http://dx.doi.org/10.7840/kics.2015.40.10.1947

국제 연동 테스트베드에서 데이터 전송 지연을 위한 CCN-Helper 성능 분석

최 원 준◆, 람 닉*, 석 우 진°

CCN-Helper Performance Analysis for Data Transmission Delay over Federated Testbed Environment

Won-jun Choi◆, Ramneek*, Woo-jin Seok°

요 약

Content Centric Networking 은 많은 연구자들이 관심을 가지고 연구되고 있는 미래 인터넷 중의 하나이다. 가상 네트워크 환경을 지원한 빠른 네트워크의 성장은 폭넓은 서비스를 가능하게 하였고 네트워크가 발달할수록 원거리 고 대역폭에서의 성능 또한 중요한 이슈가 되었다. 원거리 규모의 테스트 환경에서는 원거리의 특성으로 인하여 데이터 전송 지연이 발생할 수 있고 CCN 노드 간에는 혼잡 또는 낮은 연결 속도로 인하여 데이터 전송이 일어날 수 있다. 본 논문에서는 이러한 네트워크 환경에서 CCN-Helper 프로토콜을 제안함으로써 시뮬레이터를 통한 성능 테스트와 국제적으로 연동된 네트워크 테스트베드에서 이기종 머신 간의 데이터 전송 평가를 CCN-Helper 프로토콜을 통하여 검증해 보고자 한다. 국제 연동 네트워크 테스트 환경은 국내의 KREONET 과 벨기에의 iLab.t 네트워크를 사용하였다.

Key Words : CCN, Federation, GENI, ProtoGENI, Cloud Computing

ABSTRACT

CCN networking is one of the future internet. With the rapid development of network platforms emerged a wide range of services through a virtual network environment. The performance of CCN in the high bandwidth delay is important to provide various services between nations. We can check the high bandwidth delay product in federation. Federation between the nations has the advantage of being effective to do a various services for the users, and the remote heterogeneous physical hardware engine supports various environments. Data transmission delay can be occur in long distance network even though the network is federated network. In this paper, we propose CCN-Helper protocol for data transmission delay in ling distance networks. We connect from KREONET in Korea to iLab.t in Belgique in order to test CCNx over heterogeneous federated environment.

Ⅰ. 서 론

네트워크의 기술이 발전함에 따라 네트워크 IP 기반의 기술이 아닌 콘텐츠의 정보를 사용한 네트워크 기술이 이슈가 되고 있다. 그 대표적인 예로 ICN(Information Centric Network), CDN(Content

※ 본 연구는 KISTI 연구과제(사용자 기반 연구망 플랫폼 서비스 개발 및 적용: K-14-L01-C03-S03) 지원으로 수행되었습니다.
◆ First Author : Korea University of Science and Technology, cwj@ust.ac.kr, 학생회원
° Corresponding Author : Korea Institute of Science and Technology Information, wjseok@kisti.re.kr, 정회원
* Korea University of Science and Technology, ramneek@ust.ac.kr, 학생회원
논문번호 : KICS2015-07-236, Received July 24, 2015; Revised September 7, 2015; Accepted October 20, 2015

Delivery Network), NDN(Named Data Network)등이 있으며, 이러한 기술들은 주로 목적지 기반 주소 네트워킹이 아닌 콘텐츠 즉, 데이터를 요구하는 사용자에 중심이 된 네트워킹 구조를 이루고 있다. 콘텐츠 이름으로 네트워킹을 실현하는 기술 중에 차세대 미래 인터넷으로 떠오르고 있는 기술 중의 하나인 CCN(Content Centric Network)은 유무선 네트워크에서 원하는 사용자에게 필요한 자료만 전송한다는 기본적인 생각을 현실화 시킨 기술이라고 볼 수 있다. 초고속 인터넷과 컴퓨터 하드웨어의 발달로 원거리에서 사용자에게 서비스하기 위한 성능 평가는 중요한 이슈가 되었다. 우리는 원거리 네트워크를 Federation 환경을 통하여 구현해 볼 수 있으며 이러한 환경에서 적용될 수 있는 서비스는 다음과 같다. 컴퓨팅 서비스와 스토리지 서비스를 통하여 사용자에게 서비스되는 대표적인 서비스는 SaaS(Software as a Service), PaaS(Platform as a Service), IaaS(Infrastructure as a Service)로 나눌 수 있다. IaaS는 서버, 스토리지, 네트워크를 가상 환경으로 만들어, 필요에 따라 Infra 자원을 사용할 수 있게 제공하는 서비스 형태이며, PaaS는 필요한 개발 요소들을 웹에서 쉽게 빌려 쓸 수 있는 서비스이다. SaaS는 IaaS와 PaaS 위에 올라가는 소프트웨어이다. 현재 대표적인 open source cloud 플랫폼은 오픈스택 (OpenStack)[1,5], 클라우드 스택 (CloudStack), 유칼립투스 (Eucalyptus)[4], 오픈네뷸라 (Open Nebula)[4] 이 있다. 클라우드 스택 (CloudStack)[2]은 미국의 Cloud.com사가 개발한 오픈소스 클라우드 환경 구축 소프트웨어이며, 유칼립투스 (Eucalyptus)[3]는 클라우드 컴퓨팅 시스템의 활발한 연구를 위해 미국 UC 산타바바라 대학에서 시작된 오픈소스 프로젝트이다. 오픈네뷸라(OpenNebula)[4]는 2008년 3월에 TP1을 최초로 릴리즈하면서 탄생하였으며, 현재는 C21Labs에서 운영 및 유지보수를 하고 있다. 오픈스택(OpenStack)는 2010년 7월 NASA와 Rackspace가 같이 시작한 오픈 소스 프로젝트로써, 컴퓨팅 서비스인 Nova, 오브젝트 스토리지 서비스인 Swift, 이미지 서비스인 Glance로 구성이 되어 있으며, 이 외에도 네트워크를 담당하는 Quantum, 인증을 담당하는 Keystone, 블록 스토리지 서비스인 Cinder, 웹UI인 Horizon이 있다. 클라우드 컴퓨팅 기술을 통해서 사용자 요구에 대한 서비스를 지원하고 있으며 서버와 스토리지 그리고 네트워크를 가상화 환경으로 만들어, 필요에 따라 인프라 자원을 사용하게 만든 IaaS(Infrastructure as a Service)의 종류도 다양화 되고 있다. 클라우드 컴퓨팅 오픈소스 프로젝트인 오픈스택(OpenStack)과 국내의 KT Ucloud[8], 클라우드 스택, 아마존 웹서비스 등이 대표적이며, 이와 같은 리소스 자원의 장점은 급변하는 비즈니스에 신속하게 대응할 수 있도록 컴퓨팅 자원의 유연한 확장과 축소가 가능하다는 점이다. 국내 최초의 테스트 베드 망 관리 시스템인 dvNOC(Distributed Virtual Network Operations Center)[26]는 독립적인 자원관리가 가능한 협력 구조로 되어 있다. 이러한 이기종 자원 서비스가 증가할수록 사용자 관리와 리소스 자원관리가 어렵고 이를 위해 추가로 중복된 개발 비용이 소요된다.[19] 이러한 문제는 국제간 컴퓨팅 자원의 활용에서도 문제화 되었다. 이기종 클라우드에 중복된 데이터가 발생하여 필요 없는 자원의 낭비로 추가적인 비용이 발생하게 되고 다양한 플랫폼을 관리할 수 있는 모니터링 시스템이 필요하다.[20,21] 따라서 이러한 이슈를 해결해 보고자 SFA(Slice-based Federation Architecture)를 기반으로 한 테스트베드 간의 연동을 페더레이션이라는 용어로 사용되게 되었다. 페더레이션의 실현이 가능하도록 하기 위해서 연동 서비스가 가능한 클라우드 시스템[21]와 단일 인터페이스에서 다중 네트워크 가상 플랫폼을 지원하는 페더레이션[22] 등 다양한 연구가 진행되고 있다. GENI[5]는 Programmability, Resource Sharing(Virtualization), Federation, Slice-Based Experiment라는 핵심 개념을 토대로 실제 시스템을 구현해 나가는 나선형 개발 모델로 진행되고 있다. 독립적으로 운영되는 이기종 플랫폼의 생태계를 구성하기 위하여, 슬라이스 기반 페더레이션 아키텍처(SFA)를 정의한다. 미국과 유럽은 GENI(Global Environment for Network Innovation)[19]를 기반으로 XML포멧의 RSpec을 사용하는 PlanetLab[11], ProtoGENI[5], PDF기반의 RSpec을 사용하는ORCA[19] 등의 프로젝트를 통해 페더레이션 아키텍처를 정의해 나가고 있으며 FED4FIRE[12], Fibre[7], SmartFire[13] 프로젝트를 통해서 페더레이션 구현을 현실화 하고 있다. 이와 같은 원거리 네트워크 환경에서는 소비자가 데이터를 다운로드 할때에 전송 지연이 발생할 수 있다. 전송 지연이 발생하는 원인은 다양하지만 일반적인 CCN에서 혼잡 또는 낮은 링크율로 인하여 데이터 전송 지연이 발생할 수 있고[17] 원거리의 특성으로 데이터를 전송하는 도중에 전송 지연이 발생할 수 있다.[18] 본 논문에서는 이러한 문제로 발생하는 데이터 전송 지연 문제에 도움을 주기 위하여 CCH-Helper 프로토콜을 사용한다. 그 동안 CCN에서 소비자의 처리량을 향상시키기 위한 다양한 연구가 진행되었는데 그 예는 다

음과 같다. 높은 처리량과 낮은 지연시간을 추구하여 CCN 패킷을 처리하는 방법을 제안한 연구[23]가 있으며, 사용자에게 수신되는 지연 시간을 감소시키기 위하여 캐쉬 파이프라인을 조절한 연구[24]가 있다. 또한, 포워딩 전략, 캐쉬의 분리, 공평성, 오버로드 제어, 라우터 전략 등의 CCN 방안 연구[25], 캐쉬 PIT를 혼잡 상황에서 다양하게 조절함으로써 작업 로드를 향상시킨 연구[26] 그리고 다운링크와 업링크의 혼잡을 피하고 패킷의 전송 시간을 최소화하여 처리량을 향상시킨 연구[27]가 있다. 본 논문에서는 시뮬레이션을 통하여 CCN-Helper 프로토콜의 성능을 테스트해 보고 국제간 테스트베드의 연동을 설계 및 구축하여 소비자의 Data 패킷 다운로드의 처리량을 증대시키기 위하여 CCN-Helper 프로토콜의 성능을 검증해본다. 이를 위하여 세계적 수준의 고성능 하이브리드 네트워크 인프라인 KREONET[9]을 활용하여 구현해 보고자 한다. 한국과학기술정보연구원에서 운영중인KREONET은 세계 11개국과 협력하여 글로벌 환경 네트워크인 GLORIAD[10]를 구축하여 국제 연구 망 서비스를 제공한다. GLORIAD는 한국, 미국, 중국, 러시아. 캐나다, 네덜란드 등 6개국이 공동으로 참여하여 세계 최초의 10기가급의 글로벌 연구 망을 제공하고 대용량 데이터 전송을 필요로 하는 첨단과학기술 분야의 과학기술협업 연구망이다.

Ⅱ. CCN-Helper 프로토콜

본 논문에서 제안하는 CCN-Helper 프로토콜은 주변의 CCN 노드에서 Data 청크 파일을 다운로드 하는데 도움을 주는 프로토콜이다. CCN 노드는 수요자가 요청하는 콘텐츠를 청크 단위로 전송하게 된다. 청크 단위의 패킷을 요청하기 위해서 수요자는 Interest 패킷을 네트워크에 전송하게 되고 네트워크에 있는 중간 CCN 노드에서는 Interest 패킷의 정보를 받아서 CS에 관련 데이터가 없으면 다시 다음 CCN 노드로 포워딩을 한다. 이때, 중복 Interest 패킷일 경우와 dummy 파일이 CCN 노드에 수신될 경우에는 Prefex 확인 단계에서 걸러지게 된다. 하지만, 제안하는 방법은 중복 Interest 패킷이더라도 요청하는 청크 데이터의 시퀀스가 다르게 되면 정보를 저장한다. 이렇게 하는 이유는 주변 CCN 노드에서도 같은 prefix에 대해서 다운로드 처리에 도움을 주게 하기 위해서이다. Interest 패킷이 주변 CCN 노드에 전송되면 속도가 빠른 링크의 Interest 패킷이 먼저 다음 CCN 노드에 도착하게 되고 속도가 느린 링크에 속한 Interest 패킷은 CCN 노드에서 제외되어 처리된다. 기존 방법은 이렇게 네트워크의 속도가 가장 최적화된 링크로만 데이터를 전송한다. 하지만, 네트워크에 혼잡이 발생하게 되면 오히려 처리량의 저하를 불러올 수도 있다. 이럴 때에는 주변 CCN 노드에서 CS에 여유용량이 허용된다면 해당 prefix를 다운로드하는데 도움을 주도록 설정할 수 있다. 만약, 주변 CCN 노드에서 Data 패킷을 생산자로부터 다운로드 하는데 도움을 준다면 소비자가 수신하는 Data 패킷의 처리량은 증가하게 될 것이다.

그림. 1은 CCN 노드에서 데이터 패킷을 수신하는 경우를 나타낸 것이다. CCN 노드에 Interest 패킷이 수신되면 prefix와 CS capacity를 확인하고 CS에 관련 데이터가 있으면 주변 CCN 노드에 청크 데이터를 나누어서 전송하게 된다. 만약, 관련 데이터가 없다면 주변 CCN 노드에게 Interest 패킷을 전송한다. CCN 노드에서 Data 패킷이 수신되면 Interest 패킷에 명시된 reverse path로 Data 패킷을 전송한다.

그림. 2는 CCN의 기본 forwarding 전략과 제안하는 CCN-Helper의 전략을 비교한 것이다. CCN에서는 주변 CCN노드에 Interest 패킷을 broadcast 방식으로 전송한다. 만약, n3 CCN노드에서 보낸 Interest 패킷이 n6에 먼저 도착하여 Data 패킷을 전송하기 위한 reverse path가 n6 → n3 → n2 → n1으로 설정되었다면 다음 청크 데이터도 같은 path를 사용하게 될 것이다. 하지만, 네트워크 환경에 따라 링크가 불안정하게 되면 수요자가 받는 데이터의 수신 시간이 감소하여 처리량이 저하되는 원인이 될 것이다. 제안하는 방법은 생산자 노드 n6에서 Interest 패킷을 받을 때 같은 prefix에 대해서 예외 처리를 하지 않고 시퀀스가 다

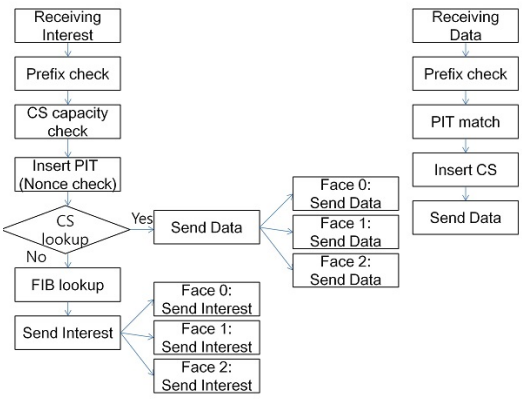


그림 1. CCN 노드에서 데이터 패킷 수신의 경우
Fig. 1. In case CCN nodes receive Data packet

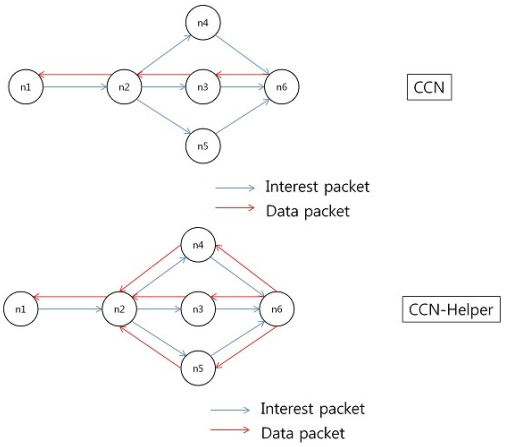


그림 2. CCN과 CCN-Helper 의 비교
Fig. 2. Comparison between CCN and CCN-Helper

른 청크 데이터를 나누어서 분산하여 전송함으로써 n2 노드와 n1이 단위 시간 당 받는 청크 데이터의 크기를 향상시켜 수요자 노드 n1에서의 처리량을 증가시킨다. CCN-Helper에서 Interest 패킷을 다음 CCN 노드에게 전송할 경우에 CCN의 브로드캐스팅과 같은 방식으로 전송하지만, 생산자가 Interest 패킷을 다운로드 하고 요청 패킷이 들어온 Face로 Data 패킷을 전송할 때에 기존의 방법은 그 Face로만 전송하게 된다. 만약, 해당 역 경로가 전송률이 낮은 구간이거나 혼잡이 발생하는 구간이라면 지속적인 패킷 전송 시에 데이터 전송 지연은 불가피하다. 따라서 제안하는 CCN-Helper 프로토콜에서는 생산자는 데이터 패킷을 전송할 때에 청크 패킷 단위로 나누어서 전송하게 되고 연결된 모든 Face로 Data 패킷을 전달한다. 이 청크 패킷을 다운로드 받은 주변 CCN노드는 prefix 이름과 chunk 시퀀스를 확인한 후에 CS의 capacity를 확인한다. prefix 이름이 같더라도 chunk 시퀀스가 다르게 되면 CS의 용량을 확인한다. 만약, CS에 남은 여유 공간이 있으면 PIT 테이블의 Interest 패킷을 받았던 역 경로로 패킷을 전달한다. 하지만 CS에 여유 공간이 없다면 Data 패킷을 저장하지 않고 주변 CCN 노드로 Data 패킷을 전달한 후에 FIB 테이블의 포워딩 경로로 생산자 노드에게 CS의 여유 공간이 없음을 알린다. 생산자 노드는 다음 Chunk 시퀀스부터 해당 CCN노드와 연결된 Face로는 Data 패킷을 전송하지 않는다. 기본적으로 CCN은 Data 패킷을 수신하기 위한 Interest 패킷을 전송할 때 브로드캐스팅 방식으로 전송하게 된다. 이 의미는 CCN의 각 노드의 PIT 테이블에는 Interest 패킷에 대한 reverse path 가 존재하

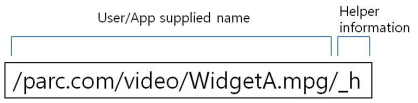


그림 3. Interest 패킷 컴포넌트
Fig. 3. Human readable Interest prefix component

는 것을 의미한다.

제안하는 방법이 가능한 CS의 공간이 부족할 때 데이터 패킷의 저장 없이 주변 노드에 패킷을 전달하고 패킷을 소유한 생산자 노드 혹은 Data 패킷을 전달받은 노드에게 버퍼 공간이 없음을 알려주기 위하여 PIT 테이블에 존재하는 Interest 패킷의 정보를 참조하여 해당 Interest 패킷을 다시 전송하되 "_h" 마커를 콘텐츠 이름에 추가하여 전송한다. 마커가 붙은 Interest 패킷을 전달 받은 노드 혹은 생산자 노드는 "_h" 마커를 확인하게 되면 해당 Face 로는 Data 패킷 전송을 보류하고 다른 Face로만 전송한다. alpha 시간이 경과하면 다시 해당 Face로 전송할 수 있으며, 이와 같이 동작하는 이유는 CS capacity는 네트워크 트래픽에 유동적이기 때문에 가능할 때에는 다시 CCN-Helper 메카니즘을 사용할 수 있다. Data 패킷을 수정하는 것이 아니기 때문에 콘텐츠 소유자에 대한 보안 이슈는 발생하지 않는다.

Interest 패킷을 사용하는 이유는 Interest 패킷의 함수는 TCP ACKs와 비슷한 방식으로 동작하며 패킷 자체의 크기는 Data 패킷보다 작기 때문에 네트워크에 큰 부하를 주지는 않는다. 그림. 3은 Interest 패킷의 "_h" 마커 위치를 설명한 것이다.

알고리즘 1는 Interest 패킷이 CCN 노드에 들어올 때의 알고리즘을 나타낸 것이다. 패킷을 처리하기에 앞서 prefix 이름과 청크 시퀀스를 확인하고 Interest 패킷의 정보를 PIT 테이블에 저장하고 FIB 테이블에 연결된 각 Face를 참조하여 연결된 경로로 전송한다. CCN 노드에 연결된 모든 Face 에 대해서 Interest packet을 전송한다. 만약 Data 패킷이 CS에 존재하면 FIB 테이블의 모든 Face 정보로 Data 패킷을 전송하고 전송하는 양은 연결된 홉 개수를 전체의 청크의 수로 나눈 값을 사용한다. 알고리즘 2은 CCN 노드로 Data 패킷이 전달될 때 처리되는 방법을 나타낸 것이다. Data 패킷을 처리하기 전에 청크의 시퀀스 번호 그리고 prefix 이름을 확인한다. prefix 이름이 같더라도 chunk 시퀀스의 번호가 다르다면 데이터 처리를 검토한 후에 도착한 Data 패킷을 처리하기 위한 CS 허용 용량을 확인한다. CS에 여유 용량이 있으면 Data 패킷을 저장하고 PIT 테이블의 역 경로를 통해

Algorithm 1 OnIncomingInterest

Ensure: incoming packet is not dummy packet
1: **if** prefix name and chunk sequence exist in PIT **then**
2: **if** _h marker exist in Name component **then**
3: except Interest packet
4: **end if**
5: **end if**
6: insert Interest packet information into the OutRecord in PIT table
(save prefix name and chunk sequence of Interest packet)
7: **if** Data packet is in CS Table **then**
8: send Data packet to all linked face of FIB table with amount of total chunk count / linked hop count
9: **else**
10: **for** linked face in the list **do**
11: send Interest packet to all linked face in FIB table
12: **end for**
13: **end if**

Algorithm 2 OnIncomingData

Ensure: incoming packet is not dummy packet
1: check PIT match(prefix name, chunk sequence)
2: **if** prefix name and chunk sequence exist in PIT **then**
3: except Data packet
4: **else**
5: **if** CS capacity is available **then**
6: insert Data packet to CS -> send Data packet refer to the reverse path of PIT table and all linked face in FIB
7: **else**
8: send Data packet to the neighbor CCN node and send Interest packet(with _h marker) to forwarding path refer to FIB table
9: **end if**
10: **end if**

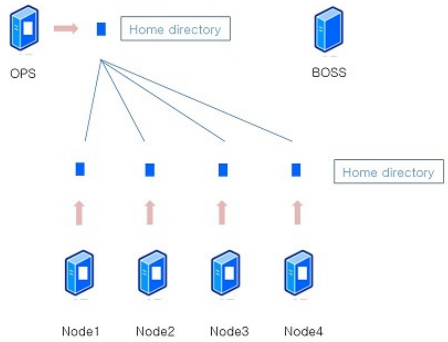
서 요청 자에게 전달한다. 만약 CS에 여유 용량이 없으면 주변 CCN노드에게 Data 패킷을 전달하고 FIB 테이블의 Interest 패킷 포워딩 경로로 생산자에게 CS에 여유 용량이 없음을 알린다. 생산자는 다음 청크 시퀀스부터는 Data 패킷을 해당 Face로 전송하지 않는다.

Ⅲ. 유럽과의 연동을 위한 ProtoGENI 설정

국제적인 페더레이션 환경을 구축하기 위하여 현재 구축되어 있는 Emulab 시스템에 ProtoGENI를 설치하여 테스트해 보고자 한다. Emulab시스템의 구조는 Boss서버와 Ops 서버 그리고 사용자가 만든 토폴로지의 각 노드로 구분이 된다. 각 노드의 홈 디렉토리는 Ops 홈 디렉토리와 마운트 되어 있으며 각 노드에서 홈 디렉토리로 접속이 가능하다. 그림. 4는 Emulab 시스템 내부 구조를 도식화 한 것이다. 운영 서버인 Boss와 Ops서버가 있으며 각 실험노드는 Home directory가 공유되어 있다.

ProtoGENI에 의해 구현된 SFA의 컴포넌트는 CH(Clearing House), AM(Aggregate Manager), SA(Slice Authority), CM(Component Manager), RSpec으로 구성된다. 컴포넌트는 에지 컴퓨터, 라우터, 프로그래머블 액세스 포인트 등에 해당하는 GENI 인프라스트럭처의 기본 구성 요소로서, CPU나 메모리와 같은 물리 자원과 포트번호나 파일 디스크립터와 같은 논리 자원, 그리고 두 형태가 결합된 패킷 포워딩 패스와 같은 자원 등으로 구성되며, 특정한 자원이 속한 컴포넌트는 최대 한 개만 존재할 수 있다.

컴포넌트는 컴포넌트 매니저(CM)에서 관리한다. Aggregate는 여러 개의 컴포넌트가 하나의 그룹으로 묶은 것이며, Aggregate Manager(AM)이 관리한다. 복수 개의 컴포넌트를 관리하는 점을 제외하면 컴포넌트와 유사하다. CH(Clearing House)는 중앙 등록지의 역할을 담당하며, 각 지역의 테스트베드 사용을 위해 사용 자격을 부여하는 역할을 담당한다. AM(Aggregate Manager)는 티켓을 발행하여 리소스 사용의 권한을 부여한다. 사용자는 AM을 통하여 티켓을 받게 되고 AM은 Sliver를 생성한다. Sliver는 각



System Configuration
- System node:
  Dell PowerEdge R710, 2.4GHz 64-bit Quad Core Xeon processor
  12GB 1066 MHz DDR2 RAM (6 x 2GB modules)
- Experiment control server:
  Dell PowerEdge R710, 24GB Memory, HDD 2TB
- File server:
  Dell PowerEdge R710, 24GB Memory, HDD 10TB
- Device control server:
  Dell PowerEdge R710, 6GB Memory, HDD 250GB
- Switch device:
  Cisco 4507R+E, Supervisor 7, 48port 10/100/1000 Interface Card
- Rack device:
  PowerEdge Rack 4220 full set

그림 4. Emulab 시스템 구조
Fig. 4. The structure of Emulab system

지역에서 제공하는 리소스 자원을 의미한다.

그림. 5는 KREONET Emulab 에서 구현된 RSpec 구조이다. 여기에서 사용자는 특정 노드를 할당할 수도 있으며 컴포넌트 ID의 지정을 통해서 Emulab CM 자원을 사용하도록 설정한다. RSpec은 데이터 교환 포맷으로써 테스트베드의 리소스 자원을 명세하고 있다. 특정 노드에 리소스 자원을 할당할 수 있으며, 컴포넌트 ID를 통해서 지역 테스트베드 자원 변경이 용이하다. 실험자는 실험의 네트워크 토폴로지를 포함하는 RSpec에서 선호하는 자원을 설정하여야 하며, 포맷이 제출되고 난 후, 요청된 자원은 자동적으로 할당되고 노드 간의 가상 링크가 실험을 토폴로지를 통해 연결된다. 실험자는 SSH로 해당 자원에 접속이 가능하다.

```
<rspec    xmlns="http://www.geni.net/resources/rspec/3">
  <node    client_id="node0"  exclusive="true"

component_manager_id="urn:publicid:IDN+emulab.KREONET.net+a
uthority+cm">
        <sliver_type  name="raw-pc"/>
        <location          xmlns="http://jfed.iminds.be/rspec/ext/jfed/1"
x="53.0"  y="67.5"/>
      </node>
</rspec>
```

그림 5. KREONET Emulab RSpec 구조
Fig. 5. The structure of KREONET RSpec

3.1 ProtoGENI 페더레이션 구성

국제적으로 연동된 테스트베드를 구현하기 위해 ProtoGENI를 KREONET Emulab[15] 테스트베드에 설치하여 실험을 수행하였다. 작동 방식은 다음과 같다. 실험을 수행하기 위해 사용자는 슬라이스를 생성하고 슬라이스를 생성하기 위해 사용자는 SA에 접근하여 새로운 슬라이스를 요청한다. 만약 요청이 수락되면, SA는 사용자에게 새로운 슬라이스의 이름을 주고 XML-RPC를 통해서 관리에 필요한 인증서를 부여한다. 각 사이트의 보스 시스템은 두개의 신뢰된 XML-RPC서버를 운영하는데 하나는 슬라이스 권한 API와 다른 하나는 컴포넌트 매니저 API이다. 인증서는 Clearinghouse의 GID(Geni Identifier, 증명서)를 사용하여 서명된다. ProtoGENI 페더레이션은 Clearinghouse에 의한 X.509 공개키 구조에 의존한다. 일반적으로 증명(certification) 권한을 가지는 각 사이트들은 CA처럼 root 증명서(certificate)를 발급하고 CH(Clearing House)를 통해서 배포된다. 잘 알려진 root certificates은 genica.bundle과 genicrl.bundle 파일에 저장되며 지속적으로 업데이트된다. 사이트 간에 개별적인 페더레이션도 bundle의 수정을 통하여 가능하다. 추가된 certificates은 openssl명령어를 사용해서 확인해 볼 수 있다. 자원을 찾기 위해서 사용자는 네트워크를 만들기 위한 컴포넌트를 선택한다. 이 방법은 다양한 방식으로 수행될 수 있다. 간단한 방법은 각 AM에게 물어보는 방식이다. 사용자는 CH에게 공유된 AM리스트를 요청한다. 슬리버를 생성하기 위해 사용자는 컴포넌트를 선택하고 요청을 생성한다. AM에게 요청을 보내서 필요한 자원을 요구한다. AM은 자원 사용에 필요한 티켓을 전달한다. 만약 모든 티켓 요청이 수락되지 않으면 사용자는 실패한 요청과 새로운 패킷 교체를 시도한다. 리소스의 사용을 위해서 사용자는 슬리버에 로그를 남기고 실험을 생성한다. 많은 컴포넌트는 ssh를 통해서 접속한다. 사용자는 실험이 실행되는 동안에 slice를 수정할지도 모른다. 실험을 마치면 모든 슬리버는 삭제된다.

Ⅳ. 성능 평가

4.1 시뮬레이터를 통한 CCN-Helper 성능 테스트

본 논문에서 제안하는 CCN-Helper 방법론을 증명하기 위해 ndnSIM 2.0[25] 도구를 사용하여 테스트하였다. 실험에 사용된 혼잡 제어 토폴로지는 그림. 6와 같으며, Interest패킷에 대한 Data 패킷의 수신시간을 비교하기 위해 C1요청자에서 Interest 패킷을 전송한다.

전송되는 패킷 크기는 100개의 청크로 설정한다. 요청자와 송신자에서 전송되는 페이로드 크기는 1040 바이트로 설정하였다. 전송이 빠른 링크는 속도가 빠른 대신에 지연 시간이 길어서 오히려 성능 저하를 불러올 수 있도록 설정하였고 전송이 조금 느리긴 하지만 지연 시간이 짧은 링크의 경우를 설정하여 데이터 전송을 도와주는 링크로 설정하였다. 속도가 빠른 링크는 지연시간을 30ms로 설정하였고 느린 링크에 대

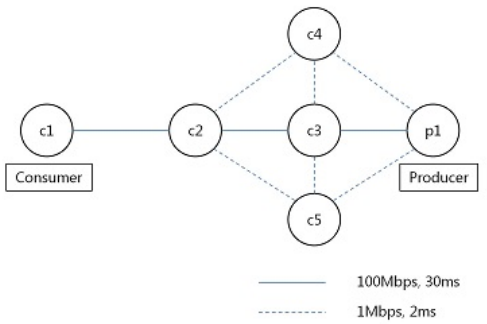


그림 6. 시뮬레이션 토폴로지
Fig. 6. Simulation topology

한 전송 Delay는 2ms로 설정한다. Bandwidth는 전송이 빠른 링크의 경우에는 100Mbps로 설정하고 느린 링크에는 1Mbps로 설정한다. 버퍼의 크기는 도착시간에 영향을 덜 주기 위해서 1000 으로 설정하고, 패킷 손실을 방지하기 위해 링크의 큐 크기는 500으로 입력하여 테스트하였다. CCN의 도착 시간은 비례적으로 증가하는 반면, CCN-Helper는 주변 노드의 도움을 받아 청크 데이터를 받기 때문에 도착시간이 거의 절반 정도 줄어든 것을 확인할 수 있다. N4와 N5의 중간 라우터에서 Data 패킷을 나누어서 전송함으로써 다운로드 시간을 줄일 수 있었다.

그림. 8는 C1 노드에서 처리량을 비교한 것이다. 링크의 지연시간이 길기 때문에 처리량은 기본적으로 감소하는데 CCN 방법의 처리량은 지수적으로 감소하는 반면, CCN-Helper는 처리량이 CCN보다 조금씩 감소하는 것을 확인 할 수 있다. 주변 노드의 도움을 받아서 패킷을 받게 되면 처리량을 증가시킬 수 있다는 사실을 확인하였다.

그림. 7는 C1 노드에서 Interest 패킷에 대한 Data 패킷의 다운로드 시간을 그래프로 비교한 것이다.

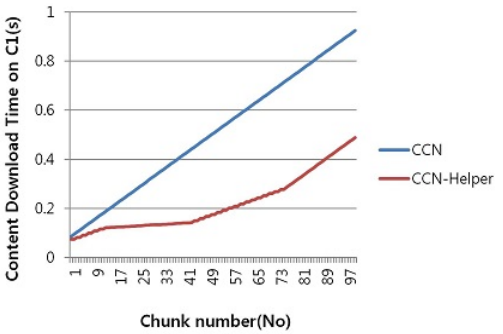


그림 7. C1노드에서 다운로드 시간 비교
Fig. 7. Comparison for download time on C1

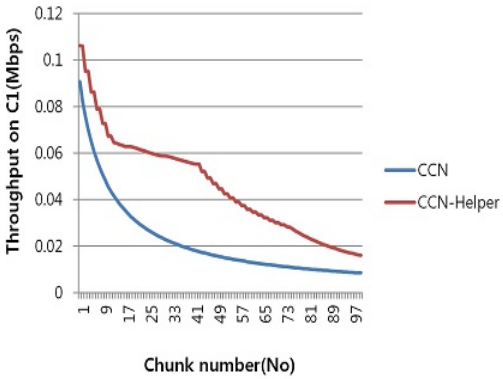


그림 8. C1노드에서 처리량 비교
Fig. 8. Comparison for throughput on C1

4.2 KREONET 환경에서의 CCNx 성능 테스트

Content Centric Network(CCN)을 구현하기 위해서 PARC(Palo Alto Research Center)의 CCNx 오픈소스 프로젝트를 사용하였다. 페더레이션 환경에서 CCN-Helper를 테스트하기에 앞서 KREONET 환경에서 Emulab 테스트베드 환경에서의 CCNx의 성능을 테스트하였다. 그림. 9와 같이 CCN 테스트를 위한 토폴로지를 구성하였다. 각 노드는 가상 머신이 아닌 실제 Ubuntu 12.04 OS를 탑재한 머신이며 각 노드가 CCN 콘텐츠를 주고 받도록 각 노드에 CCNx를 설치하고 CCN-Helper 알고리즘을 적용하였다. N0노드는 *ccnputfile* 명령어로 콘텐츠를 제공하는 노드로 설정하였으며, N9 노드는 콘텐츠 요청자로 *ccngetfile* 명령어를 사용하여 테스트하였다.

그림. 10은 KREONET Emulab 테스트베드에서 CCNx를 테스트하고 그 결과를 나타낸 것이다. 처음에는 파일크기 900KB 텍스트 파일을 N9에서 콘텐츠를 가지고 있는 N0로 콘텐츠를 요청하였고, N7에서 같은 콘텐츠를 요청한 시간과 다시 N9 에서 같은 콘텐츠를 요청한 파일의 다운로드 시간을 측정하였다. 900KB 텍스트 콘텐츠를 요청할 경우에는 전송 시간의 차이가 없었지만 20MB의 파일을 요청할 경우에는 처음에 N9에서 요청할 경우 많은 시간이 소요되었고 두 번째 N7에서 같은 파일을 요청할 경우에는 시간이 줄어든 경향을 보이고 있다. 이것은 N7이 N9 에서 파일을 다운로드 받아서 전송 시간이 줄어들었으며 다시 N9에서 같은 콘텐츠를 다운로드 받았을 때에는 N7에서 받기 때문에 전송 속도의 감소로 인한 성능 향상을 확인할 수 있다.

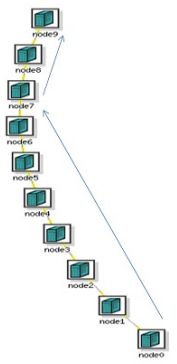


그림 9. KREONET Emulab 토폴로지 구성
Fig. 9. The composition of KREONET topology

Bandwidth		100MB
Delay		20ms
File Size	Forwarding	Transmission Time
900KB	N0->N9	723ms
	N0->N7	633ms
	N0->N9	400ms
20MB	N0->N9	505205ms
	N0->N7	4171ms
	N0->N9	3997ms

그림 10. KREONET Emulab 테스트베드에서 CCNx 결과
Fig. 10. CCNx results over KREONET testbed

4.3 국제적 연동 환경에서의 CCNx 성능 테스트

실험 생성 및 구현을 위해서 현재 KISTI에서 서비스 되고 있는 네트워크 자원 Emulab[10]을 활용하여 ProtoGENI[5]를 적용함으로써 외부와의 연동 테스트를 시행하였다. 외부 연동 페더레이션 GUI는 벨기에의 iMinds[16]에서 제공하고 있는 JAVA 기반의 Experimenter GUI(jFed)[14]를 활용하였다. Experimenter GUI를 사용하기 위해서는 로그인 계정이 필요한데 KREONET[9] Emulab 계정의 SSL 키를 생성해서 encrypted.pem 파일로도 로그인이 가능하다. 유럽과의 테스트베드 연동을 실현하기 위해서는 상호간에 신뢰할 수 있는 루트 인증서를 공유하면 쌍방의 테스트베드 사용자가 테스트 베드 자원을 할당하고 사용할 수 있게 된다. 상호 신뢰 가능한 모델을 위해서 각 사이트에서는 루트 권한 인증서를 가지고 있다. 이러한 인증서를 공유해주게 되면 다른 사이트에서 한국 사이트로의 접속이 가능하다. 페더레이션 사이트에서 등록된 사용자는 GENI API를 사용할 수 있기 때문이다. ProtoGENI와의 인터페이스는 신뢰 가능한 XMLRPC 서버를 통해서 수행되며, 하나는 Slice 권한 API로써, 다른 하나는 컴포넌트 매니저(CM, Aggregate 컴포넌트 매니저)로써 동작하게 된다. 슬리버는 하나의 클러스터 단위로 구성될 수 있으며 VLAN과 GRE 터널 등으로 각 사이트가 연결되어 있다. Clearing House(CH)는 리소스 자원의 중앙 등록지로써 컴포넌트 ID를 가지는 각 사이트의 리소스 정보를 제공한다. CH로 접속하기 위한 권한은 인증서를 발급받은 사람에게 가능하게 되고 인증서는 CH의 GID를 사용하여 발급된다. 이러한 인증서의 요청은 XML-RPC를 통해서 가능하기도 하지만 GID를 포함한 권한자도 인증서를 생성해서 접속 가능하다. ProtoGENI 확인자인 GID는 X.509 인증서를 사용한다. 로그인을 한 후 GUI 메뉴를 통해서 node0-node1 토폴로지의 실험을 생성한다. 벨기에와 신뢰적인 협력 관계 확인 후 root certificate 을 서로 교환한 후node0와 node1은 KREONET 네트워크의 시스템 자원을 할당하고 node2는 벨기에의 시스템 자원을 할당한다.

그림. 11과 같이 토폴로지를 구성하고 RSpec을 정의한다. 컴포넌트 매니저 아이디를 통해서 시스템 자원을 명시한다. RSpec에서는 특정 번호의 노드를 지정하여 테스트 할 수도 있다. 사용자는 RSpec의 component_manager_id를 통해서 시스템 자원을 확인할 수 있고 client_id를 통해서 특정 시스템 자원을 설정 가능하다. 그림. 7은 연동된 실험의 RSpec 설정을 나타낸 것이다.

시스템 로그는 그림. 12와 같이 시스템의 admin에게 전송된다. iMind의 사용자가 KREONET Emulab에 접속하기 위해 Sliver를 생성, 종료 등의 요청을 하게 되면 시스템 로그에서 RSpec과 Certificate 일치 여부 등을 확인할 수 있다.

접속을 설정하고 RSpec이 재대로 되었는지 확인한

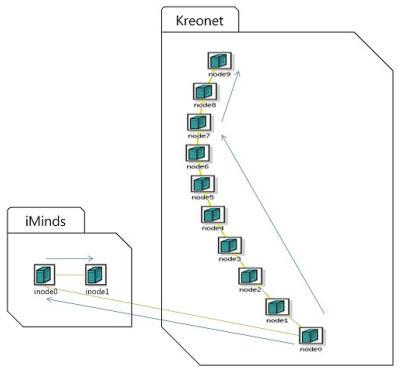


그림 11. 유럽과 연동된 토폴로지 구성
Fig. 11. The composition of federated topology

그림 12. 유럽과 연동된 RSpec 구성
Fig. 12. The RSpec configuration of federated topology

```
URN          : urn:publicid:IDN+wall2.ilabt.iminds.be+user+ftester
Module       : am
Method       :    CreateSliver
Version      : 2.0
StartTime    :    17:16:28:946654
slice_urn                                                    :
urn:publicid:IDN+wall2.ilabt.iminds.be:fed4fire+slice+sMO0Rr5pv60uj7
slice_idx    : 787
slice_uuid   :    aaccf520-0a92-11e5-bd9f-001517becdc1
EndTime      :    17:16:43:467952
Elapsed      : 14.52
LogURN       :              urn:publicid:IDN+emulab.KREONET.net
+log+6b420d06c5749c5d0e1c353a50cd2d0f
LogURL                                                       :
https://www.emulab.KREONET.net/spewlogfile.php3?logfile=6b420d0
6c5749c5d0e1c353a50cd2d0f
Code         :    0
```

그림 13. ProtoGENI log
Fig. 13. ProtoGENI log

후에 정상적으로 실행이 되면 그림. 13와 같이 특정 노드로ssh 접속 화면을 확인할 수 있다. 컴퓨터에 putty가 설치되어 있으면 자동으로 실행된다. 실험에 사용된 컴포넌트 매니저 ID는 KREONET Emulab으로 구성하였다.

그림. 14은 유럽과 연동된 테스트베드 환경에서 CCNx의 성능을 테스트한 실험 결과를 나타낸 것이다. 같은 테스트베드 환경에서는 다운로드 시간의 차이가 거의 없었다. iMinds의 할당된 노드 inode0, inode1 링크는 대역폭 100Mbps 와 전송 지연 20ms의 환경으로 설정하였다. Inode0는 KREONET 의 node0와 연결되어 있으며 inode1은 node1과 연결되어 있다. CCNx의 장점인 캐쉬 메커니즘을 사용하여 다운로드 속도의 차이를 비교하여 보았다. 우선 inode0(I1) 에서 파일 크기 900KB를 다운로드 하여 보았고 inode2(I2)에서 같은 파일을 다운로드 하였다. 전송시간의 향상을 결과로 확인할 수 있다. 20MB의 경우에도 비슷한 결과를 가져왔다. 이처럼 고대역폭 네트워크 환경에서 CCNx의 캐쉬 메커니즘을 통해서 성능의 향상을 확인할 수 있다.

Bandwidth		100MB
Delay		20ms
File Size	Forwarding	Transmission Time
900KB	N0->I1	46401ms
	N0->I2	761ms
20MB	N0->I1	999191ms
	N0->I2	4397ms

그림 14. 연동된 테스트베드에서의 실험 결과
Fig. 14. The test results over federated testbed

4.4 국제적 연동 환경에서의 CCN-Helper 성능 테스트

본 논문에서 제안하는 CCN-Helper 프로토콜의 성능을 테스트하기 위하여 CCNx를 사용하여 CCN-Helper 프로토콜을 적용하고 Data 패킷의 다운로드 시간을 비교해 보았다. 테스트를 위한 실험 토폴로지는 그림. 15와 같으며, C1과 C2는 유럽 iMinds의 테스트베드로 설정하고 C3, C4, C5, P1은 KREONET 테스트베드의 노드로 설정하였다. 로컬 테스트베드의 연결 링크의 설정은 100Mbps로 설정하였고 지연시간은 0ms로 설정하였다. 요청하는 Data 패킷의 크기는 20MB로 테스트하였고 국제적으로 연동된 테스트베드에서 각 경로로 전송되는 다운로드 시간을 확인하였다. 전송 경로가 P1 →C3 →C2 → C1으로 전송될 경우에는 파일을 다운로드하는데 1004868ms 이라는 시간이 소요되었다.

그림. 16는 C1노드에서의 다운로드 시간과 처리량을 비교한 것이다. 일반적인 CCN 알고리즘으로 Data 패킷의 전송 경로를 P1 →C3 →C2 → C1 으로 전송하였을 경우에는 초당 25692Byte를 전송할 수 있었으며, C1노드에서 요청한 20MB를 C1노드에서 전송 받는 데에 1004868ms(b)가 소요되었다.

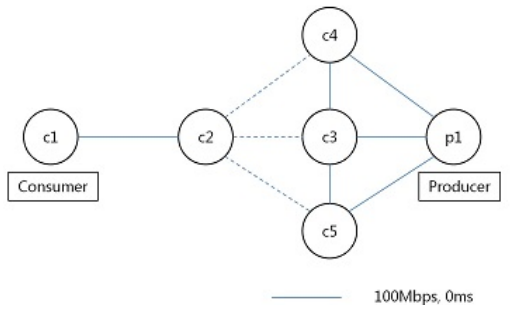


그림 15. CCN-Helper 성능 테스트를 위한 토폴로지
Fig. 15. Topology for CCN-Helper test

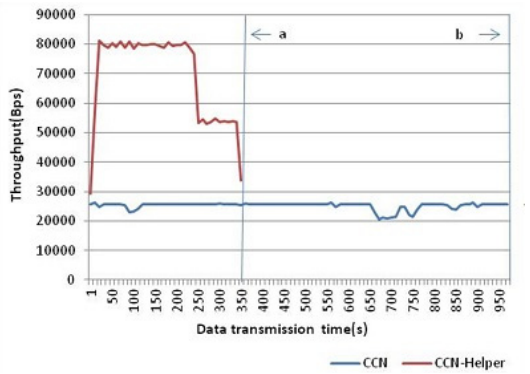


그림 16. C1노드에서의 다운로드 시간과 처리량 비교
Fig. 16. Comparison for download time and throughput on C1 node

CCN-Helper 프로토콜을 사용해서 Data 패킷을 요청하는 Interest 패킷을 C1노드에서 전송하면 P1 생산자노드에서는 Data 패킷을 C4노드와 C3노드 그리고 C5노드에 청크 데이터를 나누어서 전송하게 되고 속도가 빠른 링크의 데이터부터 C2노드에 순차적으로 도착하여 C1노드에 전달된다. 20MB 데이터를 P1 → C3 → C2 → C1으로 전달할 경우에는 1000초의 시간이 걸렸지만 6MB 데이터를 P1 → C3 → C2 → C1으로 전송하였을 경우에는 25236ms 가 소요되었다. 7MB를 P1 → C4 → C2 → C1으로 전송하였을 경우에는 347286ms가 소요되었고, 같은 7MB의 데이터를 P1 → C5 → C2 → C1으로 전송하였을 경우에는 347498ms가 걸려 전체 Data 패킷의 다운로드 시간은 350초(a)가 걸렸다.

CCN-Helper 프로토콜을 통해서 Data 패킷을 주변 노드에서 나누어서 전송하였을 경우에 전체 Data 패킷의 다운로드 시간을 650초 정도 절약할 수 있었다. CCN-Helper를 사용하여 원거리 네트워크에서 실험 결과와 같이 소비자의 Data 패킷 다운로드 처리량을 주변 CCN노드의 도움으로 증대시킬 수 있다.

V. 결 론

본 논문에서는 국내의 시험 망에서 CCN-Helper 프로토콜을 사용하여 원거리 네트워크에서 소비자의 다운로드 시간을 줄여 전체 처리량을 증가시키는 방법을 제안하였다. 또한. 국제간 테스트베드의 연동을 ProtoGENI 도구를 사용하여 구현하여 보았으며, CCN-Helper의 성능을 테스트 및 평가하였다. CCN-Helper 프로토콜은 원거리 네트워크뿐만 아니라 네트워크의 대역폭이 높지만 전송 지연이 긴 경우 또는 네트워크의 혼잡으로 인한 패킷 손실률이 증가할 경우에 주변 노드에서 패킷 전송을 일부 지원함으로써 소비자의 Data 다운로드 처리량을 증가시킬 수 있을 것이다. 본 연구의 실험 결과를 토대로 국제간 테스트베드에서 다양한 연구가 가능하도록 하기 위한 CCN 성능 향상에 도움이 되리라 생각되며 이러한 계기를 기초로 국제 시험 망을 활용한 통합적인 연구가 가능하고 사용자 관리의 편리성과 이기종 테스트베드의 관리 측면에서 높은 효율을 기대한다. 또한, SFA를 기초로 한 다양한 테스트 베드의 플러그인을 통해서 연구자와 일반 사용자에게 의미 있는 클라우드 서비스를 제공할 수 있을 것으로 생각된다. 향후 연구는 미국과의 연동 테스트와 CCN-Helper 프로토콜의 다양한 연구가 진행될 예정이다.

References

[1] Openstack http://www.openstack.org
[2] Cloudstack, http://www.cloud.com
[3] Eucalyptus, http://www.eucalyptus.com
[4] Open Nebula, http://opennebula.org/
[5] ProtoGENI, http://www.protogeni.net/
[6] GENI, http://www.geni.net/
[7] Fibre, http://www.fibre-ict.eu/
[8] KT UCloud https://ucloudbiz.olleh.com/
[9] KREONET, http://www.KREONET.re.kr/
[10] GLORIAD, http://www.gloriad.org/
[11] PlanetLab, http://www.planet-lab.org/
[12] FED4FIRE, http://www.fed4fire.eu/
[13] SmartFire, http://eukorea-fire.eu/
[14] jFed, http://jfed.iminds.be/
[15] Emulab, https://www.emulab.KREONET.net
[16] iMinds, https://www.iminds.be
[17] J. Lee, J. Lee, and D. Kim, "Delay data control scheme for enhancing network throughput in CCN environments," in *Proc. Advanced Sci. Technol. Lett.,* vol. 60, pp. 6-9, Feb. 2014.
[18] G. Tyson, J. Bigham, and E. Bodanese, "Towards an information-centric delay-tolerant network," *IEEE INFOCOM 2013*, pp. 387-392, Turin, Apr. 2013.
[19] S. Seo, M. Kim, and Y. Cui, "Cloud federation monitoring system based on SFA for integrating heterogeneous cloud platform services," in *Proc. KIISE*, pp. 1132-1134, Nov. 2013.
[20] S. Seo, M. Kim, and Y. Cui, "Resource monitoring system based on SFA for a heterogeneous cloud federation," *KIISE*, vol. 41, no. 3, pp. 124-131, Jun. 2014.
[21] R. Ricci, J. Duerig, L. Stoller, G. Wong, S. Chikkulapelly, and W. Seok, "Designing a federated testbed as a distributed system," in *Proc. Soc. Informatics and Telecommun. Eng.*, vol. 44, pp. 321-337, 2012.
[22] Y. Kanada and T. Tarui, "Federation-less federation of ProtoGENI and VNode platforms," in *Proc. ICOIN 2015*, pp. 271-276,

Cambodia, Jan. 2015.

[23] A. Ooka, S. Atat, and K. Inoue, and M. Murata, “Design of a high-speed content-centric-networking router using content addressable memory,” in *Proc. IEEE INFOCOM*, pp. 458-463, Apr. 2014.

[24] S.-H. Lim, Y.-B. Ko, G.-H. Jung, J. Kim, and M.-W. Jang, “Inter-chunk popularity-based edge-first caching in content-centric networking,” *IEEE Commun. Lett.*, vol. 18, no. 8, pp. 1331-1334, Aug. 2014.

[25] S. Oueslati, J. Roberts, and N. Sbihi, “Flow-aware traffic control for a content-centric network,” in *Proc. IEEE INFOCOM*, pp. 2417-2425, Orlando, FL, Mar. 2012.

[26] A. W. Kazi and H. Badr, “PIT and Cache dynamics in CCN,” in Proc. *Proc. IEEE GLOBECOM 2013*, pp. 2120-2125, Atlanta, GA, Dec. 2013,

[27] D. Byun, B.-J. Lee, and Y. Park, “Adaptive interest adjustment in CCN flow control,” in *Proc. IEEE GLOBECOM 2014*, pp. 1873-1877, Dec. 2014.

최 원 준 (Won-jun Choi)

2006년 2월 : 원광대학교 수학통계학과 학사

2013년 2월~현재 : 과학기술연합대학원대학교 과학기술정보과학학과 석박통합과정 재학중

<관심분야> 통신공학, 네트워크 통신, TCP 성능 분석, CCN 프로토콜, 페더레이션 네트워크

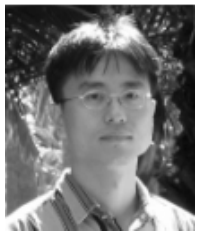
람 닉 (Ramneek)

2010년 2월 : Univ. Guru Nanak Dev 컴퓨터 공학과 졸업

2013년 8월~현재 : 과학기술연합대학원대학교 과학기술정보과학학과 박사과정 재학중

<관심분야> 무선 통신, 모바일 통신

석 우 진 (Woo-jin Seok)



1996년 2월 : 경북대학교 컴퓨터공학과 학사

2003년 2월 : Univ. North Carolina, Computer Science 석사

2008년 2월 : 충남대학교 컴퓨터공학과 박사

<관심분야> 무선/이동 QoS, TCP 성능 분석