Content Delivery over SDN: incremental vs. clean-slate approaches

권태경, 장덕현, 서준호
서울대학교

요약

최근 10년 간의 인터넷 트래픽 변화를 살펴 보면, 호스트 중심의 응용과 서비스에서 콘텐트 중심으로 이동해 온 경향이 가장 두드러진다. 특히 네트워크의 트래픽 양은 비디오 트래픽의 증가에 기인하여 폭발적으로 늘어나고 있다. 따라서 이러한 콘텐트들을 어떻게 망에 부하를 줄이면서 사용자에게 빨리 효율적으로 전달하는지 여부가 현재 인터넷 사업자들에게 최대 관심사이다. 콘텐트들을 인터넷에서 효율적으로 전송하기 위해서는 근본적으로 망 장비들이 어떤 콘텐트가 전송되고 있는지를 파악해야 하는데, 현재 망 장비들은 콘텐트의 내용을 전혀 모르고 IP 주소만 보고 패킷을 전송하기 때문에 문제가 생긴다. 본 고에서는 라우터/스위치 들이 콘텐트의 내용을 알도록 하기 위해서 2가지 방법을 소개하는데, 하나는 콘텐트별로 IP 주소를 할당하는 점진적인 접근 방법이고, 또 하나는 혁신적인 콘텐트 중심 네트워킹 기술을 구현하는 방법이다. 두 가지 다 OpenFlow/SDN 기술을 사용해서 구현 가능하다.

1. 서론

원격 접속이나 파일전송 등 호스트 간의 자원 공유를 주요 응용으로 하는 인터넷은 host-to-host 통신 패러다임을 기반으로 설계되었으나 최근에는 웹, 비디오 전송, P2P 기반의 파일 공유 등의 콘텐트 중심의 응용들이 인터넷 트래픽의 대부분을 차지하고 있다. 하지만 현재의 인터넷은 수십 년 전에 설계 된 아키텍처 및 프로토콜 구조를 그대로 유지하고 있기 때문에 이러한 콘텐트 중심의 통신 응용들로 변화되는 환경에 효과적으로 대처하기 어렵다는 한계점을 지니고 있다. 따라서 이러한 현재 인터넷의 근본적인 한계와 약점을 해결할 수 있는 새로운 형태의 미래인터넷에 대한 연구가 전 세계적으로 활발하게 진행 중이다.

대표적인 미래인터넷 연구로는 정보 중심의 네트워킹(Information-Centric Networking, ICN) 연구를 들 수 있다[1][2]. 호스트 중심의 인터넷 디자인과 실제 인터넷 사용 패턴의 괴리

는 수백만의 사용자가 동일한 콘텐트를 동시에 요청하는 경우에 서버 성능 저하 문제를 발생시키고, 또한 동일한 콘텐트가 사용자 수만큼 반복되어 전송되는 중복 다운로드로 인해 네트워크 자원 낭비 문제를 심화시킨다. 이러한 문제점을 해결하고, 효율적인 콘텐트 전달을 가능하게 하기 위해 콘텐트 중심으로 인터넷을 새롭게 설계하고자 하는 연구 방향이 바로 정보 중심의 네트워킹(Information-Centric Networking, ICN) 연구이다. 대표적인 연구로는 미국 UC 버클리의 Data-Oriented Network Architecture (DONA) [3], PARC의 Content-Centric Networking (CCN) [4][5][6], EU FP7 프로젝트의 Publish-Subscribe Internet Routing Paradigm (PSIRP) [7], 4WARD의 Networking of Information (NetInf) [8] 등이 있다.

한편 software-defined networking(SDN) [9][10]기술은 크게 두 가지 문제를 해결하기 위해 등장하였다. 첫째는 클라우드 컴퓨팅, 멀티미디어 등 대용량 트래픽 증가, 트래픽 동적인 변화 증가 등 기존의 네트워킹 기술로는 급변하는 망 변화에 대처를 효율적으로 못하는 것이 하나고, 둘째는 망 장비들이 proprietary 기술로 구현되어 있는 경우 새로운 기술이나 프로토콜들을 개발, 운영하는 것이 매우 어려운데 이를 개방형 인터페이스로 표준화하면 동적인 환경이나 새로운 요구사항에 맞추어 망의 운영을 쉽게 중앙 서버에서 제어할 수 있다. 가장 대표적인 OpenFlow 기술은 중앙 서버 controller와 OpenFlow 장비들간의 인터페이스를 OpenFlow 프로토콜로 표준화하고 이를 이용해 각 flow를 어떻게 처리할지 controller가 제어하는 기술이다 [11][12][13]. 스탠포드 대학의 주도로 시작된 OpenFlow 기술의 상용화를 촉진시키기 위해서 2011년 3월 표준화 단체인 ONF가 만들어 졌다. ONF는 Deutsche Telekom, Facebook, Google, Microsoft, Verizon, Yahoo!에 의하여 설립되었으며, 여기서 OpenFlow 기술을 표준화하고 있다.

본 고에서는 OpenFlow 기술을 이용하면 콘텐트 중심 응용의 트래픽들을 효율적으로 전송하기 위한 두 가지 접근 방법을 설명한다. 첫째는 incremental 접근 방법으로 기존 OpenFlow 기술에서 구현 가능한 기술로 각 콘텐트 이름을 하나의 IP주소로 매핑시켜서 같은 콘텐트는 같은 IP주소의 flow로 처리하는 방법니다. 둘째는 혁신적인(clean slate) 접근 방법으로 PARC의 CCN 기술을 향후 정의될 OpenFlow 기술을 사용해서 구현하는 것이다. 즉 웹 주소와 같은 콘텐트 이름이 각 패킷에 있으면 OpenFlow 장비들이 이를 분석하고 콘텐트 캐쉬와 pending interest table에 관련 데이터를 처리하는 것이다. 2장과 3장에서 두 가지 방법을 자세히 소개한다.

2. 점진적인 (Incremental) 접근 방법

본 장에서는 IP 기반의 네트워크에서 OpenFlow를 이용하여 콘텐트 전송을 효율적으로 하기 위한 네트워크 구조를 제시한다. 이를 위해서 각각의 콘텐트에 IP 주소를 동적으로 할당하고 OpenFlow 프로토콜을 이용하여 할당된 IP를 통해 콘텐트 요청 (i.e. HTTP GET) 및 콘텐트 데이터가 전송되도록 한다. 이러한 방식을 통해 기존의 IP를 이용한 전송 방식을 해치지 않

고 route-by-name을 에뮬레이션 하는 것이 가능하다.

먼저 하나의 도메인 내에서 오리지널 콘텐트 서버로부터의 콘텐트 전송 및 캐쉬된 콘텐트 전송이 어떻게 이루어지는가에 대해 설명하고, 그 후 서로 다른 도메인 사이의 콘텐트 전송이 어떻게 이루어지는 가에 대해 설명하도록 한다.

2.1 도메인 내에서의 콘텐트 전송

그림 1은 제안된 OpenFlow 기반의 구조에서 콘텐트가 어떻게 전송되는지를 보여준다. End user 1이 콘텐트 요청을 위해 HTTP GET 메시지를 자신의 액세스 노드 R1에 보내면 (1) R1은 해당 패킷에 대한 내용이 자신의 OpenFlow Flow Table에 정의되어 있지 않으므로 OpenFlow 프로토콜에 의해 이를 자신의 콘트롤러에 전달한다 (2). 콘트롤러는 해당 패킷을 받은 후 HTTP GET 메시지 안의 콘텐트 이름을 추출하고 해당 콘텐트가 네트워크 상에 캐쉬되어 있는지를 확인한다. 이를 위해 콘트롤러는 각각의 콘텐트가 네트워크 상의 캐쉬에 저장될 경우 콘텐트의 이름과 저장된 노드의 위치 정보를 가지는 content distribution metadata table을 유지한다. 따라서 해당 콘텐트의 캐쉬가 존재한다면 content distribution metadata table의 검색을 통해 캐쉬된 노드의 위치를 알 수 있다. 본 시나리오에서는 해당 콘텐트가 네트워크 상에 처음 요청되었기 때문에 요청된 콘텐트의 캐쉬를 발견하지 못하고, Domain Name Server (DNS)를 통해 오리지널 콘텐트 서버 (youtube.com) 의 위치를 알아낸다. 그 후 요청된 콘텐트의 이름으로 사용될 IP 주소 (i.e. 10.1.2.3)를 콘텐트에 할당하고 이를 content metadata table에 저장한다. 그리고 OpenFlow 노드인 R1, R2, R3 에 할당된 IP 주소에 대한 flow table entry를 생성함으로써 end user 1과 콘텐트 서버 사이의 경로를 생성한다 (3). End user 1이 보낸 HTTP GET 메시지는 R1에서 destination 주소가 10.1.2.3으로 치환이 되고, R2 를 거쳐 R3까지 전달된다. 여기서 R3는 콘텐트 서버에 메시지를 전달하기 전 destination 주소를 원래의 destination 주소인 콘텐트 서버의 주소로 치환한 후 콘텐트 서버인 youtube.com에 전달한다 (4). HTTP GET 메시지를 받은 콘텐트 서버는 요청에 대한 데이터를 자신의 액세스 노드 R3에 전송한다. R3는 해단 패킷의 source 주소를 콘텐트에 할당된 IP인 10.1.2.3으로 치환한 후 콘트롤러에 의해 세팅된 경로를 통해 해당 패킷을 전달한다. 따라서 해당 데이터는 R3, R2을 거쳐 R1에게까지 전달이 된다. 여기서 R1은 source 주소를 다시 콘텐트 서버의 주소인 youtube.com의 주소로 치환한 후 end user 1에게 전달한다 (5). 이러한 과정을 통해 end user와 콘텐트 서버에 별도의 수정없이 route-by-name을 에뮬레이션 하여 콘텐트를 전달할 수 있다.

이러한 방식으로 콘텐트의 캐쉬 및 캐쉬된 콘텐트의 전송도 가능하다. 만약 R2에 해당 콘텐트가 캐쉬되었다고 가정하면, end user 2에서 동일한 콘텐트에 대한 요청이 이루어졌을 때 캐쉬된 콘텐트를 전송할 수 있다. End user2가 HTTP GET 메시지를 자신의 액세스 노드인 R2에 보낼 경우 R2는 위의 경우와 마찬가지로 자신의 OpenFlow flow table에서 해당 패킷에 대한 entry를 찾을 수 없으므로 OpenFlow 프로토콜에 의해 콘트롤러에 이를 전달한다. 콘트롤러는 이번에는 자신의 content metadata table을 통해 이미 해당 콘텐트에 특정 IP가 할당되었음을 알고 content metadata distribution table의 검색을 통해 R2로부터 캐쉬된 콘텐트 (10.1.2.3)가 전송되도록 한다. 따라서 end user 2는 R2로부터 캐쉬된 콘텐트를 전송받는다 (8).

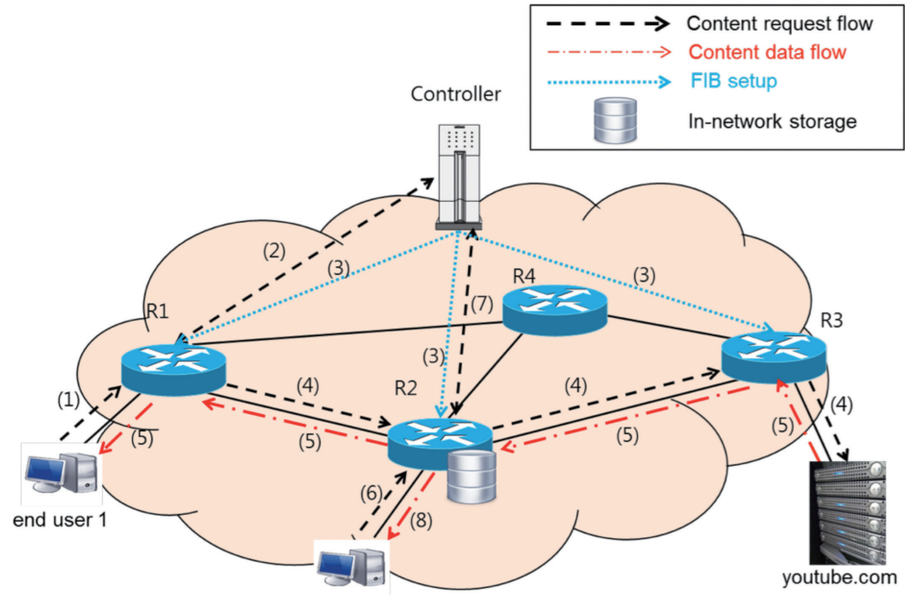

그림 1. 하나의 도메인 내에서의 콘텐트 전송

2.2 인터도메인 콘텐트 전송

본 장에서는 2-1에서 설명한 도메인 내에서의 콘텐트 전송을 기반으로 하여 서로 다른 도메인 사이의 콘텐트 전송을 위한 프레임워크를 제시한다. 이를 위해 본 연구에서는 inter-controller agent (ICA)를 도입한다. ICA는 다른 SDN의 controller와의 통신 및 inter-domain 콘텐츠 전송을 위한 기능을 담당한다. 구체적으로 다른 도메인의 ICA와의 connection setup, 그리고 inter-domain content distribution metadata 및 도메인 간 콘텐트 전송을 위한 policy등을 유지한다.

그림 2는 서로 다른 도메인 사이의 콘텐트 전송이 이루어지는 과정을 보여준다. 해당 시나리오에서는 하나의 도메인 (ISP A)에 있는 end user가 다른 도메인 (ISP B)에 있는 콘텐트 서버로부터 콘텐트를 전송 받는다. 각각의 ISP의 ICA 사이에는 이미 connection setup이 이루어져 있다고 가정한다 (e.g, SSL).

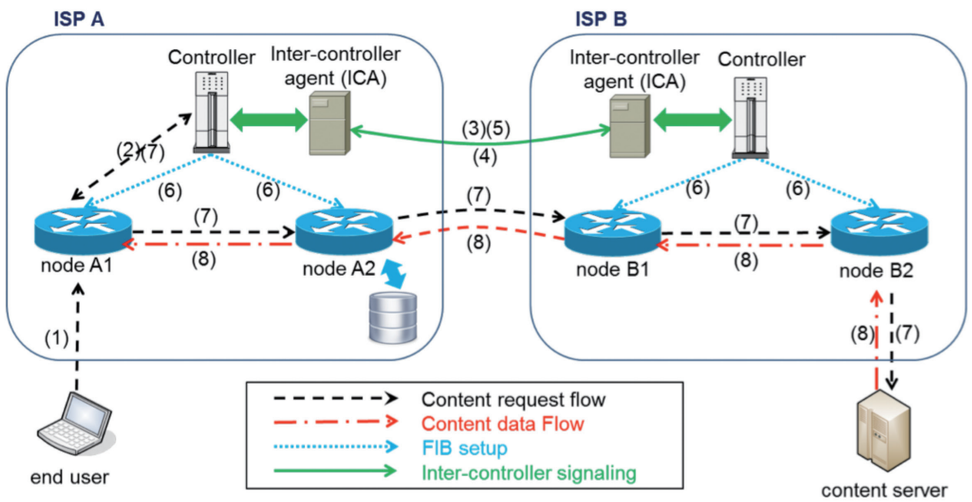

그림 2. 인터도메인 콘텐트 전송

먼저 end user가 콘텐트 요청을 위해 HTTP GET 메시지를 자신의 액세스 노드인 A1로 보내면 (1), A1은 해당 ISP 내에서의 콘텐트 전송 시나리오와 마찬가지로 자신의 OpenFlow flow table을 먼저 확인하고 해당 패킷에 대한 entry가 없으므로 이를 자신의 콘트롤러에게 전달한다 (2). 콘트롤러는 먼저 자신의 content metadata 및 content distribution metadata를 확인하여 해당 콘텐트에 대한 요청을 처리한적이 있는지 그리고 도메인 내에 해당 콘텐트의 캐쉬가 있는지에 대해 확인한다. 도메인 내에서 해당 콘텐트가 확인되지 않으면 자신의 도메인의 ICA가 유지하고 있는 inter-domain content distribution metadata를 통해 다른 ISP 내의 콘텐트를 요청한다. 여기서 설명의 편의를 위해 ICA 사이의 통신을 통해 inter-domain content distribution metadata 항상 최신의 상태로 유지되고 있다고 가정한다. 즉, 콘트롤러가 ICA에게 특정 콘텐트에 대한 정보를 요청할 때 ICA는 해당 콘텐트가 존재하는 다른 도메인을 이미 알고 있다고 가정한다. 여기서는 ISP B가 콘텐트를 가지고 있다고 가정하므로 ISP A의 콘트롤러는 ICA를 통해 ISP B에 콘텐트 전송에 대한 요청을 보낸다. 콘텐트 전송 요청에는 해당 콘텐트에 대해 ISP A에서 할당한 IP 주소가 포함된다. 또한 필요에 따라 QoS에 대한 요구사항이 포함될 수 있다 (3).

CDN Interconnection (CDNI) metadata와 같이 inter-domain content distribution에는 도메인 사이의 콘텐트 전송에 대한 policy가 미리 설정되어 있으므로 요청을 받은 ISP B의 ICA는 해당 policy에 따라 콘텐트 전송에 대한 가부를 결정한다. 본 시나리오에서는 ISP B가 콘텐트를 제공하기로 하였다고 가정한다. 따라서 ISP B는 ISP 내에 있는 콘텐트의 위치 및 네트워크 상황을 체크하고 ISP A에서 콘텐트에 할당한 IP 주소를 기반으로 자신도 해당 콘텐트에 IP 주소를 할당하고 이를 ISP A에게 알린다 (4). 이 경우 할당된 IP 주소는 2-1에서 설명한

바와 같이 콘텐트의 ID로 사용된다. 만약 ISP A와 동일한 IP를 사용할 수 없을 경우 다른 IP 주소를 사용하고 이를 ISP A에게 알린다. ISP A는 ISP B가 보낸 응답을 체크한 후 이에 대한 확인 메시지를 보낸다 (5).

그 후 각각의 ISP는 자신 내의 OpenFlow 노드의 flow table entry를 추가함으로써 end user와 콘텐트 서버 사이의 경로를 설정한다. 만약 ISP A와 ISP B가 동일한 콘텐트에 대해 서로 다른 IP를 할당하였다면 이를 node A2와 node B1에서 이를 치환할 수 있는 action을 OpenFlow 프로토콜을 통해 추가한다 (6). 경로가 설정되고 나면 end user가 보낸 HTTP GET 메시지의 destination 주소는 노드 A1에서 해당 콘텐트의 ID로 할당된 IP 주소로 치환된 후 노드 A2, B1를 거쳐 노드 B2까지 전달된다. 여기서 노드 B2는 destination 주소를 다시 원래의 주소인 콘텐트 서버의 주소로 치환한 후 콘텐트 서버에 전달한다 (7). HTTP GET 메시지를 받은 콘텐트 서버는 콘텐트를 노드 B2에 전송한다. 노드 B2는 source 주소를 해당 콘텐트의 ID로 할당된 IP 주소로 치환한 후 설정된 경로를 통해 데이터를 전송한다. 데이터는 노드 B1, A2를 거쳐 A1에게까지 전달된다. 여기서 A1은 source의 주소를 다시 원래의 주소인 콘텐트 서버의 주소로 치환하여 end user에게 전달한다 (8).

만약 요청되어 전달되는 콘텐트가 노드 A2에 캐쉬될 경우, 2-1에서와 마찬가지로 캐쉬된 노드로의 콘텐트 요청 전송 및 응답이 가능하다.

3. 혁신적인 (Clean Slate) 접근방법

본 장에서는 기존의 호스트 중심의 플로우 콘트롤에서 데이터 중심의 플로우 콘트롤을 하기 위한 구조를 제시한다. 이를 위해 OpenFlow 프로토콜 확장 및 기존의 commodity 하드웨어 스위치에서 지원해야 할 요구 사항들에 대해 기술한다.

3.1 NDN 포워딩

이번 섹션에서는 오픈플로우 프로토콜을 지원하는 네트워크상에서 NDN 네트워크를 실현하기 위한 방법을 제시함에 앞서, NDN 네트워크의 구성요소인 NDN 노드들이 수행하는 NDN 포워딩에 대해 알아볼 필요가 있다. 이를 통해 NDN 네트워크를 실현하기 위한 각 노드들의 핵심적인 NDN 포워딩 기능들을 도출할 것이다. 이렇게 도출된 기능들은 다음과 같다: (i) 이름 기반의 테이블 룩업, (ii) 인터레스트/데이터 패킷 프로세싱, (iii) 네트워크 내재 캐슁 지원. 또한 이렇게 도출된 기능들과 기존의 오픈플로우 스위치의 포워딩 기능들과 차이점을 살펴보고, 오픈플로우 (오픈플로우 프로토콜과 오픈플로우 스위치) 확장시 고려사항을 도출할 것이다.

3.1.1 핵심적인 NDN 포워딩 기능

그림 3은 NDN 네트워크의 구성하는 NDN 노드의 구성요소에 대해 도식화한 그림이다. NDN 노드는 패킷을 포워딩하기 위해서 세개의 서로다른 데이터구조를 유지하고 있다: (i)

Content Store (CS), (ii) Pending Interest Table (PIT), (iii) Forwarding Information Base (FIB). 이 외에 룩업을 빠르게 하기 위한 각 데이터구조의 인덱스 테이블을 유지한다. 또한 다른 NDN 노드와 연결하기 위한 여러 인터페이스로 구성되어 있다.

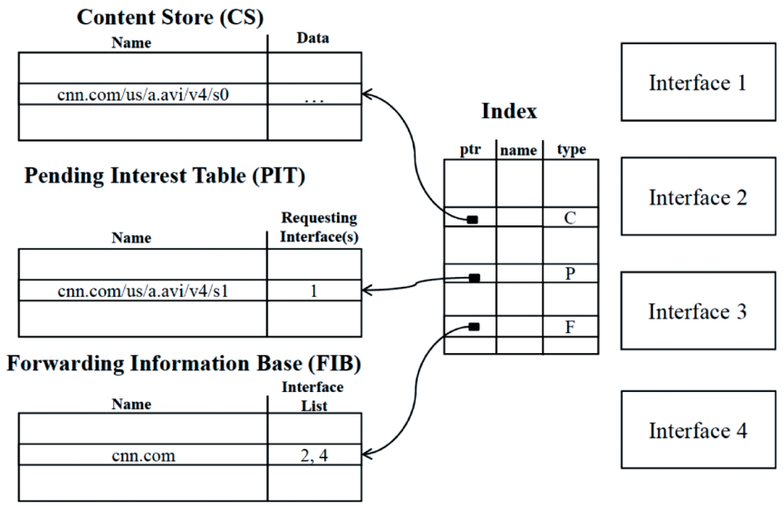

그림 3. NDN Forwarding Engine

이름 기반의 테이블 룩업: NDN 에서 콘텐츠의 이름은 계층구조를 갖는 alphanumeric의 인간이 읽을 수 있는 형태를 갖는다. 한 예로 현재 쓰이고 있는 URI 를 들 수 있는데, 그 형태는 cnn.com/us/a.avi/_v4/_s8 와 같다. 그 의미는 해당 이름을 갖는 콘텐츠 (a.avi) 는 cnn 에서 처음 게시하였으며 미국 서버에서 생성되었으며 버전 4의 세그먼트 8번을 의미한다.

이런 계층적 이름 구조를 가정하는 이유는 같은 이름 접두사 (name prefix)를 갖는 이름들을 하나의 동일한 이름 접두사 (name prefix) 로 줄일 수 있어 FIB 테이블의 사이즈를 획기적으로 줄일 수 있다. 이렇게 줄여진 이름 접두사(name prefix) 는 추후에 이름 전부 (full name)가 인터레스트/데이터 패킷에 담겨서 오면 가장 길게 매칭이 되는 이름 접두사를 택하는 것을 약속한다. 이를 longest prefix matching (LPM) 이라 칭한다. 하지만 LPM은 exact matching에 비해 복잡도가 크다. 그 이유는 full name 의 name prefix 를 미리 알지 못하기 때문에 테이블에 있는 비슷한 name prefix 를 갖는 엔트리를 전부 비교해야 한다. 현재까지 고정길이의 IPv4/IPv6 주소체계에 대해 LPM 을 위한 많은 연구가 이뤄졌다. 하지만 이 또한 고정 길이(32비트IPv4/128비트 IPv6)의 현 주소 체계와 비교해 보았을 때 alphanumeric 이고 가변의 URI 스타일의 이름 구조는 LPM 을 적용하기가 어렵다.

인터레스트/데이터 패킷 프로세싱: FIB 는 인테레스트 패킷이 NDN 노드로 들어왔을 때 요청하는 콘텐츠가 존재하는 방향으로 가는 길을 알려주는 역할을 한다. 이를 위해 FIB 에는 해당되는 콘텐츠의 aggregation 된 name prefix 와 다음 노드 (또는 face 정보) 정보를 갖고 있다. 반면에 PIT 는 FIB와는 다르게 데이터 패킷이 NDN 노드로 들어왔을 때 해당 콘텐츠를 요청한 노드로 데이터 패킷을 전달하기 위한 정보를 갖고 있다. 이를 위해 PIT 에서는 해당 콘텐츠 이름 (full name) 과 들어온 페이스 번호를 유지한다. NDN 에서는 이를 breadcrumb 이라 칭한다. 지금까지 NDN에서 두 노드간에 네트워킹을 위한 중요한 데이터구조 2가지에 대해 살펴 보았다. 지금부터 CS 테이블에 대해 알아 보자. 데이터 패킷은 전달 과정 중에 미래의 추가적인 요청을 수행하기 위해 중간의 NDN 노드들에 저장/캐싱 될 수 있다. 캐싱이 된 데이터 패킷은 NDN 노드의 스토리지에 저장이 되고 해당 리스트를 CS 테이블에 기록한다.

이제부터 인터레스트 패킷이 전달되는 과정에 대해 알아보자. 인터레스트 패킷이 NDN 노드에 들어와서 CS 에 매칭이 될 수 있다. 이는 해당 NDN 노드에 콘텐츠가 이미 존재 (캐싱) 한다는 의미 이므로 더 이상 인터레스트 패킷을 다음 노드로 전달하지 않고 자신이 갖고 있는 데이터를 돌려준다. 만약 인터레스트 패킷이 PIT 에 매칭이 되면 이는 이미 이전에 같은 콘텐츠 요청이 있었다는 이야기 이므로 해당 패킷을 더 이상 다음 노드로 전달하지 않고 기존의 PIT 엔트리에 새로운 정보 (해당 리퀘스트가 들어온 인터페이스)를 추가한다. CS 또는 PIT 에 매칭이 안됐을 경우 새로운 리퀘스트로 간주하고 매칭된 FIB 엔트리에 담긴 정보를 이용하여 다음 노드로 전달한다.

이제부터 데이터 패킷이 전달되는 과정에 대해 알아보자. 인터레스트 패킷이 전달되는 과정에 비해 데이터 패킷 전달은 간단하다. 데이터 패킷이 NDN 노드로 들어오면 인터레스트 패킷과 마찬가지로 CS와 PIT 테이블을 룩업한다. 한가지 다른 점은 데이터 패킷은 FIB 테이블을 룩업하지 않는다. CS 에 매칭이 되면 해당 데이터는 이미 NDN 노드의 스토리지 안에 캐싱이 되어 있으므로 전달하지 않는다. PIT에 매칭이 되면 해당 콘텐츠 데이터에 대해 요청이 있다는 의미 이므로 PIT 엔트리의 정보를 이용하여 요청한 노드쪽으로 데이터를 전달한다.

네트워크 내재 캐싱: NDN 에서 캐쉬를 함으로써 많은 이득을 볼 수 있는데 네트워크 트래픽 절감 효과와 delay를 줄일 수 있다.

NDN 노드의 캐쉬 스토리지가 꽉 찾을 경우 새로운 데이터를 캐쉬하기 위해서는 기존에 존재하는 데이터 중 하나 또는 여러 개를 골라 교체해주어야 하는데 주로 LRU 또는 LFU를 많이 쓴다.

3.1.2 오픈플로우와의 차이점 및 확장시 고려사항

위 섹션에서 알아본 세가지의 핵심적인 NDN 포워딩 기능을 지원하기 위해 오픈플로우를 확장하는건 쉬운일이 아니다. 그 이유는 오픈플로우 포워딩을 위한 데이터구조와 NDN 포워딩을 위한 데이터구조가 근본적으로 다르기 때문이다. 따라서 이번 절에서는 그 차이를 살피고 오픈플로우 확장시 고려할 사항에 대해 알아본다.

가변적 name 성질을 고려한 오픈플로우 확장: 현 오픈플로우 버전 1.0은 12개의 튜플을

사용해서 플로우를 구별한다. 또한 각 튜플은 고정적인 크기를 갖는다. 하지만 앞서 언급했듯이 NDN 네트워크 주소체계는 가변적이고 alphanumeric 한 name 기반이다. 13번째 튜플로 NDN 의 name 을 추가하기 위해서는 2가지를 반드시 고려해야 한다: (i) 가변적인 name 을 담기 위한 고정적인 튜플, (ii) 오픈플로우에서 제공하는 와일드카드 메커니즘과 name prefix 를 LPM 하기 위한 메커니즘과의 호환성. 따라서 우리는 다음장에서 이 두가지 고려사항을 해결하기 위한 새로운 방법을 고안할 것이다.

플로우 테이블과 NDN의 세 데이터구조의 차이: 오픈플로우 버전 1.1의 multiple tables 기능을 이용하면 NDN 포워딩 엔진의 세 데이터구조를 쉽게 확장할 수 있다. 하지만 아직까지 버전 1.1을 구현할 스위치가 만들어지지 않아 우리는 single table 인 버전 1.0을 이용해 확장할 수 밖에 없다. 또한 multiple tables 기능을 이용해 확장을 한다고 하더라도 이를 NDN 전용 데이터구조로 이용하게 되면 메모리를 비효율적으로 사용하게 되고, 오픈플로우의 플로우의 개념에 위배된다. 따라서 우리는 앞서 언급한 인터레스트/데이터 패킷 프로세싱을 오픈플로우 액션으로 추상화 하는 방법에 대해 설명할 것이다.

오픈플로우 스위치/라우터에서의 네트워크 내재 캐쉬 지원: 네트워크 내재 캐쉬는 NDN 네트워크만의 독특한 특징이다. 따라서 현재 상용화된 스위치/라우터에는 네트워크 내재 캐싱 기능이 존재하지 않는다. 또한 상용 스위치/라우터에는 패킷 버퍼링을 위해 보통 용량은 작지만 (몇 킬로바이트) 엑세스 속도가 빠른 메모리 (SRAM/0.45ns, DRAM/55ns) 를 탑재하고 있다. 하지만 NDN 네트워크에서는 데이터 캐싱을 위해 이 보다 더 많은 메모리 (몇 기가바이트) 를 요구한다. 하지만 보통 용량이 큰 메모리는 엑세스 속도가 느리기 때문에 high speed 스위치/라우터 상황에서는 적합하지 않을 수 있다. 따라서 다음절에서 이를 고려한 방법을 고아할 것이다.

3.2 ND-Flow

3.2.1 FIELD AND MATCHING EXTENSION

오픈플로우 스위치로 하여금 컨텐츠 이름을 인식하게 하기 위해선 기존의 12개 튜플의 플로우 필드를 확장할 필요가 있다. 따라서 우리는 위 그림과 같이 NDN을 위한 이름 필드와 패킷 타입 필드를 새롭게 추가 하였다.

또한 위에서 언급하였듯이 기존의 고정적인 필드들에 반해 NDN 에서의 이름은 가변적이어서 많은 문제점을 야기 하였다. 따라서 가변적 길이의 이름을 고정적 길이의 라벨로 변환할 필요가 있다. 이를 위해 ND-Flow 에서는 해싱 기법을 이용한다.

PIT는 exact matching 을 요구하고, FIB 및 CS 는 wildcard matching 을 요구한다. 이를 위해 PIT는 패킷의 전체 이름에 대한 해싱을 한 뒤 해당된 해싱값을 이용하여 exact matching 을 한다.

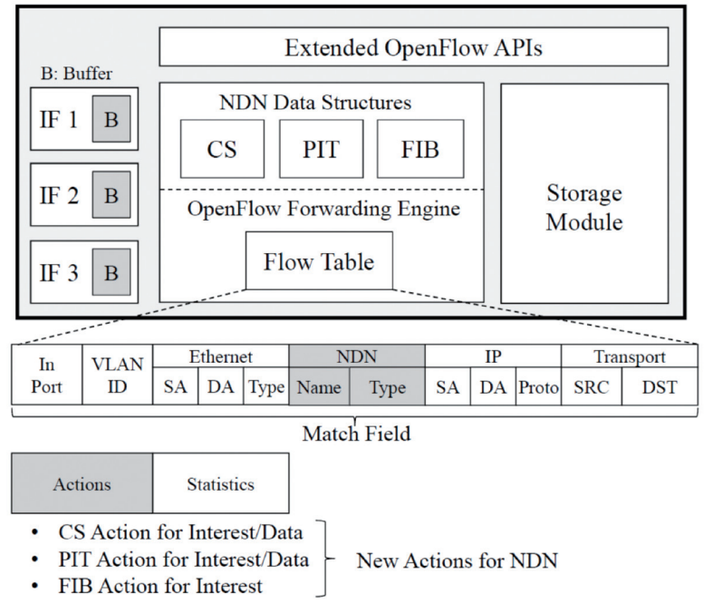

그림 4. OpenFlow에서 NDN 지원을 위한 ND-Flow 구조

3.2.2 New Actions for Packet Processing

NDN 의 세가지 데이터 스트럭처를 오픈플로우 상에서 구현하기 위해 새로운 데이터 스트럭처를 정의하지 않고, 각각의 데이터 스트럭처를 오픈플로우의 액션으로 추상화 하였다. 이를 위해 NDN 을 위한 다섯 가지 새로운 액션을 추가 하였다: CS Action for Interest, CS Action for Data, PIT Action for Interest, PIT Action for Data, FIB Action for Interest. 각각의 액션들의 오퍼레이션은 아래 수도코드에 도식화 되어 있다.

- If (msg_type is Interest)
  - Retrieve from the storage module for caching data (*load*)
  - Sending it to incoming port# of Interest (*output*)
  - Consume Interest (*drop*)
- Else // msg_type is Data
  - Duplicated data (*drop*)

- If (msg_type is Interest)
  - Update its incoming port# of Interest into an existing entry (*flow-mod with PIT action for Data*)
  - Consume Interest (*drop*)
- Else
  - Forwarding Data to the recorded interfaces (*output*)
  - Caching Data (flow-mod with CS action for Interest and CS action for Data)

- If(msg_type is Interest)
  - Forwarding Interest to corresponding ports# (*output*)
  - Update flow table (flow-mod with PIT action for Interest and PIT action for Data)

또한 새로운 액션들은 기존 오픈플로우에 존재하는 아토믹한 액션 (그림에서의파란글씨)들로 정의하였다. 이렇게 함으로써 확장성 및 이식성을 높일 수 있다.

3.2.3 In-network Caching Support with Two-level Memory Hierarchy

기존의 스위치들은 적은 용량의 빠른 메모리를 버퍼로 제공한다. 하지만 NDN 만의 새로운 기능인 캐싱으로 인해 더 많은 용량의 메모리를 요구한다. 이를 위해 우리는 스위치의 시스템 메모리를 활용한다.

하지만 시스템 메모리는 용량이 큰 반면 속도가 현저히 느리기 때문에 고속으로 들어오는 컨텐츠 요구량을 처리하지 못한다. 따라서 우리는 스위치의 각 라인카드에 있는 고속의 메모리도 이용한다. 따라서 현재 CPU 설계처럼 Two-level Memory Hierarchy 로 설계하였다.

3.2.4 Routing population

중앙 콘트롤러는 그가 관장하고 있는 네트워크의 스위치 토폴로지뿐만 아니라 각 엔드 호스트 (publisher)가 공개하고 있는 name-prefix 정보를 갖는다. 이를 이용하여 shortest-path 알고리즘을 이용하여 각 name-prefix 로 향하는 모든 루트를 계산한 뒤 해당 루트들을 스위치의 플로우 테이블에 삽입한다.

4. 결론 및 토의

이상으로 OpenFlow/SDN 기술을 사용하여 어떻게 콘텐트를 효율적으로 전송할 수 있을지 관련된 두 가지 접근 방법을 살펴보았다. 콘텐트 전송 이 외에도 이동성 지원, 멀티캐스팅, DDoS관련 수비 기법 등을 OpenFlow/SDN 기술을 통해서 쉽게 구현 가능할 것으로 전망된다. 향후 관련 분야에서 원천 기술을 확보하기 위해 산학연이 같이 노력해야 할 것으로 생각된다.

참고문헌

[1] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox. Information-centric networking: seeing the forest for the trees. In ACM Workshop on HotNets ‘11.

[2] L. Muscariello, G. Carofiglio, and M. Gallo. Bandwidth and storage sharing performance in information centric networking. In ACM SIGCOMM Workshop on ICN ‘11.

[3] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica, “A Data-Oriented (and Beyond) Network Architecture” , In Proceedings of SIGCOMM 2007

[4] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. “Networking named content,” In Proceedings of ACM CoNEXT, 2009.

[5] V. Jacobson, D. K. Smetters, N. H. Briggs, M. F. Plass, P. Stewart, J. D. Thornton, R. L. Braynard, VoCCN: Voice Over Content-Centric Networks, ReArch ’09, Rome, December, 2009.

[6] D. k. smetters and v. jacobson. “Securing network content”, october 2009. parc technical report. http://www.parc.com/content/attachments/securingnetwork-content-tr.pdf.

[7] Dimitrov, V. and Koptchev, V. PSIRP project – publish-subscribe internet routing paradigm: new ideas for future internet. Printed in a separate ACM DL issue in the “ACM International Conference Proceeding Series (ICPS)”, Vol. 471 (ISBN 978-1-4503-0243-2), Pages: 167-171.

[8] B. Ahlgren, M. D’Ambrosio, C. Dannewitz, A. Eriksson, J. Golic, B. Gronvall, D. Horne, A. Lindgren, O. Mammela, M. Marchisio, J. Makela, S. Nechifor, B. Ohlman, K. Pentikousis, S. Randriamasy, T. Rautio, E. Renault, P. Seittenranta, O. Strandberg, B. Tarnauca, V. Vercellone, and D. Zeghlache, \Second netinf architecture description,” 4WARD EU FP7 Project, Deliverable D-6.2 v2.0, Apr. 2010, fP7-ICT-2007-1-216041-4WARD / D-6.2, http://www.4ward-project.eu/.

[9] Kate Greene (March/April 2009). “TR10: Software-Defined Networking”. Technology Review (MIT). Retrieved November 20, 2011.

[10] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: rapid prototyping for software-defined networks. In ACM Workshop on HotNets ‘10.

[11] Google and facebook support openflow standards. http://www.webestigate.com/2011/04/15/googleand-facebook-support-openflow-standards/.

[12] Openflow switch specification, version 1.1. http://www.openflow.org/documents/openflowspec-v1.1.0.pdf.

[13] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. SIGCOMM Comput. Commun. Rev., 38(2):69–74, Mar. 2008.

[14] M. Baugher, B. Davie, A. Narayanan, and D. Oran. Self-verifying names for read-only named data. In INFOCOM Workshops on NOMEN ‘12, 2012.

[15] K. Cho, H. Jung, M. Lee, D. Ko, T. T. Kwon, and Y. Choi. How can an isp merge with a cdn? IEEE Communications Magazine, 49(10):156–162, 2011.

[16] K. Cho, M. Lee, K. Park, T. T. Kwon, Y. Choi, and S. Pack. Wave: Popularity-based and collaborative in-network caching for content-oriented networks. In INFOCOM Workshops on NOMEN ‘12, 2012.

[17] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee.

Devoflow: scaling flow management for high-performance networks. In ACM SIGCOMM '11.

[18] S. Podlipnig and L. B¨osz¨ormenyi. A survey of web cache replacement strategies. ACM Comput. Surv., 35(4):374–398, Dec. 2003.

[19] M. . Ruiz-s˜A , anchez, I. S. Antipolis, E. W. Biersack, S. Antipolis, W. Dabbous, and I. S. Antipolis. Survey and taxonomy of ip address lookup algorithms. IEEE Network, 15:8–23, 2001.

저자 소개

권 태 경

- 1993: 서울대학교 컴퓨터공학 학사
- 1995: 서울대학교 컴퓨터공학 석사
- 2000: 서울대학교 컴퓨터공학 박사
- 1998: IBM 왓슨 연구소 방문연구원
- 1999: University of North Texas 방문연구원
- 2000~2002: UCLA 박사 후 연구원
- 2002~2003: City University New York Principal Engineer
- 2003~2004: City University New York 박사 후 연구원
- 2004~2008: 서울대학교 전기컴퓨터공학부 조교수
- 2008~현재: 서울대학교 전기컴퓨터공학부 부교수
- 관심분야: 미래 인터넷, 무선 네트워크, Mobile/ubiquitous computing

장 덕 현

- 2004: 서울대학교 컴퓨터공학 학사
- 2004~현재: 서울대학교 컴퓨터공학 석박통합과정
- 관심분야: 미래 인터넷, 센서 네트워크, Software defined networking (SDN)

서 준 호

- 2008: 한국정보통신대학교(ICU, 현 KAIST) 학사
- 2008~현재: 서울대학교 컴퓨터공학 석박통합과정
- 관심분야: Content centric networking (CCN), Software

OSIA
Standards & Technology Review
Journal