SDN-based Distributed Mobility Management (DMM) Software 매뉴얼

OF@TEIN Document No. 12
Version
Date 2013-12-21
Author(s) Korea Univ. OF@TEIN 팀

■ 문서의 연혁

버전	날짜	작성자	비고
초안	2013.10.11	이기원, 장인선, 신민수, 김원태, 주석진, 백상헌	
1차 수정	2013.11.13	이기원, 장인선, 신민수, 김원태, 주석진, 백상헌	
2차 수정	2013.12.13	이기원, 장인선, 신민수, 김원태, 주석진, 백상헌	
3차 수정	2013.12.21	이기원, 장인선, 신민수, 김원태, 주석진, 백상헌	

본 문서는 한국정보화진흥원(NIA)의 미래네트워크연구시험망(KOREN)사업 지원과제의 연구결과로 수행되었음 (13-951-00-001).

This research was one of KOREN projects supported by National Information Society Agency (13-951-00-001).

Contents

그림 목차

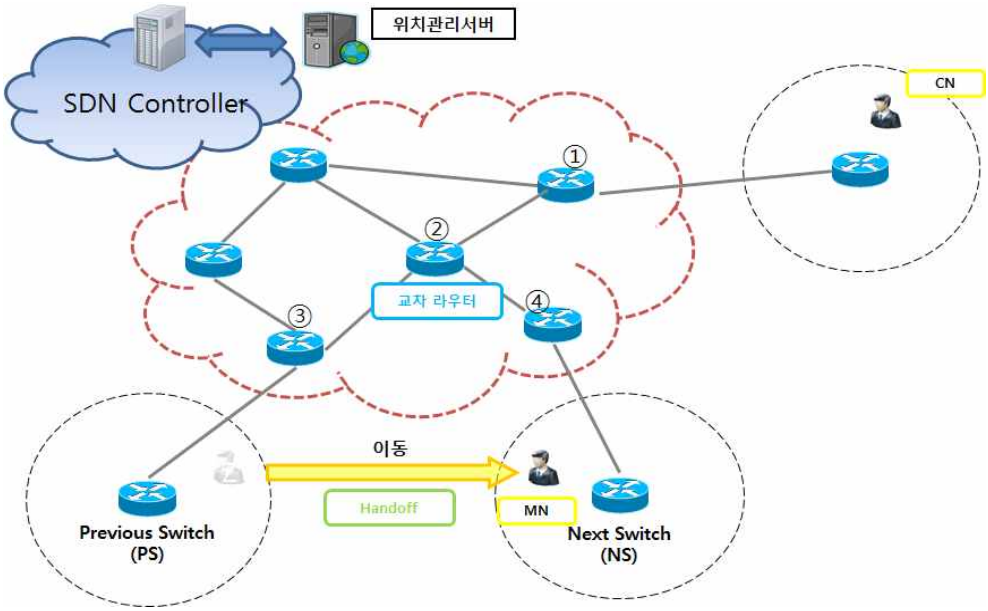
1. Introduction

1.1. Conventional DMM & SDN-based DMM

최근 모바일 트래픽의 폭증으로 인하여 이동 단말 (MN: Mobile Node)에 대한 이동성 관리를 분산적으로 제공하는 분산 이동성 관리 기법 (DMM: Distributed Mobility Management) [1]-[3]에 대한 연구가 활발히 이루어지고 있다. DMM 네트워크는 MN들의 위치정보를 관리하고 있는 다수의 액세스 라우터 (AR: Access Router)들로 이루어져 있으며, 각 MN들은 AR과 연결되어 있다. 이로 인해 DMM은 이동성 관리를 위한 제어 트래픽을 분산시키는 효과가 있다. 특히, DMM에서는 MN의 이동에 의하여 다른 AR로의 핸드오프가 발생한 경우, 이전의 AR에서 MN이 이동한 AR 사이에 터널링을 통해 패킷을 전달하여 세션을 유지시킨다. 하지만 MN의 이동성이 커져 핸드오프가 많이 발생하게 되면 데이터 전달 경로가 길어진다는 단점을 가지고 있다. 즉, 이웃한 AR간의 포워딩 기능에 의존하고 있기 때문에 최적의 경로를 사용하지 못한다.

SDN-based DMM은 기존의 DMM의 데이터 전달 경로를 최적화하는 것에 목적을 둔다. SDN-based DMM은 중앙 집중형 컨트롤러에 존재하는 위치관리 서버에 의해서 MN들의 위치 정보가 관리되고, 데이터 패킷이 최적 경로 계산에 따라 컨트롤러가 flow table을 생성할 수 있도록 도와준다. 특히, 핸드오프 발생 시 스위치 (DMM에서의 AR) 간의 터널링이 아닌 컨트롤러의 제어에 의해서 적절한 스위치에서 버퍼링이 수행이 되고 최적의 경로로 데이터가 전달될 수 있어, 기존 DMM에 비해 패킷 전달 지연 시간을 줄일 수 있다.

[그림 1]은 핸드오프 시 컨트롤러에 의해서 경로가 최적화되는 것을 보여준다. 기존 DMM은 핸드오프 발생 시 스위치간의 터널링을 통한 패킷 전달로 세션을 유지하기 때문에 MN이 핸드오프 하는 동안 데이터가 PS에 버퍼링되고, 핸드오프 후에 다시 NS로 전달된다. 따라서 기존 DMM에 의한 데이터 이동 경로는 CN - ① - ② - ③ - PS - ③ - ② - ④ - NS - MN이 된다. 반면, SDN-based DMM은 핸드오프 발생 시 컨트롤러가 PS와 NS 사이에 존재하는 ② 교차 라우터에 버퍼링을 명령하여 CN - ① - ② - ④ - NS - MN의 경로로 데이터가 이동하기 때문에, 기존 DMM에 보다 최적의 경로로 데이터가 전달될 수 있다. 컨트롤러는 네트워크 토폴로지를 알고 있기 때문에 MN이 이동하는 임의의 두 스위치 사이에 교차 라우터를 미리 알 수 있다. 금년 2013년에는 OF@TEIN 테스트베드를 통해 컨트롤러의 위치관리 서버가 각 MN들의 위치정보를 관리하며, 핸드오프 시 컨트롤러에 의해 기존 DMM보다 SDN-based DMM의 경로가 최적화되는 것을 보여준다.



[그림 1] MN의 핸드오프시 데이터 전달 과정.

1.2. Overview of SDN-based DMM software

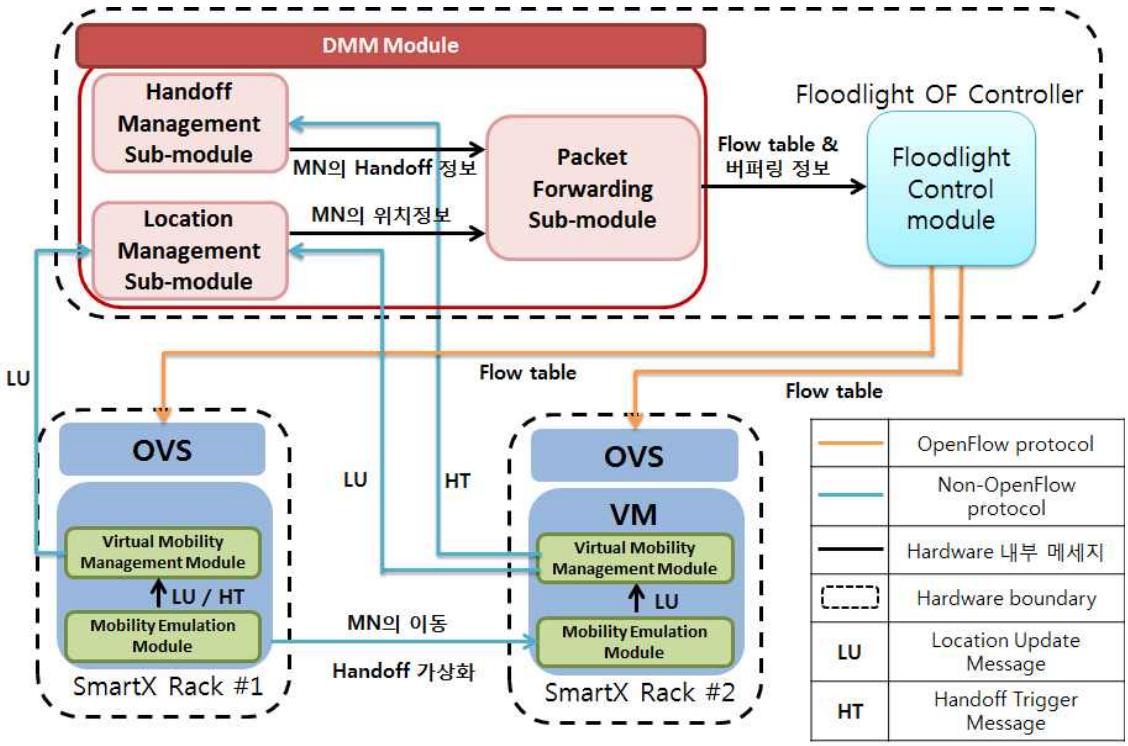
SDN-based DMM을 위한 SDN-based DMM software는 다음의 요구사항을 만족해야 한다. 우선 SDN-based DMM은 중앙 집중형 컨트롤러에 의해서 데이터 전달에 대한 최적의 경로를 제공해야 한다. 이를 위해 [그림 2]와 같이 컨트롤러 부분에 위치관리 서버를 두어 DMM module이 구현된다. 컨트롤러 부분에 구현되는 DMM module은 기능적으로 크게 세 가지의 sub-module (location management, handoff management, packet forwarding)로 분류된다. Location management sub-module과 Handoff management sub-module은 스위치로부터 위치 정보 (Location Update: LU) 및 핸드오프 트리거 (Handoff Trigger: HT) 메시지를 전달받아 MN의 위치 및 이동 정보를 유지한다. DMM module의 packet forwarding sub-module은 이러한 정보들을 이용하여 최적의 데이터 전달 경로를 계산하여 컨트롤러가 경로 상의 스위치들에게 플로우 엔트리 생성을 명령하게끔 도와준다. 이러한 동작과정은 sub-module들의 유기적인 연동을 통해서 이루어지고, MN의 위치 정보 관리 및 핸드오프 시 적절한 제어를 수행하여 최적 경로를 제공할 것이다.

또한, SDN-based DMM에서 스위치는 DMM 네트워크의 AR과 같은 기능을 수행해야 한다. 따라서 스위치는 MN으로부터 전송되는 LU 메시지 또는 핸드오프 시 전송되는 HT 메시지를 수신하여 컨트롤러로 전달해줘야 하고, 핸드오프 시 버퍼링을 수행하는 기능 또한 요구된다. 이러한 스위치의 기능은 SmartX rack 내부의 스위치 VM에 그림 2와 같이 Virtual Mobility Management (vMM) module로 구현된다.

SmartX rack 기반의 실험환경에서는 무선 인터페이스에 대한 고려되어 있지 않

기 때문에 MN의 실질적인 이동에 대한 실험 수행이 어렵다. MN의 역할은 smartX rack 내부의 호스트 VM에서 이루어지기 때문에 그림 2와 같이 호스트 VM 내부에 MN의 이동성 및 핸드오프 상황을 가상화하기 위한 Mobility Emulation (ME) module이 구현된다.

SDN-based DMM software에서는 module들의 동작이 적절히 수행되는 지 확인할 수 있는 visualizer가 구현되며, 이는 이용자들에게 추가적인 세부 정보 (MN의 이동 시 플로우의 흐름 변화, AR의 버퍼링 상태 등)를 제공한다.



[그림 2] SDN-based DMM 소프트웨어 구조

2. Structures of SDN-based DMM software

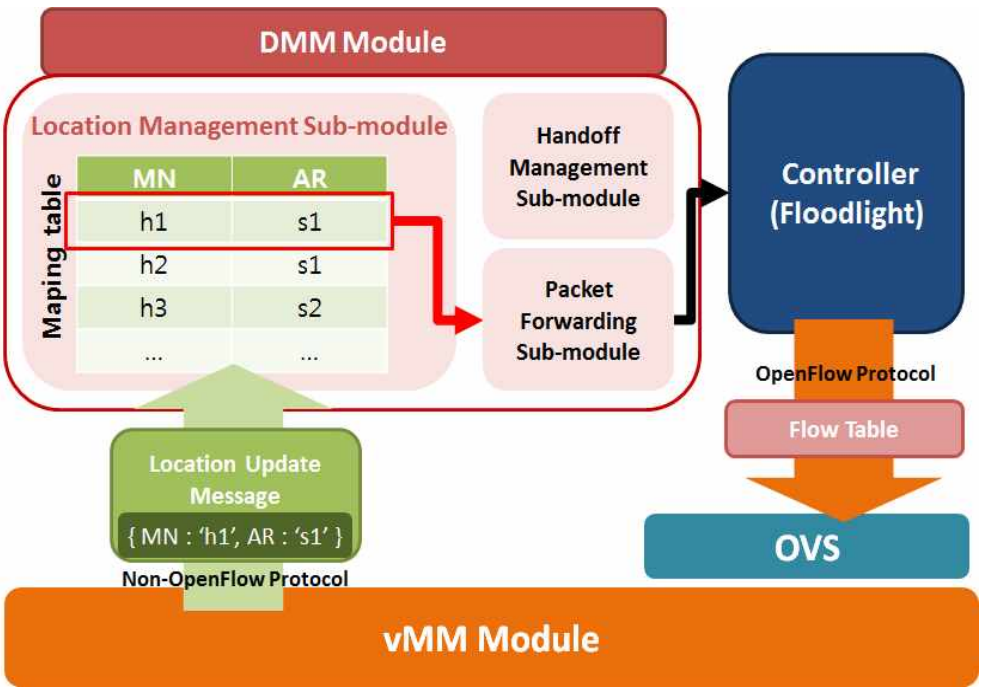
2.1. DMM Module for Controller

2.1.1. Location Management Sub-module

Location management sub-module은 위치관리 서버 내부에 구현되는 module로 MN들의 위치 정보 관리 (즉, MN과 MN이 위치한 스위치를 mapping) 기능과 적절한 flow table 생성을 위해 MN의 위치정보를 전달하는 역할을 한다. 이 location management sub-module은 [그림 3]과 같이 설계된다.

이 module은 SmartX rack에 존재하는 vMM module로부터 MN의 위치 정보가 담겨 있는 location update (LU) 메시지를 전달 받는다. LU 메시지는 MN과 MN이 위치한 스위치를 mapping하는 형태로 { MN : 'h1', AR : 's1' }의 JavaScript Object Notation (JSON)으로 작성된다. Location management sub-module은 이 LU 메시지를 바탕으로 MN들의 위치정보를 mapping table로 관리한다. Mapping table도 { 'h1' : 's1', 'h2' : 's1', 'h3' : 's2', ...}와 같이 JSON 형태로 유지된다.

LU 메시지에 의해 mapping table의 entry가 수정되거나 추가되면 location management sub-module은 packet forwarding module에게 MN들의 변화된 위치정보를 전달한다.



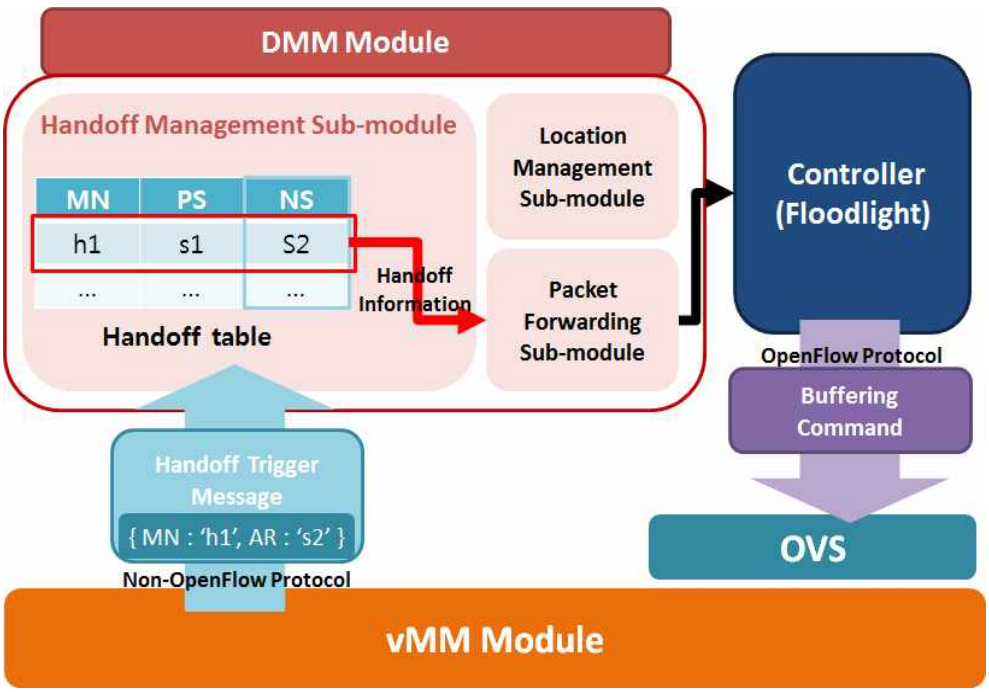
[그림 3] Location Management Sub-module

2.1.2. Handoff Management Sub-module

Handoff management sub-module은 위치관리 서버 내부에 구현되는 module로 MN들의 handoff를 관리, 즉, MN들의 다른 스위치로의 handoff 발생 시에 데이터 버퍼링을 위해 그 정보를 저장하고, 전달하는 역할을 한다. 이 handoff management sub-module은 [그림 4]와 같이 설계된다.

Handoff management sub-module은 vMM module로부터 Handoff Trigger (HT) 메시지를 받는다. HT 메시지는 MN이 이동하기 전/후 스위치 (PS/NS)의 정보를 포함하여 { MN : 'h1', PS : 's1', NS : 's2'}의 JSON 형태로 작성된다. Handoff management sub-module은 이 HT 메시지를 바탕으로 { MN : 'h1', AR : { PS : 's1', NS : 's2'} }와 같이 MN들의 handoff 정보를 관리한다.

또한, 이 module은 HT 메시지에 의해 MN들의 handoff가 발생한다는 사실을 알게 되면 packet forwarding sub-module에게 MN들의 handoff 정보를 전달하여 버퍼링 기능을 수행하게 한다.



[그림 4] Handoff Management Sub-module.

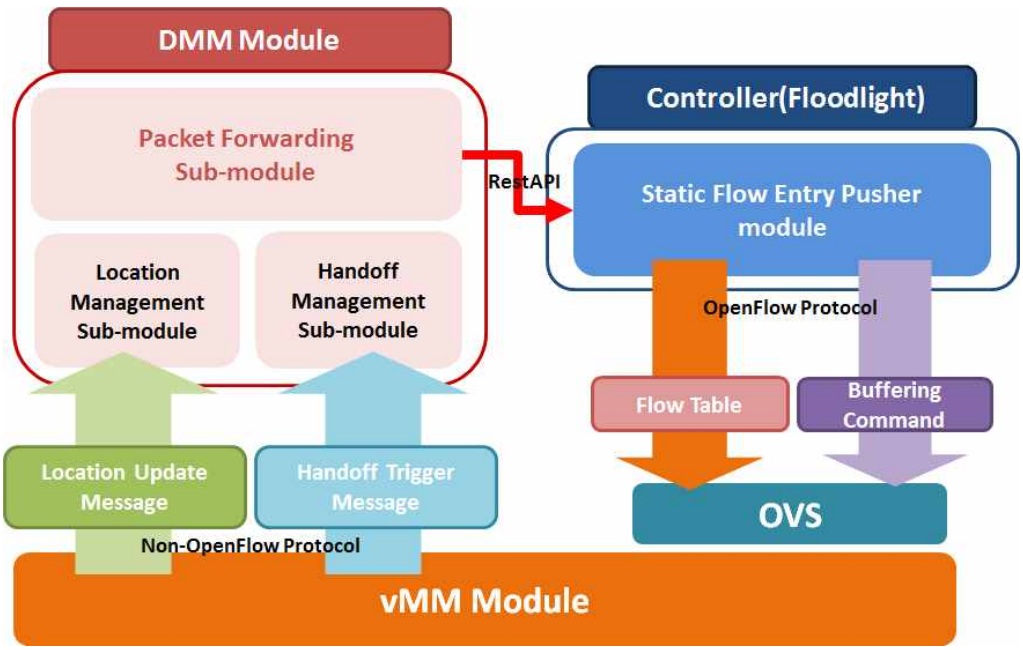
2.1.3. Packet Forwarding Sub-module

Packet forwarding sub-module은 location management sub-module에 의해 전달된 MN들의 위치정보를 바탕으로 최적의 라우팅 경로를 계산하고, handoff

management sub-module에 의해 전달된 MN들의 handoff 정보를 바탕으로 버퍼링 기능을 수행할 수 있도록 도와주는 역할을 한다. 이 packet forwarding sub-module은 [그림 5]와 같이 설계된다.

MN들의 위치정보를 바탕으로 최적의 라우팅 경로를 설정하는 기능은 다음과 같다. 먼저 location management sub-module로부터 mapping table이 변화되었다는 정보를 전달 받는다. Packet forwarding sub-module은 global view를 이용하여 최적의 경로를 찾아 Floodlight 컨트롤러의 control module인 Static Flow Entry Pusher의 Rest API를 이용하여 컨트롤러가 각 스위치에게 적절한 flow table을 생성할 수 있도록 도와준다.

다음으로, MN들의 handoff 정보를 바탕으로 버퍼링 수행을 도와주는 기능은 다음과 같다. 먼저 packet forwarding sub-module로부터 MN이 다른 스위치로 handoff가 되었다는 정보를 받고, 컨트롤러가 handoff와 관련된 두 스위치 사이(즉, handoff 전/후 스위치)에 적절한 교차 라우터에게 데이터 버퍼링을 명령할 수 있도록 도와준다.



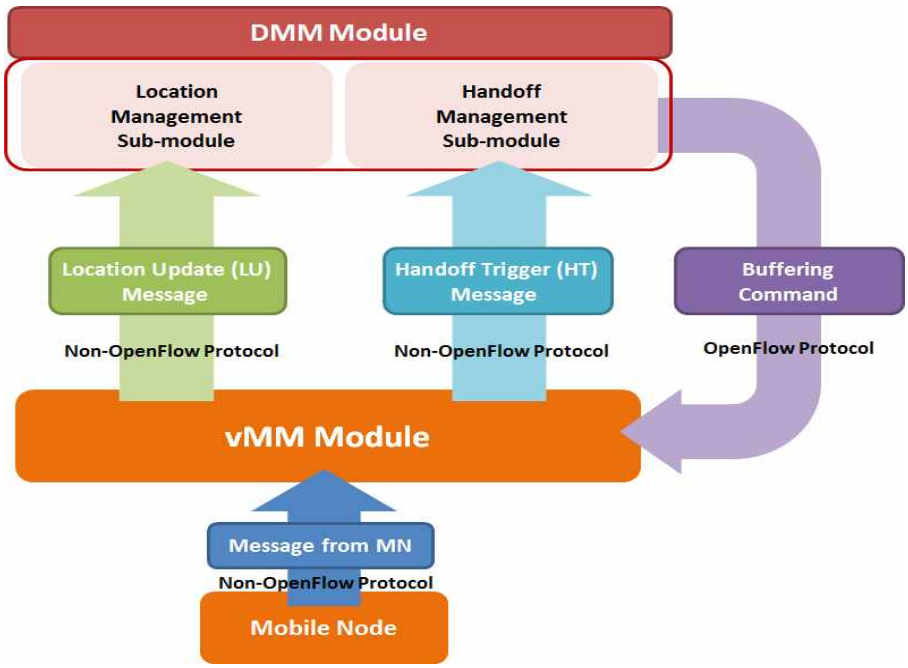
[그림 5] Packet Forwarding Sub-module

2.2. Modules for SmartX Rack

2.2.1. Virtual Mobility Management (vMM) Module

vMM module은 SmartX rack 내부의 스위치 VM에 구현되는 module로 DMM의 AR로 동작하기 위한 기능을 수행하며 [그림 6]과 같이 설계된다. vMM module은 우선적으로 MN에서 발생하는 LU 및 HT 메시지를 수신해야 한다. 해당 메시지 수신은 OpenFlow 프로토콜과는 별개로 MN과의 non-OpenFlow 프로토콜 세션을 통해서 이루어진다. 또한, vMM module은 MN으로부터 수신한 LU 및 HT 메시지를 컨트롤러의 적절한 sub-module로 전달하기 위해서 우선 MN으로부터 수신한 메시지가 LU인지 HT인지 판단한다. 만약 LU 메시지인 경우 컨트롤러의 location management sub-module과 세션을 만들고, HT 메시지인 경우에는 handoff management sub-module과 세션을 만든다. vMM module과 컨트롤러의 각 sub-module 사이를 연결하는 세션은 MN과 vMM module 사이의 세션과 같은 non-OpenFlow 프로토콜을 이용한다.

앞서 1.2에서 언급한 것과 같이 vMM module에는 핸드오프 시 버퍼링을 수행하는 기능이 추가적으로 구현된다. 컨트롤러는 핸드오프 발생 시 전달되는 HT 메시지를 수신한 후, 핸드오프 이전과 이후의 데이터 이동경로 사이에 교차지점에 있는 스위치에게 버퍼링 명령을 내린다. 버퍼링 명령을 받은 스위치는 핸드오프가 완료될 때까지 MN으로 가는 패킷을 버퍼링을 수행하게 된다.

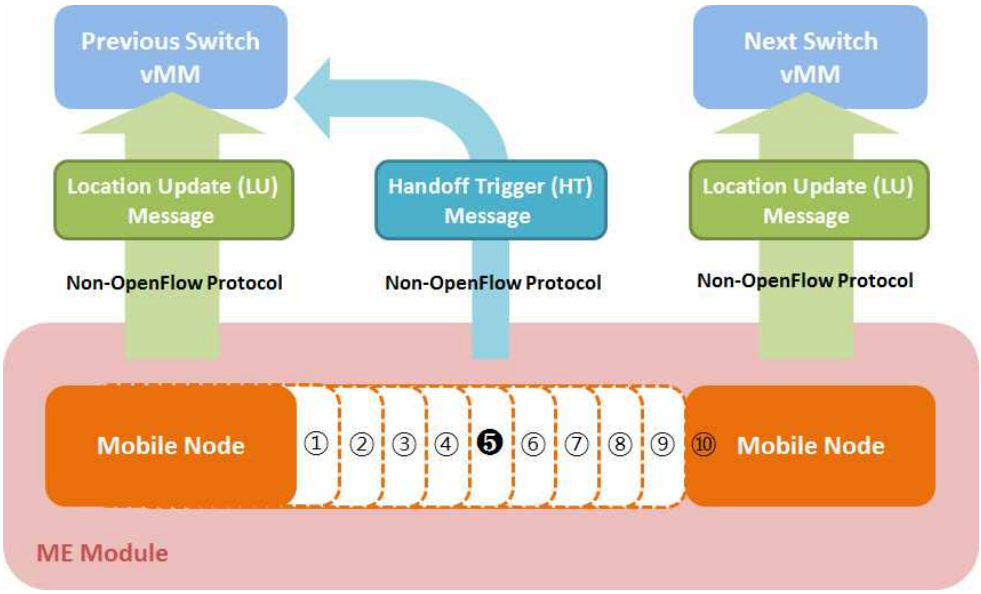


[그림 6] Virtual Mobility Management Module.

2.2.2. Mobility Emulation Module

ME module은 SmartX rack 내부의 호스트 VM에 구현되는 module로 MN의 역할을 수행하며 [그림 7]과 같이 설계된다. ME module은 MN이 새로운 스위치와 연결되는 순간 해당 스위치에게 자신이 연결되었다는 LU 메시지를 전송한다. LU 메시지 전송은 앞서 2.2.1에서 언급한 것과 같이 vMM module과 non-OpenFlow 프로토콜 세션을 이용하여 이루어진다.

한편, SmartX rack 내부의 호스트 VM은 무선 인터페이스가 없을 뿐만 아니라 MN의 필수적인 역할인 이동성을 제공하지 못한다. ME module은 이러한 이동성을 가상으로 제공하기 위해서 타이머를 이용하여 설계된다. MN이 다른 스위치로 이동하려는 순간 ME module의 타이머가 작동되기 시작한다. 타이머가 시작시점부터 종료시점까지의 중간 지점 (핸드오프 트리거 시점)에 이르렀을 때, ME module은 핸드오프 트리거 이벤트를 발생시키고 이동하기 전 스위치의 vMM module로 HT 메시지를 전송한다. HT 메시지 전송 이후 ME module은 타이머를 계속적으로 카운팅 하며, 타이머가 종료되는 시점에 MN은 새로운 스위치와 연결되고 LU 메시지를 전송한다.

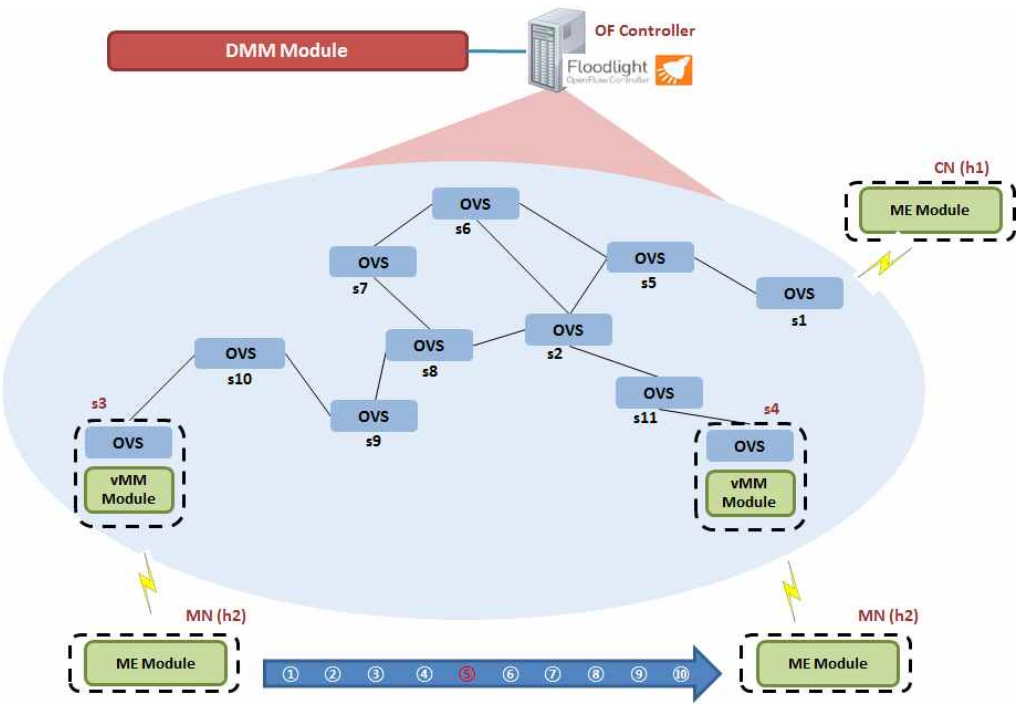


[그림 7] Mobility Emulation Module.

3. Mininet Emulation for SDN-based DMM software

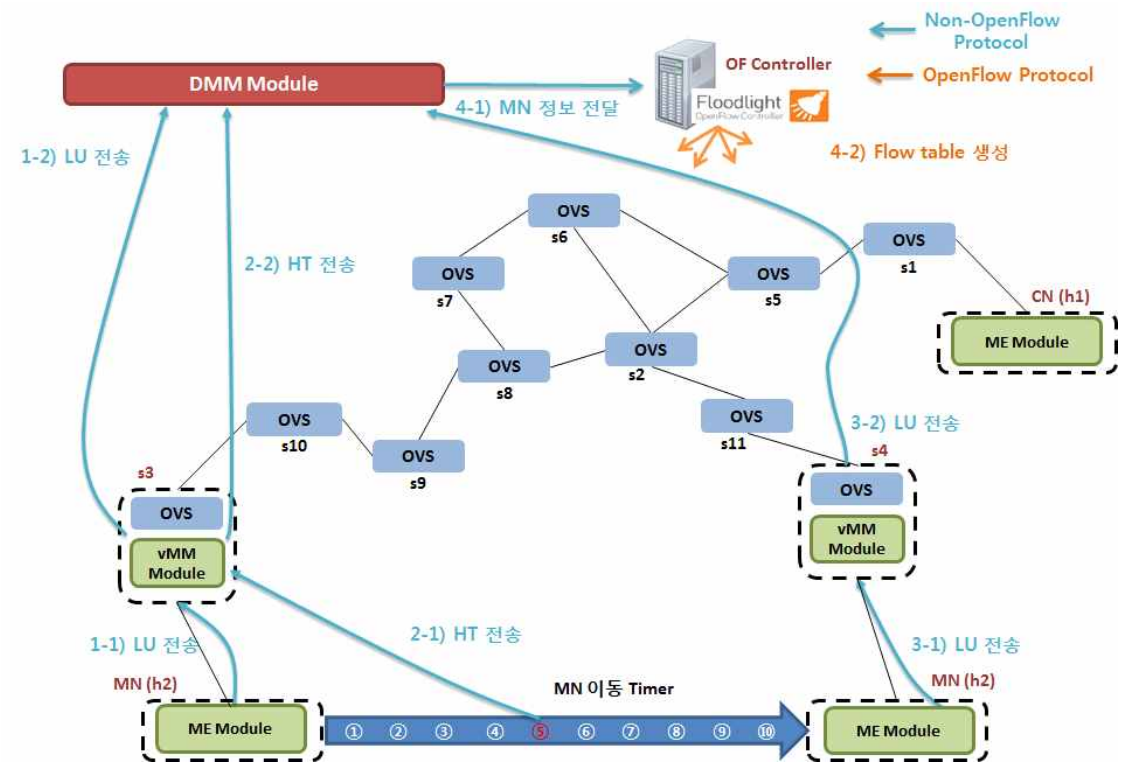
3.1. Environment

SDN-based DMM을 위한 실험환경은 [그림 8]과 같다. 우선 컨트롤러 부분에는 floodlight 네트워크 운영체제를 탑재한 컨트롤러와 앞서 설명한 DMM module이 구현된 서버로 구성되고, 스위치 및 호스트들은 Mininet을 이용하여 가상으로 생성한다. Mininet [4]은 SDN 환경의 네트워크 토폴로지를 가상으로 생성해주는 에뮬레이션 도구로, 실제 환경에 적용하기 전에 실험을 수행하는 목적으로 폭 넓게 이용되고 있다 . Mininet으로 구성된 네트워크는 실제 컨트롤러에 의해서 제어 가능하고, 각각의 스위치 및 호스트들은 굉장히 가벼운 프로세스로 가상화된다. 또한, Mininet에서 생성되는 스위치 프로세스들은 선택적으로 Open vSwitch (OVS) 기반으로 동작시킬 수 있다. 본 실험에서는 스위치 프로세스들을 OVS 기반으로 동작하도록 선택하여 구성하였다. 그리고 스위치 프로세스들 중 s3와 s4에는 앞서 설명하였던 vMM module이 탑재된다. 한편, 호스트 프로세스들은 이동을 하는 MN과 트래픽을 발생시키는 CN으로 구성하여, 앞서 설명하였던 ME module를 탑재한다. Mininet에서는 무선 인터페이스를 갖는 이동 단말을 생성할 수 없으므로, 일반적인 호스트로 MN을 생성하여 무선 인터페이스를 갖는다고 가정하였다. Mininet으로 구성된 네트워크 토폴로지는 SDN-based DMM의 기능을 검증할 수 있는 규모로 설계하였다.



[그림 8] Experiment environment

3.2. Scenario



[그림 9] Experiment scenario

SDN-based DMM의 실험 시나리오는 [그림 9]와 같이 진행된다. 우선 MN의 ME module이 이동하기 전 연결되어 있던 스위치인 s3에서 동작하는 vMM module로 LU 메시지를 전송한다. s3의 vMM module은 수신한 LU 메시지를 DMM module의 이 탑재된 서버로 전달한다. 이는 DMM module의 sub-module인 Location Management sub-module은 MN과 MN이 위치한 스위치를 mapping 한다. 이때 LU 메시지는 JSON 형식으로 되어있으며 python에서 기본적으로 제공하는 JSON API를 통하여 전달되며 파일의 형태로 저장된다.

위와 같은 동작이 완료된 후, MN이 이동하는 핸드오프 상황이 발생된다. 핸드오프는 MN에 탑재되어 있는 ME module의 타이머가 작동되는 순간 시작된다. 그리고 타이머가 시작시점부터 종료시점 사이의 중간 지점 (핸드오프 트리거 시점)에 이르렀을 때, MN의 ME module은 s3의 vMM module로 HT 메시지를 전송하고, HT 메시지는 앞서 LU 메시지와 같이 DMM module로 전달된다. 그리고 DMM module의 sub-module인 Handoff Management sub-module에 의해 MN의 핸드오프 정보가 LU 메시지와 동일한 방법으로 저장된다. 이와 같은 동작이 진행되는 동시에 ME module에서는 계속적으로 타이머가 동작하고, 타이머가 만료되었을 때, 즉 핸드오프가 완료되었을 때, MN은 새로운 스위치 s4에 연결된다. 이러한 MN의 이동성은 ME module의 타이머가 중간지점을 지나는 순간 이동하기 전의 가상 네트워크에서 이동

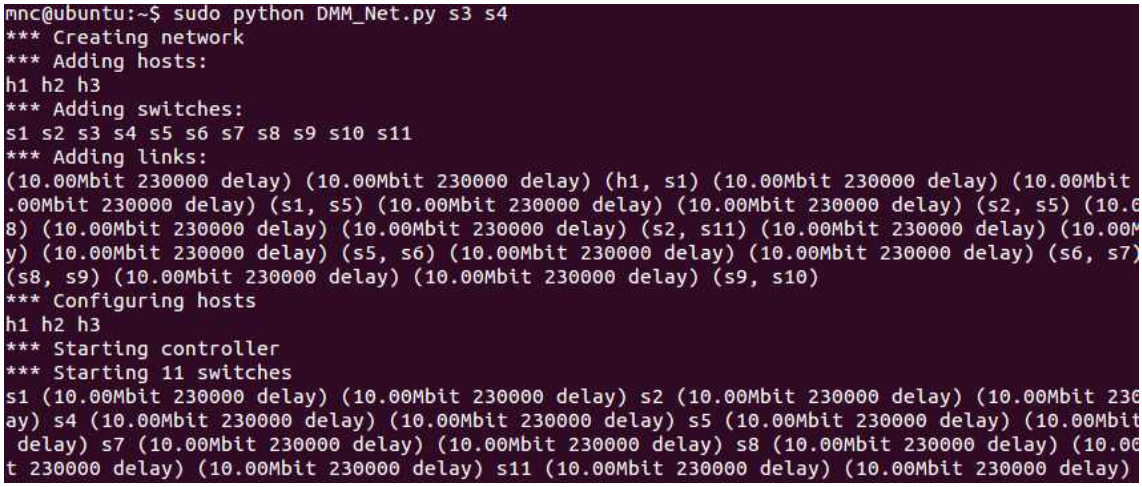
한 후의 가상 네트워크로 전환시켜 구현된다.

MN이 이동한 스위치 s4에 연결되는 순간 MN의 ME module은 s4의 vMM module로 LU 메시지를 전송한다. 해당 LU 메시지는 DMM module에 전달되어 MN이 스위치 s4에 연결되었다는 것을 알려준다. DMM module의 Packet Forwaring sub-module은 MN의 위치정보와 핸드오프 정보를 이용하여 CN에서 보내는 플로우에 대한 최적화된 경로를 결정하여 Rest-API를 통해 컨트롤러가 스위치들의 플로우 엔트리 생성을 명령하도록 한다.

위와 같은 시나리오에서 LU 및 HT 메시지의 전달은 TCP 기반의 소켓을 통해서 이루어져 있으며, vMM module와 ME module 내부에 구현되어 있다. 또한, DMM module은 MN에 대한 정보를 바탕으로 Rest-API를 통하여 컨트롤러가 플로우 테이블 생성 명령을 내릴 수 있도록 도와주며, 이는 OpenFlow 프로토콜을 기반으로 수행된다.

3.3. Execution & Results

본 실험에서는 MN의 위치등록 및 핸드오프한 결과가 DMM module에게 성공적으로 전송되는 지와 MN의 핸드오프 후, CN에서 MN에게 데이터를 전송할 때, conventional DMM (기존 DMM)과 SDN-based DMM에서의 데이터 경로에 따른 전송 딜레이를 측정하였다. 먼저 [그림 8]의 Mininet에서 구성되는 토폴로지를 만들고, ME module과 vMM module을 실행시키는 DMM_Net.py를 먼저 실행한다. DMM_Net.py 뒤쪽에는 핸드오프 과정에서 이전 스위치와 이후 스위치에 대한 ID를 입력한다. 즉, [그림 10]과 같이 s3 s4를 입력하게 되면 MN은 s3에서 s4로 핸드오프를 하게 된다. DMM_Net.py를 실행하면, 먼저 토폴로지가 생성되고, MN에 의한 위치 정보 등록 및 핸드오프 과정이 진행된다. 또한, LU 및 HT 메시지들의 전송이 성공하였다는 로그들이 출력되며, 각 메시지들에 의해서 전달된 내용은 DMM module이 동작하고 있는 서버에서 확인된다. 이 로그들은 [그림 10]과 [그림 11]과 같이 출력된다.

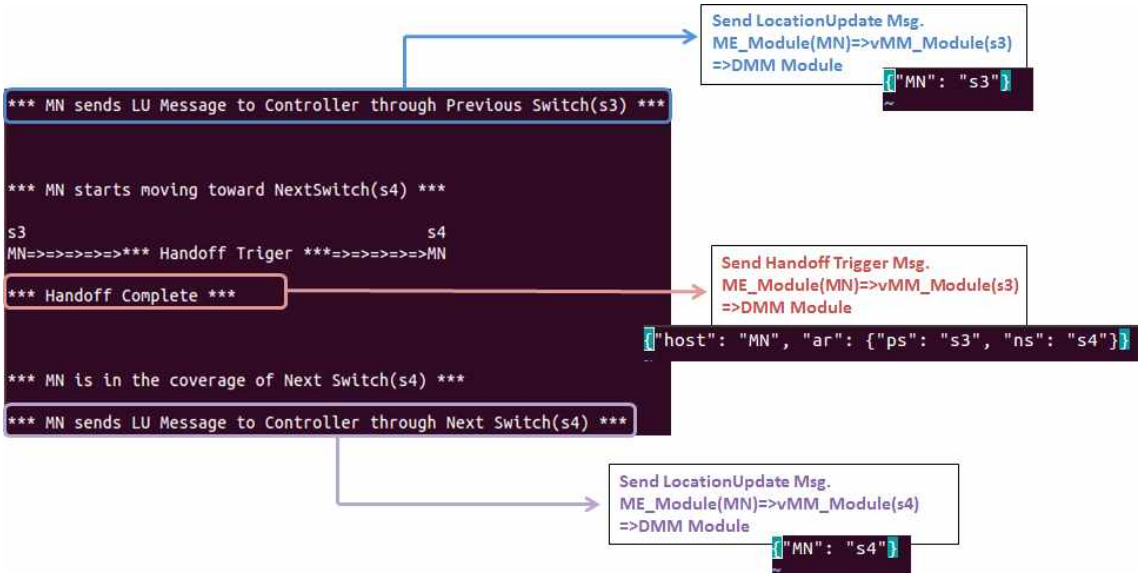


[그림 10] DMM_Net.py 실행화면

[그림 10]은 [그림 8]의 Mininet에서 구성되는 토폴로지가 생성되는 로그들을 보여준다. 특히, DMM_Net.py에서는 토폴로지안의 링크에 대한 대역폭과 딜레이를 설정할 수 있으며, 위 시나리오에서는 10Mbps, 230ms로 설정되었다.

[그림 11]에서의 청색 상자는 MN이 이동하기 전의 스위치인 s3에 연결되어 있다는 LU 메시지가 DMM module에 전송되었을 시 출력되는 로그이고, 해당 LU 메시지 내용에는 h2 호스트가 s3 스위치에 위치해 있다는 정보가 담겨져 있다. DMM module은 LU 메시지의 내용을 바탕으로 MN의 위치정보를 {“MN” : “s3”} 형태로 맵핑한다. 다음으로, MN이 s3 스위치에서 s4 스위치로 이동하고 있다는 MN의 핸드오프 발생에 대한 로그를 확인할 수 있다. 이는 ME module의 타이머가 증가할 때마다 ‘=>’ 로그가 출력되어 MN이 이동하고 있음을 확인할 수 있다. 적색 상자는 타이머가 중간지점까지 증가하였을 때, 핸드오프 트리거 이벤트가 발생하였다는 것을 나타내며, 핸드오프 트리거 이벤트에 의한 HT 메시지의 내용에는 MN이 s3 스위치에서 s4 스위치로 이동하고 있다는 핸드오프 정보가 담겨져 있다. DMM module은 HT 메시지의 정보를 바탕으로 MN의 이동 정보 (핸드오프 정보)를 {”host“ : MN”, “ar” : “ps” : “s3” , “ns” : “s4”} 형태로 저장한다. 그리고 ME module의 타이머가 계속적으로 증가하여 만료되면, MN의 이동이 끝났다는 로그를 출력한다. 이동이 완료된 MN은 새로운 스위치인 s4와 연결되어 있다는 LU 메시지를 DMM module로 전송한다. 보라색 상자는 이를 확인할 수 있는 로그이고, 해당 LU 메시지 내용에는 MN이 s4 스위치에 위치해 있다는 정보가 담겨져 있다. 이러한 정보를 통해서 DMM module은 MN의 위치 정보를 {“MN” : “s4”} 형태로 새롭게 갱신한다.

다음으로 저장된 위치 정보 및 핸드오프 정보를 바탕으로 Rest API를 통해 컨트롤러가 플로우 테이블을 내려줄 수 있게 하는 DMM module을 실행한다. DMM module 실행방법은 [그림 12]와 같이 실행하고, conventional DMM (기존 DMM)은 tra, SDN-based DMM은 sdn을 이용하여 실행한다.



[그림 11] 위치 정보 및 핸드오프 과정 결과 화면

```
mnc@ubuntu:~$ sudo python DMM_module.py tra
mnc@ubuntu:~$ sudo python DMM_module.py sdn
```

[그림 12] DMM module 실행방법

먼저, DMM_module.py tra, 즉 conventional DMM에 대해서 실행하게 되면 [그림 13]의 오른쪽 화면과 같이 MN의 핸드오프 후, CN에서 MN으로의 데이터 경로에 대한 예상 hop수가 출력되고, 컨트롤러가 그에 맞는 플로우 테이블을 각 스위치에게 내려준다. 본 시나리오에서 conventional DMM에 대해서는 14 hop으로 나오게 된다. 다음으로, Mininet상에서 CN에서 MN으로 ping을 보내어, conventioanl DMM에서의 MN의 핸드오프 후 전송 딜레이를 측정하였다. 이 측정값은 [그림 13]에서와 같이 출력되고, 10번 전송하였을 경우, 평균 딜레이가 644.584ms가 나오게 되는 것을 알 수 있다.

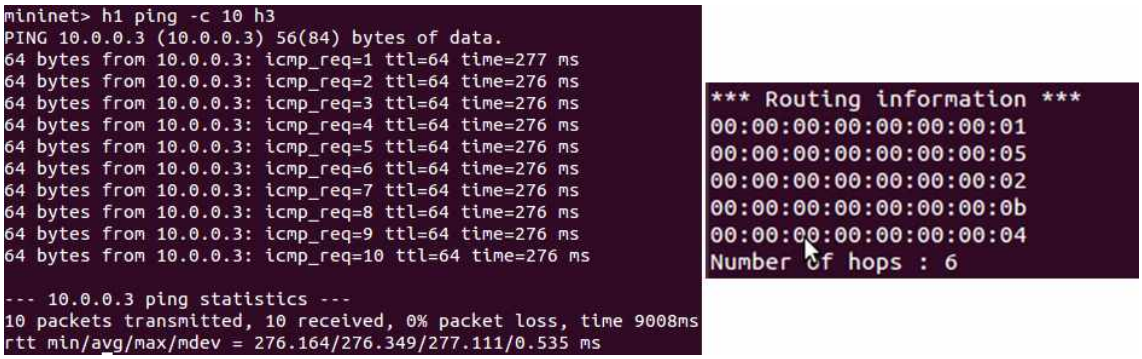
```
mininet> h1 ping -c 10 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_req=1 ttl=64 time=646 ms
64 bytes from 10.0.0.3: icmp_req=2 ttl=64 time=644 ms
64 bytes from 10.0.0.3: icmp_req=3 ttl=64 time=644 ms
64 bytes from 10.0.0.3: icmp_req=4 ttl=64 time=644 ms
64 bytes from 10.0.0.3: icmp_req=5 ttl=64 time=644 ms
64 bytes from 10.0.0.3: icmp_req=6 ttl=64 time=644 ms
64 bytes from 10.0.0.3: icmp_req=7 ttl=64 time=644 ms
64 bytes from 10.0.0.3: icmp_req=8 ttl=64 time=644 ms
64 bytes from 10.0.0.3: icmp_req=9 ttl=64 time=644 ms
64 bytes from 10.0.0.3: icmp_req=10 ttl=64 time=644 ms

--- 10.0.0.3 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 8998ms
rtt min/avg/max/mdev = 644.347/644.584/646.587/0.838 ms
```

```
*** Routing information ***
00:00:00:00:00:00:00:01
00:00:00:00:00:00:00:05
00:00:00:00:00:00:00:02
00:00:00:00:00:00:00:08
00:00:00:00:00:00:00:09
00:00:00:00:00:00:00:0a
00:00:00:00:00:00:00:03
00:00:00:00:00:00:00:0a
00:00:00:00:00:00:00:09
00:00:00:00:00:00:00:08
00:00:00:00:00:00:00:02
00:00:00:00:00:00:00:0b
00:00:00:00:00:00:00:04
Number of hops : 14
```

[그림 13] Conventional DMM의 경로에 대한 hop 수 및 ping test 결과

마지막으로, DMM_module.py tra, 즉 SDN-based DMM에 대해서 실행하게 되면 마찬가지로 [그림 14]의 오른쪽 화면과 같이 MN의 핸드오프 후, CN에서 MN으로의 데이터 경로에 대한 예상 hop수가 출력되고, 컨트롤러가 그에 맞는 플로우 테이블을 각 스위치에게 내려준다. 본 시나리오에서 SDN-based DMM에 대해서는 6 hop으로 나오게 된다. 다음으로, conventional DMM의 경우와 똑같이 Mininet상에서 CN에서 MN으로 ping을 보내어, SDN-based DMM에서의 MN의 핸드오프 후 전송 딜레이를 측정하였다. 이 측정값은 [그림 14]에서와 같이 출력되고, 10번 전송하였을 경우, 평균 딜레이가 276.349ms가 나오게 되는 것을 알 수 있다.



[그림 14] SDN-based DMM의 경로에 대한 hop 수 및 ping test 결과

Conventional DMM과 SDN-based DMM에서의 전송 딜레이의 차이는 약 368ms정도 차이가 나며, 예상 경로의 hop수는 8 hop차이가 난다. 즉, SDN-based DMM을 이용하게 되면 MN의 핸드오프 시 기존 DMM에 비해 최적화된 경로를 설정할 수 있으며, MN의 이동성이 많은 환경에서 더 향상된 성능을 보일 수 있다.

4. Experiment for SDN-based DMM software

본 장은 SmartX Rack을 이용한 실험이 완료된 후 작성될 예정이다.

4.1. Environment

4.2. Scenario

4.3. Execution & Results

5. Conclusion

SDN-based DMM 소프트웨어는 기존 분산 이동성 관리 기법이 MN의 이동성이 발생할 때의 최적화된 경로를 설정하지 못하는 문제에 대한 솔루션을 제공한다. 소프트웨어 사용자들은 각 스위치 (SmartX Rack)에 위치 정보 및 핸드오프 트리거 정보 전달 등의 기능을 갖고 있는 vMM module과 ME module을 탑재하고, DMM module과 컨트롤러를 이용하여 쉽게 SDN-based DMM을 실험할 수 있다. 이 소프트웨어는 현재는 한정된 네트워크, 즉 단일 도메인에서의 제한적인 실험이 이루어지지만, 추후 대규모 네트워크에서의 분산 이동성 관리 실험 뿐만 아니라 모바일 네트워크에서의 다양한 서비스들을 가상화된 모듈 형태로 실험할 수 있는 기반이 될 것으로 기대된다.

6. References

[1] IETF Distributed Mobility Management Working Group: http://datatracker.ietf.org/wg/dmm/charter/.

[2] B. Sarikaya, "Distributed Mobility IPv6," *IETF Internet-Draft*, draftsarikaya- dmm-dmipv6-00, work in progress, August 2012.

[3] C. Bernardos, A. Oliva, and F. Giust, "A PMIPv6-Based Solution for Distributed Mobility Management," *IETF Internet-Draft,* draftbernardos-dmm-pmip-03, work in progress, July 2013.

[4] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. ACM SIGCOMM Workshop on Hot Topics in Networks 2010*, Monterey, USA, October 2010.

SDN-based DMM Software 매뉴얼

- 이 문서의 기술적인 내용은 OF@TEIN 프로젝트의 진행과 함께 별도의 예고 없이 변경될 수 있습니다.
- 최신의 정보로 갱신된 매뉴얼에 대한 정보와 매뉴얼에 대한 문의 사항은 아래의 정보를 참조하시길 바랍니다.
  - Homepage: trac.nm.gist.ac.kr/tein_public
  - E-mail: tein_admin@nm.gist.ac.kr

작성기관: 광주과학기술원
작성년월: 2013/12