앙집중식으로 운용되기 때문에 확장성을 제공하지 못한다. 이 문제를 극복하기 위해 통합된 ID 를 관리하는 방식으로 운용되는 FIdM (Federated Identity Management)이 제안되었다. 또 다른 방안으로 역할기반으로 운용되는 RBAC(Role-Based Access Control)이 제안되었다[14]. RBAC 은 소규모 시스템에 적용하기에는 적합하지만, 많은 규칙들을 적용하기에는 한계가 있으며, 다양한 콘텍스트를 지원하지 못한다. 이러한 단점을 극복하고 세밀한 접근 제어를 위해 속성기반으로 운용되는 ABAC (Attribute-Based Access Control) [15, 16] 논의되고 있다. 또한 MS 의 AZURE 에서 운용되는 CBAC (Claim-Based Access Control) [17]과 다양한 인자(factor)에 의한 정의되는 위험(Risk)에 의해 운용되는 RAdAc (Risk Adaptive Access Control) [18]도 함께 논의되고 있다. 그러나 이러한 접근 제어 기법들은 컴퓨팅과 네트워크 자원에 대한 속성의 의미를 동의하기가 어렵다는 이슈가 제기되어, 이를 해결하기 위한 방안으로 인증을 기반으로 운용되는 NBAC (autheNtication-Based Access Control)[19]과 권한에 의해 운용되는 ZBAC (authoriZation-Based Access Control)[20] 등이 제안되었다.

3 Computing-centric Resources for Experimental Facilities

3.1 An Overview on Cloud Computing Resources

3.1.1 Basic Concept on Cloud Computing

클라우드 컴퓨팅이라는 단어는 그 동안 그리드(Grid) 컴퓨팅, 유틸리티(Utility) 컴퓨팅, thin-client 기반의 컴퓨팅 등의 다양한 모습으로 발전해 왔으며, 2008 년 ~ 2010 년간 이를 활용한 상용 서비스의 개시와 실질적인 성능과 장점 등의 입증이 가시화됨에 따라서 많은 관심을 받고 있다. 특히 일반 기업체에서 보안적인 측면을 고려하여, 모든 자료가 서버에 저장되고 개인이 유출하기 어려운 장점으로 인하여 긍정적으로 평가되고 있는 실정이다. 아울러 Green IT 에 대한 관심 증가와 맞물려서 기업의 서버 등 인프라 비용 감소와 함께 전력 사용량 감소라는 부수적인 효과까지 고려되어 활발하게 도입을 고려하고 있다. 특히 클라우드 컴퓨팅은 client 기기에 대한 하드웨어 및 소프트웨어 측면의 요구사항이 대부분 웹 브라우저의 구동만 이루어지면 되는 수준으로 낮아 졌다는 점, 그리고 최근 들어 노트북과 Apple iPhone/iPad 류의 모바일 휴대기기에서의 인터넷 사용이 급격하게 확산되는 점과 맞물려 모바일 비즈니스 및 콘텐트 시장에서 큰 관심을 받고 있다. 이러한 추세는 현재 일반화된 3 세대 이동통신을 넘어서, 표준이 거의 완료된 3GPP LTE(Long-Term-Evolution)/LTE-Advanced 와 Mobile-WiMAX 16m 의 상용화가 본격화 되면, 현재보다 더 많은 부문으로의 확산이 이루어 질 것으로 보인다. Juniper Research 에 의하면 cloud 기반의 모바일 어플리케이션은 2009 년 $400 million 이었으며, 2014 년에는 $9.5 billion 으로 증가할 것으로 예상하고 있다. 이러한 클라우드 컴퓨팅은 사용자의 환경 밖에서 서비스로서 제공된 확장 가능한 컴퓨팅 자원을 사용한 양에 따라 비용을 지불하고 사용하는 것이라고 할 수 있다. 사용자는 사용한 자원에 대한 비용만을 지불하며 클라우드 환경에 있는 모든 자원을 언제 어디서나 인터넷을 통해 엑세스할 수 있다. 사용자는 더 이상 실제

소프트웨어와 인프라를 관리할 필요가 없으며 이러한 관리는 클라우드 서비스 제공자가 담당한다. 클라우드 컴퓨팅에서는 컴퓨팅과 정보 기술 서비스를 완전히 다른 방식으로 바라본다. 클라우드 컴퓨팅의 장래는 필요한 컴퓨팅 자원을 쉽게 엑세스하고 확장할 수 있도록 하는 데 있다[34].

현재의 클라우드 컴퓨팅의 기본 서비스 형태로 일반적으로 세가지 형태로 분류 할 수 있다. 그 중 첫 번째 형태로는 Sofrware as a Service(SaaS)로 클라우드 서비스 중 가장 일반적인 유형이며 대부분의 사용자가 언젠가 사용해 본 경험이 있는 유형이다. SaaS 클라우드 모델에서는 서비스 제공자가 모든 인프라와 소프트웨어 제품을 제공한다. 사용자는 웹 기반의 프론트엔드를 사용하여 서비스와 상호작용한다. 이러한 서비스는 Google Apps 와 같은 웹 기반의 사용자 소프트웨어에서 Microsoft Office Live 와 같은 사무용 소프트웨어에 이르기까지 다양한 범위에 적용된다.

클라우드 컴퓨팅 서비스의 두 번째 형태로는 Platform as a Service(PaaS)로 서비스 제공자가 자체 하드웨어 인프라에서 호스트하는 소프트웨어와 제품 개발 도구를 제공하는 클라우드 서비스이다. 사용자는 제공된 API 와 플랫폼 또는 개발용 그래픽 사용자 인터페이스를 사용하여 애플리케이션을 개발할 수 있다. 이러한 유형의 서비스에 대한 일반적인 사례로는 Salesforce.com 의 Force.com 과 Google App Engine, 그리고 Microsoft Azure 를 들 수 있다.

마지막 클라우드 컴퓨팅 서비스의 세 번째 형태로는 Infrastructure as a Service(IaaS)로 가상 서버, 데이터 스토리지 및 데이터베이스와 같은 일련의 빌딩 블록이나 기본 서비스를 제공한다. 사용자는 이러한 서비스를 조합하여 애플리케이션을 전개하고 실행할 수 있는 플랫폼을 구성할 수 있다. 또한, 시스템을 편리하게 구축하거나 해체할 수 있다. IaaS 서비스는 SOAP 나 REST 기반 메시지를 사용하는 API 를 통해 엑세스 할 수 있다. IaaS 와 관련된 가장 유명한 사례는 Amazon Web Services(AWS)와 Rackspace 이다.

3.1.2 Latest Trends on Cloud Computing

최근 클라우드 컴퓨팅의 트렌드는 기존의 서비스의 확장 형태로서 다양한 형태의 서비스로 진화하고 있다. 첫 번째 서비스 형태로는 Network as a Service(NaaS)로 하나의 가상 네트워크나 다수의 가상 네트워크에서 두 개의 가상 머신 인스턴스 간의 상호 연결을 위하여서 네트워크 상에서의 연결에 대한 네트워크 추상화를 가능하도록 한다. 현재 NaaS 에 대한 제공 사례로는 Cisco 에서 OpenStack 클라우드를 활용한 네트워크 서비스 제안을 통해 NaaS 에 대한 구체적인 개념과 요구사항들이 정립되고 있다[33]. 두 번째 서비스 형태로는 Communication as a Service(CaaS)로 클라우드를 기반으로 VoIP, 인스턴스 메신져, 이메일, 비디오 컨퍼런스와 같은 서비스를 제공하도록 한다. 이러한 CaaS 를 이용한 제공 사례에는 Global Crossing 의 오디오 컨퍼런스, 8x8 Inc 의 비디오 컨퍼런스가 대표적인 사례로 꼽히고 있다. 마지막 최근 서비스 트렌드 형태로서 Database as a Serbice(DaaS)는 클라우드 상에서의 데이터베이스 자체를 호스팅해준다. 이러한 DaaS 의 사례로는 Amazon 의 Simple/Relational DB 서비스, Salesforce.com 의 Database.com 을 대표적인 사례로 들 수 있다.

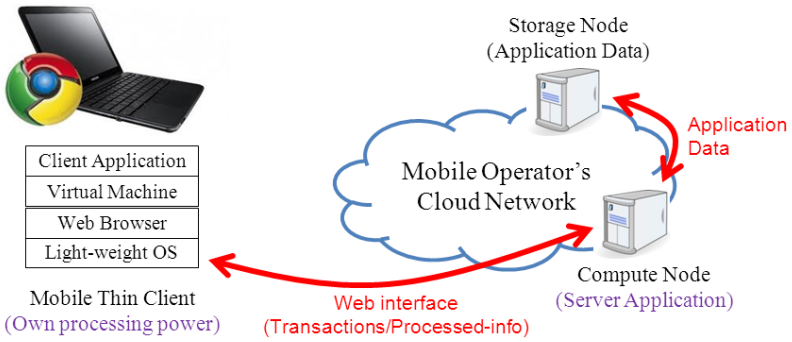

그림 7. Mobile Thin-client Service.

또한, 최근 모바일 비즈니스 및 콘텐츠 시장의 급격한 성장 및 4 세대 이동통신의 시작과 맞물려서 기존의 클라우드 컴퓨팅은 모바일 환경에 맞게 진화하고 있다. 이러한 모바일 클라우드 환경을 기반으로 하는 최근 모바일 클라우드 컴퓨팅의 형태로는 다음과 같다.

첫 번째로 모바일 Thin-client 서비스 형태로 그림 7 과 같이 모바일 노드에서는 프로세싱 전원을 가지고 있고, 클라이언트 애플리케이션을 실행하는 형태이다. 또한, 모바일 클라우드 상에서 서버 애플리케이션을 구동하며, 모바일 노드와 모바일 클라우드 내의 compute 노드 사이에는 기존의 웹 기반의 클라이언트/서버 인터페이스를 사용하여 transaction 이나 프로세스 정보들을 교환한다. 이러한 모바일 Thin-client 형태로 클라우드 컴퓨팅 서비스를 제공하는 사례로는 구글의 Chromium OS 와 기존의 스마트패드와 스마트폰이 있다. 또한, IBM Cloud School, West Sprinfield High School, Coleman Tech Chater High School, New York Columbia High School, Minnesota Online High School, Horizon Project, Google Back to School 에서는 모바일 Thin-client 서비스를 활용하여 교육분야에서 클라우드 컴퓨팅의 활성화가 이루어지고 있다.

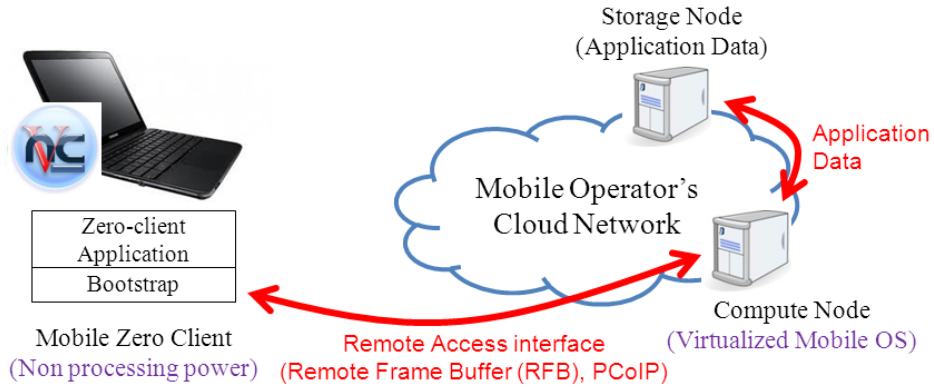

그림 8. Mobile Zero-client Service.

두 번째는 그림 8 와 같이 모바일 Zero-client 서비스 형태로 모바일 노드에서는 프로세싱 전원을 가지고 있지 않고, 오로지 VNC-terminal 과 같은 원격 뷰어를 실행하는 모바일 Zero-client 애플리케이션과 bootstrap 을 가지고 있다. 또한 모바일 클라우드 내의 compute 노드에서는 가상화된 모바일 OS(Android, iOS)를 지원함으로써, 모바일 노드의 클라이언트와 클라우드 서버 사이에서는 Remote Frame Buffer(RFB)와 같은 프로토콜을 이용하여 원격 접속 인터페이스를 제공한다[37]. 이를 통하여 클라이언트는 클라우드 서버로 원격 뷰어를 통해 접근 할 수 있으며, 애플리케이션의 데이터들은 클라우드 내의 스토리지 노드 상에 저장할 수 있다.

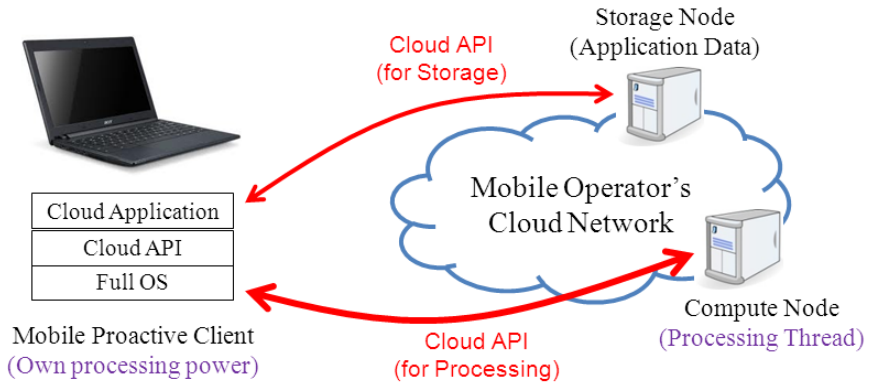

그림 9. Mobile Proactive-client Service.

마지막은 모바일 Proactive-client 서비스 형태로 그림 9 과 같이 모바일 노드 상에서 클라우드 서비스를 실행시키며 클라우드 노드와 클라우드 API 를 이용하여 서비스를 제공하며, 클라우드 노드에서는 확장된 형태의 CPU 노드와 스토리지 노드를 제공한다. 모바일 Proactive-client 에서 사용하는 클라우드 API 는 모바일 클라이언트와 클라우드 간의 전용화된 API 로서 클라우드 서비스의 형태에 맞춰 프로세싱을 위한 API, 스토리지를 위한 API 를 제공하여 클라이언트와 클라우드 간에 최적화된 서비스 형태를 제공하도록 한다. 이와 같은 모바일 Proactive-client 를 이용한 서비스로는 클라우드를 이용하여 파이썬 라이브러리 인터페이스를 제공하는 PiCloud API, 클라우드를 이용하여 PHP 인터페이스를 제공하는 SimpleCloud API 가 있다. 이 중 SimpleCloud API 에서는 파일 스토리지 서비스, 도큐먼트 스토리지 서비스, 심플 큐 서비스와 같은 클라우드 애플리케이션을 위하여 PHP 를 이용한 공통의 클라우드 인터페이스를 제공한다.

3.1.3 Standardization and Federation

클라우드 컴퓨팅을 위한 표준화로 현재 IEEE 에서는 P2301 Working Group 을 통하여서 Cloud Profiles 에 대한 표준화, 그리고 P2302 Working Group 을 통하여서 Intercloud 에 대한 표준화를 진행 중에 있다[38].

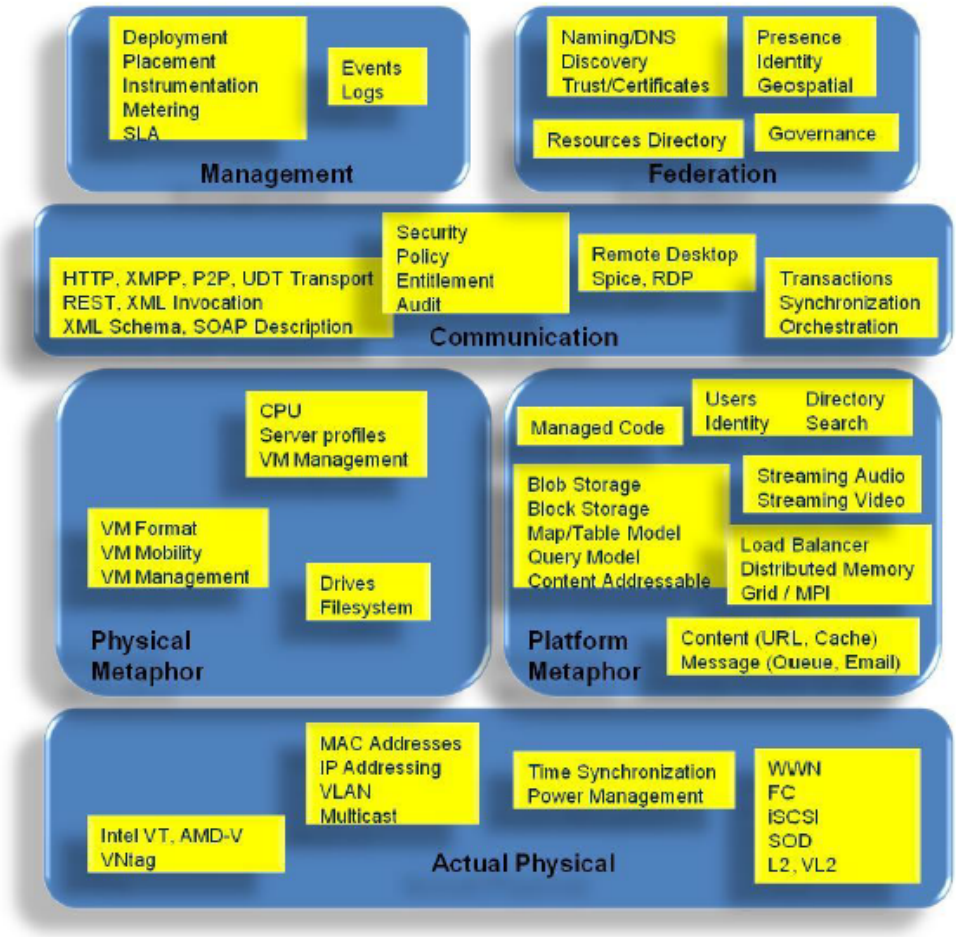

그림 10. P2301 Profile Examples

먼저 P2301 Working Group 에서 제공하는 Cloud Profiles 에 대한 표준화 가이드는 "Guide for Cloud Portability and Interoperability Profiles (CPIP)"라는 이름을 통해 그림 10 와 같이 제공되며, 가이드의 개발 목적으로는 클라우드 컴퓨팅 벤더와 개발 및 빌딩 그리고 표준화 기반의 클라우드 컴퓨팅 제품과 서비스를 제공하려는 사용자에게 향상된 portability, commonality, interoperability 를 지원하도록 한다. 클라우드 컴퓨팅 시스템에서는 다양한 요소들을 포함하고 있다. 각 요소에서는 다양한 옵션과 인터페이스 그리고 파일 포맷과 운영 규칙들이 있다. 이렇게 많은 종류의 인터페이스, 포맷, 규칙들은 다른 의미들을 가지고 있다. P2301 가이드에서는 다양한 옵션들을 열거하고, 인터페이스, 포맷, 그리고 규칙들을 "profiles"로

그룹화한다. 이러한 방법으로, 클라우드 에코시스템 참여자들에게 더욱 portability, commonality, interoperability 경향과 클라우드 컴퓨팅 성장 비율에 도움이 되도록 한다.

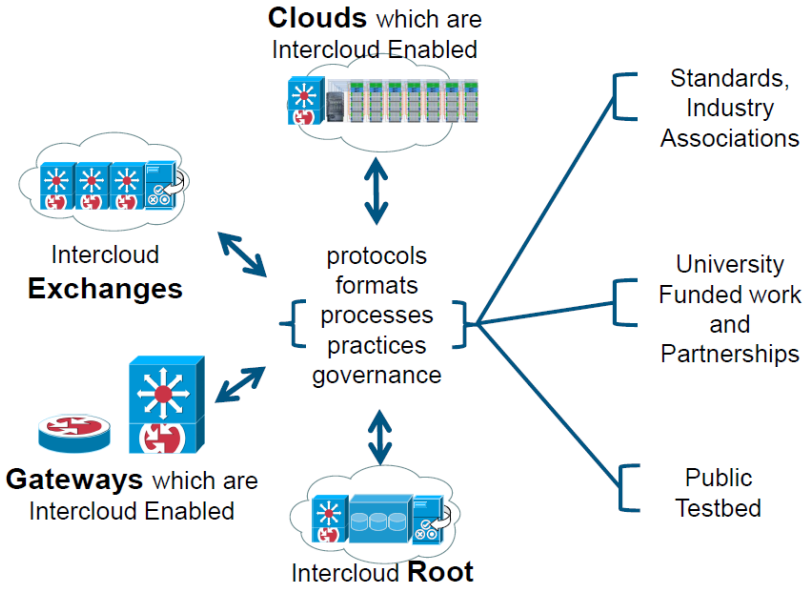

그림 11. P2302 Federation Standard Examples.

다음으로 P2302 Working Group 에서 제공하는 Intercloud 에 대한 표준화 가이드는 "Standard for Intercloud Interoperability and Federation (SIIF)"라는 이름을 통해 제공되며, 그림 11 와 같이 해당 가이드의 목적으로는 동적인 클라우드 인프라스트럭쳐를 위하여 클라우드 제공자들 사이에서의 상호운용성과 federation 을 위한 표준을 제시한다[39]. 본 가이드에서는 토폴로지, 함수, 클라우드와 클라우드 사이의 interoperability 과 federation 을 위한 관리를 정의한다. 위상학적 요소들로 클라우드들과 roots, 그리고 클라우드 간의 중재 관리를 위한 exchanges 마지막으로 클라우드 간의 데이터교환 중재를 위한 gateways 를 포함한다. 기능학적 요소들은 이름 공간, presence, messaging, reousrce ontologies 와 인프라스트럭쳐를 포함한다. 또한, 관리 요소들로서 등록, geo-independence, trust anchor, potentially compliance, audit 를 포함한다. 본 표준화에서는 클라우드 내부에서의 운영에 대해서는 기술하지 않고 있다.

마지막으로 HP, Intel, Yahoo!와 함께 연구되는 Open Cirrus 는 오픈 클라우드 컴퓨팅 연구 테스트베드로서 글로벌환경과 분산된 데이터센터 규모에서의 설계와, 프로비져닝, 서비스 관리,

federation 을 지원하도록 설계되었다[40][41]. 본 연구에서는 클라우드 컴퓨팅에서의 foster 시스템 레벨의 연구를 목표로 하며, 새로운 클라우드 컴퓨팅 애플리케이션들과 애플리케이션들의 연구를 장려하고 실험적인 데이터의 수집을 제공한다.

3.1.4 Future Internet and Cloud Computing

미래인터넷과 클라우드 컴퓨팅을 위하여 유칼립투스 클라우드 플랫폼의 네트워크 기능의 확장을 통해 RENCI 에서는 Network Extensions to Eucalyptus(NEuca)를 제공한다. NEuca 를 통해서 가상머신 상에서의 다수의 네트워크 인터페이스를 VLAN 을 통해 연결하고 호스트 노드의 물리적인 인터페이스를 연결하도록 설정 및 생성한다. 또한, 게스트 가상 머신에서 부팅 이후에 대한 액션을 임의로 수행한다[42].

다음으로 GENICloud 는 그림 12 과 같이 클라우드 컴퓨팅을 위한 오픈소스 소프트웨어 프레임워크(Framework) 중 하나인 유칼립투스를 GENI 의 미래인터넷 테스트베드와 이기종 자원(Resource)간의 페더레이션(Federation)을 목적으로 한다[13]. 유칼립투스와 PlanetLab 의 페더레이션으로 미래인터넷 테스트베드를 이용하는 실험자들에게 개발분야, 연산분야, 데이터 생성분야와 같이 클라우드를 이용함으로써 더욱 더 포괄적인 플랫폼을 제공한다. 또한, 클라우드 컴퓨팅의 장점을 이용하여, GENI 의 실험자에게 기존의 제약적인 서비스의 요구사항들을 동적인 스케일(Scale)로 사용할 수 있을 뿐 아니라, 클라우드와 다른 서비스들의 연계를 통하여서 이익을 볼 수 있다. 이러한 GENICloud 는 유칼립투스 클라우드 플랫폼과 PlanetLab 미래인터넷 테스트베드의 이기종 자원간의 연동을 통하여서 상호 보완적인 요소로 둘 사이의 제약적인 요소들을 보완한다.

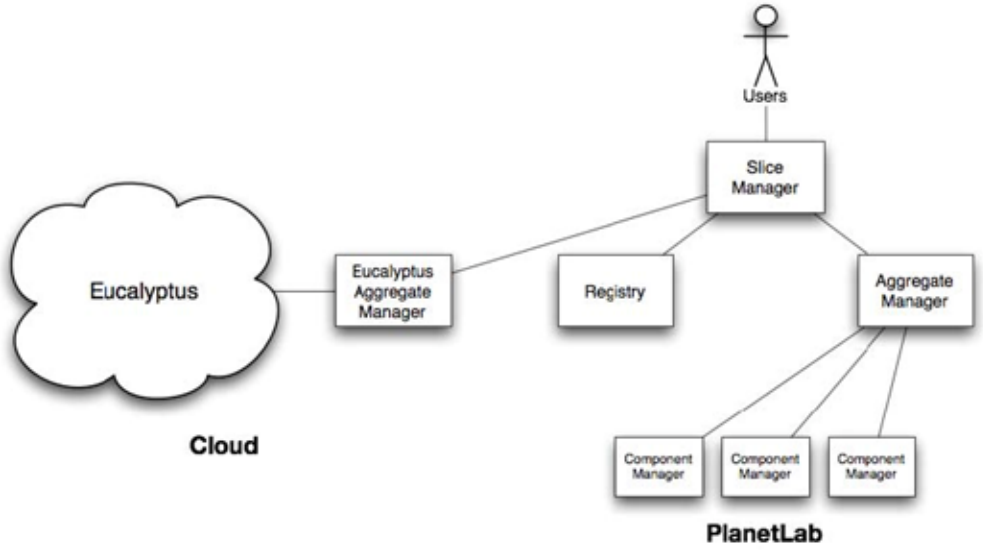

그림 12. GENICloud 시스템 구조도.

미래인터넷 테스트베드인 PlanetLab 에서는 분산된 가상 머신들의 집합체로서 'slice'라는 개념으로 정의하고 있으며, PlanetLab 오버레이 네트워크의 슬라이스 형태로 각기 다른 노드에서 동작중인 가상 머신들을 말한다. 또 다른 용어로서 각각의 슬라이스 내에 개별 가상 머신들을 가리켜 'sliver'라는 개념으로 정의 된다. GENICloud 에서는 유칼립투스 가상 머신을 슬라이스의 개념에서 확장하게 되며, GENICloud 의 슬라이스는 유칼립투스 클라우드 플랫폼을 통하여서 PlanetLab 의 리소스와 가상 머신들을 연동할 수 있으며, 실험을 진행 할 수 있도록 제공 된다.

Slice-based Federation Architecture(SFA)는 사용 가능한 슬라이스 기반의 네트워크를 연동 및 호환하기 위한 인터페이스의 집합과 데이터 타입으로 정의된 구조이다[12]. 이러한 SFA 에서는 아래와 같이 3 가지 형태의 인터페이스를 정의하고 있으며, Registry 인터페이스, Slice 인터페이스, 컴포넌트 관리 인터페이스이다.

- Registry 인터페이스: 슬라이스 기반의 네트워크상에서 Registry 인터페이스는 사용자, 슬라이스, 노드와 같은 정보들을 계속적으로 추적하며 데이터베이스 또는 파일 형태로 유지한다.

- Slice 인터페이스: 슬라이스를 관리하기 위한 인터페이스로, 슬라이스의 초기화 및 리소스 공급과 제어를 하도록 한다. 해당 인터페이스는 오버레이 네트워크상의 슬라이스에 대한 제어를 하며 슬리버 기반의 제어는 컴포넌트 관리 인터페이스에서 제공한다.
- 컴포넌트 관리 인터페이스(Component Management Interface): 컴포넌트 관리 인터페이스에서는 물리적인 자원인 CPU, 논리적인 자원인 소켓(Socket), Synthetic 리소스인 네트워크 경로지정(Route)에 대한 캡슐화(Capsulation)를 제공한다.

GENICloud 의 구현에 대한 중점 사항으로는 유칼립투스의 Aggregate Manager(AM)에 대한 구현을 중점을 두고 있다. AM 은 PlanetLab 과 유칼립투스 클라우드의 중재자 역할을 수행하며, 슬라이스에서의 유칼립투스 인스턴스(Instance)의 생성을 관리하고, 사용자가 슬라이스 상에 리소스 할당에 관한 명령을 질의할 때, 슬라이스와 인스턴스 사이의 맵핑(Mapping)을 유지 한다. Eucalyptus Aggregate Manager 는 그림 5 와 같이 유칼립투스 클라우드에 대한 제어와 관리에 대한 기능을 제공하도록 설계되었다. 사용자에게 슬라이스 상의 유칼립투스 가상 머신 인스턴스에 대한 연계를 하도록 하며, 인스턴스 타입에 대한 상세 규격 형태로 제공되는 RSpec(Resource Specification)을 SFI tool (SFA API)을 이용하여 맵핑/번역을 하고, PlanetLab 상에서 슬라이스 관리를 하도록 한다. Eucalyptus Aggregate Manager 는 사용자가 정의하여 제출한 RSpec 을 받게되고, RSpec 에 대한 파싱(Parsing)작업을 거친 후에 AM 에서는 제출된 RSpec 에 따라 클라우드 서버 상에 인스턴스를 할당하며, 할당된 인스턴스의 정보를 바탕으로 슬라이스와 인스턴스를 맵핑을 제공한다.

인스턴스 타입에 대한 상세 규격 형태로 RSpec 은 Eucalyptus Aggregate Manager 와 사용자 사이에서 중요한 상호작용 역할을 수행한다. RSpec 은 AM 에 의해서 사용되면서 XML 문서 형식으로 제공되고 있으며, 사용자에게 정보를 반환하거나 AM 에게 정보를 보낼 시에 RSpec 을 사용하게 된다. 특별한 네트워크를 위한 RSpec 의 포맷 형태는 네트워크를 정의하기 위하여

사용되며, 다양한 네트워크에 대한 정의를 할 수 있고, 그 결과 현재 PlanetLab, VINI, ProtoGENI 의 네트워크에 대한 RSpec 포맷을 지원 가능하다.

GENICloud 에서는 유칼립투스 클라우드를 연동하기 위하여 RSpec 을 정의 하였고, 사용자의 자원 요청들을 XML 포맷형태로 표현이 가능하다. 일반적으로 RSpec 에서는 사용자의 요청에 따라서 3 가지 타입의 정보를 가지고 있다. 첫 번째, RSpec 에서는 네트워크의 Capacity 와 Capability 가 충분한지에 대한 것을 사용자에게 알리는 역할을 한다. RSpec 은 사용자에게 어떤 리소스가 사용가능한지에 알린다. 두 번째, RSpec 은 사용자에게 리소스를 위하여 요구사항에 대한 표현을 허락한다. 사용자들이 RSpec 상에 사용자가 원하는 리소스에 대한 요구사항을 정의한다면, Slice Manager 와 AM 은 이에 대한 요청에 대하여 처리를 진행한다. 마지막으로, RSpec 에서는 사용자에게 이미 연계된 리소스에 대한 정보를 포함하고 있다.

3.2 Public Cloud (Computing, Storage) Resources

3.2.1 Amazon Web Services (AWS)

Amazon 에서 제공하는 Amazon Web Services(AWS)는 웹 호스팅 및 컴퓨팅 자원을 서비스하는 Elastic Compute Cloud(EC2), 스토리지 서비스인 Simple Storage Service(S3), Queue 서비스를 위한 Simple Queue Service(SQS), 데이터베이스 서비스를 위한 SimpleDB 와 Relational Database System(RDS)등 다양한 서비스를 제공하고 있다. 이는 모두 초기 서비스인 AWS 를 기반으로 운영하고 있다. 개발자들은 AWS 웹 페이지와 Simple Object Access Protocol(SOAP), Representational State Transfer(REST)와 같은 API 를 제공하여 클라우드 서비스를 여러 언어에 맞춰 연동 및 사용할 수 있는 편의를 제공한다[43].

AWS 에서는 Compute 자원을 위한 서비스로 EC2 를 제공한다[44]. EC2 는 Elastic Compute Cloud 의 약자이며, S3 와 함께 AWS 의 핵심 서비스 중의 하나이다. 유연성과 서버 운영에 대한 상당한 노하우와 안정성이 필요한 중요 서비스이다. 일반적인 서버 환경과 EC2 를 비교하자면,

서비스 운영을 위한 받게 되는 일반 서버와 비교할 수 있다. 추가되는 개념은 사용자의 요구사항에 맞는 컴퓨팅 자원(Computing Resource)과 운영체제를 개발자가 선택하도록 하고 그 비용을 청구 받도록 한다. 게다가 EC2 는 한 곳에 모여있지 않고 유럽, 미국 동부, 미국 서부, 싱가폴, 일본 등 5 개 지역(Region)으로 나누어서 관리되고 있다. Region 에는 다시 Zone 으로 분류되어 관리되고 있다. EC2 서비스를 시작하기 위해서는 어떤 운영체제를 사용할지를 먼저 결정한다. 이를 Amazon Machine Image(AMI)라고 불리는데, EC2 에 동작할 수 있는 다양한 운영체제를 아마존 AWS 에서 선택할 수 있다. Amazon 과 파트너 사의 도움으로 만들어진 윈도우 서버, 오픈 솔라리스, 리눅스 배포판 등 다양하게 지원되고 있으며, 필요하다면 개발자가 직접 만든 운영체제와 애플리케이션을 부팅 이미지로 만들고 이를 S3 에 올려 AMI 와 연동하여 설치할 수 있도록 되어 있다. Apache HTTP 서버, 데이터 베이스, 개발환경, Hadoop 등 다양한 애플리케이션을 운영체제와 함께 설치할 수 있다. 이러한 EC2 에서 AMI 는 여러 개의 CPU 를 가지고 있는 중형서버에서 여러 개의 서로 다른 OS 를 설치하여 운영할 수 있는 가상화(Virtualization) 기술인 Xen 기반으로 되어 있다. Xen 은 MS 의 Hyper-V, SpringSource 을 인수한 VMWare 사의 가상화 솔루션과 함께 전세계적으로 사용하는 가상화 기술 중 하나이다. Xen 은 반가상화 (para-virtualization)기법을 사용하는데, 운영체제가 직접 하드웨어를 제어하지 않고 중간 레이어(Layer)인 '하이퍼바이저(Hypervisor)'에게 요청하고 하이퍼바이저는 하드웨어에 요청해서 응답 받는 수직 구조를 가지고 있다.

다음으로 AWS 에서 Storage 자원을 위한 대표적인 서비스로 S3 를 제공하고 있다[45]. 이러한 S3 는 대용량 Blob 데이터에 대한 저장을 위해서 디자인 되었다. 파일, 이미지, 동영상과 같은 큰 사이즈의 데이터를 저장하며, 저장될 수 있는 데이터의 수와 크기는 제한이 없고, 저장되는 데이터의 크기는 레코드당 1byte 에서 최대 5GB 를 지원한다. 또한, S3 는 Bucket(도메인 이름)을 생성하고, 파일을 올리는 구조로 되어 있다. HTTP, HTTPS, Torrent 에서 접근할 수 있는 API 인터페이스를 제공하고 있다.

AWS 에서는 Storage 자원을 위한 서비스로 인스턴스 상에서 볼륨을 제공할 수 있도록 하는 서비스인 Elastic Block Storage(EBS) 서비스를 제공한다[46]. EBS 는 EC2 인스턴스에 확장 할 수 있는 가상의 하드디스크(정확히 말하면 S3 보다는 작은 분산 스토리지)이다. 하나의 EC2 인스턴스에는 여러 개의 EBS 볼륨이 마운트 될 수 있으며, 하나의 볼륨 크기는 1GB~1TB 이다. 실제로 저장될 때는 S3 서비스를 이용해서 저장되는데, 흥미로운 점은 분산 파일 구조를 채택하기 때문에 IO 성능이 상당히 높은 편일 뿐만 아니라, EBS 는 부팅 파티션(Booting Partition)으로도 마운트가 가능하다. 또한 특정 시점에 EBS 의 이미지를 S3 에 저장하여 백업용도로 사용 가능하다

3.2.2 Google App Engine

Google App Engine 은 2008 년부터 제공된 클라우드 컴퓨팅 서비스로서 Google 인프라 상에서 사용자가 자신의 웹 서비스를 만들 수 있도록 되어 있다. 이를 위하여 Google 은 표 1 과 같은 다양한 API를 사용자에게 제공하고 있다[47].

표 1. Google App Engine API.

API	주요 기능
Python 런타임 API	응용프로그램이 실행되는 Python 환경 정보 (CGI, 샌드박스 기능, 응용 프로그램 캐싱, 로깅)
데이터 저장소 API	확장 가능한 데이터저장소 및 이를 효율적으로 사용하는 방법에 대한 정보
이미지 API	이미지 데이터 조작 서비스
메일 API	응용프로그램에서 이메일 전송
Memcache API	분산 메모리 캐시
URL 가져오기 API	응용프로그램에서 다른 인터넷 호스트에 액세스
사용자 API	사용자 응용프로그램을 Google 계정과 통합

또한, Google App Engine 은 그림 13 과 같이 Software Development Kit (SDK)가 제공되어 자신의 로컬 환경에서 소프트웨어의 개발과 시험을 할 수 있으며, 테스트가 완료되면 해당 서비스를 Google 로 업로드 할 수 있다. 일단 업로드 된 서비스는 관리자 인터페이스를 통하여

언제 어디서든 서비스 상태, 시스템 및 데이터 로그, 방문자 이력 등을 조회할 수 있다. 실행환경은 Python 에 기반하고 있으며, 추가적으로 Java 를 지원한다. 인프라 내부는 Google 이 확보한 서버 클러스터 상에, GFS 를 통한 분산 파일 시스템이 동작하고 있으며, 이 위에 Bigtable 분산 데이터 베이스를 구동하고 있다. 글로벌 메모리 캐쉬로는 Memcache 가 지원된다. 수많은 서비스들이 동일한 인프라를 공유하는 환경에서는 확장성 있는 데이터 저장소의 필요로 인하여 App Engine 은 기존 관계형 DB 대신 구글의 분산 데이터 저장소인 Bigtable 을 기본 저장소로 사용하여 확장성 문제를 해결하였다. 개발자는 SQL 인터페이스 대신 datastore API 와 GQL (Google Query Language)의 새로운 데이터 인터페이스를 사용 해야 한다.

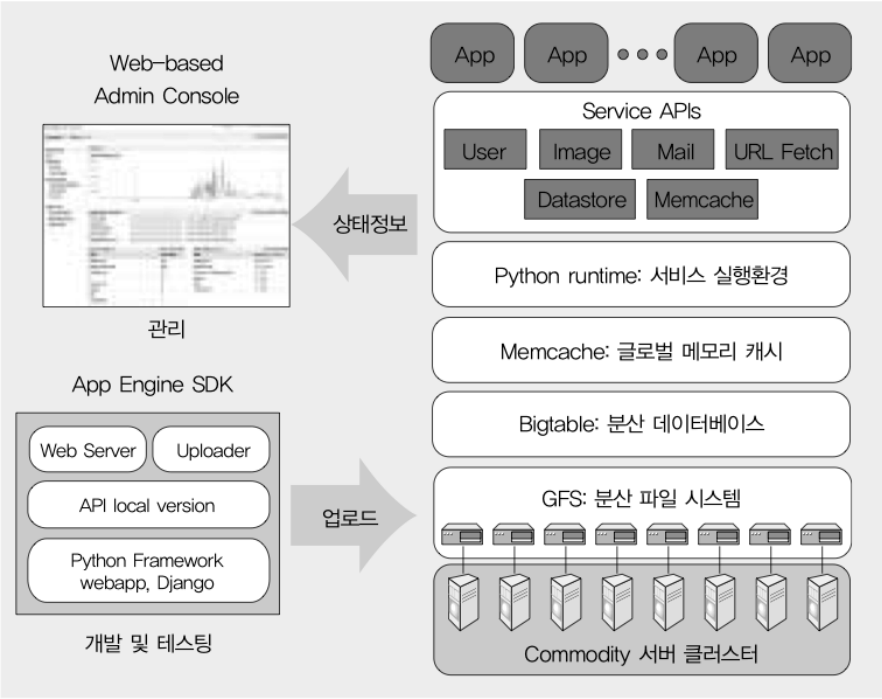

그림 13. Google App Engine 플랫폼 구조도[48].

3.2.3 Microsoft Azure

Microsoft 는 그림 14 와 같은 클라우드 컴퓨팅 환경을 구축하고 있으며, 사용자가 원하는 만큼의 컴퓨팅 프로세서와 저장장치를 필요한 시간만큼 사용하고, 사용량에 대하여 사용료를 내는

모델의 Azure 플랫폼을 제공하고 있다[49]. Azure 는 기존의 Microsoft 의 .NET 서버, Visual Studio 개발 환경 등을 재활용할 수 있기에 Microsoft 의 환경에 익숙한 사용자에게 장점이 있다.

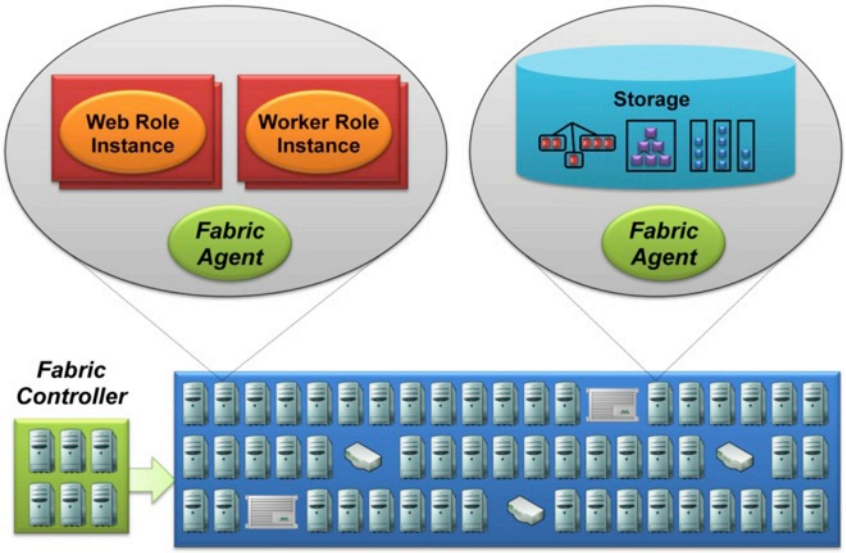

그림 14. Microsoft Azure 구조도.

Microsoft 가 확보한 서버들과 저장장치에 접근하기 위하여, 사용자는 본인이 필요한 컴퓨팅 프로세서와 저장장치에 대한 요구사항을 온라인 인터페이스를 통하여 Microsoft 에 요청하면, Fabric Controller 가 물리적인 컴퓨터와 저장장치를 묶어서 사용자가 요청한 논리적인 가상 서버 환경을 제공하여 준다. 이를 위하여 Fabric Controller 는 클라우드 자원인 컴퓨터와 저장장치에 위치한 Fabric Agent 와 통신하여 각 자원을 모니터링 한다. 사용자가 요청한 컴퓨팅 프로세서는 역할에 따라서 웹 어플리케이션 타입의 Web Role 과 .NET 기반의 어플리케이션 타입인 Worker Role 로 나뉘어 지원된다. 동일한 애플리케이션의 일부분인 전체 Web Role 인스턴스 간에 요청을 분산할 수 있도록 빌트인 로드밸런싱(load balancing)을 제공한다. 또한, Woker role 인스턴스는 외부로부터 요청을 직접 받아들 일 수가 없게 되어있다. 어떤 유형의 네트워크 연결도 허용되지 않고 IIS 가 가상 머신 안에서 구동되지도 않는다. 대신 Web role 인스턴스에서 입력이 들어와서 Windows Azure Strorage 에 Queue 형태로 저장된다. 작업 결과값이 Windows Azure Storage 에 저장되거나 외부로 보내지고 외부로 나가는 연결을 허용한다. 이러한 Azure 는 HTTP 를 통해 RESTful 스타일로 BLOBS, TABLES, QUEUES 를 지원한다. Windows Azure

Storage 는 다른 곳에서 구동되는 애플리케이션을 통해서도 접근이 가능하다 데이터를 노출시키는데 REST 방식을 사용한다. 모든 것은 URI 로 이름이 부여되어 있고 표준 HTTP 방식으로 처리된다.

다음으로 Azure 의 Storage 서비스인 SQL Azure 는 마이크로소프트의 SQL 서버 기술을 기반으로 제공되는 클라우드 서비스용 관계형 데이터베이스이다. SQL Azure 는 마이크로소프트의 SQL Server 가 제공하는 각종 Relational 한 기능들, 즉 리포팅, 분석, 데이터 동기화, Relational 쿼리 등을 클라우드 서비스로 제공하는 것이다. SQL Server 기반으로 구현되어 있는 SQL Azure 는 TSQL 기반으로 구축되어 있기 때문에 일반적인 Relational 한 데이터베이스 접근 방식을 그대로 제공하고, REST 및 SOAP 기반의 웹 서비스 인터페이스 방식도 제공하고 있다. 데이터는 REST 를 통해 접근될 수 있고(HTTP), 어떤 플랫폼상의 애플리케이션도 SOAP 를 통해 사용 가능하다. 플랫폼의 종류와 관계없이 애플리케이션은 SQL DATA Services 에 미리 정의된 사용자이름/패스워드 또는 Access Control 서비스 STS 에 의해 만들어진 TOKEN 에 의해 사용자를 인증 할 수 있다.

3.3 Private Cloud (Computing, Storage) Resources

3.3.1 Eucalyptus Cloud Platform

Eucalyptus(Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems)는 컴퓨팅 클러스터나 워크스테이션 팜을 사용하여 탄력적이고 유용한 클라우드 컴퓨팅을 구현할 수 있는 오픈소스 소프트웨어 인프라이다[35][36]. Eucalyptus 는 캘리포니아 산타바바라 대학 전산학과에서 연구 프로젝트로 시작했으며 Eucalyptus Inc.에서 최근에 상업화했다. Eucalyptus 는 계속해서 오픈 소스 프로젝트로 유지되고 개발된다. Eucalyptus Systems 에서는 오픈 소스 Eucalyptus 를 기반으로 하는 추가 제품을 제작하고 있으며 또한 지원 서비스를 제공한다. Eucalyptus 는 그림 15 와 같이 다섯 개의 기본 구성요소로 이루어지며 이

구성요소는 서로 협력하여 필수 클라우드 서비스를 제공한다. 이러한 구성요소는 SOAP 메시징을 WS-Security 와 함께 사용하여 서로 안전하게 통신한다.

이러한 Eucalyptus 에서 Compute 를 위한 구성요소로서 Cloud Controller(CLC), Cluster Controller(CC), Node Controller(NC)를 정의한다. 먼저, CLC 는 Eucalyptus 클라우드에서 전체 시스템 관리를 책임지는 기본 제어기 구성요소이다. Eucalyptus 클라우드에 액세스하려면 모든 사용자와 관리자는 CLC 를 거쳐야 한다. 모든 클라이언트는 SOAP 나 REST 기반의 API 를 사용해야 CLC 와 통신할 수 있다. CLC 는 요청을 수집하여 올바른 구성요소에 전달하고 구성요소에서 보내온 응답을 다시 클라이언트에 전송하는 기능을 담당한다. 이 기능은 Eucalyptus 클라우드의 일반적인 기능이다. 두 번째로, CC 구성요소는 Eucalyptus 내에서 전체 가상 인스턴스 네트워크를 관리하는 기능을 담당한다. 요청은 SOAP 나 REST 기반의 인터페이스를 통해 CC 에 전달된다. CC 는 시스템에서 실행하는 노드 제어기에 관한 모든 정보를 유지하며 인스턴스의 라이프사이클을 제어하는 기능을 담당한다. CC 는 사용 가능한 자원을 사용하여 가상 인스턴스를 시작하는 데 필요한 요청을 노드 제어기에 라우트한다. 세 번째 구성요소로 NC 에서는 호스트 운영 체제와 하이퍼바이저(현재는 Xen 이나 KVM 을 지원하며 VMWare 는 차후 지원할 예정임)를 제어한다. NC 인스턴스는 사용자가 CC 의 요청에 따라 인스턴스화된 실제 가상 인스턴스를 호스트할 각 시스템에서 실행해야 한다.

다음으로 Eucalyptus 에서 Storage 를 위한 구성요소로서 Walrus, Storage Controller(SC)를 정의한다. 먼저 구성요소는 Eucalyptus 에서 스토리지 서비스에 대한 액세스 권한을 관리한다. 요청은 SOAP 나 REST 기반의 인터페이스를 사용하여 Walrus 에 전달된다. 다음으로 SC 는 Eucalyptus 내부에 있는 스토리지 서비스로 Amazon 의 S3 인터페이스를 구현한다. SC 는 가상 머신 이미지를 저장하고 액세스하는 데 사용된다. VM 이미지는 공용이나 사설로 사용될 수 있으며 처음에는 압축되고 암호화된 양식으로 저장된다. 이 이미지는 노드에서 새로운 인스턴스를 시작해야 하고 해당 이미지에 대한 액세스 권한을 요청하는 경우에만 암호가 해독된다.

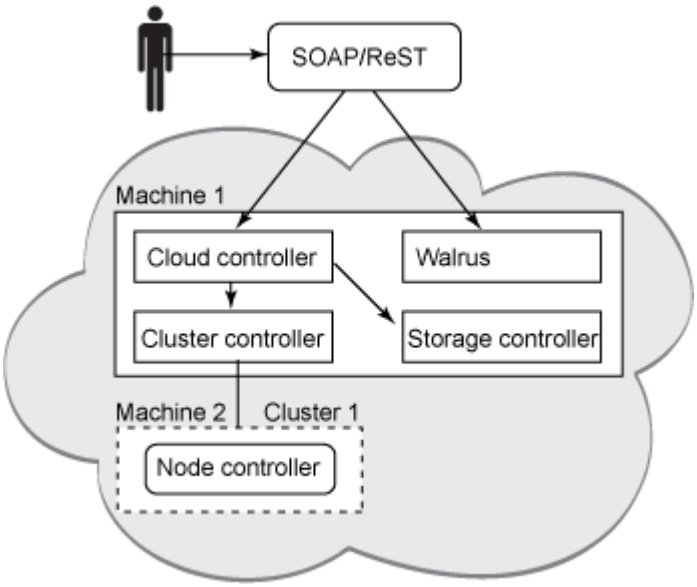

그림 15. Eucalyptus 시스템 구조도.

3.3.2 OpenStack Cloud Platform

OpenStack 은 Public 과 Private 클라우드를 위하여 국제적인 개발자들의 협력과 클라우드 컴퓨팅 기술자들에게 유비쿼터스 오픈 소스 클라우드 플랫폼을 생산한다. 이 프로젝트는 심플하게 구현하고, 확장 가능하도록 하며, 다양한 기능을 통하여 모든 타입의 클라우드를 위한 솔루션을 제공하는 것을 목표로 한다. Rackspace Hosting 과 NASA 에 의하여 설립되었으며, OpenStack 은 표준과 확장 가능한 오픈소스 클라우드 운영 시스템을 기반으로 개발자들의 국제적인 소프트웨어 커뮤니티로 성장하고 있다. OpenStack 을 위한 모든 코드는 Apache 2.0 라이센스 아래 무료로 이용할 수 있다.

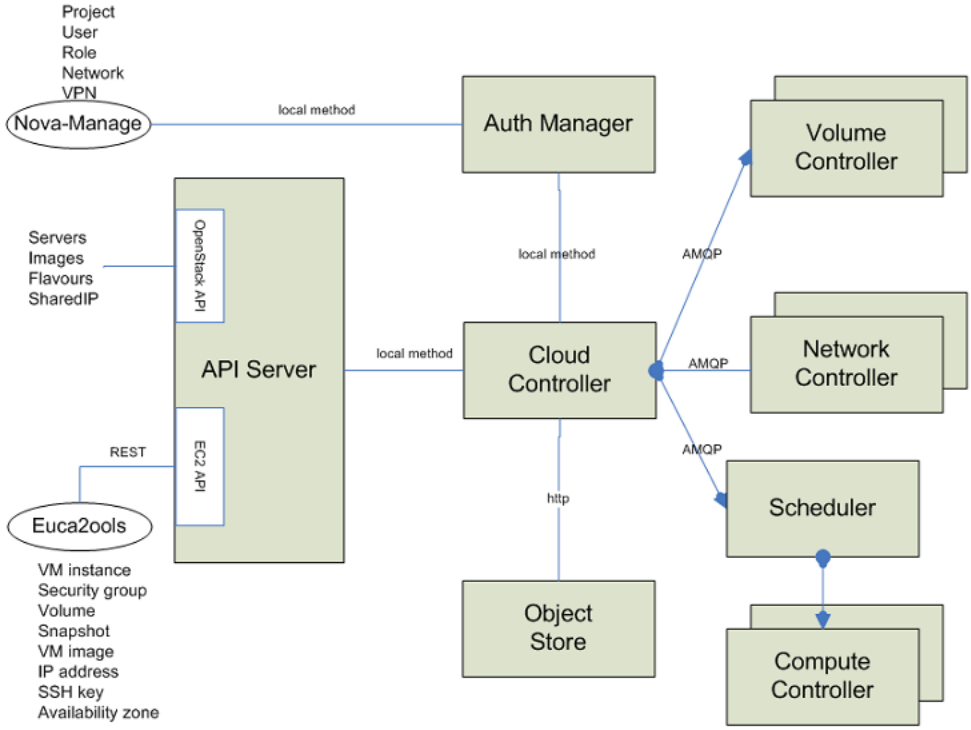

그림 16. OpenStack NOVA 시스템 구조도.

OpenStack Compute 라고 알려진 Nova 는 다수의 그룹이 사용할 수 있는 서버상의 가상 머신을 운용, 관리하기 위한 인프라 클라우드 관리 소프트웨어이다[50]. Nova 는 그림 16 과 같이 가상 머신의 운용과 관리를 담당하지만 자체적으로 가상화 소프트웨어를 포함하고 있지는 않으며, 단지 호스트 시스템상의 가상화 메카니즘을 구동하기 위한 드라이버를 정의하고 웹 API 를 통해 기능을 제공하고 있다. 이러한 Nova 는 7 개의 메인 컨포넌트로 구성되어 있다. Cloud Controller 는 모든 다른 컴포넌트들과의 인터렉션(Interaction)을 담당하며 클라우드의 글로벌 상태 정보를 나타낸다. API 서버는 Cloud Controller 의 프론트엔드(Front-end)로써 웹 서비스 형태로 동작한다. Compute Controller 는 클라우드 서버의 리소스를 제공하는 역할을 담당하며, Object Store 는 이미지나 템플릿 등의 저장을 위한 스토리지 서비스를 제공한다. Auth Manager 는 인증을 담당하며, Volume Controller 는 Compute 서버에 빠르고 끊김 없는 블록 레벨의 스토리지 서비스를 제공한다. Network Controller 는 Compute 서버간과 퍼블릭 네트워크 간의 상호 통신을 가능하게 하는 가상 네트워크를 제공한다. 마지막으로 Scheduler 는 호스트

서버에서 하나의 인스턴스를 생성하기 위해 가장 접합한 Compute Controller 를 선택하는 역할을 한다. Nova 는 shared-nothing, Message 베이스 구조로 설계되었다. Compute Controller, Volume Controller, Network Controller, Object store 와 같은 주요 컴포넌트들은 멀티서버상에서 동작된다. Cloud Controller 는 Object store 와 HTTP 통신을 하고, Scheduler, Network Controller, Volume Controller 와는 AMQP 통신을 한다. 통신 중 응답을 기다리는 동안 각각의 컴포넌트들이 블락되는 것을 피하기 위해, Nova 는 트리거 방식의 콜 백 기능을 가진 비동기 호출 방식을 사용한다

OpenStack Storage 라고 알려진 Swift 는 NASA 와 Racespace 가 공동으로 진행한 OpenStack Object Storage 프로젝트로써 대표적인 클라우드 스토리지 서비스 업체인 Rackspace 의 핵심 기술을 오픈 소스화하여 Amazon 의 S3 서비스와 비슷한 Object Storage Service 를 제공한다[51]. 이러한 Swift 가 제공하는 서비스로는 REST 기반의 S3 와 호환 가능한 API 제공하며, 인증과 Storage Account/Container/Object 를 위한 API 를 제공한다. Swift 의 구조는 5 가지 컴포넌트로 구성되어 있으며, 각 컴포넌트의 기능은 다음과 같다. Auth middle 에서는 사용자 생성 및 로그인, 인증 토큰을 생성하고 관리하며, Proxy Server 는 사용자의 요청에 따라 알맞은 서버에 연결하여 서비스 제공하도록 한다. 다음으로 Account Server 에서는 사용자 계정 관리 및 계정 별 컨테이너 조회하도록 하며, Container Server 는 사용자 계정의 컨테이너를 관리, 컨테이너가 가진 오브젝트를 조회한다. 마지막으로 Object Server 에서는 컨테이너 내의 오브젝트(Blob)를 관리한다.

3.4 FiRST Cloud

3.4.1 FiRSTCloud AM

FiRSTCloud Aggregate Manager (AM)은 미래인터넷 테스트베드와 오픈소스 오픈스택(OpenStack) 클라우드 컴퓨팅 플랫폼 간의 연동을 위한 GENI (Global Environment for Network Innovation) AM API (Application Programming Interface)를 기반으로 한다[52][53].

기존 미래인터넷 테스트베드인 PlanetLab 과 유칼립투스 클라우드 시스템과의 연동을 목표로 개발된 GENICloud 에서는 GENI AM API 보다 제한적인 API 의 기능를 제공한다. 따라서 FiRSTCloud 에서는 기존의 GENICloud 보다 많은 수의 API 의 기능을 제공하도록 하며, 국내의 미래인터넷 프로젝트로 수행중인 FiRST@PC 를 바탕으로 테스트베드와의 연동을 진행한다.

FiRSTCloud AM 은 오픈스택 클라우드와 미래인터넷 테스트베드 사이의 중재자 역할을 수행하며, 사용자가 슬라이스 상에서의 리소스 할당에 관하여 명령어를 통하여 질의 할 때에 오픈스택 클라우드 컴퓨팅 플랫폼을 통하여서 슬라이스 상에 인스턴스 생성을 관리한다. 이로 인하여 FiRSTCloud AM 은 슬라이스와 인스턴스 간의 맵핑을 유지하며, 이러한 슬라이스와 인스턴스 간의 맵핑을 하기 위해서는 오픈스택 인스턴스와 슬라이스 정보 간의 데이터베이스 테이블을 FiRSTCloud AM 상에서 표 2 와 같이 생성한다.

표 2. Slice 와 OpenStack Instance 의 DB 테이블.

Slice		
Name	Type	Key
id	INTEGER	PRIMARY KEY
Slice urn	TEXT	

Openstack instance		
Name	Type	Key
id	INTEGER	PRIMARY KEY
Instance id	TEXT	
Kernel id	TEXT	
Image id	TEXT	
Ramdisk id	TEXT	
Inst type	TEXT	
Key pair	TEXT	
Slice id	INT	REFERENCES slice(id)

다음으로 FiRSTCloud AM 이 제공하는 API 는 GENI AM API 에서 제공하는 총 7 개의 API 인 GetVersion(), ListReousrce(), CreateSliver(), DeleteSliver(), SliverStatus(), Shutdown(), RenewSliver() 중 RenewSliver()을 제외한 6 개의 API 를 기존의 API 를 바탕으로 FiRSTCloud 만의 추가적인 기능을 보완하였다. 이러한 FiRSTCloud 의 지원하는 기능에 대한 정리를 표 3 에서 나타낸다.

표 3. FiRSTCloud 지원 기능 및 GENICloud 와의 비교.

API	FiRST Cloud 지원 사항	GENI Cloud
GetVersion()	• 가짜 자원과 관련하여 의미 없이 만들어진 가짜 정보들 제거 • 클라우드 정보를 포함하는 버전 정보를 반환	X
ListReousrce()	• OpenStack 상에 Zone, Image, Keypair, Instance 정보 호출 • Zone, Image 정보를 파싱하는 코드 • 오픈스택 클라우드의 정보들을 RSpec 으로 생성	O
CreateSliver()	• Boto 라이브러리를 이용하여 오픈스택 클라우드 연결 • 슬라이스와 인스턴스를 맵핑하기 위한 DB 생성/호출 • 요청한 RSpec 의 정보에 맞게 인스턴스 생성	O
DeleteSliver()	• Boto 라이브러리를 이용하여 오픈스택 클라우드 연결 • 슬라이스와 인스턴스를 맵핑하기 위한 DB 생성/호출 • 요청된 Slice_urn 에 해당하는 인스턴스 삭제	X
SliverStatus()	• 슬라이스와 인스턴스 정보를 DB 에서 호출 • 반환되는 Face Resource 리소스 정보에 오픈스택 인스턴스 정보 추가 • 오픈스택 인스턴스 정보 반환시 각 인스턴스에 현재 상태에 기반한 Sliver 상태 결정	X
Shutdown()	• Boto 라이브러리를 이용하여 OpenStack 클라우드 연결 • 슬라이스와 인스턴스를 맵핑하기 위한 DB 생성/호출 • 요청된 Slice_urn 에 해당하는 인스턴스 삭제	X
RenewSliver()	• 기존의 함수와 변동사항 없으며 다른 함수들과 연동 테스트	X

또한, FiRSTCloud AM 에서는 그림 17 과 같이 사용자에 의해 제출되는 인스턴스의 구체적인 정보 및 리소스 정보와 관련된 설명을 위한 RSpec 을 정의한다. 이러한 RSpec 은 FiRST Cloud AM 에서 파싱된 이후에 각 항목에 맞춰 다른 목적으로 관리된다. 사용자에 의해 인스턴스를 요청할 시에 RSpec 은 사용자가 필요로 하는 클라우드 리소스를 정의하도록 한다. 논리적인 의미에서

사용자의 응용프로그램, 운영체재와 같고, 물리적인 의미에서 CPU, RAM, DISK 와 같은 하드웨어를 RSpec 으로 하여금 XML 포맷으로 FiRSTCloud AM 에게 전달한다. 전달 받은 FiRSTCloud AM 에서는 XML 을 파싱(Parsing)하여서 해당 되는 항목에 맞게 사용자에게 클라우드 자원을 제공하도록 한다.

```
<RSpec type="GCF">
<resource>
  <urn>urn:publicid:IDN+geni:gpo:gcf:resource:Cloud01</urn>
  <id>0</id>
  <type>OpenStack</type>
  <available>true</available>
  <cloud id="OpenStack">
    <ipv4>198.55.yy.xx</ipv4>
    <bundles>
      <bundle id="fc12" />
      <bundle id="fc11" />
    </bundles>
    <images>
        <image id="ami-88112222">
            <type>machine</type>
            <arch>x86_64</arch>
            <state>available</state>
            <location>images/ttylinux.img.manafest.xml</location>
        </image>
        <image id="aki-88112222">
            <type>kernel</type>
            <arch>x86_64</arch>
            <state>available</state>
            <location>images/vmlinux-2.6.16.33.manafest.xml</location>
        </image>
        <image id="ari-88112222">
            <type>ramdisk</type>
            <arch>x86_64</arch>
            <state>available</state>
            <location>images/ramdisk-2.6.16.33.manafest.xml</location>
        </image>
    </images>
```

그림 17. FiRSTCloud 를 위한 RSpec.

3.4.2 FiRSTCloud AM API

- FiRSTCloud AM 의 GetVersion() API: FiRSTCloud AM 은 그림 18 과 같이 GENI AM API 의 버전을 반환한다. 버전 정보는 FiRSTCloud 와 연동되어 있는 오픈스택 클라우드 버전을 포함한다.

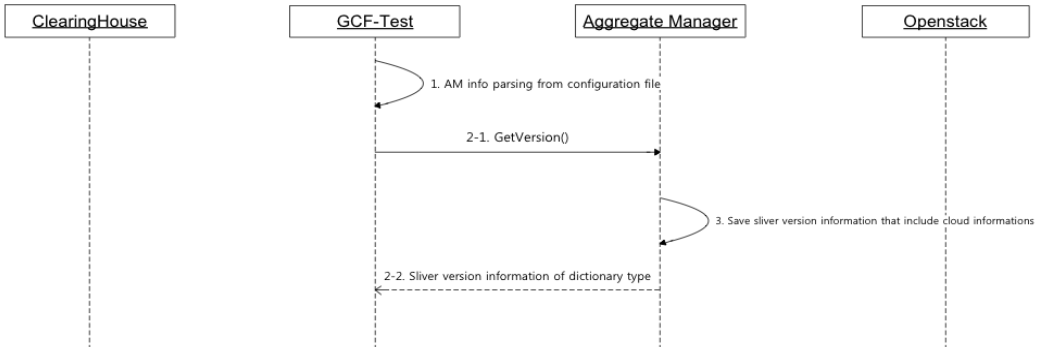

그림 18. GetVersion() API.

- FiRSTCloud AM 의 CreateSliver() API: FiRSTCloud AM 은 그림 19 와 같이 가장 먼저 슬라이스에 리소스를 할당한다. 또한 이 동작은 동작이 성공적으로 완료된 이후 리소스의 할당이 비동기적으로 진행된다. 이후 사용자는 SliverStatus() API 를 사용하여 리소스의 상태를 확인할 수 있다. 오픈스택 클라우드와 연결하기 위하여, 첫 번째로 EC2 와 호환 가능한 Boto 라이브러리를 사용하여 오픈스택의 환경 정보를 바탕으로 연결을 수행한다. 연결이 완료가 되면, 인스턴스와 슬라이스의 데이터베이스 정보를 새롭게 초기화한다. 새로운 인스턴스를 생성하기 위해 사용자로부터 요청된 RSpec 에서 이미지 정보, 가상머신의 종류 및 key-pair 정보를 오픈스택 클라우드로 파싱한다. 이렇게 인스턴스의 생성이 완료가 되면, 생성된 인스턴스의 ID 를 반환한다. 이후 인스턴스 리소스에 대한 새로운 RSpec 을 생성하고, 사용자에게 반환한다.

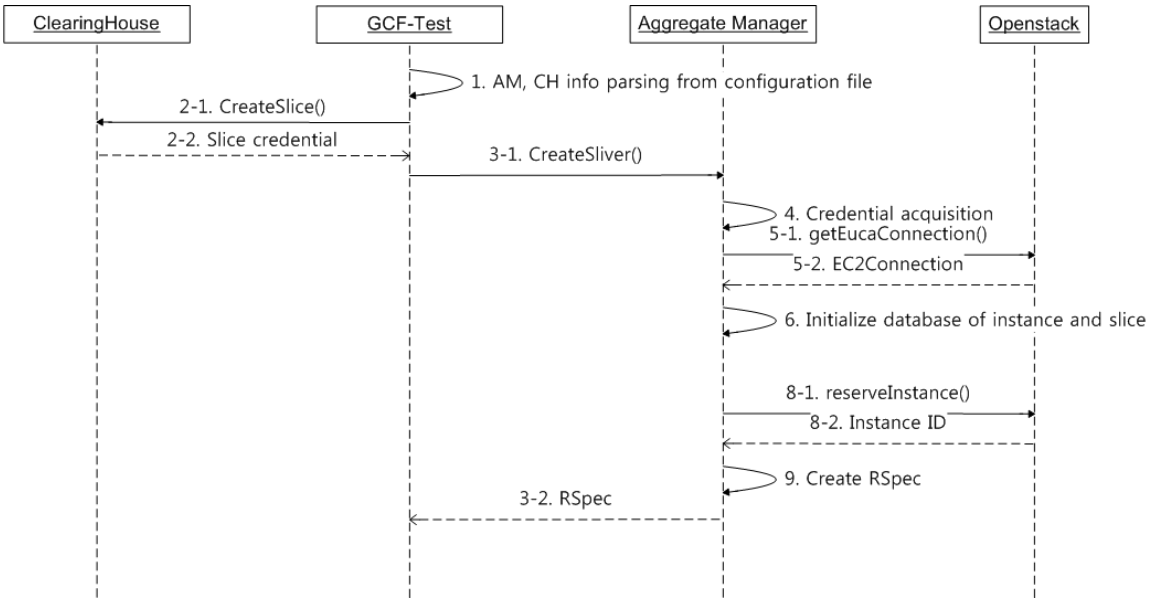

그림 19. CreateSliver() API.

- FiRSTCloud AM 의 ListResources() API: FiRSTCloud AM 은 그림 20 과 같이 가용한 클라우드 리소스에 대한 정보 또는 슬라이스에 할당된 리소스에 대한 정보를 반환해 준다. 오픈스택 클라우드와 연결하기 위하여, 첫 번째로 EC2 와 호환 가능한 Boto 라이브러리를 사용하여 오픈스택의 환경 정보를 바탕으로 연결을 수행한다. 이 후, 오픈스택 클라우드의 가용한 zone 의 정보와 등록된 이미지 정보, key-pair 정보 및 인스턴스 정보를 요청하고 AM 에서는 해당 정보들을 리스트의 형태로 값을 생성한다. 최종적으로 AM 에서 위의 리스트에 있는 정보를 바탕으로 오픈스택 클라우드의 정보들을 새로운 RSpec 을 생성함으로써 클라우드 리소스 정보를 반환한다.

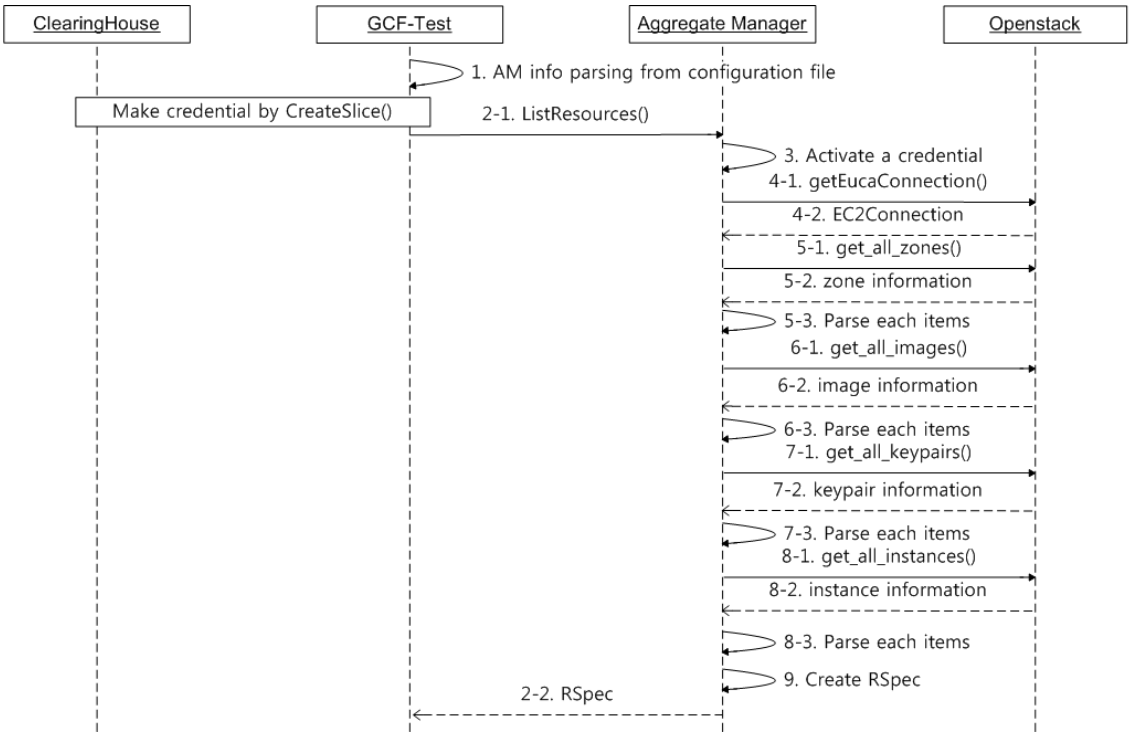

그림 20. ListResource() API.

- FiRSTCloud AM 의 DeleteSliver() API: FiRSTCloud AM 은 그림 21 과 같이 슬리버가 구동중이라면 이를 멈추고 할당된 리소스를 할당 해제함으로써 슬리버를 삭제한다. 해당 API 에서는 주어진 슬라이스 에 해당하는 대상에 대하여 앞서 언급한 것과 같이 오픈스택 상에서 인스턴스의 동작 중지와 자원 할당 해제를 수행한다. 이는 AM 이 데이터베이스에서 해당 슬라이스와 매핑되어 있는 오픈스택의 인스턴스 정보를 찾아 해당 인스턴스의 종료 명령을 내린다.

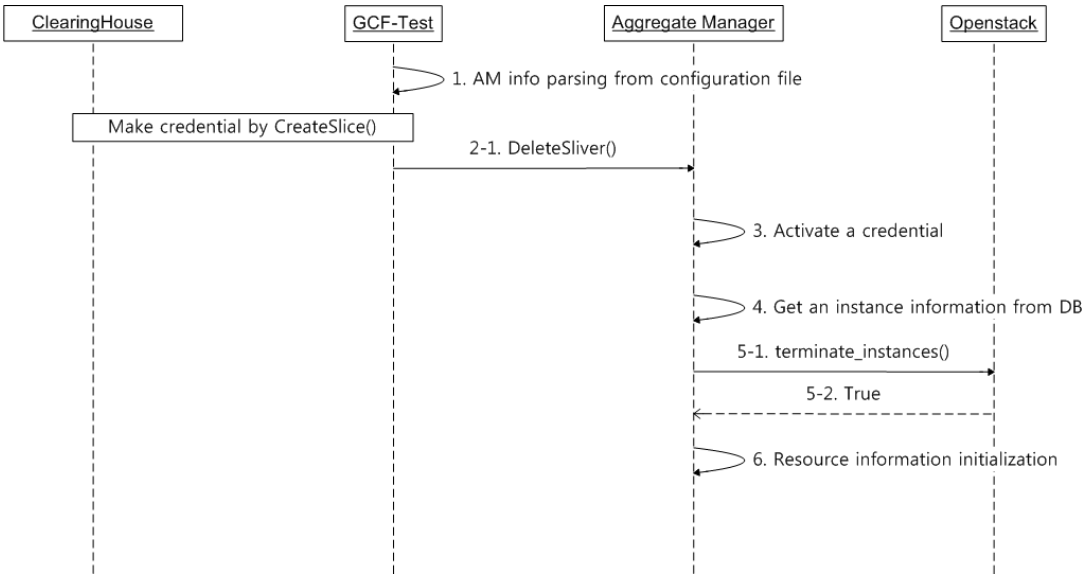

그림 21. DeleteSliver() API.

- FiRSTCloud AM 의 SliverStatus() API: FiRSTCloud AM 은 그림 22 와 같이 슬리버의 상태 정보를 얻는다. 먼저, 오픈스택 클라우드 인스턴스의 상태를 확인하기 위한 연결을 요청하고, 해당 슬라이스와 연결된 모든 인스턴스에 대한 정보를 받아온다. 이러한 인스턴스의 정보는 슬리버의 최종 상태로 간주되어 사용자에게 전달된다.

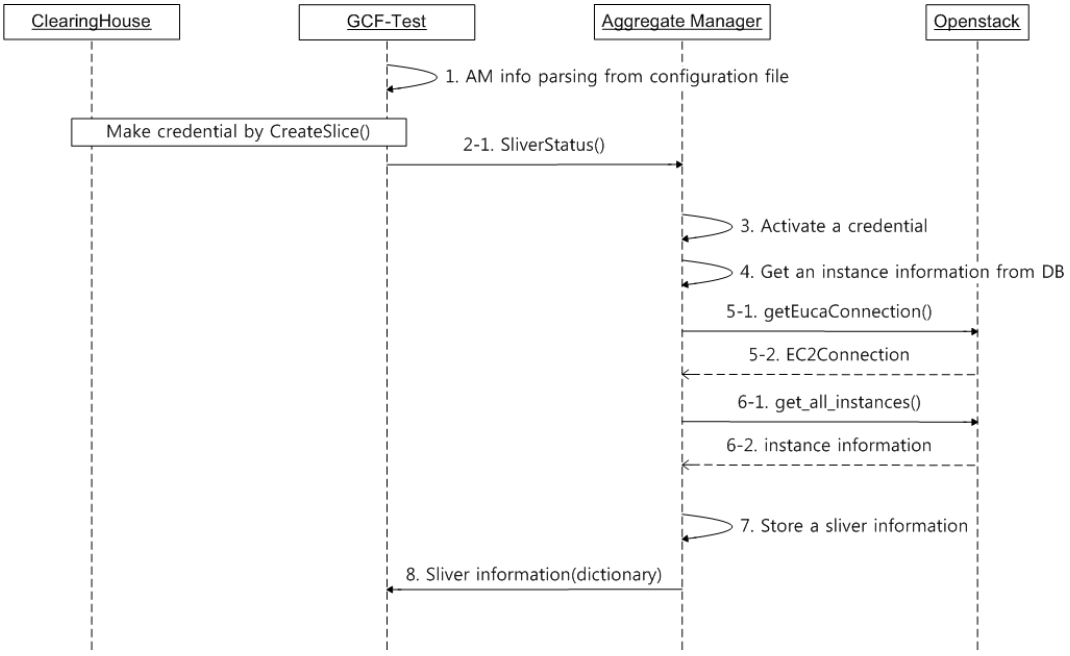

그림 22. SliverStatus() API.

- FiRSTCloud AM 의 Shutdown() API: FiRST Cloud AM 은 그림 23 과 같이 긴급한 상황에서의 슬리버에 대한 shutdown 을 수행한다. 이 동작은 관리상의 용도를 위하여 만들어진 것이다. 추가적으로, 이 API 는 슬라이스 및 인스턴스와 관련된 데이터베이스를 얻은 뒤에 해당 인스턴스에 대한 종료와 관리를 할 수 있다.

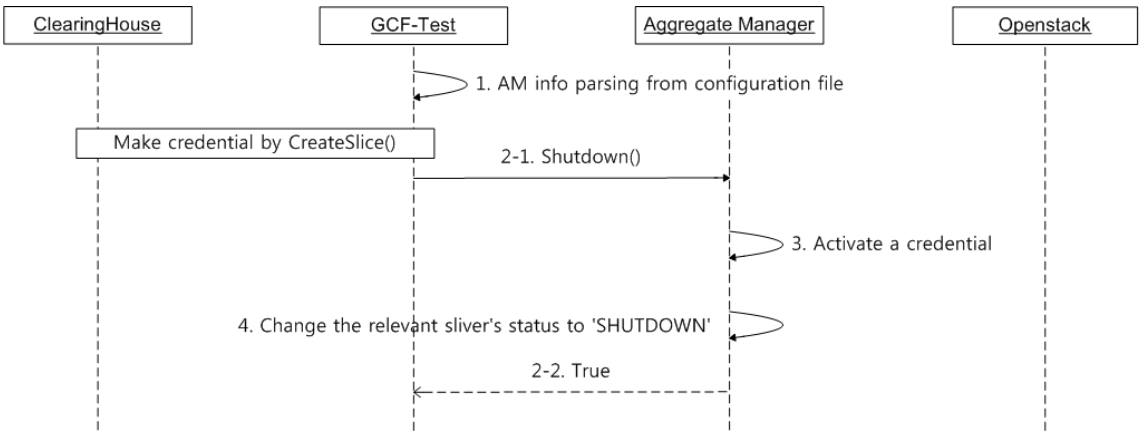

그림 23. Shutdown() API.

3.4.3 FiRSTCloud 활용 환경

FiRST Cloud 테스트베드는 기존의 Aggregate Manager 인 PlanetLab AM, ProtoGENI AM, OpenFlow AM 의 제어를 지원하는 GENI AM API 를 이용하여 사용자에 의해 제어된다. 그림 24 에서는 어떻게 API 를 활용하는지에 대하여 보여주고 있다. GENI 실험자들은 소프트웨어 사용자와의 상호작용을 하며, 사용자는 GENI clearinghouse 와의 통신을 통하여서 슬라이스를 생성하고, 슬라이스 credential 을 받으며, 어떠한 aggregate 가 사용가능한지에 대한 정보를 전달 받는다. 슬라이스가 생성이 되면, 사용자 소프트웨어는 GENI AM API 를 사용하여 실험 자들과의 상호작용을 한다. 따라서, 클라우드 컴퓨팅을 사용하기 위한 사용자들은 기존의 AM 들과 같은 방법으로 FiRST Cloud AM 에게 접근할 수 있다.

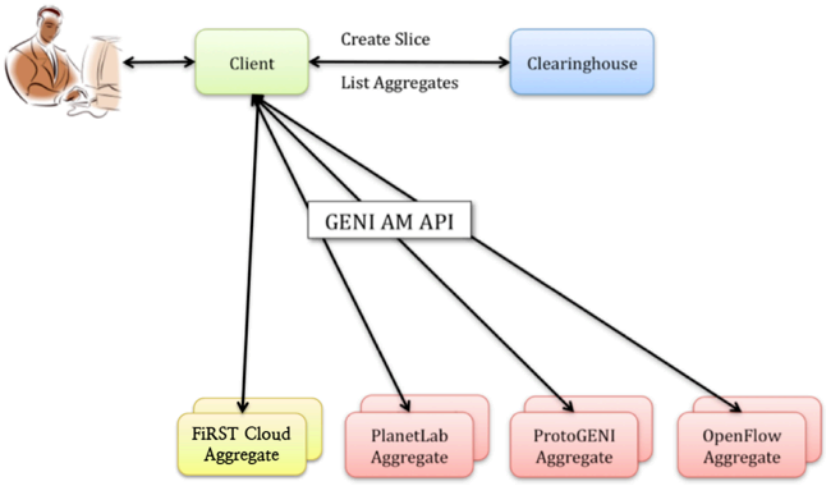

그림 24. FiRSTCloud 실험환경.

OpenStack 클라우드 플랫폼은 FiRST 클라우드 컴퓨팅 환경에 사용된다. 그림 25 와 같이 OpenStack 서버는 Front-end 서버와 Node 서버로 구성이 되며, Private Switching Hub 를 사용하여 클라우드 환경을 관리할 수 있도록 한다.

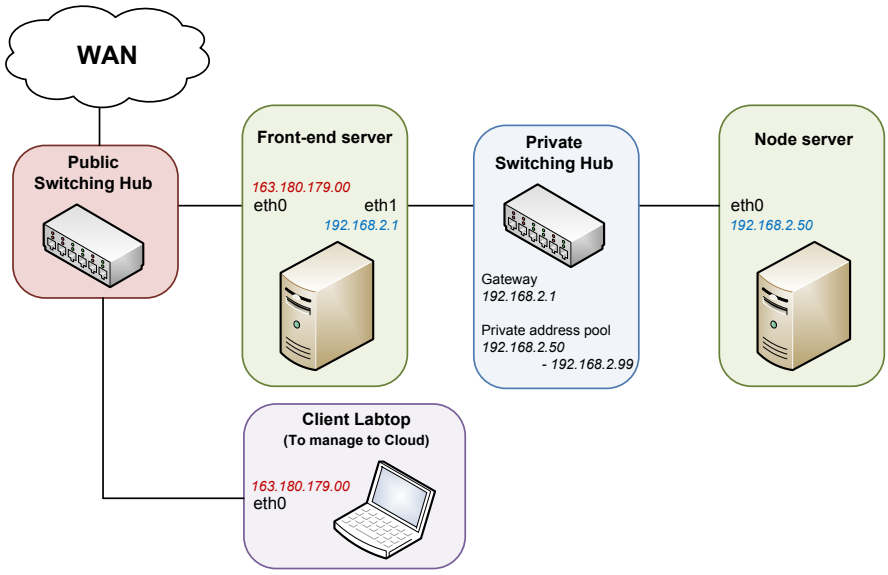

그림 25. OpenStack 클라우드 테스트베드 구조.

FiRSTCloud 는 미래인터넷 테스트베드와 오픈스택 클라우드 컴퓨팅 플랫폼 간의 연동을 위한 GENI AM API 를 기반으로 개발을 하였다. FiRSTCloud 에서는 기존의 GENICloud 보다 많은 수의

API 의 기능을 제공하도록 하며, 국내의 미래인터넷 프로젝트로 수행중인 FiRST@PC 를 바탕으로 테스트베드와의 연동을 진행할 수 있도록 실험을 하였다. 해당 FiRSTCloud 의 실험을 위하여서 GENI Control Framework(GCF) tool 을 이용하여 실험을 진행하였고, 실험 결과 개발한 API 에 대하여 모든 기능이 오픈스택 클라우드 상에서 정상적으로 동작하였다.

4 Networking-centric Resources for Experimental Facilities

미래인터넷 테스트베드는 연구자들이 창의적인 서비스 실현을 실증할 수 있도록 연결용 네트워크와 이에 연동된 컴퓨팅 장비들, 그리고 이를 지원하는 각종 실험용 소프트웨어 일체를 지칭하는 일종의 가상적인 실험실이어야 한다. 활용되는 컴퓨팅/네트워킹 자원들이 여러 네트워크 계층에서 프로그램이 가능해야 하며, 다수의 실험자들이 자신들의 실험을 동시에 수행하도록 자원의 가상화도 필요하다. 하지만 네트워킹 자원들은 프로그램화/가상화 부분에서 매우 미진한 상황이며, 네트워킹 자원들이 사용자가 원하는 대로 활용하려면 새로운 패러다임에 따라서 네트워킹 장비를 개발하는 혁신이 필요하다는 주장이 강하게 대두되고 있다. 4 장에서는 이러한 요구 사항을 만족시키기 위한 여러 가지 기술과 방법론을 살펴본다.

4.1 An Overview on Software-Defined Networking

4.1.1 OpenFlow/SDN 을 통한 네트워킹 분야의 혁신이란?

미래에 인터넷을 통해서 사용자가 원하는 서비스를 제공하기 용이한 새로운 차원의 인프라로 나가기 위한 방법들의 하나로 네트워킹 장비 개발과 운용 분야에 혁신 (innovation)이 필요하다고 한다. 전세계 사람들을 항상 연결해주고 있는 인터넷을 떠 받치고 있는 근간인 각종 라우터와 스위치 장비들이 계속적으로 고도화 되면서 지속적으로 늘어나는 인터넷 트래픽을 비교적 무난하게 처리하고 있는 실정인데 말이다. 네트워킹 분야의 혁신은 사용자가 원하는 새로운 네트워킹 아이디어가 네트워킹 장비에 빠르게 적용되고 또한 손쉽게 운용될 수 있는 환경이 조성되어야 가능할 것이다. 그런데 지금의 네트워킹 장비 개발과 운용은 "프로토콜" 중심의 고전적인 패러다임이 요구하는 지나치게 많은 (또한 대부분 실제로 사용되지 않는) 연동성과 각종 호환성 검증이라는 장애물을 넘어서야만 사용자의 선택을 받을 수 있는 상황이다.

장비의 사용자와 개발자 간의 다음 대화를 일례로 살펴보자. "MPLS-VPN 기능이 지원되나요? *네.* 그런데 지원이 제대로 되는지 검증 자료는 있으신가요? *네, 현재 검증이 진행되고 있습니다.* 그러면 IPv6 지원에는 문제가 없으신가요? *네, 지원하고 있고 검증 자료도 가지고 있습니다.* 그런가요? 그럼 저희가 보유하고 있는 C 사의 장비와의 호환성은 어떠한가요? *네, X 사에서 연동해서 사용 중인데 큰 문제가 없습니다. …*" 이와 같이 야심 차게 개발한 새로운 네트워크 장비를 고객에게 팔기 위해서는 위와 같은 질문들 모두에 대한 만족스러운 대답들을 제시해야 비로소 구매를 위한 검토 대상에 포함되는 영광을 누리게 된다. 이와 같이 네트워크 장비 개발과 이의 시장 확보라는 과제를 수행함에 있어서 넘어야 할 산은 너무나 많은 것이 현실이고, 이는 기존에 개발, 검증, 적용/보완의 전반에 걸쳐 생태계를 구축해 놓은 장비 회사들만의 독과점을 조장하게 된다. 새로운 장비의 개발과 시장 적용을 위해서 노력을 경주했던 네트워크 장비 업체들을 제외한 대다수 당사자에게 소위 "검증된" 제품에 대한 구매는 편안한 선택일 수 밖에 없기 때문이다.

앞에 논의한 비합리성이 존재할 수 밖에 없는 이유를 기존의 네트워킹 패러다임이 가지는 근본적인 문제점으로 인식하고, 이를 소프트웨어를 정의한다는 시각으로 네트워킹을 새롭게 바라보면서 소프트웨어 정의 네트워킹 (Software Defined Networking: SDN)으로 불리는 새로운 네트워킹 패러다임을 함께 개척하자는 목소리가 나타나고 있다. 즉 복잡성을 즐기는 엔지니어들이 설계하는 네트워킹을 벗어나서 단순함만이 통하는 사용자들이 손쉽게 이해하고 운용할 수 있는 새로운 네트워킹 패러다임으로 혁신해 보자는 것이다.

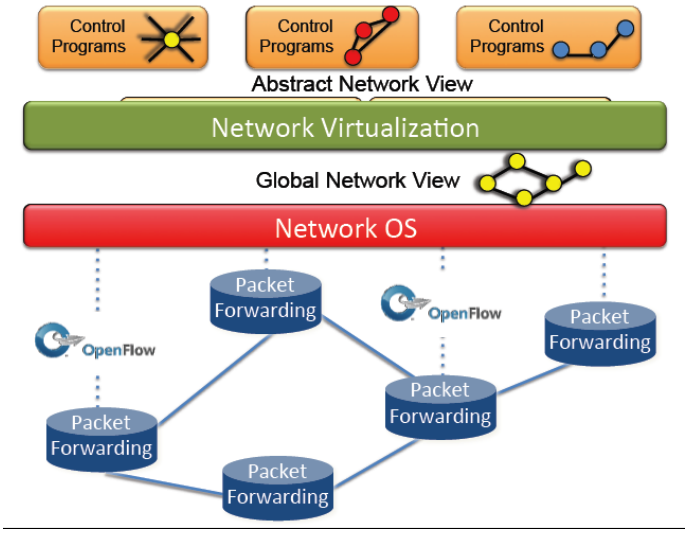

그림 26. Software-Defined Networking and OpenFlow [from Nick McKweon's ONS Presentation Slide].

좀 더 구체적으로 살펴보자면, 지금의 네트워킹 패러다임은 네트워크 장비 하나 하나가 여러 계층에 걸친 수 많은 프로토콜을 지원해서 동작하도록 만들고, 이의 성능을 개별적으로 최적화한 후에, 이들 각각에 대한 검증 과정을 거치는 개별 장비 중심으로 형성되어 있다. 따라서 이를 벗어나기 위해서는 개별 장비들에게는 일반 상용 PC 와 같은 핵심 자원 기능들에 대한 검증된 성능만을 요구하고, 핵심적인 프로토콜들을 제외한 수 많은 프로토콜들 자체가 불필요하도록 상황을 바꾸는 발상의 전환이 요구된다. 이를 위해서 SDN 을 채택하면 그림 2 에 제시한 바와 같이 네트워크 OS (Network Operating System, 즉 네트워크를 위한 Windows 또는 Linux 로 생각하면)를 간단하게 프로그래밍을 제어하기 위한 오픈플로우(OpenFlow) 라고 불리는 인터페이스 부분만 표준적으로 합의하여 만들어서 이를 통해서 개별 장비들은 네트워크 OS 와의 포팅에 전념하게 한다. 그리고 수 많은 프로토콜에 대한 대응은 논리적으로 중앙집중된 콘트롤러를 통해서 "합의된, 즉 표준화된 프로토콜은 필요가 없는 상태로" 해당 환경에 부합되는 적절한 알고리듬의 직접적인 구현으로 손쉽게 해결하도록 하면 된다. 예를 들면 교과서적인 네트워크 라우팅의 구현 사례인 Dijkstra 의 알고리듬이 장비들의 상태 맵(map)을 파악하고 있는 중앙집중형 콘트롤러에 동작하는 원리는 4 페이지 문서면 모두 설명할 수 있다. 그런데 이를 분산 시스템에서 네트워크 맵을 최신으로 일관되게 분산 방식으로 업데이트하면서 구현하려면

101 페이지에 걸친 문서에 근거해서 구현해야 하면, 실제 장비에 적용되는 표준화된 OSPF 라우팅을 위한 RFC 2328 문서는 245 페이지에 걸쳐서 프로토콜을 정리하고 있다. 즉 4 페이지의 간단한 알고리듬 구현을 50 배나 복잡하게 만들어서 구현하고 또한 이에 부합하는 지, 그리고 유사 장비들과 호환이 되는 지를 일일히 검증하는 넌센스가 벌어지고 있는 것이다. 따라서 SDN 을 기반으로 한 새로운 패러다임을 만들어 내어서 아래와 같은 혜택들을 누리는 네트워크 장치 생태계를 만드는 것이 필요하다고 예기되고 있다.

- 포워딩(forwarding) 추상화에 따라서 오픈플로우 표준에 의해 검증된 하드웨어를 이용하고, 또한 이에 연동하여 소프트웨어 기반으로 제공되는 네트워킹 특성이 요구되는 모든 절차들에 대해서 각각의 절차마다 충분하게 증빙되어 있는 견실한 네트워킹을 실현할 수 있는 토대를 구축할 수 있다.
- 장비 사용자들이 자신의 필요에 따라 네트워킹을 유연하게 구성(customize)하고 불필요한 구성요소들은 과감하게 제거하면서 자신만을 위해서 가상화된 네트워크를 생성하기 쉽도록 지원한다.
- 하드웨어 추상화(abstraction)에 따라 확장성을 고려한 상태에서 공통화된 (즉 commodity 형식으로) 하드웨어를 구입하고, 또한 소프트웨어도 분리해서 구입하도록 하여 기존의 폐쇄적인 네트워크 장치 공급자 체인을 벗어나서 자체 개발, 외주 개발, 오픈 소스 형식을 모두 포함하는 다변화된 공급자 체인으로 체질 개선을 유도할 수 있다.
- 소프트웨어를 개발하는 속도로 혁신이 일어나도록 하고, 표준은 구현된 소프트웨어의 확산을 위해 뒤따라가는 방식을 취하고, 소프트웨어 적인 개방성에 근간하여 기술의 공유 협력을 쉽도록 함으로써 혁신의 속도를 가속하도록 지원한다.

이러한 실용주의적인 SDN 을 함께 추진하기 위해 ONF (Open Networking Foundation)이 2011 년 상반기에 Deutsche Telekom, Facebook, Google, Microsoft, Verizon, Yahoo 를 이사회 멤버로 하여 정식으로 출범했다. 즉 ONF 라는 회원사 공동체를 중심으로 SDN 기술을 개발하고

주요 표준도 구현에 기반하여 빠르게 제정해서 확산을 도모해 보자는 것이다. 또한 지난 달(2011 년 10 월 17-19)에는 OpenFlow 를 최초로 제안했던 미국 Stanford 대학에서 ONF 주관으로 첫 번째 Open Networking Summit 행사가 개최되었다. 주최측이 예상을 두 배가 넘는 폭발적인 관심으로 제한된 400 여명의 인원들만 참가할 수 있었으며, 다양한 영역들에서 온 참가자들이 3 일에 걸쳐서 관련 데모와 발표를 공유하면서 많은 논의를 나눴다. 특히 SDN 이 가지는 융합적인 성격을 반영하여, 추상화개념부터 시작해서 분산 시스템, 데이터베이스, 소프트웨어 공학 등 다양한 컴퓨터사이언스(Computer Science)의 개념을 접목해 보려는 참가자들이 다수 눈에 뛰었다. 이번 행사의 keynote 발표에서는 기존의 프로토콜 중심의 네트워크 구축 방법을 버리고 오픈플로우 인터페이스를 통해서 소프트웨어 기반으로 새로운 네트워킹으로 혁신하는 노력을 함께 시작해 보자는 메시지가 전달되었던 행사였다. 또한 현재 시점의 오픈플로우 지원 장비 현황과 이에 기반한 SDN 의 개념을 보여주는 12 개의 데모가 3 차례에 걸쳐서 진행되었으며, 한국에서도 FIF 테스트베드 WG 에서 FIRST/KOREN/KREONET 공동으로 준비한 OF@Korea 데모[1]를 진행하였다.

또한 실제 네트워크 인프라에 SDN 을 어떻게 적용해서 개선할 수 있을 것인가 대해서는, 지금 막 태동하는 노력이므로 현실을 반영하여 데이터센터, 엔터프라이즈, 네트워크 서비스 제공자의 순서로 점차적으로 적용이 확산되는 것에 대해 공감대를 형성되고 있다. 특히 오픈플로우 기반 SDN 이, 다양한 장치들과 가상 머신들이 병존하면서 네트워크 토폴로지를 묶고 있는 데이터센터, 개인화된 클라우드, 그리고 캠퍼스랜에서 유용하다는 주장이 많았다. SDN 이 클라우드/데이터센터에서 요구되는 로드밸런싱, 플로우제어, 가상화된 네트워킹을 지원하는 손쉽게 운용함에 효과적이라는 판단인 것이다. 데이터센터나 엔터프라이즈 네트워크 전반에 걸쳐서는 네트워크 관리 측면의 혜택이 부각되었으며, 앞에서 설명한 4 페이지 대 수백 페이지 비유에서 설명된 바와 같이, 현재 장비가 가진 기능들의 극히 일부만 보수적으로 운영하는

[1] http://opennetsummit.org/demonstrations.html

분위기가 SDN 도입으로 보다 풍부한 기능을 활용하면서 동적으로 관리될 것으로 전망되었다. 즉 새로운 SDN 기반 패러다임에서는 다양하고 독창적인 구성을 지닌 라우터와 스위치들로 구성되지만 제공하는 네트워킹에 대한 통합된 제어가 일사불란하게 가능하도록 만드는 "네트워크 OS" 같은 역할이 핵심적이라는 것이다.

그러므로 한국에서도 이러한 비전에 대한 대응 전략에 대해서 곰곰이 생각해 볼 시점이 아닌가 생각된다. 한국의 국가적 장점인 초고속 인터넷 보급과 확산에 있어서 세계를 선도했던 경험과 이를 지원했던 네트워킹 하드웨어에 대한 신속한 적응력에도 불구하고, 지속적으로 유지하고 관리하면서 구색을 갖추어야만 하는 소프트웨어 적인 부분의 부진으로 인해서 돌파구를 잃고 있는 국내의 네트워크 산업에게도 새로운 기회를 부여할 수 있는 다가오는 큰 물결(Big Wave)과 같은 역할을 할 수도 있기 때문이다. 스마트 폰 개발에 있어서 시기를 놓쳐서 힘겹게 쫓아가면서 깨달은 소프트웨어 마인드를 "스마트 네트워크" 장비의 개발에는 선제적으로 전용할 수 있을지도 모르기 때문이다.

4.1.2 OpenFlow 인터페이스 및 이에 기반한 SDN 구현 현황

OpenFlow 의 개발을 주도하고 있는 스탠포드 대학에서는 OpenFlow 스위치의 요구사항을 담은 스펙(specification) 문서를 공개하여, 누구나 다른 OpenFlow 프로토콜 기반의 플로우 기반 네트워킹 능력을 지닌 장치를 개발할 수 있도록 하였다. 그 결과, 소프트웨어 형태의 OpenFlow 스위치뿐만 아니라, Cisco, HP, IBM 과 같은 많은 상용 네트워크 장비 벤더들도 기존 장비의 firmware 업데이트 수준에서 OpenFlow 스위치를 지원하고 있다. OpenFlow 스위치의 스펙 문서는 2010 년 1 월에 1.0 버전의 스펙[26]이 참조구현(reference implementation)과 함께 공개되었고, 그 후 2011 년 3 월에 1.1 버전의 스펙[27]을 공개하였다. 하지만, 1.1 버전에 대응되는 장치들은 매우 제한된 상태이므로 현 시점까지는 1.0 버전이 주로 활용되고 있는 상황이다. 본 문서에서는 OpenFlow 1.0 과 1.1 버전의 주요 기능들을 살펴보고, 그에 기반한 SDN 구현 현황을 소개한다.

4.1.2.1 OpenFlow 1.0/1.1

OpenFlow 1.0 스펙 문서는 수 년간의 작업을 통해 안정화된 OpenFlow 스위치의 모습을 담고 있다. OpenFlow 스위치로 동작하기 위한 기본적인 내용과 함께, 플로우 테이블의 형태와 네트워크 컨트롤러와 연결을 위한 인터페이스인 secure channel 을 구성하기 위한 메시지 포맷을 정의하고 있다. 플로우 테이블의 각 엔트리는 플로우와 매칭할 패킷의 헤더 필드와 해당 플로우에 대한 스위치의 행동 (action) 목록, 그리고 일치하는 패킷에 대한 카운터를 포함한다.

Secure channel 을 구성하기 위해 controller-to-switch, asynchronous, symmetric 의 세 가지 메시지 타입을 정의하고 있다. Controller-to-switch 메시지 타입은 네트워크 컨트롤러가 스위치에게 보내는 메시지로, 스위치의 능력에 대한 정보를 요청하는 features, 플로우 테이블 엔트리를 추가/삭제/수정하기 위한 modify-state 메시지, 그리고 스위치의 상태 정보를 수집하기 위한 read-state 등을 포함하고 있다. 또한, 초기적인 형태의 QoS 제공을 위한 슬라이싱에 대한 내용을 담고 있다. Asynchronous 메시지 타입은 스위치가 상태 변화가 생겼을 때 이를 네트워크 컨트롤러에게 알리기 위해 사용하는 것으로, Packet-in, Flow-removed, Port-status, Error 메시지 등이 정의되어 있다. Packet-in 메시지는 플로우 테이블에 매칭되는 엔트리가 없을 시에 이를 네트워크 컨트롤러에게 전달하기 위해 이용되고, flow-remove 메시지는 정해진 시간 동안 패킷이 입력되지 않아 플로우 엔트리가 지워질 경우에 이용된다. Symmetric 메시지 타입은 네트워크 컨트롤러나 스위치 누구든지 요청할 수 있는 것으로, Hello, Echo, Vendor 메시지가 있다. 이 중 Vendor 메시지는 각 스위치 벤더들이 향후에 추가적인 기능을 OpenFlow 메시지 타입 공간 안에서 구현할 수 있도록 표준화된 방법을 준비해 둔 것이다.

OpenFlow 1.1 의 스펙 문서는 2011 년 3 월에 공개되었는데 OpenFlow 1.0 과의 호환성을 유지하면서 파이프라인 프로세싱과 그룹의 개념을 추가해 다중 경로 (multi-path) 지원이나, 개별 플로우의 프로세싱을 보다 다양하게 제어하기 위한 지원이 확대되었다.

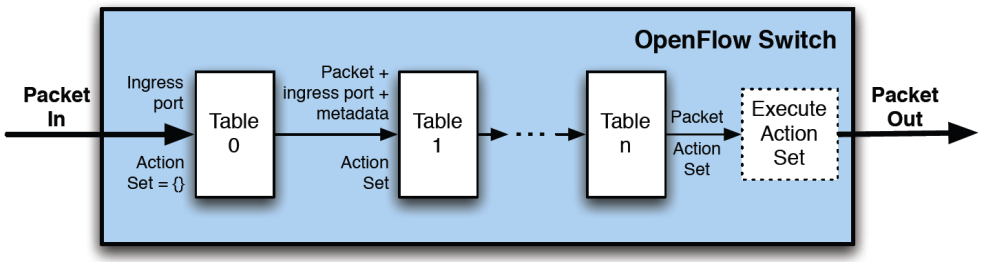

그림 27. 파이프라인을 통한 플로우 처리.

먼저 가장 큰 변화는 플로우에 대한 파이프라인 프로세싱을 추가한 것으로, 1.0 버전에서는 하나의 플로우 테이블만을 이용하였으나, 그림 3 과 같이 다수의 플로우 테이블을 이용하도록 변경되었다. 파이프라인 프로세싱을 위해 먼저 각 패킷별 행동 정보를 Action Set 을 통해 정의한다. Action Set 은 패킷이 각 플로우 테이블에 의해 처리되면서 추가되는 행동의 집합으로 패킷이 프로세싱 파이프라인을 빠져 나왔을 때 수행될 액션들을 나타낸다. 플로우 테이블은 1.0 버전과 같이 여러 플로우 엔트리를 가지고 있지만, 플로우 테이블 엔트리가 행동 목록을 직접 표현하는 대신 instruction 의 목록을 담고 있다는 점이 달라졌다. Instruction 은 각 패킷에 연결된 Action Set 을 수정하거나 파이프라인 프로세싱을 위한 지시사항을 표현하는데, Apply-Actions, Clear-Actions, Write-Actions, Write-Metadata 그리고 Goto-Table 등의 메시지가 있다. 이 중 Write-Actions 는 패킷의 Action Set 에 행동을 추가하고, Goto-Table 은 파이프라인의 다음 테이블을 가리켜 기본적인 파이프라인 프로세싱을 지원한다.

또한 새로운 행동으로 group 이 추가되어 다양한 형태의 포워딩을 지원한다. 그룹은 하나의 고정된 출력 포트를 지정하는 형태가 아닌, 여러 개의 출력 포트를 동시에 이용하거나, 선택적으로 이용하는 것을 지원한다. 이를 통해 멀티캐스트나 브로드캐스트 뿐만 아니라, 다중 경로 중에 선택적으로 하나의 경로를 이용하는 것을 쉽게 지원하고 있다. Group 의 개념을 지원하기 위해 플로우 테이블 이외에 그룹 테이블이 새로 추가되었다.

4.1.2.2 OpenFlow 에 기반한 SDN 구현 사례

OpenFlow 가 소개된 이후로 다양한 네트워킹 환경에 OpenFlow 를 적용한 구현 사례들이 소개되고 있다. 초기의 OpenFlow 의 활용은 경로의 제어에 집중되어 있었지만, 차츰 OpenFlow 의 프로그래머블 네트워킹 능력을 응용이 요구하는 네트워킹을 위해 활용하는 결과물들이 많이 소개되고 있다. 현재는 OpenFlow 의 활용성이 높을 것으로 예상되는 데이터센터 내부에서 이용될 수 있는 데이터 센터 전원 관리, 다중 경로 전송 등의 기능들이 SDN 을 기반으로 구현되어 소개되고 있다.

가장 많이 소개되고 있는 구현사례로 웹서버의 로드 밸런싱을 제공하는 Aster*x [28]가 있다. Aster*x 는 스탠포드 대학에서 OpenFlow 를 기반으로 만든 결과물로 네트워크의 혼잡 상황과 더불어 서버의 부하까지 고려한 효율적인 로드 밸런싱을 제공한다. 서버와 네트워크 상태 정보들을 수집하고, 이를 바탕으로 사용자의 웹 서버에 대한 요청을 제어하는 데 이용함으로써 Aster*x 는 클라이언트의 응답 시간을 줄이고, 시스템 비용을 줄일 수 있는 효율적인 로드 밸런싱 시스템을 선보였다.

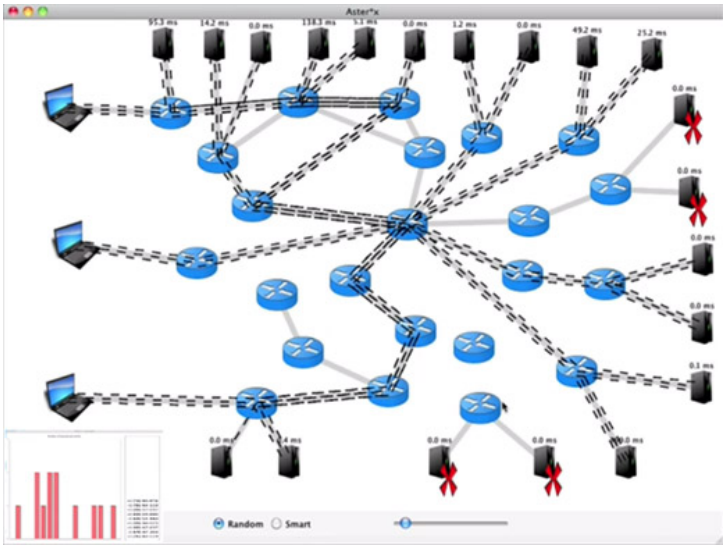

그림 28. Aster*x 의 Front-end GUI.

무선 환경에서 OpenFlow 를 적용한 대표적인 사례로 N-casting [29]을 들 수 있다. N-casting 은 무선 단말이 지닌 다수의 Wi-Fi, WiMAX 인터페이스를 동시에 활용하여 전송의 신뢰성을 높일 수 있음을 보였다. OpenFlow 를 지원하는 WiFi AP 와 스위치, 라우터 들을 활용하여 패킷 복제를 동적으로 제어함으로써 단말이 지닌 다수의 무선 인터페이스를 통해 패킷 전송의 신뢰성을 높인다.

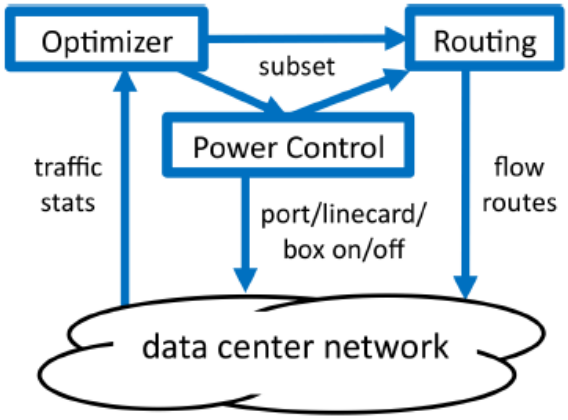

그림 29. Elastic Tree 의 시스템 다이어그램.

데이터 센터 내의 네트워크 관리에 대해서는 OpenFlow 의 논리적으로 중앙집중된 구조와 프로그래머블한 네트워킹을 활용하여 기존의 분산처리로 인해 불필요하게 복잡했던 부분을 단순화 시키면서 효율성을 높이는 일이 진행되고 있다. 대표적인 구현으로 데이터 센터 내의 트래픽 상황에 따라 전원 관리 기능을 제공하는 ElasticTree [30]가 있다. 그림 5 는 Elastic Tree 의 시스템 다이어그램으로 데이터 센터 내부의 트래픽 상황에 대한 모니터링을 기반으로 현재 상황을 처리하기에 적합한 최적의 토폴로지를 구하고, 이에 따라 네트워크 링크와 스위치와 같은 네트워크 구성 요소의 활성화 여부를 동적으로 제어한다. 또한, 이를 바탕으로 패킷들의 경로를 결정함으로써 네트워크 구성 요소의 동적인 비활성화가 네트워크 연결에 미치는 영향을 최소화하면서 에너지를 효율을 높인다.

다른 한편으로는 기존에 릴레이 노드와 같은 부가적인 미들박스가 처리했던 부분들을 네트워크 컨트롤러를 활용하여 제공함으로써 쉽게 네트워크 서비스들을 관리하고 보급할 수 있는 방법에

대해서도 연구가 진행되고 있다. 그 중 Phonenet [31]은 그룹 통신을 위해 필요한 기능들인 위한 세션관리 기능과 패킷 릴레이 기능을 모두 네트워크 컨트롤러를 통해 제공함으로써 추가적인 노드 없이 네트워크 컨트롤러와의 통신을 통해 그룹 통신이 가능함을 보였다.

4.1.3 ONF 를 중심으로 한 SDN 확산 및 향후 전망

4.1.3.1 ONF 소개 및 SDN 확산 계획

ONF(Open Networking Foundation)는 2011 년 도이치 텔레콤(Deutsche Telecom), 페이스북(Facebook), 구글(Google), 버라이존(Verizon), 야후(Yahoo!)를 창립 멤버로, SDN 을 통한 네트워킹의 혁신을 추구하면서 OpenFlow 를 비롯한 관련 기술과 표준을 개발하고 공유하기 위해 설립된 비영리 기관 목적이다 [21]. 따라서 지금까지 진행되던 OpenFlow 규격은 이제 ONF 에서 관리하게 되며 스탠포드 대학에서 운영하던 OpenFlow 사이트(http://www.openflow.org/)와 통합 및 조정 작업이 진행중이다.

4.1.3.1.1 회원 구성

현재 회원사는 설립 멤버(Foundings and Board Members)와 일반 회원으로 구분되어 있다. 설립 멤버는 앞서 언급한 초기 멤버와 Microsoft, 그리고 최근 가입한 NTT Communications 로 구성되고, 일반 회원사로는 Cisco 와 Juniper Networks 를 비롯한 기존 장비 업체와 HP, IBM, Intel, NEC, Nokia Siemens, Ericsson, Huawei 을 비롯한 주요 IT 업체와 Big Switch Networks, Nicira 와 같은 신생 벤처(이 두 회사는 OpenFlow 핵심 멤버가 창업한 회사임) 등과 같이 다양하게 구성되어 있으며, 우리나라에서는 ETRI 와 Samsung, KT 가 현재 가입한 상태다. 이처럼 네트워크 장비 사업자와 망 사업자뿐만 아니라, 인터넷 서비스 업체와 기존 주요 IT 업체가 모두 포진한 상태로서, 커뮤니티 자체의 구성과 잠재력은 상당히 강력한 편이다. 특이한 점은 설립 멤버의 구성을 통해 알 수 있듯이, Cisco 와 같은 기존 장비 업체보다는 구글이나 페이스북과 같은 인터넷 서비스 업체나 기존 IT 솔루션 회사가 주도하고 있으며, 기존 장비 업체는 SDN/OpenFlow 가 점차 호응을 얻는 추세에 동참하는 차원의 성격이 없지 않다.

4.1.3.1.2 성격 및 활동

ONF 는 표준화 기구로 설립됐지만, OpenFlow 와 같은 표준 규격 제정 활동에만 머무르지 않고, SDN 이라는 새로운 패러다임을 실현하기 위한 전반적인 아키텍처와 요소 기술을 개발하는 목적도 중요시하고 있다. 이러한 관점에서 OpenFlow 는, 지금까지 SDN 문맥에서 제시된 유일하면서 구현 가능한 표준 기술이며, SDN 을 구성하는 여러 가지 하부 기술 중 하나로서 강조되고 있지만, ONF 의 탄생이 OpenFlow 프로젝트의 성공에 기반을 둔 만큼, 현재까지는 OpenFlow 표준 개발이 ONF 활동의 중심축에 있다. 올해 12 월 5 일자로 OpenFlow 1.2 버전이 완료된 상황이며, 1.4 버전까지 2012 년도 중반에 완료할 계획을 갖고 있다. 회원사의 의지와 실행력을 바탕으로 올해부터 상당히 빠른 속도로 버전 업이 이루어지고 있으므로, 2.0 까지 일정대로 무난히 진행될 것으로 예상된다.

4.1.3.1.3 오픈 소스가 아닌 오픈 커뮤니티

ONF 는 집단 지성과 생태계 중심의 성격이 강하여 종종 오픈 소스 단체로 오해하기 쉽지만, 어느 한 회사가 기술과 시장을 장악하지 않고 여러 회사가 서로 공동으로 협력하는 오픈 커뮤니티를 추구한다는 의미일 뿐, ONF 의 활동으로 파생되는 IPR 은 멤버쉽을 가진 회사간의 공유의 성격이 강하다. 이러한 멤버쉽은 연 3 만달러로 유지되며, OpenFlow 프로토콜을 비롯한 SDN 관련 기술 및 라이센스에 비용을 포함한다고 볼 수 있다.

4.1.3.1.4 IETF 의 SDN

ONF 는 네트워킹 기술 및 표준 개발을 목적으로 한다는 점에서 IETF 과 유사한 부분이 있다. 최근 IETF 에서도 SDN BoF 가 결성됐는데, 여기서 말하는 SDN 은 Software Defined 가 아닌 Driven 으로서 소프트웨어 및 표준 인터페이스 중심의 접근 방법이란 점에서만 유사할 뿐, 실제 ONF 와는 공식적인 연계가 없는 독립적인 활동으로 볼 수 있고, 오히려 현재는 약간의 긴장 관계가 엿보이기는 하나, 이는 향후 두 단체가 어떻게 합의하는가에 따라 달라질 수 있다. 현재 ONF 는 SDN 및 OpenFlow 의 초창기 멤버를 중심으로 결성되어 있으며, 개인적인 차원에서

ONF 와 IETF 모두 활동하는 사람은 다수 존재한다. ONF 의 주요 표준 문서는 IETF 와 달리 저자 명시없이 공동체의 결과로 귀속된다.

4.1.3.1.5 구성

ONF 는 최상위 결정 기구인 Board of Directors 와 전반적인 기술 자문을 담당하는 Technical Advisory Group(TAG), 세부 기술별 Working Group(WG)과 각 WG 의장의 모임인 Council of Chairs, 그리고 ONF 를 총괄하는 Executive Director(Dan Pitt)과 홍보 및 운영 조직으로 구성되어 있다. 다른 표준 단체와 마찬가지로 주요 기술적인 활동은 WG 를 통해 이루어지고 있으며, 산업체 소속이 아닌 SDN/OpenFlow 초기 멤버는 TAG 을 통해 ONF 활동의 전반적인 방향에 조언하고 있다. WG 의장을 비롯한 주요 직책에 대한 결정은 OpenFlow 초기 멤버에 의해 결정됐지만, 이제 독립적인 ONF 가 설립된 이상 TAG 를 통한 조언 외에는 철저히 커뮤니티 중심으로 이루어질 예정이다.

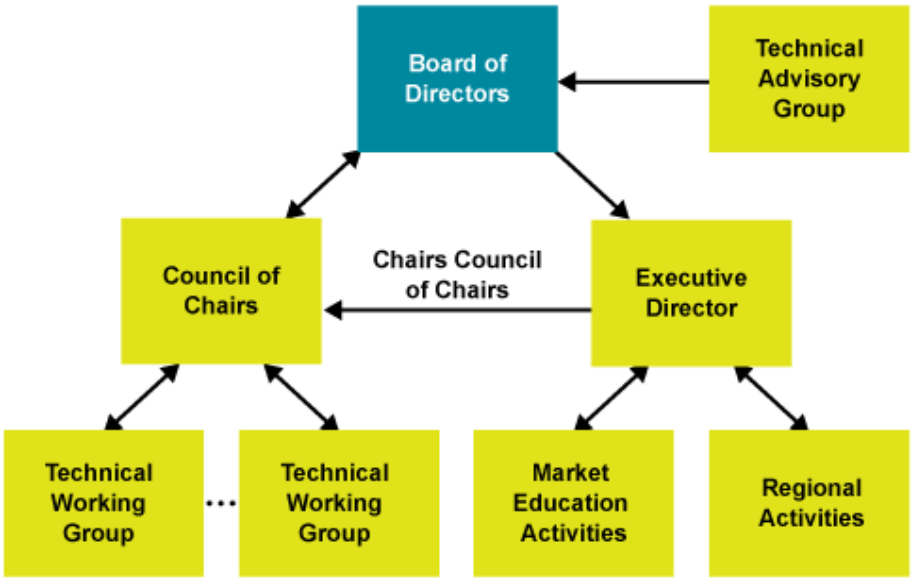

그림 30. ONF 구성.

현재 다음과 같은 WG 가 결성되어 있다. (각 WG 의장은 괄호 안에 명시) 현재 각 WG 은 오프라인 미팅보다는 메일링리스트와 간헐적인 컨퍼런스 콜 중심으로 이루어지고 있으며,

아시아측 참여자가 아직 적은 관계로 주로 미국 및 유럽 시간대에 진행되며, 주요 결과물이 나온 시점에 이뤄지는 오프라인 미팅도 샌프란시스코 및 산호세 지역에서 개최되고 있다.

- Extensibility (Jean Tourrihes, HP): 모듈화, 인코딩, 프로토콜 독립성을 위한 핵심 요소를 개발하며, OpenFlow 규격 문서를 개발함. 가장 많은 관심을 받고 있고, 다른 WG 에 비해 활동이 적극적인 편임
- Configuration and Management (Deepak Bansal, Microsoft): OpenFlow 스위치의 설정 및 관리 이슈를 다룸
- Testing and Interoperability (Michael Haugh, Ixia): 호환성 검증 및 테스팅 이슈를 다룸
- Hybrid (Bruce Davie, Cisco Systems): 기존 장비와 OpenFlow 스위치의 혼용 환경에 대한 이슈를 토의

또한 공식 WG 형태는 아니지만 메일링 리스트 등으로 다음과 같은 여러 가지 주제에 대한 토의도 활발히 진행되고 있다. 메일링 리스트 및 기타 자료에 대한 접근은 회원사 로 제한되어 있다.

- OpenFlow Future (OF 2.0): 현재 가장 활발히 활동하고 있으며, OpenFlow 2.0 의 핵심 구조와 이를 표현하기 위한 정형 언어를 논의하고 있음
- Match-Action Table: IPv4 나 IPv6 와 같은 특정 프로토콜을 처리하기 위한 방법을 논의
- Northbound API: 네트워크 제어 및 관리 애플리케이션 측면의 이슈를 다룸
- Use-case: OpenFlow 를 비롯한 여러 기술의 활용 사례를 논의

4.1.3.2 향후 전망

ONF 는 SDN 의 확산 및 OpenFlow 규격을 개발하는 것을 주요 목표로 두고 있는 만큼, 정기적인 ONS 및 ONF 미팅을 통한 커뮤니티 교류와 주요 기술 및 표준 공동 개발을 좀 더 적극적으로 개발할 것이다. 실제로 제 1 회 ONS 및 ONF 미팅을 가진 이후로 OpenFlow 규격 개발 속도가 급격히 빨라졌으며, 최근 신설된 OpenFlow-Future 그룹을 통해 OpenFlow 2.0 에 대한 논의도

가속화되고 있다. Scott Shenker, Nick McKweon 을 비롯한 주요 SDN 창시자가 강조한 학제간 연결을 위해 ONF 의 TAG 에도 현재 공석인 분산시스템 전문가가 충원될 예정이며, 기존 CS 기술과의 시너지는 보다 극대화될 것으로 전망한다.

표준 측면에서는 현재로선 IETF 의 SDN 활동과의 연계는 예측할 수 없으며, 당분간 ONF 회원사 중심으로 주요 기술과 표준을 빠른 속도로 개발해나갈 것으로 예상된다. 이러한 표준 기술 규격과 더불어 NEC, HP, Cisco, Juniper, Big Switch Networks 등을 비롯한 회사를 통해 OF 스위치 장비가 빠르게 개발될 것으로 예상되며, OpenFlow 1.0 이후 상당히 복잡해진 구조로 인해 규격의 효율적인 하드웨어 구현 문제를 해결하고, 범용적인 추상 모델과 현실적인 구현의 간격을 채우기 위한 노력도 계속 진행될 것이다. 또한 기존 ONS 데모 및 기타 연구 활동을 통해 제시된 여러 가지 라우팅 기술과 오버레이 네트워크 기술, 그리고 IPv6 지원을 비롯한 기존 기술과의 연계가 구체적으로 진행될 것이다.

OpenFlow 가 미래 인터넷을 위한 Clean slate 방식을 추구한 반면, ONF 에서는 좀 더 현실적인 접근을 취하여, 기존 인터넷 기술보다 융통성있는 기술 혁신이 가능한 데이터센터나 클라우드 분야에 적용되는 사례가 늘어날 것으로 예상되며, 실제로 구글이나 페이스북을 비롯한 자체 데이터 센터 및 클라우드 구축 기술을 보유한 업체에서는 이미 상당한 실험을 진행하고 있다. 또한 GENI 프로젝트에서도 OpenFlow 기술을 주요 요소 기술로 삼고 있는 만큼, 테스트베드 차원의 OpenFlow 망 구축도 보다 고도화될 예정이며, 국내 테스트베드 또한 SDN/OpenFlow 연계에 대해 적극적인 행보를 보이고 있는 만큼, 내년에는 미래인터넷 테스트베드의 SDN/OpenFlow 지원이 보다 구체화될 것이다.

현재의 추세대로 SDN 에 대한 커뮤니티의 기대가 큰 만큼, 조만간 다양한 SDN 관련 기술과 솔루션, 그리고 구축 및 활용 사례도 점차 늘어날 것이며, 국내에도 ETRI, Samsung, KT 등이 최근 ONF 에 참여한 만큼 향후 국내 전문가의 ONF 활동도 기대해볼 수 있다.

4.2 OpenFlow-enabled Programmable Switches: OF@Korea

OpenFlow 가 소프트웨어 형태로 공개되고, 이를 지원하는 상용 스위치가 등장하면서 국내에도 몇몇 연구 기관과 대학에서 OpenFlow 를 기반으로 관련된 연구를 진행하는 그룹들이 생기면서, 각자가 독립적인 OpenFlow 테스트베드를 구축해서 활용하고 있다. 하지만, 각자 구축한 테스트베드 내에서 OpenFlow 를 지원하는 노드의 수는 실제 네트워킹 환경에 비교하기에는 규모가 작아 다양한 네트워킹 실험을 진행하기에는 부족한 점이 많다. 2011 년 상반기부터 시작된 OF@Korea 는 국내에서 서로 독립적인 많은 연구기관과 대학에서 관리되고 있는 OpenFlow 를 지원하는 자원들을 통합하여 약결합된 (loosely-coupled) 대규모의 OpenFlow 테스트베드를 구축하여 규모가 있는 OpenFlow 기반의 네트워킹 테스트베드를 구축하는 것을 목표로 한다. 또한 이를 OpenFlow 기반의 SDN 을 연구하는 많은 실험자들이 공유하여 활용할 수 있도록 지원하고자 한다.

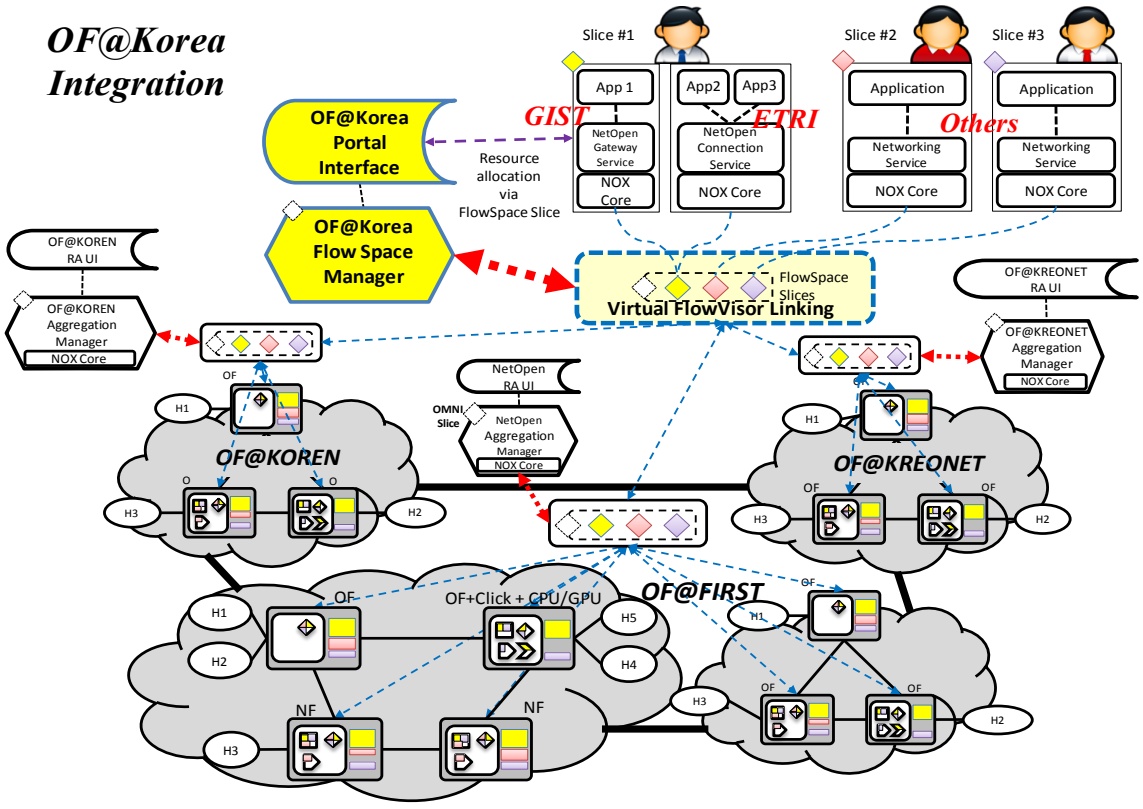

그림 31. OF@Korea 통합 개요도 (진행중).

4.2.1 OF@Korea 를 위한 자원 통합

그림 31 은 2011 년 봄부터 진행 중인 OF@Korea 라고 이름 지은 국내 OpenFlow 자원의 통합 개요를 보여주는 그림이다. 먼저, OF@Korea 는 다수의 연구기관과 학교, 그리고 과제 수준에서 관리되고 있는 OpenFlow 자원들의 집합 (RA, Resource Aggregate)을 연결하고 있다. 현재 참여하고 있는 OpenFlow 가 활성화된 국내 자원들로는 연구망을 제공하는 KOREN (Korea Advanced Research Network)과 KREONET (Korea Research Environment Open NETwork)에서 관리하고 있는 OpenFlow 자원들인 OF@KOREN 과 OF@KREONET, 그리고 미래 인터넷 과제인 FIRST (Future Internet Research for Sustainable Testbed) 에서 관리하고 있는 OpenFlow 자원인 OF@FIRST 가 있다. 2011 년 여름에 FIF-TB WG 미팅을 통해서 이러한 자원들의 통합을 위해 세 기관의 담당자들이 실무 수준의 협력을 시작하였다.

OF@FIRST RA 는 미래 인터넷 과제인 FIRST 에 의해 구축되어 관리되고 있는 OpenFlow 자원으로, ETRI 의 FIRST 팀과 GIST, 충남대, 포항공대, 경희대의 대학들이 팀을 이룬 FIRST@PC 팀의 협력을 통해 관리되고 있다. OF@FIRST RA 내부적으로는 FIRST@PC 팀은 NetFPGA/OpenFlow 를 이용한 PC 기반의 가상화된 컴퓨팅/네트워킹 플랫폼을 프로토타입을 만드는 연구를 진행 중이며, ETRI 는 네트워크 프로세서를 기반으로 컴퓨팅/네트워킹 자원을 제공하는 'FIRST 플랫폼'이라고 부르는 ACTA 기반의 가상화 플랫폼을 구축하고 있다. 연구망인 KREONET 과 KOREN 은 OpenFlow 를 지원하는 자원들을 기존의 R&D 망에 VLAN 기반으로 구성하여 L2 연결을 제공하고 그 들을 각각 OF@KREONET 과 OF@KOREN 자원으로 OF@KOREA 통합을 위해 제공하고 있다.

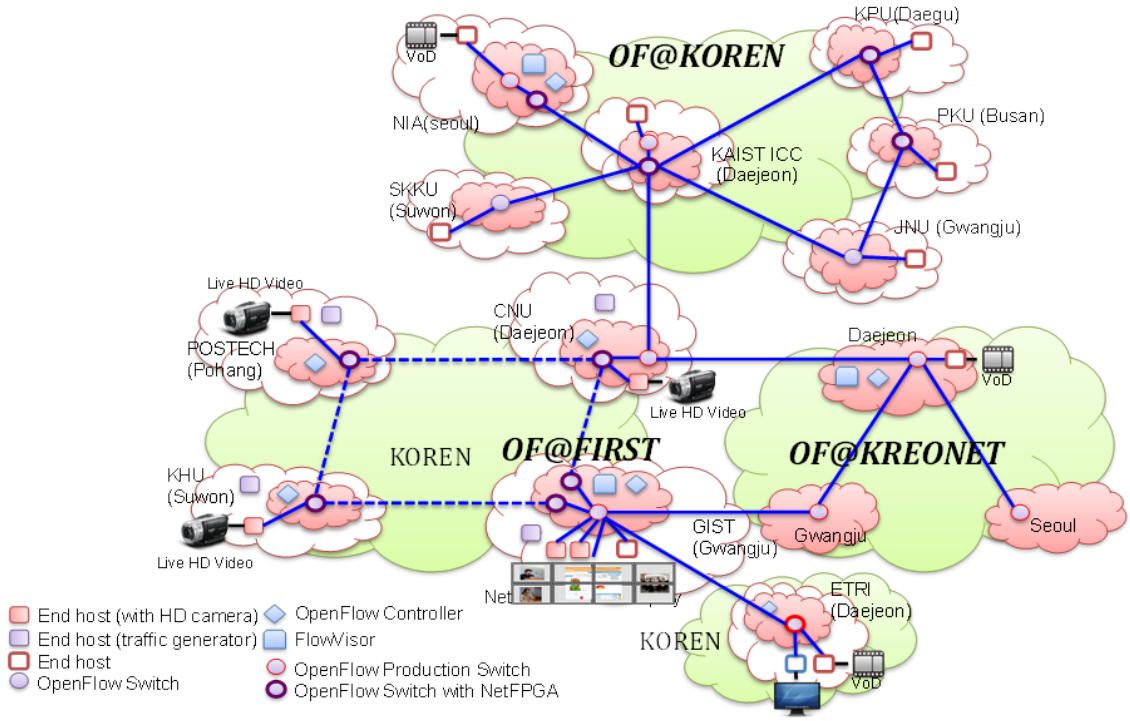

그림 32. OF@Korea 네트워크 하부 구성도.

각 참여 기관의 OpenFlow RA 는 그림 31 에서와 같이 각자의 Aggregation Manager 와 UI 를 통해 관리된다. 각 Aggregation Manager 들은 관심있는 플로우들의 집합인 플로우 스페이스 수준의 가상화를 지원하는 FlowVisor 와 그 들 사이의 연결을 통해 서로 약결합된다. 또한 이렇게 약결합된 자원을 포탈 형태의 인터페이스를 통해 실험자에게 제공한다.

그림 32 는 OF@Korea 의 네트워크 하부의 구성을 보여주는 그림으로, 세 참여기관의 OpenFlow RA 의 상호 연결을 보여준다. 각 참여기관 사이의 연결은 내부적으로 2 계층의 VLAN 연결과 터널링을 통해 구성되어 있다. 우리는 다양한 형태의 OpenFlow 를 지원하는 (PC 기반이나 상용) 스위치들을 FlowVisor 수준으로 연결하고, 이를 포탈 스타일의 인터페이스를 통해 실험자들에게 제공한다. 아직은 시작인 수준이지만, 현재 OpenFlow@Korea 는 국내에 약결합 형태로 통합된 OpenFlow 테스트베드를 구축하였고, 현재는 각 참여기관의 실험자에 의해 그 활용성을 검증하고 있다.

4.2.2 OF@Korea 활용

OF@Korea 의 초기적인 형태를 구축하고 이를 기반으로 미국 스탠포드 대학에서 처음 열린 SDN (Software-Defined Networking) 에 대한 행사인 Open Networking Summit (ONS) 2011 에 참여하여 한국의 OpenFlow 자원들을 연결한 OpenFlow@Korea 시연을 통해 OF@Korea 의 활용성을 검증하였다.

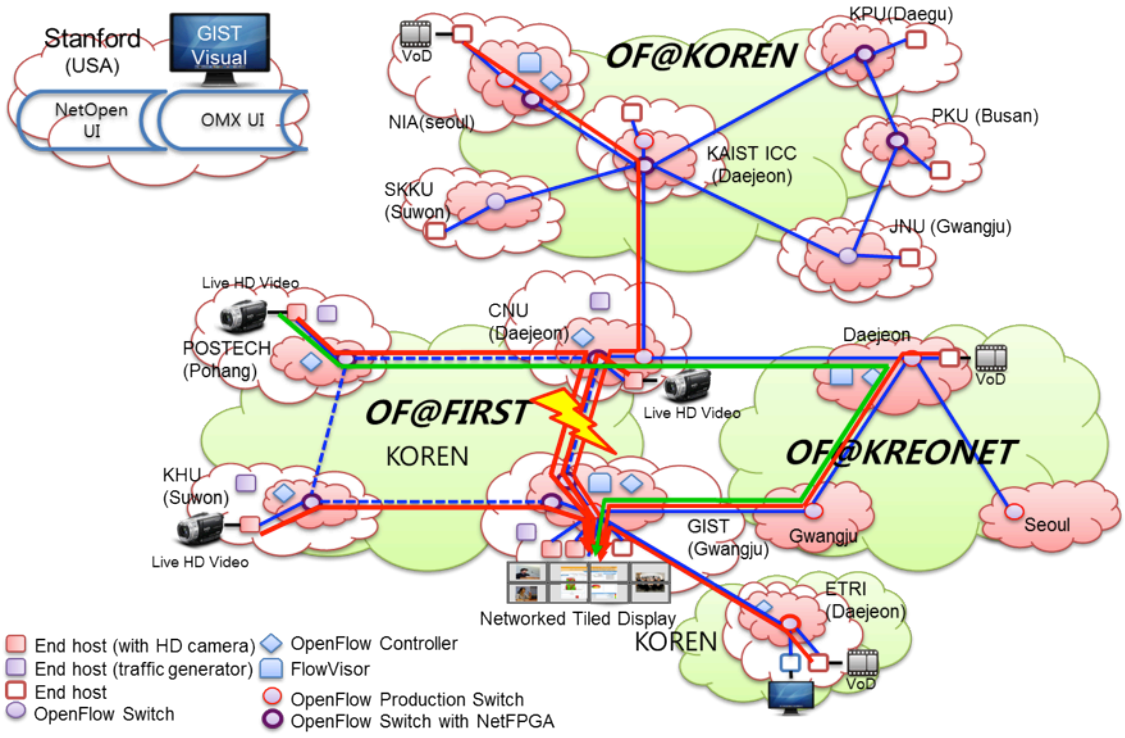

그림 33. Open Networking Summit 2011 에서의 QoS Control 시나리오.

그림 33 은 ONS 시연에서 활용한 QoS Control 시연 시나리오를 보인 것이다. 먼저 각 참여기관, 즉 OF@FIRST 의 GIST, 포항공대, 경희대, 충남대에서 250Mbps 정도의 대역폭을 필요로 하는 HD 비디오를 GIST 의 네트워크로 관리되는 격자 형태의 디스플레이 장치인 NeTD (Networked Tiled Display)로 전송한다. 그리고, OF@KOREN, OF@KREONET 그리고 ETRI 에서도 30Mbps 정도의 대역폭을 차지하는 HD video 를 마찬가지로 GIST 의 NeTD 로 가시화한다. 그림 상의 붉은색 화살표는 이 때 발생하는 비디오 플로우들의 전달 경로로 GIST 와 충남대 (CNU) 사이의 링크에

많은 플로우들이 집중됨을 알 수 있다. 하지만, 실제 플로우들이 이용 가능한 경로는 포항공대 (POSTECH)와 경희대 (KHU)를 연결하는 링크나 OF@KREONET 의 대전과 광주 스위치를 연결하는 링크가 가용하다. 본 시연에서 우리는 OF@KREONET 의 경로를 이용해 플로우들을 분산시킴으로써 모든 플로우들이 대역폭 충돌없이 전송될 수 있음을 시연하였다.

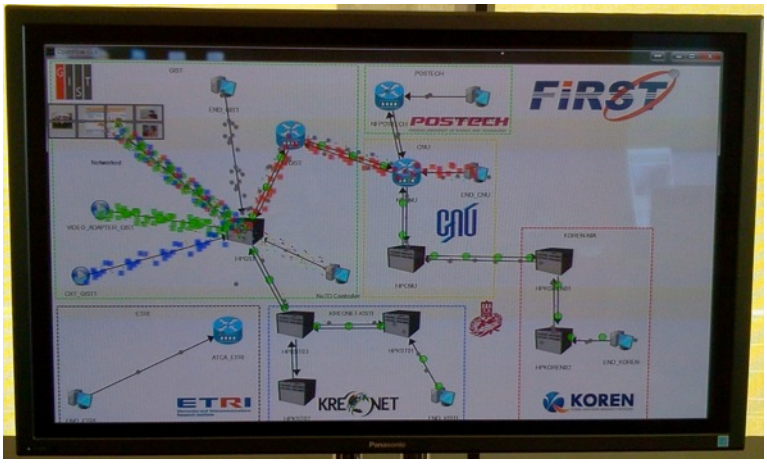

그림 34. ONS 시연 시의 NetOpen Experiment UI 모습.

그림 34 는 시연 시에 활용한 NetOpen Experiment UI 의 모습을 보여준다. 충남대 (CNU)에서 오는 비디오 플로우는 빨간색, GIST 에서 오는 플로우는 초록색으로 표현하여 쉽게 파악할 수 있도록 하였다. 이를 통해 GIST 의 NeTD (Networked Tiled Display)에 가시화된 결과를 그림 35 에 보였다.

그림 35. GIST 의 Networked Tiled Display 에 가시화된 결과.

그림 36 은 ONS 시연장에서 시연을 위한 구성으로 맨 왼쪽부터 GIST 의 NeTD 에 가시화된 결과를 보여주는 노트북과 서비스 합성 실험을 제어하는 노트북, 그리고 NetOpen Experiment UI 를 보여주는 노트북이 놓여있고, 그 뒤에 이 노트북들의 화면을 스위칭하여 보여줄 수 있는 LCD 디스플레이를 볼 수 있다.

그림 36. Open Networking Summit 시연장 모습(Stanford, USA)

4.3 Virtual Router: FIRST

혁신적인 아키텍처와 서비스를 검증하기 위해 미래인터넷 테스트베드가 요구되고 있다. 이를 위해, FiRST (Future Internet Research for Sustainable Testbed) 프로젝트에서는 엄격한 자원 격리를 지원하는 네트워크 가상화 기술과 하드웨어에 대한 구체적인 지식이 없이도 사용자가 원하는 네트워크를 쉽게 구성하도록 하는 네트워크 프로그램화 기술을 개발하고 있다. 또한, 사용자 접근을 위한 웹 기반 인터페이스, 가상 노드•네트워크의 생성•변경•삭제, 자원 규격의 설정, 다양한 정보의 검색 등을 제공하는 제어 프레임워크 기술도 함께 개발하고 있다. 한편,

미래인터넷 생태계 조성을 통한 국가기술 경쟁력 확보 차원에서 KOREN (Korea Advanced Research Network) 고도화 사업의 일환으로 KOREN 의 미래인터넷 테스트베드에 FiRST 플랫폼을 구축하였다.

4.3.1 FiRST 플랫폼

FiRST 플랫폼은 고속의 패킷 처리 장비에서 통신 자원의 격리를 통해 사용자가 원하는 가상 노드를 신속하고 독립적으로 생성•제어할 수 있도록 하는 프로그래머블 네트워크 가상화 플랫폼로써, 특정 하드웨어에서 벗어나 개방형 서브스트레이트 및 이식성을 통해 네트워크 가상화와 프로그램화 기술을 실현하는 일종의 소프트웨어 기반 네트워킹 개념을 지향하고 있다. 이 FiRST 플랫폼은 하드웨어 개발에 대한 부담을 최소화하고 신속한 플랫폼 개발을 위해 유무선 분야 네트워크 장비의 표준인 ATCA(Advanced Telecommunications Computing Architecture)를 적용하였다. 하나의 노드는 ATCA 샤시, NP 블레이드, CPU 블레이드, 스위치 블레이드를 포함하고 있다. 하나의 NP 블레이드는 2x cn5860 NP 모듈을 가지고 있고, 하나의 NP 모듈은 16 개의 멀티코어(코어당 750 MHz 속도)와 2MB 의 L2 캐쉬를 포함하고 있다. 다음 그림은 이러한 FiRST 플랫폼의 하드웨어 형상이다.

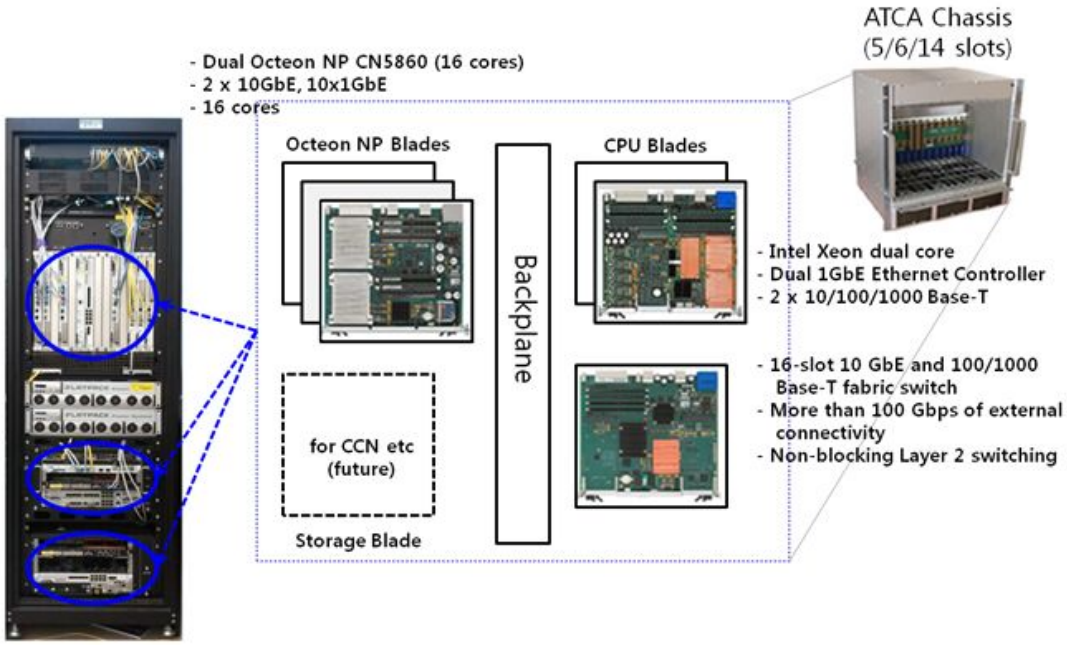

그림 37. FiRST 플랫폼의 하드웨어 형상.

FiRST 플랫폼의 핵심 기능은 데이터평면에서 네트워크 가상화와 프로그램화 기능을 제공하는 패킷바이저(Packetvisor) 기술로써, 이를 통해 사용자는 자신이 원하는 가상노드를 동적으로 신속하게 구성할 수 있다. 패킷바이저는 네트워크 가상화와 관련하여 컴퓨팅 및 네트워킹 자원(CPU, 코어, 메모리, I/O 등)의 엄격한 격리 메카니즘을 제공하는 PacketvisorVMM(Packetvisor Virtual Machine Monitor, PVMM)과, 네트워크 프로그램화와 관련하여 API (Application Programming Interface) 및 저작도구와 같은 패킷바이저 SDK (Software Development Kit) 환경을 제공하는 PacketvisorWorks(PWorks)로 구성되어 있다.

PVMM 은 NP 기반 고속의 패킷처리 하드웨어에서 사용자 프로그램을 독립적으로 실행하는 하이퍼바이저로서, 슬리버 단위로 컴퓨팅 및 네트워킹 자원을 동적으로 할당•격리하여, 독립적인 가상 네트워크의 생성•관리를 지원한다. 이 PVMM 은 제어 API, PVMM 관리자, RAMFS (Random Access Memory File System), PVMM 커널 등으로 구성되었다. 제어 API 는 PVMM 상에서 실행되는 가상머신들을 제어하기 위해 사용되는 인터페이스의 집합이며, PVMM 관리자는 제어 API 와 PVMM 커널 사이에서 가상머신들을 관리해주는 시스템 가상머신이다. 그리고 RAMFS 는

사용자 응용 프로그램을 PVMM 상에서 실행 가능한 이미지로 변환해주는 RAM 기반 파일시스템이며, PVMM 커널은 NP 기반 컴퓨팅 및 네트워킹 자원의 가상화를 통해 가상머신들이 독립적으로 실행될 수 있는 환경을 조성해준다. JVM(Java Virtual Machine)이 호스트 OS (Operating System) 위에서 하나의 프로세스를 실행하기 위한 단일-스택 머신이라면, PVMM 은 하드웨어 바로 위에서 여러 개의 프로세스를 시분할로 실행시키는 다중-스택머신이다. 다음 그림은 이러한 PVMM 구조를 나타낸다.

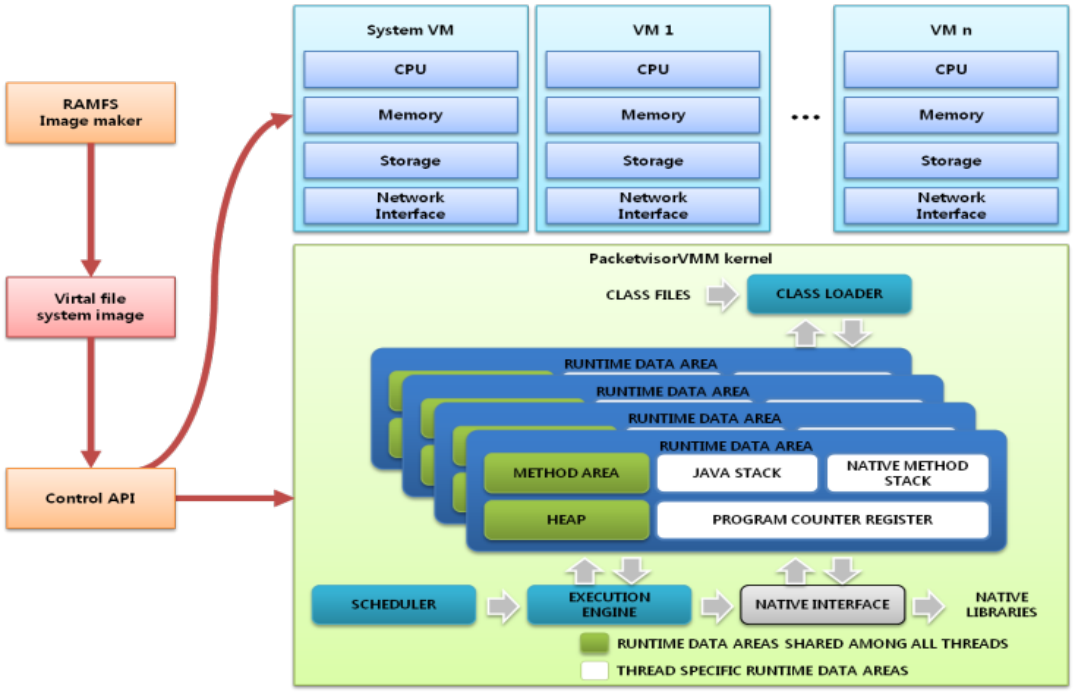

그림 38. PVMM 구조.

4.3.2 FiRST 제어 프레임워크

가상 네트워크를 위한 제어 프레임워크 구조는 GENI(Global Environment for Network Innovation)의 개념을 적용하고 있으나, 제어 프레임워크의 대상이 되는 플랫폼이 서로 다르기 때문에 자원 명세 및 구현된 기능들은 서로 다르다. 또한, 본 제어 프레임워크 구조에서는 CH (Clearinghouse), AM (Aggregate Manager), CM (Component Manager) 이외에 FiRST 플랫폼에

맞게 실험 기능을 지원하는 FH (FiRST Home)이라는 사용자 실험 에이전트 기능이 추가되어 있다. FiRST 제어 프레임워크의 구조는 다음 그림과 같다.

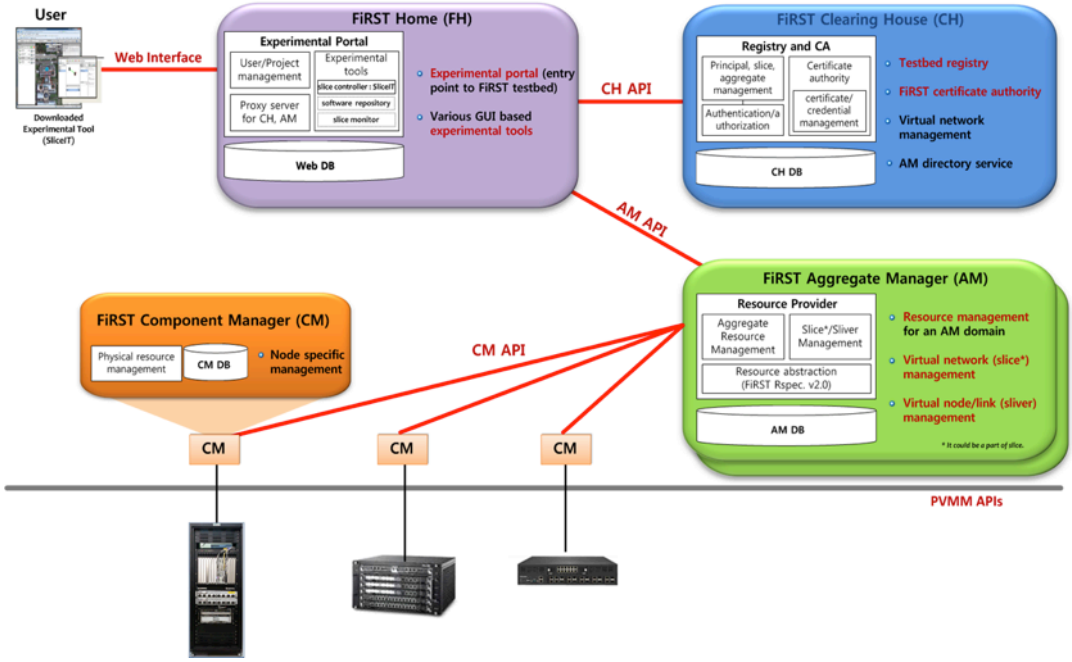

그림 39. FiRST 제어 프레임워크 구조.

여기서 FH 는 FiRST 테스트베드의 실험 포털로써 다양한 실험 툴을 보유하여 사용자로 하여금 가용 자원 검색, 슬라이스•슬리버 제어 및 자원 모니터링 등을 수행하도록 한다. CH 는 테스트베드의 사용자, 슬라이스 및 애그리게이트(컴포넌트의 집합)에 대한 정보를 관리하고, 사용자 인증 및 권한 검증을 위한 인증서(certifiacte) 및 신임장(credential)을 발급한다. 컴포넌트에 대한 및 도메인의 자원정보를 관리하고, 등록된 사용자를 인증하여 적합한 슬라이스에 대한 권한을 부여한다. AM 은 네트워크 도메인 또는 사이트 단위로 컴포넌트 자원을 관리한다. 그리고 CM 은 해당 컴포넌트의 슬리버들에 대한 CPU, 메모리, 대역폭 등의 자원 등을 관리한다. 이러한 FH, CH, AM, 그리고 CM 사이의 인터페이스는 다음 표와 같다.

표 4. FiRST 제어 프레임워크 API 리스트.

Operation	CH-API
슬라이스 제어	RegisterSlice, RemoveSlice ResolveSlice, BindToSlice AddAmToSlice 등
객체 (실험자/AM) 제어	ListAggregate, List, Register Remove, Update, Resolve 등
인증 / 권한검증	GetUserCredential, GetSliceCredential 등
버전정보	GetVersion
Operation	**AM-API**
슬리버제어	CreateSliver, DeleteSliver RenewSliver, SliverStatus 등
자원검색	ListResources 등
버전정보	GetVersion
Operation	**CM-API**
슬리버제어	CreateSliver, DeleteSliver 등
링크연결	AddLink
자원검색	ListResources

4.4 Programmable Router: Packet Shader

본 연구에서는 PC 기반의 소프트웨어 라우터를 GPU 를 사용해 효과적으로 가속하는 것을 목표로 하고, 크게 다음 두 가지의 방법론으로 나누어 접근하였다. 첫째, PC 기반의 소프트웨어 라우터는 NIC(Network Interface Card; 네트워크 인터페이스 카드)을 통해 패킷을 송수신하게 되는데, 이를 고속으로 처리할 수 있도록 하였다. 둘째, 소프트웨어 라우터에서 수신된 패킷은 IP 포워딩 룩업이나 암호화 등 패킷 처리를 수행해야 하는데, 이를 GPU 를 통해 병렬적으로 처리하여 가속 처리하는 방법을 개발하였다.

4.4.1 고성능 네트워크 I/O 스택 구현에서의 병목 진단 및 개선안 제시

일반적인 소프트웨어 라우터는 CPU 자원의 대부분을 NIC 을 통한 패킷의 송수신 과정에 사용한다. 기존의 리눅스 네트워크 스택에서는 64B 패킷의 경우, 라우팅을 하지 않고 단순 포워딩을 하더라도 서버 한 대에서 최대 13.3Gbps 내외의 성능을 보여준다. 이를 수십 Gbps 급의 성능으로 끌어올리기 위해서는 리눅스 네트워크 스택에 존재하는 비효율성을 찾아 이를 최적화해야 한다. 이를 위해, 본 연구팀에서는 패킷당 CPU 사이클 소요량을 OProfile 시스템 프로파일러를 사용해 측정한 결과, 리눅스 커널에서 패킷에 대한 메타 정보를 저장하기 위한 데이터구조와 메모리 할당과 해제를 관리하는 메모리 서브시스템에서 63% 이상의 CPU 시간을 소비하여 라우터로서는 비효율적임을 알 수 있다. 본 연구에서는 크게 세 가지 방법으로 리눅스커널의 비효율성을 제거하였다.

첫째는 Huge Packet Buffer 이다. 큰 버퍼를 하나 할당하여 각 패킷이 고정된 위치에 저장되도록 하고, 메타정보에 대해서도 마찬가지로 큰 버퍼를 하나 할당해 각 패킷에 대한 메타정보를 순차적으로 저장함으로써, 메모리 할당/해제 비용을 줄일 수 있다.

두 번째로는 여러 패킷을 일괄적으로 모아서 처리하는 방식, 즉 Batch Processing 이다. 본 연구에서는 배치 프로세싱을 하드웨어 레벨, 디바이스 드라이버 레벨, 어플리케이션 레벨의 세 단계에서 모두 구현하였다. 첫째, 하드웨어 레벨에서, NIC 이 여러 패킷들을 하나로 모아 하나의 PCI-e 트랜잭션 내에서 처리할 수 있도록 하였다. 이는 I/O 대역폭을 효과적으로 사용할 수 있게 해준다. 둘째, 디바이스 드라이버에서, RX/TX 순환 큐 조정이 패킷 단위가 아닌 배치 단위에서 처리되게 함으로써, 패킷마다 소요되는 관리 비용(per-packet bookkeeping cost)를 크게 줄였다. 셋째, 어플리케이션 레벨에서, 프로그램이 여러 패킷을 한꺼번에 처리하게끔 함으로써 함수 호출과 동기화 비용을 줄여 패킷당 더 적은 CPU 명령이 소요되도록 하였다. 또한 위의 모든 세 과정에서, 소프트웨어 prefetch(CPU 가 실제 메모리의 데이터를 접근하기 전에, 미리 그 내용을

CPU 캐시에 불러들여 캐시 미스로 인한 접근 지연을 최소화 하는 기술)를 공격적으로 사용하여, 디바이스 드라이버가 패킷 디스크립터와 데이터의 접근을 캐시 미스 비용이 거의 없도록 하였다.

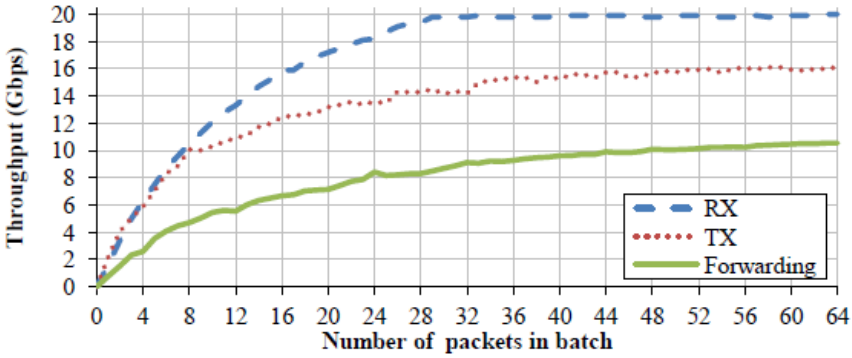

그림 40. 2 CPU, 2 10GbE 사용했을 때 batch processing 에 따른 64B 패킷의 포워딩 성능.

위의 그래프는 이러한 배치 프로세싱의 효과를 보여준다. 배치 프로세싱을 수행하지 않을 경우(배치되는 패킷의 수가 하나일 때) 그 성능이 1 Gbps 에 미치지 못할 정도로 매우 낮지만, 배치의 크기가 커질수록 그 성능이 향상되어 64 개의 패킷을 한번에 배치하는 경우에는 포워딩의 경우 하나의 CPU 코어당 10.5 Gbps 의 성능을 보여주는데, 이는 약 13.5 배의 성능 향상 효과이다.

세 번째로는 멀티코어 시스템에 대응하여 멀티코어 상에서 효율적으로 동작할 수 있는 시스템을 구현하였다. 현대 컴퓨터의 발전은 코어 하나의 속도를 높이는데 한계가 있어 여러 개의 코어를 하나의 시스템에 넣는 방향으로 진화하고 있다. 많은 수의 코어가 하나의 시스템에 존재함으로써, 코어들 간의 통신에서 발생하는 오버헤드가 시스템의 전체 성능에 큰 영향을 주기 때문에 이런 코어들 사이의 통신을 최소화 하여 코어 수에 비례해서 성능이 증가할 수 있어야 하며, 하나의 코어에 일이 몰리지 않고 여러 코어에 고루 분배해줄 수 있어야 한다. 최신 NIC 들은 코어들 사이에 트래픽을 분배해 주기위해 Receive Side Scaling(RSS) 기능을 제공한다. RSS 는 패킷헤더의 해시 값을 통해 패킷들을 코어들 사이에 분배해주는 기술이다. 또한 각 코어마다 별도의 독립된 패킷 큐를 둘 수 있는데, 이것을 유저레벨에서 직접 접근하도록 함으로써 큐의 개수에 따라 더 많은 코어를 사용할 수 있도록 하였다.

본 연구에서 제안한 위의 최적화 방안들을 구현하고 평가한 결과는 아래와 같다. 성능을 평가할 서버를 패킷 생성기와 10 Gbps 이더넷 포트 8 개를 사용하여 직접 연결하고, 패킷 생성기에서 다양한 크기의 입력 패킷을 생성해 부하를 발생시켰다. 서버에서는 크게 패킷 수신(RX), 송신(TX), 포워딩(RX+TX)의 세 가지 경우에 대해 성능을 평가하는 동시에, CPU 점유율을 측정하였다.

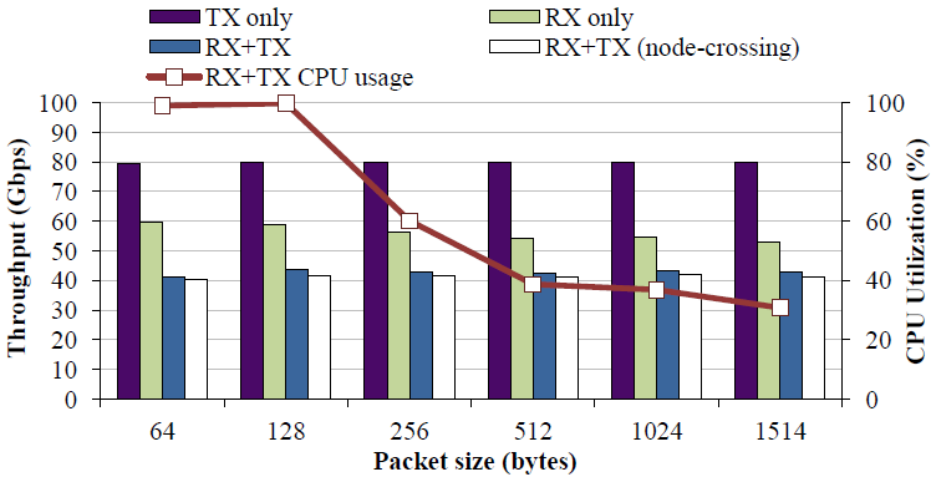

그림 41. PacketShader 에서 개발한 고성능 패킷 I/O 엔진의 성능.

실험 결과, TX 의 경우 80 Gbps, RX 의 경우 60 Gbps, 포워딩의 경우 40 Gbps 의 성능을 패킷 크기에 관계없이 보여주었다. CPU 점유율은 패킷의 크기가 작을 때 높고, 패킷의 크기가 커지면 낮아진다. 패킷의 크기가 작을 때 CPU 점유율이 100%라 하더라도, 패킷의 배치 크기에 따라 CPU 가 탄력적으로 사용되므로 추가적인 작업을 수행할 수 있는 여지가 남아 있다. 40 Gbps 의 포워딩 성능은 Intel Research Berkeley 의 기존 연구 결과인 RouteBricks 의 13.3 Gbps 의 성능에 비교할 때 대략 3 배의 성능 향상 효과를 보여준다.

4.4.2 GPU 에서의 기본적인 라우터 기능 가속 기술 개발

본 연구에서는 고성능 패킷 I/O 엔진위에 이제 GPU 를 활용할 수 있는 프레임워크를 개발하였다. GPU 를 네트워크 프로세싱에 활용하는데 있어서 핵심 기술은 네트워크 프로세싱에 존재하는 병렬성을 잘 활용하는 데 있다. 현재 인터넷 네트워크는 기본적으로 패킷 단위로 프로세싱이 이루어지므로 여러 패킷들을 동시에 처리함으로써 병렬성을 확보할 수 있다. 또한 패킷

하나에서도 암호화 알고리즘 같은 경우 패킷 크기보다 작은 블록에 대하여 동시에 처리할 수 있는 경우가 존재하며 이런 경우에도 GPU 의 병렬성을 잘 활용할 수 있다. 본 연구에서는 이런 병렬성을 효율적으로 활용할 수 있는 프레임워크를 구축하였다.

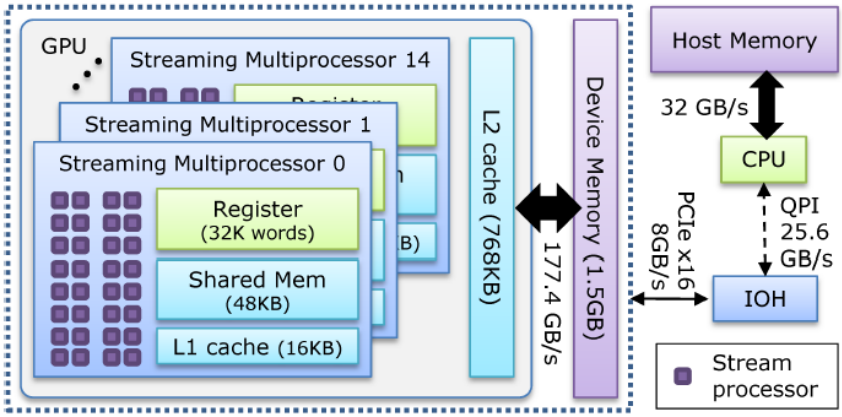

그림 42. PacketShader 구현에 사용한 NVidia GTX480 그래픽 프로세서의 구조.

GPU 의 병렬성은 소프트웨어 라우터의 성능 최적화를 위한 핵심 개념이다. 일반적인 용도의 CPU 와 달리, GPU 의 각 코어는 CPU 보다 낮은 성능을 가지고 있지만, 그 코어들이 많은 수가 배치되어 전체적인 처리량을 높일 수 있는 구조이다. 예를 들어, 본 연구에서 사용된 NVIDIA 사의 GTX480 GPU 는 도합 480 개의 코어를 가지고 있는데, 이는 일반적인 CPU 의 2~8 개의 코어에 비해 압도적으로 많은 것이다.

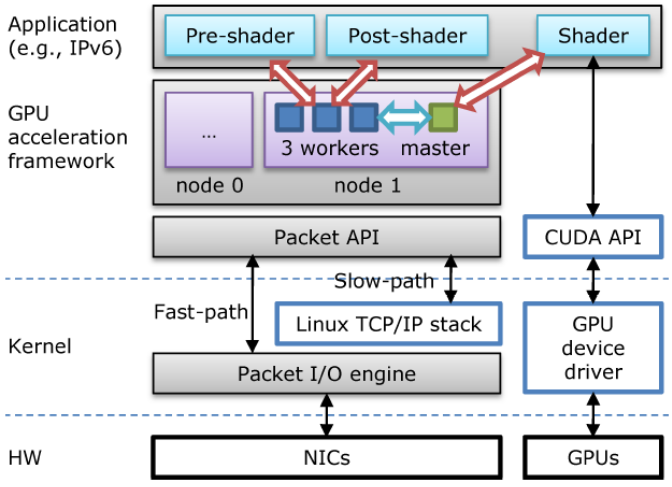

그림 43. PacketShader 의 소프트웨어 구조.

위의 그림은 본 연구에서 개발한 PacketShader 시스템의 개략적인 구조를 보여준다. 앞서 기술한 고성능 패킷 I/O 엔진이 커널 레벨에서 구현되어 있으며, 실제 GPU 가속 프레임워크는 시스템 구현의 유연성과 편리성을 도모하기 위해 유저레벨에서 구현되었다. 커널과 유저레벨 사이의 패킷 I/O 인터페이스는 커널 레벨의 디바이스 드라이버에 유저레벨 프로세스가 접근해 패킷을 송수신할 수 있도록 API 를 제공한다. GPU 가속 프레임워크는 시스템의 NUMA 프로세스 단위로 독립적으로 동작하는데, 이는 복수의 CPU 를 가지는 시스템에서의 성능 확장성을 극대화하기 위함이다.

GPU 를 통해서 프로세싱을 하기 위해서는 GPU 메모리로 데이터를 전송하고 이 데이터를 프로세싱하기 위해 GPU 에서 동작하는 커널을 실행시키고, 마지막으로 처리된 데이터를 다시 호스트메모리로 복사 하는 과정이 필요하다. 또한 GPU 로 복사하기 전 후에 처리할 데이터를 모으는 전처리 과정 그리고 GPU 에서 패킷을 처리한 후 데이터를 가지고 실제 패킷을 보내기 위한 후처리 과정이 필요하다. 이를 위해 각 NUMA 노드 내에서, CPU 코어들 중 하나는 마스터 스레드로 동작하며, 나머지는 워커 스레드로 동작한다. 마스터 스레드는 GPU 가속을 위한 인터페이스로 동작하고, 나머지 워커 스레드들은 NIC 과의 패킷 송수신 부분을 담당하게 된다. PacketShader 가 이러한 구조를 갖는 것은, 현재 GPU 가 여러 개의 CPU 코어에서 접근하게 되면 context switching 비용이 과도하게 커져 성능이 급감하는 문제를 회피하기 위함이다. 하나의 GPU 는 하나의 CPU 코어(마스터 스레드)가 전담해 처리하도록 하고, 나머지 워커 스레드들이 수신한 데이터를 마스터 스레드로 전달하여 처리하게끔 하는 일종의 프록시 구조를 가진다. 다음 그림은 마스터 스레드와 워커 스레드가 시스템 내에서 협력적으로 동작하는 방식을 보여준다. 모든 패킷 처리는 Pre-shaing, Shading, Post-shading 의 세 단계에 걸쳐 수행된다.

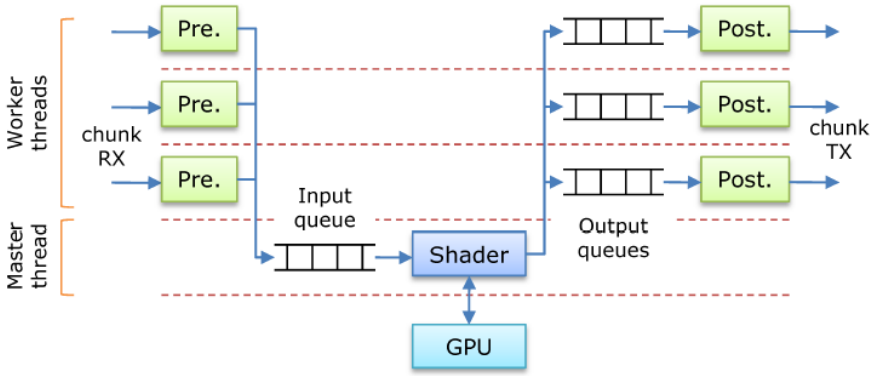

그림 44. 마스터 스레드와 워커 스레드의 동작 방식.

이렇게 구현된 프레임워크의 실제 동작 성능을 측정하기 위해, 본 연구에서는 IPv4 포워딩, IPv6 포워딩, OpenFlow 스위치, IPsec 게이트웨이의 네 가지 어플리케이션을 구현하고 그 성능을 평가하였다. 실험을 위해, PacketShader 가 동작하는 서버와 입력 트래픽을 생성하는 트래픽 생성기를 8 개의 10 Gbps 이더넷 포트로 직접 연결한 다음, 다양한 크기를 가진 패킷을 생성하여 PacketShader 시스템이 처리하도록 하였다. 위의 네 가지 어플리케이션을 본 연구에서 구현하고, 실험을 통해 그 성능을 측정한 결과는 다음과 같다.

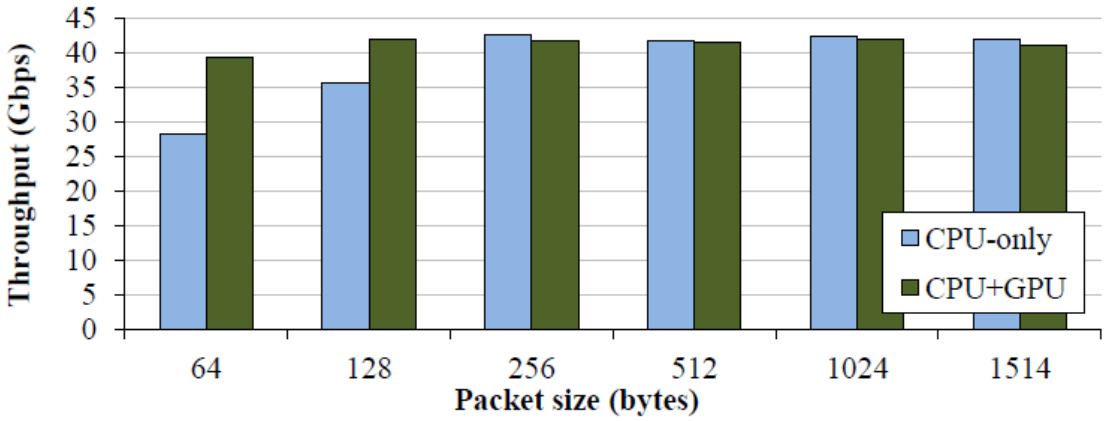

그림 45. IPv4 포워딩 성능.

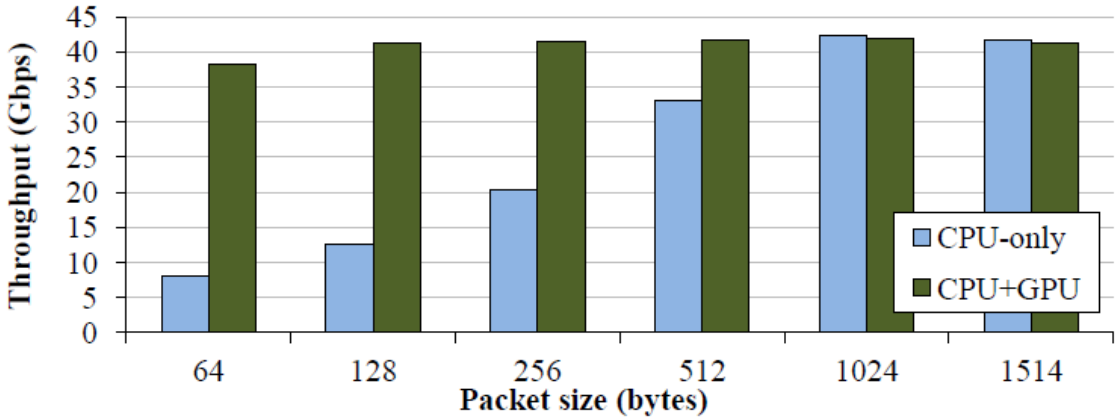

그림 46. IPv6 포워딩 성능.

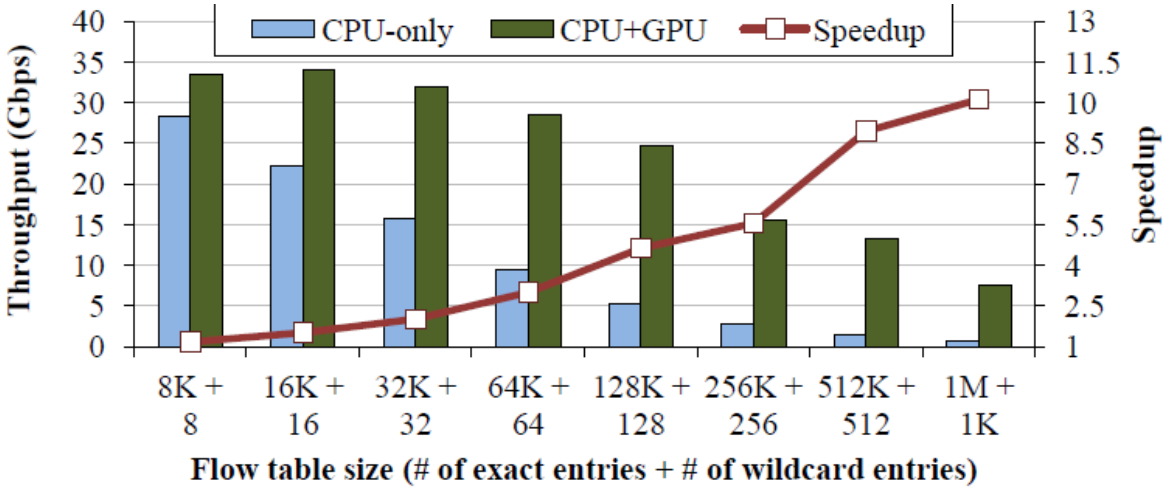

그림 47. OpenFlow 포워딩 성능.

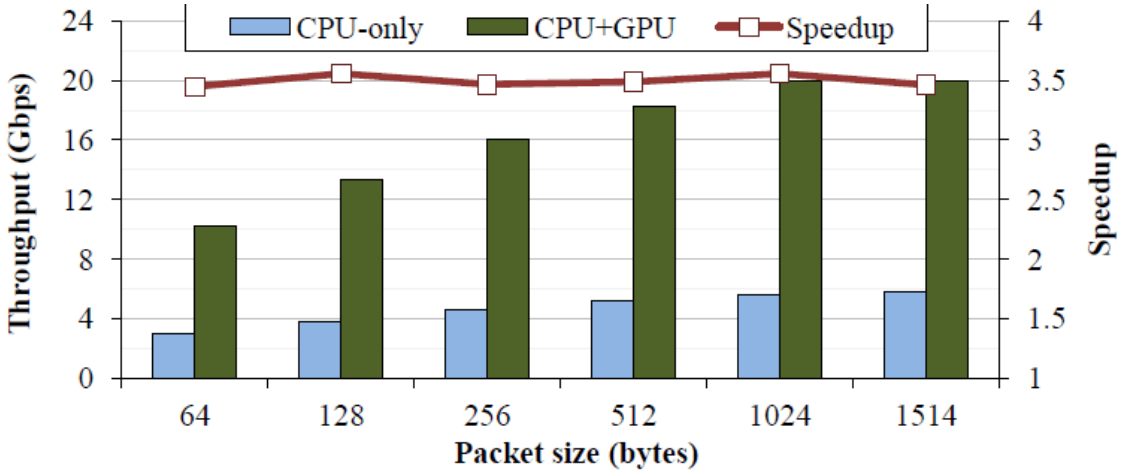

그림 48. IPsec 암호화 성능.

IPv4 트래픽 처리 성능을 보면 PacketShader 는 모든 크기의 패킷에 대해 대략 40 Gbps 의 성능을 보여주는데, 이는 Intel Research Berkeley 의 RouteBricks 시스템의 8.7 Gbps 의 대략 4.5 배의 성능이다. IPv6 의 경우에도 대략 40Gbps 의 성능을 보여주며, 특히 GPU 가속을 사용하는 경우 최대 5 배의 성능 향상을 보여준다. IPv6 포워딩의 성능을 수십 Gbps 급의 성능으로 구현하고 측정한 것은 본 연구가 세계 최초이다. OpenFlow 스위치 구현에서는 exact 매치 테이블과 wildcard 매치 테이블의 크기에 따른 성능을 보여준다. GPU 가속을 사용하는 경우 CPU 만을 사용하는 경우에 비해서 매우 높은 성능을 보여주며, 이는 테이블의 크기가 커질수록 차이가 커진다. 본 연구에서 구현된 OpenFlow 스위치는, Stanford 대학의 NetFPGA 기반 OpenFlow 스위치 구현보다 8 배 이상 빠른 것이다. IPsec 성능 그래프에서는 모든 패킷 크기에 대해서, GPU 가속이 IPsec 성능을 대략 3.5 배 정보 향상시켜주는 것을 알 수 있다. 본 연구 결과는 일반 PC 기반 서버에서 최대 20 Gbps 의 IPsec 성능을 보여주는데, 이는 시중의 수만 달러 상당의 하드웨어 기반의 IPsec 가속기보다도 더 나은 성능이다.

4.4.3 PacketShader 플랫폼의 의의

최근 미래 인터넷 테스트베드 연구의 중요성이 부각되고 있다. 기존의 인터넷에서는 불가능 했던 기능들을 미래 인터넷에서 가능하게 하기 위해 다양한 프로토콜을 실제로 실험해보고 할 수 있는 플랫폼에 대한 연구가 크게 주목을 받고 있다. 본 연구의 결과물은 미래 인터넷 테스트베드에서 네트워크 프로세싱을 할 수 있는 요소로써 고성능, 소프트웨어 수정의 용의함, 가격경쟁력 세 가지 항목에 대해 NetFPGA, ATCA 와 NP 의 조합, 일반 Software Router 에 비하여 모든 면에서 경쟁력을 가지고 있으므로 미래 인터넷 테스트베드의 기반 플랫폼으로써 활용이 가능하다. 학계에서 미래 인터넷에서 사용될 프로토콜을 실제로 구현하고 실험해 볼 수 있는 플랫폼을 한 차원 높은 성능으로 높여 주면서 쉽게 프로그래밍 할 수 있는 환경을 제공하여 많은 네트워크 연구가 우리의 플랫폼을 활용할 것으로 기대한다.

5 Use Cases

본 장에서는 상기한 자원들을 활용하여 실험자(사용자)들이 원하는 서비스(응용)을 제공하는 국내 활용 사례들을 살펴보고자 한다. 현재까지는 미래인터넷 주제에 관해서는 연구의 방향성 모색에 결부된 기본적인 수준에서의 활용 사례가 나타나는 초기 단계이어서 사례들이 제한적으로 발생되고 있다는 점에 유의하는 것이 필요하다.

5.1 FIRST@PC: 컴퓨팅/네트워킹 서비스들의 Balaced Composition

서비스 지향 컴퓨팅 패러다임에 기초한 미디어 중심 서비스 합성은 네트워크 미디어 중심적인 그리고 시간 제약적인 시스템을 유연하게 구성하는데 유망한 방법이다. 이런 관점에서 우리는 네트워크화된 임베디드 노드들의 이질적인 자원들 (예, MediaX, NetOpen)과 함께 FIRST@PC (Future Internet Research on Sustainable Testbed based on PC)를 구축하고 있다. 네트워크 임베디드 노드들의 자원들은 고도로 프로그램화되고 컴퓨팅과 네트워킹[2]의 차원에서 가상화가 된다고 가정한다. GIST 를 중심으로 충남대, 포항공대, 경희대가 함께 진행하고 있는 FIRST@PC 프로젝트에서 추구하는 연구 목표는 구축한 PC 기반의 테스트베드 위에서 사용자 원하는 미디어 중심적인 서비스 합성을 서비스 의존 그래프에 대한 컴포넌트 서비스들의 연결을 시간적으로 제어하는 적응화 프로세스로 표현하면서 실증하고자 한다.

[2] 현재는 컴퓨팅 자원들이 이미 상당히 프로그램화되고 (클라우드 컴퓨팅으로부터 관찰되는 것처럼) 부분적으로 가상화된다. 그러나 네트워킹 리소스들은 프로그램화가 시작되는 단계이며 아직 가상화가 되기위해서는 더 많은 시간이 필요하다.

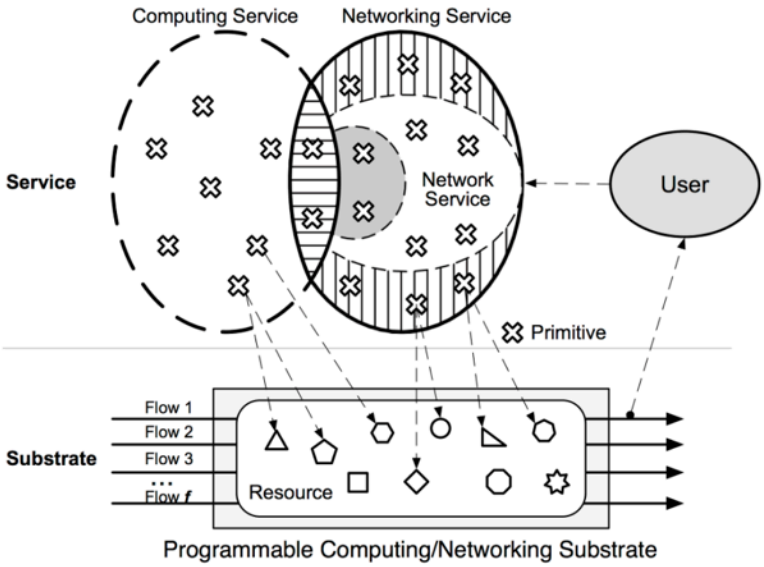

그림 49. 컴퓨팅/네트워킹 서비스들의 균형잡힌 합성에 대한 개념도.

그림 49 는 컴퓨팅/네트워킹 서비스들의 균형잡한 서비스 합성의 상위 수준 개념을 보여준다. 우리는 컴퓨팅 자원들과 엮인 기존의 컴퓨팅 지향 서비스들을 '컴퓨팅 서비스'라고 지칭하고 보통 자원들의 콘테이버 안으로 한정한다. 우리는 (웹 서버들과 클라이언트들의) 컴퓨팅 서비스들 사이에서 다양한 플로우들을 전달하도록 도와주는 네트워킹 중심적인 서비스들의 집합을 '네트워킹 서비스'라고 부른다. 플로우 수준의 네트워크 프로그램화를 활용하는, 이 서비스 합성은 기존 네트워크 서비스들에 의해 야기되는 기술적인 한계를 극복하는 시도를 한다. (오늘날 프로그램화를 지원하지 않는 네트워크 인프라에 기초한 기본적인 네트워크 연결성을 강조하기 위해 '네트워크 서비스'라고 부른다)

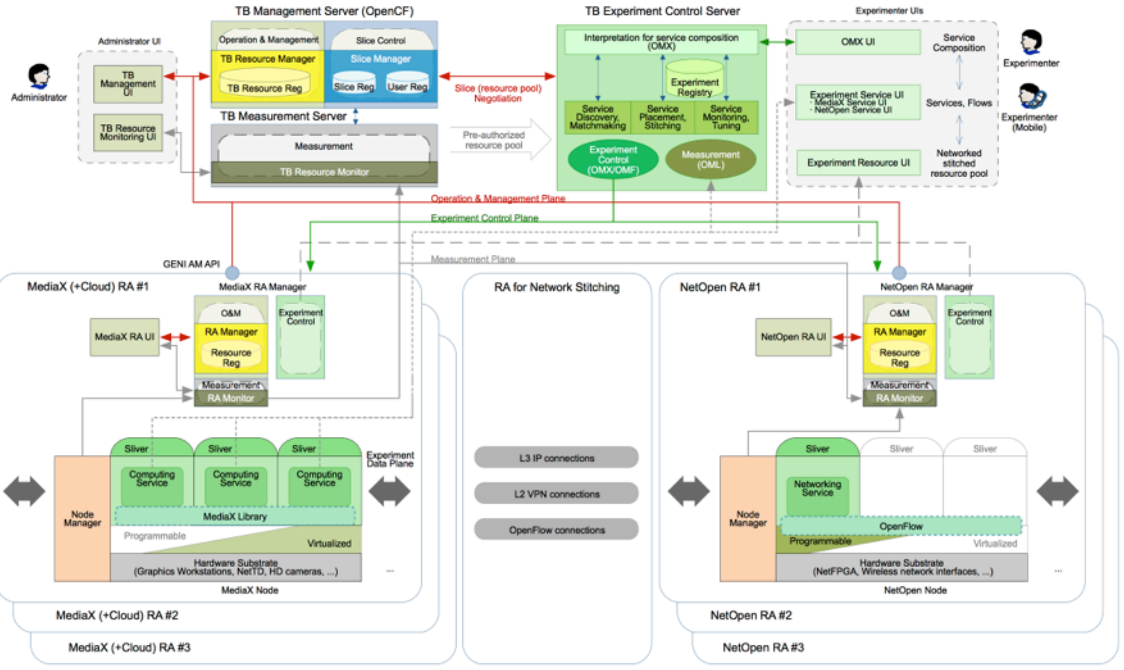

그림 50. FIRST@PC 테스트베드 플랫폼을 위한 구조.

효율적이고 유연한 서비스 합성을 위해 컴퓨팅/네트워킹 자원들을 균형있게 사용하는 개념을 검증하기 위해서, 그림 50 에서 보이는 바와 같이, 현재 우리는 FIRST@PC 테스트베드 플랫폼의 구조적인 설계를 하고 있다. 제안하는 플랫폼은 주로 RA, TB (testbed) 관리 서버, TB 실험 제어 서버들로 이루어 진다. 각 RA 는 물리적인 자원들을 캡슐화하고 자원들을 노출시기키 위해 slivers 라고 불리우는 원격에서 접근가능한 인터페이스들을 제공한다. 그림 50 에서 보이는 바와 같이, 우리는 컴퓨팅 지향 자원들을 대표하기 위해 MediaX(+Cloud) RA 를 만들며, 이는 프로그램화 가능하면서 가상화될 수 있다. 우리는 또한 네트워킹 지향 자원들을 대표하기 위해서 NetOpen RA 를 만들며, 이는 프로그램화 가능하지만 OpenFlow 기반 소프트웨어 정의 네트워크를 위해 (플로우스페이스 가상화 관점에서) 제한된 가상화를 지원한다. 더욱이 우리는 네트워크 스팃칭(network stitching)을 위한 RA 를 만들며, 이는 다른 RA 들의 노드들을 연결할 수 있다. RA 관리자는 실험자들과 관리자들의 권한에 따라 자원들에 대한 인가된 접근을 지원한다. TB 관리서버는 슬라이스 관리, 자원 관리, 자원 모니터링을 지원하여 관리자들이 테스트베드 자원들을 관리하도록 도와준다. 슬라이스 생성은 TB 관리 서버와 관련된 RA 관리자들에 의해

협력적으로 수행된다. TB 실험 제어 서버는 실험자들이 할당된 자원들을 사용하고 서비스 합성 실험들을 실행할 수 있도록 해준다. 슬라이스와 함께, 이것은 주어진 실험 기술서를 해석하고, 기술된 서비스 합성 프로세스를 수행한다. 또한 TB 실험 제어 서버는 실험 제어 평면을 통해 제어 이벤트들을 통지하고, 측정 평면을 통해 실험 상태와 결과를 수집한다. 실험자 UI 를 통해 실험자들은 할당된 자원 풀(pool)의 상태뿐만 아니라 서비스들의 상태를 점검할 수 있다.

미래지향적인 컴퓨팅/네트워킹 자원들을 위해 우리는 다음과 같은 미래의 시나리오를 상상한다. 자원들의 컴퓨팅/네트워킹 능력들의 긴밀한 통합을 강조하는 것은 우리로 하여금 단일 기기안에서 자원들을 위한 컴퓨팅/네트워킹 수요를 만족시키는 프로그램화 가능한 (그리고 궁극적으로 가상화되는) RA 들을 위한 SmartX 노드들의 개념적인 설계를 유도한다. 즉, SmartX RA 를 기반으로 우리는 단일 기기 안에서 (미디어 처리 능력이 결합된) 다양한 컴퓨팅/네트워킹 자원들을 수용하는 프로그램화 가능한 RA 들을 위한 지속적인 수요가 있을 것으로 생각한다. 기본적으로 SmartX 노드는 범용 CPU, GPU, 디지털 미디어 인터페이스와 같은 컴퓨팅 자원들을 포함하면서 (트랜스코딩을 지원하는 생방송 스트리밍과 같은) 가속화된 미디어 처리를 수행할 수 있어야 한다. SmartX 노드는 네트워킹 자원들과 함께 네트워크 스위치의 형태로 제공되며, 이것은 OpenFlow 프로토콜의 플로우 스페이스를 통해서 프로그램화되고 부분적으로 가상화된다. OpenFlow 가 지원되는 SmartX 노드들과 함께, 우리는 이들과 Click 소프트웨어 기반 확장가능한 모듈러 라우터들을 연결시킴으로써 네트워킹 자원들을 깊히 프로그램할 수 있다. 보다 상세히 말하자면, Click 요소들과 함께 in-network processing 을 가능하게 함으로써, 우리는 전달될 모든 패킷들을 위해 커스터마이즈된 컴퓨팅을 지원할 수 있다.

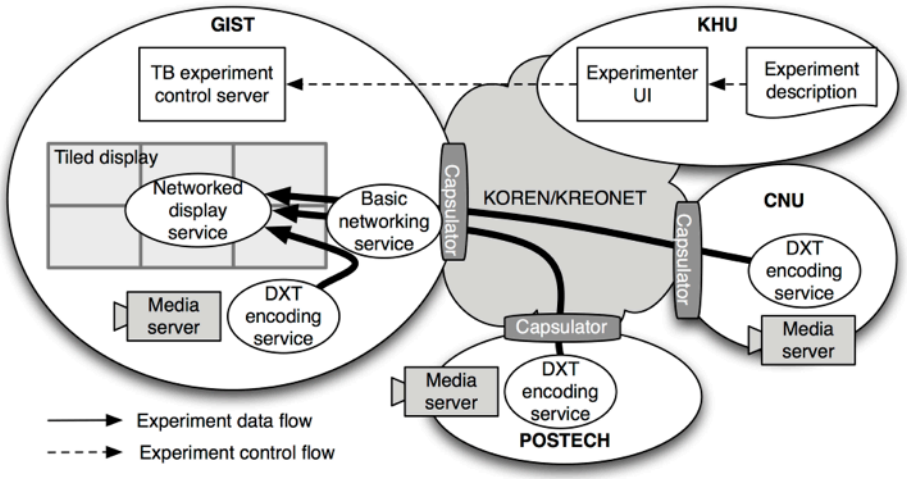

그림 51. 기본적인 수준의 서비스 합성을 실험하기 위한 FIRST@PC 테스트베드 프로토타입.

그림 53 은 FIRST@PC 테스트베드 프로토타입을 보여주며, 이것은 나선형 방식에 따라 구축되고 평가되는 중이다. 이 테스트베드 프로토타입에서, 네 개의 사이트들이 (KOREN, KREONET 과 같은) 연구망들을 통해 연결된다. 우리는 연결된 엣지 노드들의 물리적인 인터페이스들 위에 캡슐레이서들을 설치하고 IP 상의 이더넷을 실현하여, OpenFlow 가 가능한 네트워크들이 2 계층 네트워크 연결성을 가지도록 한다. 그런 다음, 미디어 콘텐츠들을 다루기 위해, 테스트베드 프로토타입은 실시간 미디어 플로우들을 공급하는 미디어 서버들, 플로우들을 위한 실시간 미디어 처리를 하는 커넥터와 어답터들, 초고해상도 비디오 월(wall)을 실현하는 네트워크 타일드 디스플레이를 포함한다.

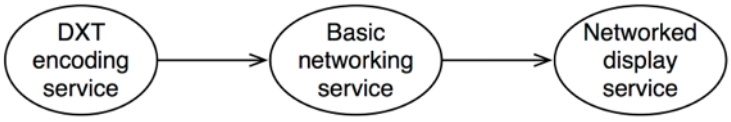

그림 52. 서비스 의존 그래프.

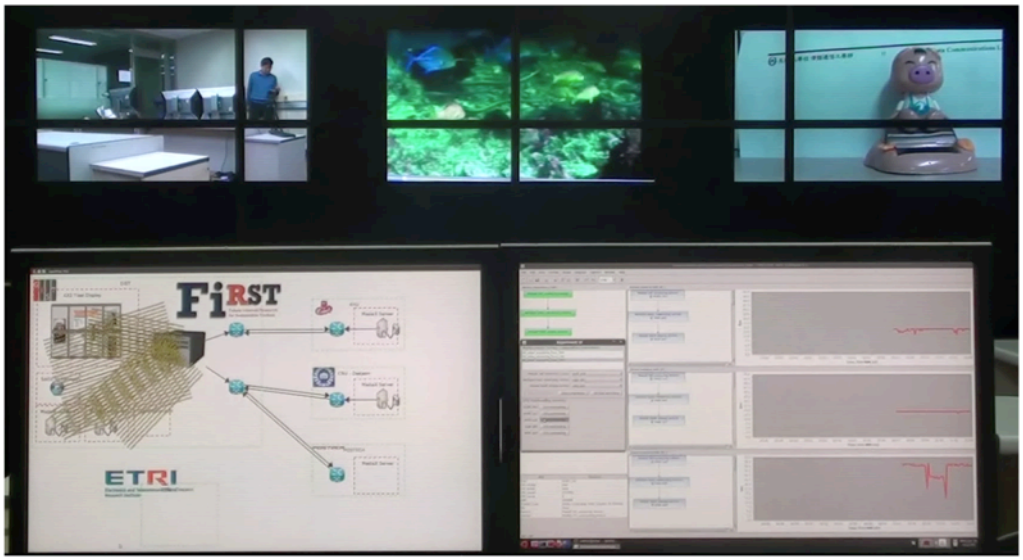

그림 53. 타일드 디스플레이에서 전달된 HD 영상 재현.

목표하는 실험으로써, 우리는 특정한 미디어 서버들과 네트워크 타일드 디스플레이 사이에 영상 플로우들을 완전하게 전송하고자 한다. 실험자 유저인터페이스를 사용하여, 실험자는 수동적으리지만 상호작용적으로 원하는 서비스들의 집합을 선택하고 서비스들을 연결한다. 이 HD 미디어 전달 실험을 위해, 실험자들은 목표로 하는 서비스 의존 그래프들을 그리며, 이것은 그림 52 과 같다. 그래프 탐색은 경량 압축된 300Mbps HD 영상을 공급하는 DXT (DirectX Texture compression) encoding service 에서 시작된다. 그런다음 basic networking service 를 통해서 미디어 플로우들이 전달되는데, 이 서비스는 OpenFlow 가 가능한 노드들 위에서 홉수 기반의 최단 경로를 찾아서 출발지 노드들에서 목적지 노드들로 이 플로우들을 전달한다. 그러면 미디어 플로우들은 networked (tiled) display service 에 도달하고, 이 영상들을 그림 53 에 보이는 바와 같이 디스플레이에 재현된다.

5.2 FiRST & FiRSTProNet

KOREN 은 광대역, 고품질의 국내외 연구시험망을 산•학•연에 제공하여 미래 네트워크 관련 기술의 시험검증과 첨단 응용분야 연구개발을 지원함으로써 연구개발촉진 및 국제공동연구 협력기반을 조성하기 위한 비영리 선도시험 네트워크 인프라이다. 이 미래네트워크 연구 시험망은 전국 6 개 대도시 지역(서울, 수원, 대전, 광주, 대구, 부산)을 10Gbps~20Gbps 로 연결하는 백본망을 구축 운영하고 있는데, 최근에는 미래인터넷 기술과 관련된 연구결과물들을 KOREN 에 설치하여 시험 및 검증 과정을 지원하고 궁극적으로는 시범사업 및 조기 상용화의 생태계 환경을 조성하기 위한 "KOREN 기반 미래인터넷 시험환경 초기인프라 구축" 사업을 진행한 바 있다. 이러한 KOREN 고도화 사업의 한 부분으로 서울 및 대전의 2 개의 PoP 에 FiRST 플랫폼을 구축하였다. 특히, 이 플랫폼들은 10G 인터페이스를 통해 POINTS(Packet Optical Integrated Transport System)라는 패킷-광 통합 장비의 클라이언트 형태로 서로 연결되어 있다. KOREN 에 설치된 FiRST 플랫폼의 구성도는 다음과 같다.

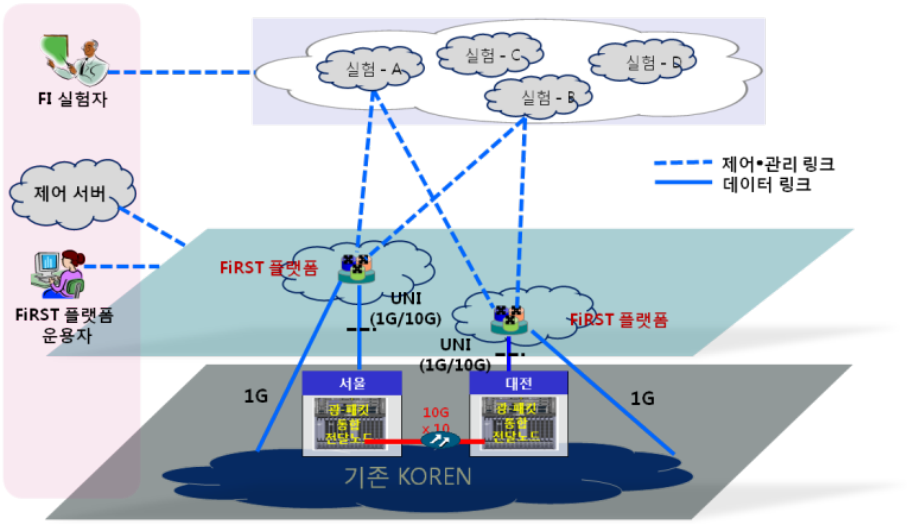

그림 54. KOREN 의 FiRST 플랫폼 구성도.

상기한 플랫폼을 구축하기 전에 자체적으로 PVMM, OpenFlow, Legacy IP 기능을 대상으로 1 차 기능 시험(단일 노드 수준) 및 2 차 기능 시험(노드간 연결 수준)을 진행하였고, KOREN 에 노드를

설치한 후 기능 시험과 서비스 연동 시험을 진행하였다. 서비스 연동 시험에서는 서울 및 대전 PoP(Point of Presence)에 서비스 환경(예: 대전에서 모니터 화면을 비디오 캠으로 촬영하여 기존 KOREN 을 통해 서울로 전송)을 미리 설치한 후, FiRST 플랫폼 상에서 TFTP 상의 브로드캐스팅 슬리버를 NP 블레이드 내의 PVMM 위로 동적 다운로딩하고 브로드캐스팅 서비스가 실행되는 과정을 시연하였다.

FiRST 프로젝트는 2012 년부터 2-단계로 진입할 예정이다. 2-단계에서는 최적화 작업과 성능 확장, 그리고 사용자 그룹 활성화, 그리고 도메인간 페더레이션에 초점을 두고 연구•개발을 진행할 것이다. 특히, 사용자 그룹 활성화를 위해 1-단계의 결과물을 연구개발망에 적용하여 미래인터넷 아키텍처 및 서비스 과제 등의 공통 테스트베드로써 활용할 수 있도록 지원할 계획이다.

5.3 PANTO

미래인터넷을 위한 새로운 구조의 현실적인 적용 여부를 검증하기 위해서는, 소규모 실험실 수준의 테스트 환경보다는 현재 인터넷에 최대한 가까운 규모의 실험 인프라가 필요하다. 모든 테스트베드 주체가 한 자리에 모여 통합 아키텍처 및 제어 인터페이스를 정립하지 않는 한, 자신의 테스트베드 입장에서는 다른 테스트베드를 블랙 박스로 볼 수밖에 없으며, 연동을 위한 최소한의 API 와 RSpec 차원의 접근이 가능한 현실이다. PANTO 는 한 기관 또는 국가 단위로 구축된 테스트베드를 하나의 논리적인 거대 실험 인프라로 연동하기 위해, 이종 테스트베드를 포괄하는 공통 의미 모델과 각 제어 인터페이스의 변환을 중심으로 구성되는 페더레이션 플랫폼이다 [54].

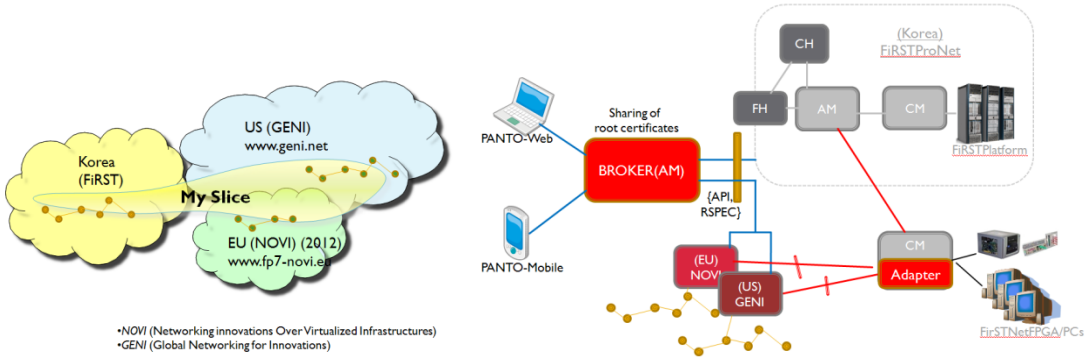

그림 55. PANTO 연동 대상 및 아키텍처.

공통 의미 모델은 GENI 의 SFA(Slice-based Federation Architecture)[10]과 FiRSTProNET 의 제어 프레임워크를 비롯한 다양한 모델에 공통적으로 적용되는 기반 아키텍처를 담고 있으며, 제어 프레임워크의 API 와 Rspec 을 비롯한 세부 인터페이스가 상이하더라도, 공통 의미 모델을 기반으로 기능상 변환 가능한 테스트베드는 모두 연동(페더레이션)의 대상이 될 수 있다. 현재 PANTO 는 분산 시스템의 표준 언어로 정착한 XML 을 공통 의미 모델 및 변환을 위한 최종 표현 언어로 사용하고 있으며, 다행히 FiRSTProNET 이나 GENI 를 비롯한 많은 테스트베드 인프라가 XML 또는 XML 로 변환 가능한 포맷을 AM API 및 Rspec 에 사용하고 있다. 그러나 XML 기반 표현은 어디까지나 문법적인 호환성을 보장할 뿐, 각 테스트베드에서 표현하는 자원과 API, 그리고 그 밖의 여러 가지 부가 기능과 Policy 가 가지는 의미를 고려한 변환이 필요하다. PANTO 에서는 자신의 테스트베드 제어 인터페이스와 연동 대상 제어 인터페이스를 모두 이해하는 사람이 변환 규칙을 작성하고, 두 테스트베드 연동 과정에서 의미 손실 및 변질이 없는지를 검사한 후, 최종적으로 XSLT 포맷으로 서술하여, 변환 규칙 정의는 수동 작업으로 이루어지지만, 일단 정의된 변환 규칙 적용 과정은 자동으로 실행되는 구조를 바탕으로 페더레이션의 기능적 연동 메커니즘을 구현한다.

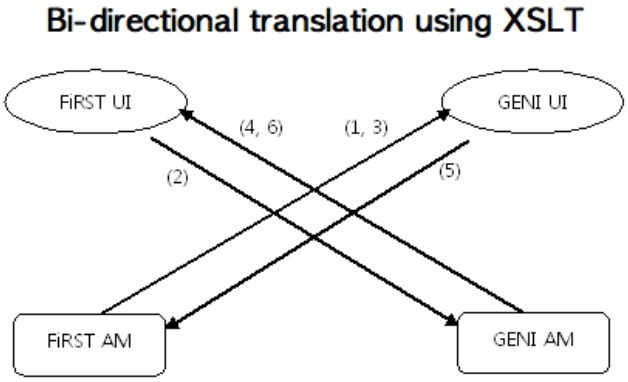

그림 56. XSLT를 이용한 양방향 변환.

FiRST v2.1 and ProtoGENI v2 Advertisement RSpecs for a PANTO Resource

```
<rspec
   xsi:schemaLocation="http://www.protogeni.net/resources/rspec/2
http://www.protogeni.net/resources/rspec/2/ad.xsd"
type="advertisement"
    generated="2011-11-18T07:35:33Z"          valid_until="2011-11-
 18T07:35:33Z"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://www.protogeni.net/resources/rspec/2">
<node
   component_manager_id="urn:publicid:IDN+etri-
cml.kreonet.net+authority+cm"
   component_name="pc0" exclusive="true"
   component_id="urn:publicid:IDN+etri-cml.kreonet.net+node+pc0">
   <hardware_type name="pc"/>
   <sliver_type name="xen-vm">
      <disk_image
         name="urn:publicid:IDN+etri-cml.kreonet.net+image+ds1"
         os=" CentOS 5.5"
         version="" description="Xen VM for CentOS 5.5"
         default="true"/>
   </sliver_type>
   <available now="true"/>
   <location          country="KR"          latitude="36.381084"
longitude="127.365210"/>
   <interface
       component_id="urn:publicid:IDN+etri-
cml.kreonet.net+interface+pc0vif0"/>
</node>
</rspec>
```

GENI Format

```
<rspec
   xmlns="http://www.etri.re.kr/first/2011"
   valid_until="2011-07-11T13:15:38.617+09:00"
   ticket_urn="urn:publicid:IDN+first.kr+ticket+myticket"
   slice_urn="urn:publicid:IDN+first.kr+slice+myslice"
   namespace="http://localhost/test" rspec_type="manifest">
<node_first_pcs><node_first_pc
   reservable="false"
   sliver_urn="urn:publicid:IDN+first.kr+sliver+atcal.etri.re.kr:12345678"
   aggregate_urn="urn:publicid:IDN+first.kr+aggregate+first.kr"
   component_type="PC"
   component_urn="urn:publicid:IDN+first.kr+component+pc1.etri.re.kr">
   <location longitude="100.0" latitude="100.0"/>
     <pc>
       <cpus><cpu usage="10000" speed="10000"/></cpus>
       <storages><storage usage="10000" capacity="10000"/></storages>
       <memorys><memory usage="10000" capacity="10000"/></memorys>
       <interfaces><interface usage="1000"
          interface_urn="urn:publicid:IDN+first.kr+interface+linecard-
interface">
            <address mac="00:00:00:00:00" ipv4="127.0.0.1"/>
       </interface></interfaces>
       <software description="This software is ...."
          boot="boot" locator="locator"
          software_urn="urn:publicid:IDN+first.kr+software+etri-cml:pc0"/>
       <configuration extra="etra" dhcp="true" vncViewer="false" vnc="false"
          virtualizationType="para" domainName="pc0"/>
     </pc>
</node_first_pc></node_first_pcs>
<links><link
   link_urn="urn:publicid:IDN+first.kr+link+mylink0"
   link_type="ethernet">
   <interfaces><interface
      interface_urn="urn:publicid:IDN+first.kr+interface+myinterface0">
      <address mac="00:00:00:00:00:00" ipv4="129.0.0.0"/>
   </interface>
   <interface
      interface_urn="urn:publicid:IDN+first.kr+interface+myinterface1">
      <address mac="00:00:00:00:00:01" ipv4="129.0.0.1"/>
   </interface></interfaces>
</link></links>
</rspec>
```

FiRST Format

그림 57. PANTO 자원에 대한 RSpec.

PANTO 의 API/Rspec 변환 기능의 1 차 버전은 FiRSTProNET 과 GENI(ProtoGENI) 의 제어 인터페이스를 단순한 문법 변환이 아닌, 서로간의 의미를 보존하는 1:1 양방향으로 변환 구조로

설계됐으며, 변환 규칙의 최종 포맷은 현재 XSLT 를 사용하고 있다[55]. 이러한 1:1 양방향 변환 어댑터는 추후 M:N 매핑을 지원하는 브로커 구조로 확장할 계획이다.

PANTO 변환 어댑터의 기본 개념과 예제, 그리고 페더레이션 된 모든 테스트베드의 자원을 조회하고 슬라이스를 생성하는 일관된 사용자 경험을 제공하기 위한 PANTO GUI 는 PANTO Wiki 를 통해 지속적으로 공개하고 있다 [56].

그림 58. PANTO Mobile 과 PANTO Web GUI

6 Conclusion

본 문서에서는 미래인터넷 테스트베드에서 프로그램화 및 가상화 가능한 실험 장비들에 관한 요구사항과 기술개발 동향에 대해 살펴보았다. 먼저 안정적인 실험자원 공급을 위한 요구사항을 도출하고, 슬라이스 기반의 실험자원 관리에 관한 동향과 이슈들을 분석하였다. 슬라이스 기반의 실험자원 공급을 위한 기본 구조와 방법론에 대해서는 정리가 되어가고 있으며, 이를 장기적으로 유지하려는 노력들이 진행되고 있다. 하지만 실험자원에 변화가 생겼을 때, 슬라이스를 안전하게 유지하면서 성능 저하를 막는 연구가 추가로 필요하다. 다음으로, 가상화 지원 컴퓨팅 자원을 공급하는 클라우드 컴퓨팅에 대한 기본 개념과 트랜드를 살펴본 뒤, 미래인터넷과 클라우드 컴퓨팅의 연계 방안을 소개하였다. 아직 클라우드 기반의 가상 머신을 미래인터넷의 컴퓨팅 자원으로 활용하는 수준에서 통합연구가 진행되고 있다. 최근에는 클라우드 내부에서 트래픽 부하 분산을 위해 SDN 을 활용하는 연구가 활발히 진행되고 있어, 효율적인 컴퓨팅/네트워킹 자원 공급을 위한 상승효과가 기대된다. 마지막으로 SDN 개념을 수용하는 미래인터넷 네트워킹 실험자원들에 대한 구현 및 사용 현황에 대해 살펴보았다. OpenFlow 의 프로그래밍 개념을 수용하는 네트워킹 테스트베드들이 설치/운용/통합되고 있으며, 네트워크 가상화를 지원하는 가상 라우터와 고속 소프트웨어 라우터들도 활발하게 개발되고 있다. 이와 같은 컴퓨팅/네트워킹 실험 장비들은 미래인터넷 테스트베드가 창의적인 네트워킹 개념들을 실험해 볼 수 있는 가상 연구실 역할을 할 것이다. 보다 다양한 실험 요구사항들을 개방적으로 수용할 수 있는 연결용 네트워크와 이에 연동된 컴퓨팅/네트워킹 장비들, 그리고 이를 지원하는 각종 실험용 소프트웨어들에 대한 지속적인 관심과 투자를 기대해 본다.

7 References

[1] 이동훈, 박주원, 김종원, "실감미디어, 서비스 합성, 프로그래머블 인프라에 근간한 미래인터넷 서비스 프레임워크," 한국정보과학회 논문지, 제 28 권, 제 1 호, pp. 31-40, 2010.

[2] J. Roberts, "The clean-slate approach to future Internet design: A survey of research initiatives," Annals of Telecommunications, vol. 64, no. 5-6, pp. 271-276, 2009.

[3] 신명기, "미래인터넷 기술 및 표준화 동향," 전자통신동향분석, 제 22 권, 제 6 호, pp. 116-128, 2007 년 12 월.

[4] P. Stuckmann, R. Zimmermann, "European research on future Internet design," IEEE Wireless Communications, vol. 16, no. 5, pp. 14-22, Oct. 2009.

[5] GENI Planning Group, "GENI Design Principles," IEEE Computer , vol. 39, no. 9, pp. 102-105, Sep. 2006.

[6] A. Gavras, A. Karila, S. Fdida, M. May, and M. Potts, "Future internet research and experimentation: the FIRE initiative," ACM SIGCOMM Computer Communication Review, vol. 37, no. 3, pp. 89-92, 2007.

[7] T. Aoyama, "A new generation network: Beyond the Internet and NGN," IEEE Communications Magazine, vol. 47, no. 5, pp. 82-87, May 2009.

[8] P. Barford, M. Blodgett, M. Crovella, and J. Sommers, "Requirements and Specifications for the Instrumentation and Measurement Systems for GENI," Tech. Report, May 2009.

[9] R. Dutta, G.N. Rouskas, I. Baldine, A. Bragg, D. Stevenson, "The SILO Architecture for Services Integration, controL, and Optimization for the Future Internet," in Proc. of IEEE International Conference on Communications, pp. 1899-1904, Jun. 2007.

[10] L. Peterson, R. Ricci, A. Falk, and J. Chase, "Slice-based Federation Architecture," 2010, working draft V2.0.

[11] GENI Plastic Slices, http://groups.geni.net/geni/wiki/PlasticSlices

[12] L. Peterson, R. Ricci, A. Falk, and J. Chase, "Slice-based Federation Architecture," 2010, working draft V2.0.

[13] M. Yuen, "GENI in the Cloud," M.S. Thesis, University of Victoria, 2010.

[14] D.F. Ferraiolo and D.R. Kuhn, "Role Based Access Control", in Proc. of National Computer Security Conference, Oct. 1992.

[15] M. Blaze, J. Feigenbaum, J. Ioannidis, "The KeyNote Trust-Management System Version 2", IETF RFC 2704, Sep. 1999.

[16] A. Pimlott and O. Kiselyov, "Soutei, a Logic-Based Trust-Management System", in Proc. of Int. Symp. on Functional and Logic Programming. Fuji-Susono, Japan, Apr. 2006.

[17] D. Baier, V. Bertocci, K. Brown, and E. Pace, and M. Woloski, "A Guide to Claims-Based Identity and Access Control," Microsoft, 2010.

[18] Alan H. Karp, Harry Haury, Michael H. Davis, "From ABAC to ZBAC: The Evolution of Access Control Models," HP Labs, 2009.

[19] J. Li and A. H. Karp, "Access Control for the Services Oriented Architecture", in Proc. of ACM Workshop on Secure Web Services, Nov. 2007.

[20] L. Fang and D. Gannon, "XPOLA - An Extensible Capability-based Authorization Infrastructure for Grids", in Proc. of PKI R&D Workshop: Multiple Paths to Trust, Apr. 2005

[21] Open Networking Foundation, http://www.opennetworking.org/fa

[22] Open Networking Summit, http://www.opennetsummit.org/

[23] Frenetic, http://www.frenetic-lang.org/

[24] Christopher Monsato, et al., A Compiler and Run-time System for Network Programming Languages, POPL'12, January 25-27, 2012, Philadelphia, PA, USA.

[25] The Yale Haskell Group, http://haskell.cs.yale.edu/

[26] OpenFlow Switch Specification Version 1.0.0., OpenFlow, Dec. 2009.

[27] OpenFlow Switch Specification Version 1.1.0., OpenFlow, Feb. 2011.

[28] N. Handigol, M. Flajslik, S. Seetharaman, R. Johari, and N. McKeown, "Aster*x: Load-Balancing as a Network Primitive," in *Proc. Architectural Concerns in Large Datacenters (ACLD '10)*, Jun. 2010.

[29] K.-K. Yap, T. Huang, M. Kobayashi, M. Chan, R. Sherwood, G. Parulkar, N. McKeown, "Lossless handover with n-casting between WiFi-WiMAX on OpenRoads." in *Proc. Annual Int. Conf. on Mobile Computing and Networking (MobiCom '09)*, Sept. 2009

[30] B. Heller, S. Seetharaman, P. Mahadevan., "ElasticTree: Saving energy in data center networks," in *Proc. USENIX Symp. on Networked Systems Design and Implementation (NSDI'10)*, Apr. 2010.

[31] T. Huang, K.-K. Yap, B. Dodson, M. S. Lam, N. McKeown, "PhoneNet: A phone-to-phone network for group communication within an administrative domain," in *Proc. ACM SIGCOMM Workshop on Networking, Systems, and Applications on Mobile Handhelds (MobiHeld '10)*, Aug. 2010.

[32] PLANETLAB, http://www.planet-lab.org/

[33] Cisco's Proposal on OpenStack: Naas, http://wiki.openstack.org/NetworkContainers/

[34] 이성원, 김현준, "모바일 클라우드 컴퓨팅 소프트웨어 기술," 한국통신학회지, 제 27 권, 제 6 호, pp. 35-43, 2010 년 6 월.

[35] Eucalyptus Systems, http://www.eucalyptus.com/

[36] Daniel Nurmi, and etc., "The Eucalyptus Open-source Cloud-computing System", Cloud Computing and Its Applications 2008, Oct. 2008.

[37] The Remote Framebuffer Protocol, http://tools.ietf.org/html/rfc6143

[38] IEEE P2301 – Guide for Cloud Portability and Interoperability Profiles (CPIP), http://standards.ieee.org/develop/project/2301.html
[39] IEEE P2302 – Standard for Intercloud Interoperability and Federation (SIIF), http://standards.ieee.org/develop/project/2302.html
[40] Avetisyan, A., and etc., "Open Cirrus: A Global Cloud Computing Testbed," IEEE Computer, vol 43, no 4, Apr. 2010.
[41] Open Cirrus, http://opencirrus.org/
[42] NEuca, https://geni-orca.renci.org/trac/wiki/NEuca-overview
[43] Amazon Web Services, http://aws.amazon.com/
[44] Amazon Elastic Compute Cloud (Amazon EC2), http://aws.amazon.com/ec2/
[45] Amazon Simple Storage Service (Amazon S3), r
[46] Amazon Elastic Block Store (Amazon EBS), http://aws.amazon.com/ebs/
[47] Google App Engine, http://code.google.com/intl/ko-KR/appengine/
[48] 한재선, "사례 위주로 살펴 본 클라우드 컴퓨팅 플랫폼 기술," 마이크로소프트웨어, 1 월호 2009 년.
[49] Windows Azure, http://www.windowsazure.com/
[50] OpenStack Compute, http://www.openstack.org/projects/compute/
[51] OpenStack Object Storage, http://www.openstack.org/projects/storage/
[52] Hyunjun Kim, Sungwon Lee, "FiRST Cloud: Interworking between Cloud Computing and Future Internet over FiRST Project.", CUTE 2011, Dec. 2011.
[53] The GENI Aggreagate Manager API, http://groups.geni.net/geni/wiki/GeniApi/.
[54] K. Nam, M. Shin, H. Kim, S. Jung, "Federating Future Internet Testbeds: An Adapter-based Approach," CFI, Seoul, Korea, June 2011.
[55] JANUS, http://janus.kreonet.net/.
[56] Panto, http://panto.kreonet.net/.