IPv6 사이트 멀티호밍을 위한 리눅스 커널 기반의 L3Shim 구현에 관한 연구

*임영빈, *류지호, *박건우, *권태경, *최양희,
**유태완, **이승윤

*서울대학교 컴퓨터공학부,
*{ybim, jhryu, kwpark}@mmlab.snu.ac.kr, *{tkkwon, yhchoi}@snu.ac.kr
**한국전자통신연구원
**{twyou, syl}@etri.re.kr

A study on the L3Shim Implementation for IPv6 Site Multihoming based on Linux Kernel

*Youngbin Im, *Jiho Ryu, *Kunwoo Park, *Taekyoung Kwon, *Yanghee Choi

**Taewan You, and Seungyun Lee

*School of Computer Science and Engineering, Seoul National University, Seoul, Korea

*{ybim, jhryu, kwpark}@mmlab.snu.ac.kr, *{tkkwon, yhchoi}@snu.ac.kr

**Electronics and Telecommunications Research Institute, Daejeon, Korea

**{twyou, syl}@etri.re.kr

요 약

본 논문은 현재 IETF 에서 표준화가 진행되고 있는 L3Shim 사이트 멀티호밍 프로토콜을 리눅스 커널에 구현하는 방법을 기술한 것이다. L3Shim 은 기존 호스트와 shim-enabled 호스트 사이의 통신 및 shim-enabled 호스트들끼리의 통신에서 단말식별자와 위치식별자를 구분함으로써 현재 맺어져 있는 커넥션을 유지하면서 멀티호밍을 지원하는 프로토콜이다. 본 연구진은 단말들이 L3Shim 을 사용할 경우 일어날 수 있는 성능저하요소를 제거하기 위하여 L3Shim 의 핵심 부분은 모듈의 형태로 리눅스 커널에 직접 구현하였다. 그리고 L3Shim 의 나머지 부분은 하나의 응용프로그램으로 구현함과 동시에 API 를 제공함으로써 사용자 또는 타 개발자가 접근하기 쉽도록 설계했다. 본 연구를 통하여 L3Shim 은 낮은 오버헤드를 가지고 동작할 수 있음을 보이고 추가적인 draft 형태로 존재하는 signaling 프로토콜을 통하여 오류 자동 감지 및 자동 복구에의 활용 가능성을 확인한다.

1. 서론

인터넷의 발달로 하나의 사이트를 2 개 이상의 인터넷서비스 제공자(ISP)로 연결함으로써 한 ISP 로부터 연결에 장애가 발생하였을 경우 다른 링크를 통하여 인터넷에 연결할 수 있도록 하는 사이트 멀티호밍(site-multihoming)이 보편적인 현상으로 나타나고 있다. 그러나 현재 수준의 멀티호밍은 단순히 외부 링크로의 다중화만 제공할 뿐 부하 분산(load sharing)이나 자동 오류 감지 및 복구(failure detection & recovery)와 같은 장점은 제공하지 못하고 있다. 또한 근본적인 문제점으로 백본 라우터에서의 라우팅 테이블의 크기를 기하급수적으로 증가시키는 문제가 발생하고 있으며 새로운 인터넷 표준인 IPv6[1]에는 적용이 불가능하다.

상기 문제들을 해결하기 위하여 IETF(Internet Engineering Task Force)의 multi6 (site multihoming in IPv6) 워킹그룹에서는 IPv6 환경에서의 사이트 멀티호밍 방안에 대한 표준으로 IP 주소가 내포하는 단말식별자(identifier)와 위치식별자(locator)를 분리하는 개념을 도입한 L3Shim[2] 이라는 솔루션을 제안하였다. 단말식별자와 위치식별자를 분리하면 DFZ(default-free zone)에서의 라우팅 테이블 크기를 줄일 수 있으며, ISP 변경시에 주소를 재할당하는 renumbering 이 용이하고, 세션의 중단 없이 이동성(mobility)을 제공할 수 있으며, 라우팅 시스템 변경 없이 트래픽을 조절할 수 있는 능력을 얻을 수 있다.

L3Shim 프로토콜은 IPv6 환경에서, 여러 가지 종류의 failure 발생시에도 기존 통신을 유지하고, 한 호스트로의 여러 커넥션들이 해당 호스트의 서로 다른 locator 를 사용함으로 load 를 분산시킬 수 있도록 locator agility 를 제공하는 프로토콜이다. 멀티호밍을 지원하기 위해서는 identifier 가 두 개 이상의 locator 와 동적으로 매핑되어야 하며 이 작업은 기존 인터넷 프로토콜 스택의 transport 계층과 IP 계층 사이에서 이루어진다. L3Shim 은 public key 와 같은 새로운 개념을 도입하는 것이 아니라 기존 IP 주소와 동일한 형태의 identifier 를 사용하게 함으로써 기존의 호스트들과도 호환된다는 장점이 있다.

L3Shim 과 마찬가지로 identifier 와 locator 의 분리 개념을 제안한 기법으로는 VIP(Virtual Internet Protocol)[3], LIN6(Location Independent Network for IPv6)[4], HIP(Host

Identity Protocol)[5] 등 다수 존재하지만 나머지 기법들이 단말에 대한 이동성 지원을 주 목적으로 하는 것에 비해 L3Shim 은 멀티호밍에 초점을 맞추고 있으며 향후 이동성 지원을 추가하는 전략으로 표준화가 진행될 것이다.

현재 L3Shim 의 표준화는 multi6 워킹그룹의 다음 단계로 만들어진 SHIM6 (site-multihoming by IPv6 intermediation) 워킹그룹에서 진행되고 있다. 현재 L3Shim 관련 구현으로는 벨기에 catholique de Louvain 대학의 Sébastien Barré 의 석사 논문을 기반으로 리눅스 상에서 L3Shim 을 구현한 LinShim6, 유닉스 기반 시스템에서 네트워크 트래픽 분석 및 스니핑을 수행하는 프로그램에서 L3Shim 트래픽을 분석하도록 패치한 Wireshark, 본 논문의 일부 저자들이 참여하여 리눅스 유저 어플리케이션 상에서 Netfilter 와 Iptables 을 사용하여 구현한 서울대학교의 구현이 있다. 그 밖에도 진행되고 있는 세션에서 패킷 전송을 위한 주소를 변경하면 많은 공격을 받을 수 있는데, 이러한 공격으로부터 공개키 암호화 등을 사용하지 않고 효율적으로 방어할 수 있는 메커니즘을 제공하는 HBA 를 구현한 Francis Dupont 의 구현 등이 있다[6].

2 장에서는 저자들이 구현한 L3Shim 프로토콜의 기본 구조 및 동작을 소개하며, 3 장에서는 개발 환경 및 툴에 대해 소개하고 개발한 시스템의 구조를 설명한다. 4 장에서는 구현한 시스템의 성능 테스트 결과를 제시한다. 마지막 5 장에서 결론을 맺는다.

2. 시스템 구조 및 동작

2.1 기본 구조

L3Shim 은 TCP 커넥션이나 UDP 스트림 등에서 failure 발생시에도 기존 설정된 communication 을 유지하며, 상위 레이어 특히 transport 프로토콜에 최소한의 영향을 미쳐서 기존의 방식대로 동작할 수 있도록 한다. 또한 여러 가지 보안 위협들을 방어하도록 하며, 일정 수의 패킷이 교환되었을 때에만 shim state 를 설정하도록 함으로써 shim state 를 설정하는데 필요한 추가의 round trip 이 세션 생성 초기에 필요하지 않도록 한다. 그 밖에도 load 분산을 가능하게 하는 것을 목적으로 한다.

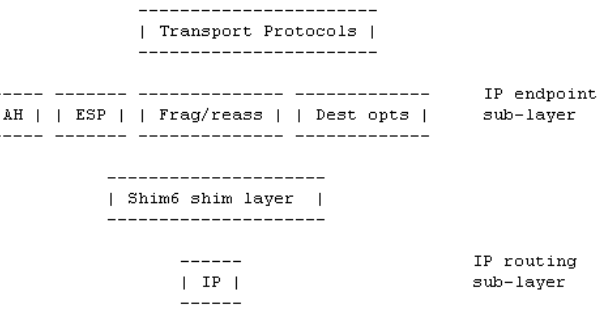

그림 1. Protocol stack

그림 1 과 같이 Shim 레이어는 IP 라우팅 프로토콜 바로 위에 위치한다. Shim 레이어 위에는 TCP, UDP 같은 transport 프로토콜과 ICMP 같은 컨트롤 프로토콜, OSPF 같은 라우팅 프로토콜, IPX, AppleTalk, 또는 IP 의 IP 상 터널링 프로토콜이 존재한다. AH, ESP 를 shim 위에 위치시킴으로써 IPsec 이 locator 의 변화를 알지 못하도록 해서 locator 의 변화 시에도 안정적으로 동작할 수 있도록 한다. 그 밖에도 fragmentation header 를 shim 위에 위치시켜서 fragment 마다 서로 다른 path 를 거침으로써 다른 source locator 를 사용하는 경우에도 프레임 재조합을 안정적으로 동작하도록 한다.

L3Shim 은 새로운 identifier 네임 공간을 도입하지 않고 remote peer 와 초기 contact 에서 지속적으로 사용할 identifier, 즉 Upper Layer Identifier(ULID)를 선택할 때 네트워크 레이어에서 사용하는 실제 주소인 locator 의 하나로 선택한다. 초기에 선택한 locator 를 사용하는데 있어서 failure 가 발생함에 따라 network 레벨에서 사용하는 locator 는 시간이 지나면서 달라질 수 있지만 상위 레이어 프로토콜들은 ULID 를 변함없이 사용한다.

2.2 기본 동작

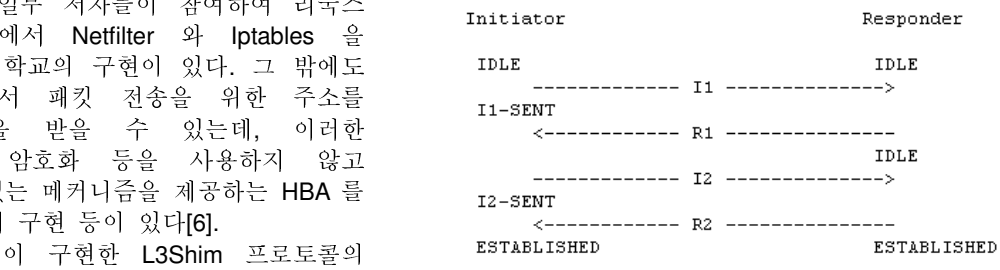

그림 2. Normal context establishment

그림 2 와 같이 L3Shim 에서는 context 를 설정하기 위해 I1, R1, I2, R2 의 네 가지 메시지를 이용하며 보통 이 순서대로 전송된다. 이때 peer 의 locator 리스트가 교환된다. 그러나 Update Request 메시지와 Update Acknowledgement 메시지 교환, Locator List Option 을 사용해서 locator set 은 동적으로 변화한다. 그 밖에도 Update Request 메시지에 preference option 을 사용해 locator 의 선호도를 조정할 수도 있다. R1bis 와 I2bis 메시지도 정의되었는데 context 가 손실되었을 때 회복하기 위해 사용한다.

L3Shim 세션이 생성된 직후에는 Shim 레이어는 아무런 동작을 하지 않는다. 그러나 failure 가 발생하여 2.3 절에서 설명되는 REAP 프로토콜[7]이 동작하면 문제가 생긴 ULID 대신 새로운 locator 가 해당 세션에 할당이 되고, Shim 레이어의 역할이 시작된다. 상위 레이어에서 패킷을 받으면 Shim 레이어는 목적지 주소를 ULID 에서 현재 사용되는 locator 로 변경한다. 수신 측에서는 패킷이 Shim 레이어에 도착하면 목적지 주소를 현재의 locator 에서 ULID 로 변경한다. 이렇게 ULID 가 실제로 동작하지 않음에도 불구하고 주소 변경을 통해 상위 레이어가 ULID 를 계속 사용해서 통신할 수 있도록 한다.

2.3 REAchability 프로토콜(REAP)

Failure 의 발생을 인지하기 위해서 상대 호스트와의 접근성을 확인하는데 이를 위해서 Forced Bidirectional Detection(FBD) 이라는 기술이 사용된다. 이것은 한쪽 방향으로 데이터 트래픽이 있을 때 다른 방향으로도 트래픽이 있도록 함으로써 이것이 실패했을 때 접근이 불가하다고 결정하는 방법이다. 즉 한쪽 방향으로 데이터를 전송했을 때 반대 방향으로 데이터가 오거나 transport 계층의 ACK 이 오면 접근 가능하다고 결정하고, 보낼 트래픽이 없을 때에는 주기적으로 Keepalive 메시지를 보내도록 함으로써 상대가 접근성을 확인할 수 있도록 한다. 특정 locator pair 를 사용해 데이터가 전송 되었으나 일정 시간 동안 되돌아오는 패킷이 없으면 해당 locator pair 에 failure 가 발생한 것이다.

locator pair 의 한쪽 address 가 사용할 수 없게 되거나 pair 자체가 운용할 수 없게 되면 통신이 재개될 수 있도록 다른 pair 를 찾으려 시도하게 된다(Full Reachability Exploration). 문제를 처음 발견하는 측에서 상대에게 probe 메시지를 보내서 자신이 가지고 있는 주소 정보의 각 주소 pair 를 시도해 본다. 메시지에는 peer 간에 연결 상태에 관한 정보가 담겨 있다. probe 메시지를 받게 되면 자신도 parallel 한 exploration 을 시작함으로써 상대의 동작을 돕는다. 가능한 후보 주소 pair 의 집합 중에 사용 가능한 것을 발견할 때까지 계속 test 를 진행한다. 또한 실패한 경우 이 과정을 반복할 수 있다. 모든 노드들은 이 과정들을 순차적으로 진행하며 exponential back-off 를 사용한다.

3. 구현 시스템

3.1 개발 환경

Linux kernel 2.6.18 에 기반하여 Netfilter 및 Iptables 을 사용하여 L3Shim 을 개발하였다. 일부 저자들이 참여하여 2006 년도에 개발했던 버전의 경우 user space level 에서 동작하기 때문에 packet 하나하나의 처리 속도가 매우 늦다는 단점이 있었다. 이를 보완 하기 위해 kernel level 에서 동작 하도록 kernel module programming 을 활용하였다.

3.2 Netfilter 및 Iptables

Netfilter 는 리눅스에서 일종의 firewall 과 같은 기능을 제공하는 tool 이다. 즉 리눅스에서는 Netfilter 를 통해서 현재 자신에게 오는 packet 들을 선별하여 받거나 버리도록 할 수 있다. 이 때 Netfilter 는 전체 network protocol stack 중에서 총 5 지점에서 패킷을 filtering 할 수 있는 point 를 제공하는데, 이를 훅이라고 한다. IPv6 를 기준으로 그림 3 과 같이 NF_IP6_LOCAL_IN(2), NF_IP6_PRE_ROUTING(1), NF_IP6_FORWARD(3), NF_IP6_LOCAL_OUT(5), NF_IP6_POST_ROUTING(4) 라는 이름으로 제공되는데, 각각의 훅에서는 패킷이 그 지점을 통과할 때 Iptables 를 통해서 해당 패킷을 조작할 수 있다.

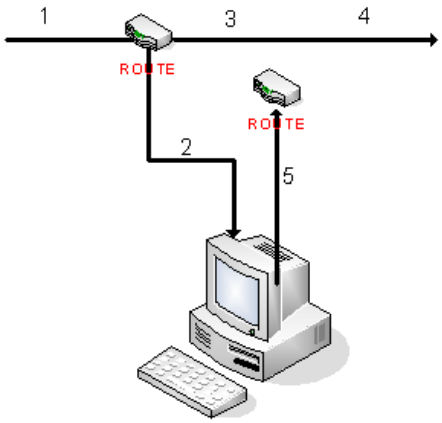

그림 3. 네트워크 아키텍처에서 netfilter 의 다섯 개의 훅

Iptables 는 Netfilter 의 각각의 훅에 규칙을 적용시킬 때 사용된다. Iptables 을 사용하여 규칙을 적용시킬 때에는 크게 MATCH, TARGET, TABLE 이 필요하다. MATCH 는 훅에서 어떠한 동작을 할 것인가를 나타내는 것으로, ACCEPT, DROP, RETURN, QUEUE 등의 동작을 할 수 있다. TARGET 은 훅에서 어떠한 패킷을 대상으로 작업할 것인가를 결정짓는 것으로 IP 주소, port, 프로토콜 등으로 구분할 수 있다. 끝으로 TABLE 에는 용도에 따라 Filter table, Mangle table, NAT table 의 3 가지가 있다.

리눅스 커널의 패킷을 직접 처리하는 함수들에서 L3Shim 을 구현할 수 있지만 커널을 직접 수정하는 것이기 때문에 커널을 재 배포하여야 하고 사용자는 새로운 커널을 설치해야만 한다. 따라서 Netfilter 와 Iptables 를 이용한 모듈 프로그래밍을 통해 구현함으로써 모듈만 설치하면 L3Shim 을 이용할 수 있도록 하였다.

3.3 IPv6 testbed

IPv6 환경 구축을 위해 6to4 터널링 기법[8]을 사용하였다. 6to4 터널링은 공인 IPv4 네트워크에 있는 Dual Stack 단말(PC 또는 라우터)이 원격지의 IPv6 네트워크까지 터널을 형성하는 방법이다.

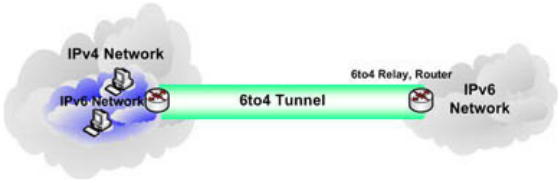

그림 4. 6to4 터널링 기법

그림 4 의 6to4 Relay 는 6to4 네트워크에서 전달된 데이터를 다른 6to4 Relay 로 전달하는 역할을 하며, 6to4 Relay Router 는 6to4 네트워크에서 전달된 데이터를 다른 IPv6 네트워크로 전달하는 역할을 한다.

3.4 시스템 구조

본 연구에서 L3Shim 은 기본적으로 커널 공간에 존재한다. 커널 내부에 상주하고 있는 L3Shim 모듈은 모든 incoming/outgoing 패킷들에 대하여 L3Shim 에서 관리하고 있는 mapping table 을 참고하여 IPv6 address translation 을 수행한다. L3Shim REAP 모듈은 L3Shim 에서 연결 복구 작업을 위해 필요한 모듈로 사용자 공간에서 커널의 L3Shim 모듈을 지원한다.

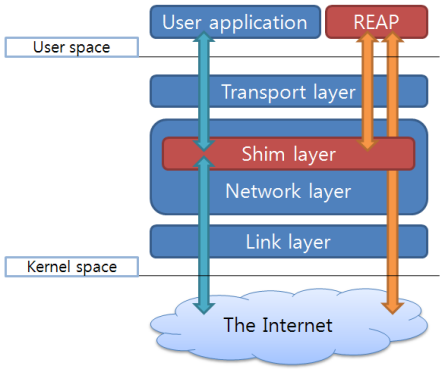

그림 5. L3Shim 구현 디자인

그림 5 에서 왼쪽의 두 화살표는 payload 가 전송되는 경로를 표시한 것이고, 오른쪽의 두 화살표는 L3Shim control 패킷이 전송되는 경로를 표시한 것이다. 사용자 응용 프로그램은 shim layer 의 존재 여부와 상관없이 동작한다. shim layer 가 존재하면 현재 연결상태에 대한 mapping table 을 구축하게 된다. 만약 연결상태에 문제가 생긴다면 REAP 모듈은 상대방 REAP 모듈과 alternative path finding 동작을 함으로써 새로운 locator 를 사용하여 연결을 복구하게 된다.

4. 성능 평가

구현한 L3Shim 프로토콜이 잘 동작하는지 확인하기 위하여 저자들은 Iperf[9] 를 사용하여 두 개의 호스트 A, B 간에 TCP traffic 을 만들어 throughput 을 측정해 보았다.

실험에 사용된 L3Shim 프로토콜의 기본 parameter 들은 다음과 같다.

I1 메시지 TIMEOUT	1 초
I1 메시지 retry limit	5
I2 메시지 TIMEOUT	1 초
I2 메시지 retry limit	5
Send TIMEOUT	A:10 초 B:15 초
Keepalive TIMEOUT	A:5 초 B:10 초
Table entry TIMEOUT	300 초

표 1. L3Shim 프로토콜 기본 parameter

위의 표 1 에서는 기본적인 parameter 값들에 대해서만 언급하였다. 위에서 언급하지 않은 parameter 들은 대부분 draft 에 나와 있는 default 값을 사용하였다. 특별히 Send TIMEOUT 이나 Keepalive TIMEOUT 의 경우 draft 기본 설정 값과 다르며, 두 호스트간에도 서로 다른 값을 선택하였다. 이는 구현에서 대부분의 parameter 값을 사용자가 임의로 선택할 수 있도록 했는데 설정 값이 차이를 보이는 경우에도 일정 수준까지는 프로토콜이 올바로 동작함을 보이기 위해서이다. 위에서 Table entry TIMEOUT 이란 L3Shim 에서 관리하는 세션 중에서 특정시간 동안 주고 받은 메시지가 없는 경우를 세션이 끝났다고 판단하여 더 이상 L3Shim 에서 세션을 관리하지 않기 위한 설정 값이다. 이는 메모리 재 사용을 위해서 필요한 값인데, 값이 너무 클 경우 너무 많은 세션을 관리하는데 따른 오버헤드가 클 수 있고, 너무 짧은 경우 순간적으로 긴 딜레이에 의한 일시적인 세션 끊김 현상이 세션이 끝난 것으로 오인될 수 있으므로 선택할 때 주의하여야 한다.

실험 시나리오는, 호스트 A 에서 iperf 를 이용해서 두 개의 TCP session 을 20 초 간격으로 만든 뒤, 30 초 뒤에 호스트 B 의 interface 중 TCP 세션들이 사용하고 있는 interface 를 강제로 다운시켰다. 단 asymmetric 링크의 경우에도 프로토콜이 제대로 동작함을 확인하기 위해 B 는 다른 호스트로 데이터를 보낼 수는 있지만 받을 수는 없도록 설정하였다. 구현한 L3Shim 프로토콜이 잘 동작 한다면 REAP 프로토콜이 동작하여 이용 불가능한 path (interface)의 주소 대신에 다른 주소를 선택하여 다시 TCP 세션을 복구 할 것이다.

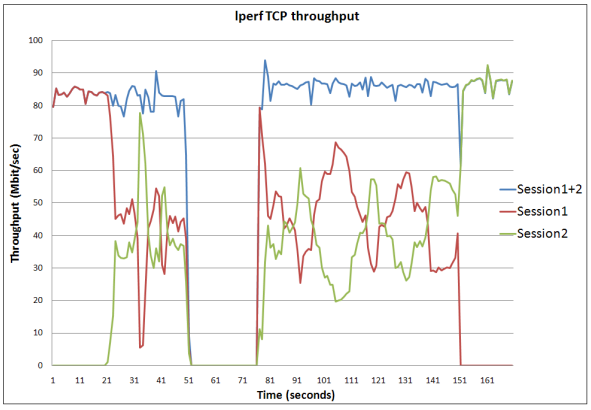

그림 6. Iperf 실험 결과

그림 6 은 앞에서 설명한 시나리오의 실험 결과를 보여준다. 두 개의 TCP 세션의 시간의 변화에 따른 throughput 과 전체 throughput 을 보여 주고 있다.

첫 번째 TCP 세션이 생기고 20 초 후에 두 번째 TCP 세션이 생겼으며 30 초 후에 사용하던 interface 가 다운 되었기에 throughput 이 0 으로 떨어진 것을 볼 수 있다. 이후 REAP 프로토콜이 동작하여 최종적으로는 약 25 초간 connection 이 끊겼다가 다시 복구된 모습을 볼 수 있다. 이 값은 parameter 설정에 따라 달라질 수 있는데 이 경우 asymmetric 링크로 인해 양방향으로 통신이 단절된 경우보다 오류 복구를 위해 더 긴 시간이 필요함을 알 수 있다. 즉 60 초경 B 가 A 에게 Keepalive 메시지를 보내게 되는데 이것이 A 에게 도달하므로 한쪽이 문제를 발견하게 되는 때는 60 초 에 B 가 A 에게 Keepalive 를 보내면서 Send timer 설정 후, 15 초 동안 A 가 패킷을 보내지 않아서 B 가 발견하거나, A 가 60 초에 Keepalive 를 받고 5 초 후에 B 에게 Keepalive 를 보내면서 Send timer 를 설정한 후 10 초가 지난 후이다. 한쪽이 문제를 발견해서 locator 를 바꿔가며 Probe 메시지를 보내면 1~2 초 후에 복구가 된다. throughput 이 다시 증가한 시점부터는 커널에서 본래 사용하던 ULID 를 새로운 locator 로 mapping 하는 작업이 이루어지고 있지만, 위 graph 에서 보듯이 L3Shim protocol 에 의한 성능 저하는 거의 없음을 알 수 있다.

5. 결론

본 논문에서 저자들은 IPv6 가 본격적으로 도입될 차세대 인터넷에서의 사이트 멀티호밍을 이용한 단말의 지속적인 인터넷 연결성 제공을 염두에 두고 현재 표준화가 진행되고 있는 L3Shim 프로토콜을 분석하고 구현하였다. Linux 라는 공개된 환경의 운영체제를 목표로, L3Shim 표준 문서에 따른 프토토콜 구현을 위하여 Netfilter 와 Iptables 라는 접근 방식을 선정하였으며 리눅스 커널의 네트워크 스택 구조에 삽입되어 동작하는 L3Shim 응용을 구현하였다. 구현된 멀티호밍 응용은 IPv6 다중 인터페이스 환경에서 Iperf 을 활용하여 통신중인 인터페이스의 failure 발생에도 L3Shim 프로토콜 및 REAP 프로토콜을 통하여 지속적인 서비스를 제공할 수 있음을 확인하였다.

L3Shim 프로토콜은 커널에 들어오는 모든 패킷 뿐만 아니라 커널에서 나가는 모든 패킷을 상대로 동작해야 하기 때문에 구현 방법에 따라 성능이 크게 달라질 수 있다. 따라서 본 연구팀은 L3Shim 프로토콜에서 가장 핵심이 되는 connection 정보 관리 및 ULID/Locator 매핑 부분을 Linux kernel 내에 모듈의 형태로 삽입함으로써 성능을 획기적으로 향상시켰다. 또한 다른 개발진이 손쉽게 접근할 수 있도록 여러가지 API 를 제공하고 있다.

6. 참조

[1] S. Deering, R. Hinden, “Internet Protocol, Version 6 (IPv6) Specification,” IETF RFC 2460, December 1998.
[2] E. Nordmark, M. Bagnulo, “Shim6: Level 3 Multihoming Shim Protocol for IPv6,” draft-ietf-shim6-proto-08, May 2007.
[3] http://www.sonycsl.co.jp/project/VIP/
[4] T. Oiwa, M. Kunishi, M. Ishiyama, M. Kuhno, and F. Teraoka, “A Network Mobility Protocol based on LIN6,” VTC Fall 2003.
[5] R. Moskowitz, “Host Identity Protocol (HIP) Architecture,” IETF RFC 4423, May 2006.
[6] http://www.shim6.org/
[7] Arkko, J. and I. Beijnum, "Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming", draft-ietf-shim6-failure-detection-07 (work in progress), December 2006.
[8] B. Carpenter, K. Moore. “Connection of IPv6 Domains via IPv4 Clouds.” RFC 3056, February 2001.
[9] http://dast.nlanr.net/Projects/Iperf/