편집위원 : 임용재 (한국방송통신전파진흥원)

미들박스 서비스를 위한 전용 소프트웨어 플랫폼과 네트워크 기능 가상화

박경수
카이스트 전기 및 전자공학과

요 약

소프트웨어기반의 네트워크 미들박스 시스템은 특정 하드웨어의 종속성을 탈피하고, 다양한 여러 기능을 유연하게 제공할 수 있는 장점이 있어 최근 큰 각광을 받고 있다. 더욱이 최근 멀티코어 및 매니코어 프로세서의 발전 및 큰 대역폭을 지원하는 네트워크 카드의 등장은 저렴한 범용 컴퓨팅 하드웨어 기반에서도 높은 성능의 미들박스 서비스를 소프트웨어만으로 쉽게 제공할 수 있는 가능성을 보여주고 있다.

하지만 기존의 소프트웨어기반 네트워크 미들박스 시스템 개발에서 쓰이는 네트워킹 소프트웨어 스택은 여러 미들박스 서비스를 쉽게 만들고 유지하기에 불편한 점이 많이 있다. 첫째로, 리눅스(Linux)와 같은 범용 운영체제는 버클리 소켓(Berkeley socket)과 같이 엔드 노드를 위한 네트워킹 스택을 지원하는 반면 네트워크 미들박스 서비스 제작을 위한 전용 스택은 지원하지 않고 있다. 이로 인해 미들박스에서 많이 쓰는 플로 관리 같은 기능을 IP 패킷처리부터 새로 구현해야 하는 부담이 생긴다. 두번째로, 전용 스택의 부재는 같은 기능을 갖는 여러 미들박스 서비스가 공존할 때에도 그 구현을 공유하지 못하는 문제를 만들어 낸다. 또, 여러 미들박스 서비스가 하나의 물리적 하드웨어 위에서 수행될 경우에도 인터페이스가 일정하지 않아 같은 연산을 중복 수행해 자원 낭비를 초래한다.

본 논문에서는 차세대 소프트웨어기반 미들박스 서비스 설계 및 제작을 용이하게 하기 위한 전용 소프트웨어 스택의 필요성을 알아보고, 이런 전용 스택이 만들어 낼 수 있는 여러 가능성을 짚어본다.

Ⅰ. 서 론

네트워크 미들박스 (middlebox) 란 네트워크 안에서 IP 패킷 포워딩 및 라우팅을 제외한 여러 부가적인 네트워크 기능들을 중간에서 도와주는 시스템을 말한다. 미들박스의 종류에는 Network Address Translator (NAT), 방화벽 (Firewall), 침입탐지시스템 (IDS/IPS), 부하 분산기 (Load balancer), WAN 가속기 및 각종 성능 향상을 위한 프락시등 많은 종류가 있으며, 큰 회사나 학교 캠퍼스에서 쓰는 미들박스의 수는 이미 라우터의 개수보다 많을 만큼 미들박스 서비스는 네트워크 깊숙이 침투해 있다[12]. 미들박스 시스템은 종종 네트워크에서 흘러 다니는 트래픽의 내용물을 중간 노드에서 변경하기 때문에, 트래픽 전송이 투명하지 못한 문제를 만들 수 있고, 그런 경우 디버깅도 쉽지 않은 단점을 초래한다. 이 때문에 미들박스는 “end-to-end 원칙”을 지지하는 네트워크 원론주의자(network purists)들의 비판을 종종 받기도 하지만 미들박스가 수행하는 성능 및 보안향상의 여러 순기능들 때문에 현대 네트워크 관리에서 없어서는 안 될 필수요소로 자리 잡았다. 그리고 미들박스에 대한 네트워크 수요는 요구하는 대역폭의 증가에 따라 급속도로 커지고 있는 추세를 보이고 있다.

네트워크 미들박스 시스템은 제공하는 기능에 따라서 IP 계층 레벨, TCP/UDP등의 프로토콜이 속하는 트랜스포트 계층 레벨, 그리고 애플리케이션 계층 레벨에서 동작하는 시스템으로 나눌 수 있다. 이중 애플리케이션 레벨에서 동작하는 네트워크 미들박스들 (예: 각종 프락시류) 은 주로 범용 운영체제에서 제공하는 소켓 API등을 써서 구현되므로, 이런 애플리케이션을 위한 개발환경은 비교적 잘 갖춰져 있는 편이지만, IP 계층 또는 트랜스포트 계층 레벨에서 동작해야 하는 미들박스의 구현 환경은 여러 문제점을 지니고 있다. 고성능 서비스 처리를 위해선, 네트워크 패킷이 인입된 후 이를 운영체제안 커널에서 처리하는 것이 일반적인데, 커널 프로그래밍은 개발, 디버깅, 유지보수가 어렵다는 단점 외에도 커널 버전에 따른 종속성까지 있어 쉽게 여러 플랫폼으로 포팅하기 어려운 점이 있다. 사실 이보다 더 큰 문제점은 한번 구현된 부분을 같은 기능을 제공하는 다른 플랫폼에서 재활용하기가 어렵다는 점일 것이다. 예를 들어 침입탐지시스템의 대명사인 Snort[6]는 패킷 단위로도 패턴 매칭을 수행하지만 TCP 플로를 재조합 (Flow reassembly)하는 역할도 수행해 여러 패킷에 걸치는 공격에 대해서도 탐지할 수 있도록 설계 되어 있다. Snort의 플로 재조합은 ‘Stream5’라

는 플로 매니지먼트 모듈에 구현되어 있는데, 기능이 비슷함에 불구하고 이를 재활용하기는 어렵게 되어 있어, 비슷한 침입탐지 시스템인 Suricata[14]는 거의 같은 기능을 별도로 구현하여 사용하고 있다. 이렇게 공개 소프트웨어끼리도 공통 모듈에 대한 구현을 공유할 수 없는 문제가 있을 뿐 아니라, 서로 다른 모듈을 사용해야 하기 때문에 이미 한 시스템 운영상에서 발견되고 수정된 여러 운영상 버그들도 각각의 시스템에서 따로 해결해야 하는 낭비적인 결과를 초래하게 한다 [1]. 또 같은 네트워크 계층에서 동작하는 여러 미들박스 서비스를 하나의 물리적 하드웨어에서 통합 운영할 때에도 공통으로 사용하는 기능들을 공유하지 못하여 같은 연산을 반복 수행해야 하는 문제점도 있다. 이런 불필요한 자원낭비는 소프트웨어 기반 미들박스 시스템에선 특히 치명적으로 작용하여, 심각한 성능저하를 가져 올 가능성이 있다.

이 모든 문제점들은 소프트웨어 기반 미들박스 구현을 위한 전용 네트워킹 스택이 없다는 데에 기인한다. 본 논문에서는 소프트웨어 기반 미들박스 설계 및 구현을 위한 공통 개발 플랫폼의 역할과 방향성을 모색해 본다. 소프트웨어 기반 미들박스 제작은 최근 소프트웨어 정의 네트워킹 (Software-defined networking)과 네트워크 기능 가상화 (Network functions virtualization) 라는 네트워크 관리의 새로운 경향과 맞물려 학계/산업계 모두가 크게 주목하고 있는 화두이다. 여러 네트워크 서비스에 쓰일 수 있는 범용 네트워크 미들박스 소프트웨어 플랫폼 제작은 기존에 있던 미들박스 시스템의 개발/유지 보수에도 도움을 줄 수 있을 뿐 아니라, 복잡한 개발 사이클 때문에 기존에 잘 만들지 못했던 특화 미들박스 (customized middlebox)의 제작에도 큰 도움을 줄 수 있을 것으로 판단된다. 즉, 침입탐지/방지시스템처럼 잘 알려진 기능을 가진 미들박스 제작 외에도 현재 관리대상의 네트워크에서 플로별 TCP 패킷 재전송률 조사나 특정 프로토콜에 대해서만 중복 전송 제거를 하는 미들박스를 쉽게 만들 수 있는 기반을 제공할 수 있지 않을까 기대해 본다.

본 논문은 우선 기존 미들박스 설계 및 구현과정을 살펴보고, 개발과정의 문제점들을 조명해 본 후, 이를 극복할 수 있는 열린 소프트웨어 스택 (open software stack)에 기초한 미들박스 제작 및 이를 위한 시스템 구조에 대해서 논의한다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 기존 미들박스 시스템 구현의 문제점 및 공통 네트워킹 스택의 필요성에 알아보고, Ⅲ장에서는 미들박스 전용의 공통 네트워킹 스택의 요구사항에 대해 알아보고, Ⅳ장에서 SDN과 NFV의 관점에서 새로 개발될 스택이 어떤 역할을 할 수 있을지 알아보고, 마지막으로 Ⅴ장에서 전체 내용을 요약 정리한다.

Ⅱ. 미들박스 플랫폼의 배경 및 공통 네트워킹 스택의 필요성

본 섹션에서는 네트워크 미들박스 구현에 주로 쓰이는 네트워킹 스택 및 운영체제상 지원에 대해 알아보고, 미들박스 타입에 따른 개발상의 문제점을 알아본다.

1. 엔드 노드를 위한 네트워킹 스택

1983년 4.2BSD Unix 배포에 포함된 버클리 소켓(Berkeley Sockets)은 컴퓨터 네트워크 프로그램을 쉽게 만들 수 있게 잘 정리된 API를 지원한다. 미들박스 시스템 중 CDN 시스템 등에 널리 쓰는 캐싱 웹 프락시, TCP 레벨 터널링 프락시, 또는 리버스 SSL 프락시 같이 트랜스포트 계층 윗 레벨의 트래픽을 받아 서비스를 수행하는 시스템은 버클리 소켓을 써서 미들박스를 제작할 수 있다. 소켓 API의 장점은 실제 서비스가 구현되는 레벨 이하의 패킷 또는 TCP레벨 프로세싱을 커널에서 해 주고, 유저 레벨에서는 커널이 지원하는 서비스를 이용해 네트워킹 스택과 별개인 실제 미들박스 고유의 서비스 구현에 집중할 수 있다는 점이다. 또 버클리 소켓 API는 운영체제의 종류 및 버전에 상관없이 표준화 되어 있어, 한번 제작해 둔 미들박스 코드(또는 다른 일반 네트워크 프로그램)들을 다른 운영체제 기반의 플랫폼으로 쉽게 옮겨 컴파일만 하면 다시 쓸 수 있다는 장점도 제공한다. 즉 작성된 코드의 이동이 편해지고, 소켓 함수의 구현은 플랫폼에 관계없이 RFC등을 따라 정형화 되어 있으므로 같은 기능 구현을 공유하는 효과를 가지고 온다. 따라서 복잡한 TCP 표준 스펙을 일일이 보면서 구현할 필요 없이 간단한 API만으로 TCP 연결 위의 데이터를 다루면 된다.

버클리 소켓의 단점은 멀티 프로세서가 일반화 되지 않았던 시절에 API가 설계 되었기 때문에, 근래에 출시된 멀티 코어(multicore) 등이 제공하는 병렬성을 제대로 활용하지 못한다는 점을 들 수 있다. 예를 들어, 멀티 코어 CPU가 있는 시스템에서 멀티 쓰레드를 지원하는 웹 서버를 제작한다고 하자. 그럼 그 웹서버는 HTTP의 잘 알려진(well-known) 포트인 80번에 bind()한, 리스닝 소켓(listening socket)을 만들고, 여러 쓰레드는 이 소켓을 공유해, accept()를 부르면서 클라이언트로부터의 새로운 TCP 연결을 기다리게 될 것이다. 물론 쓰레드의 개수는 병렬성을 극대화하기 위해 시스템에서 제공하는 CPU 코어수와 같게 하고, 각각의 쓰레드는 CPU 코어에 고정시켜(affinitized), 코어에서 제공하는 CPU 캐시의 효율을 높이게 할 수 있다. 하지만 이런 구조에서의 문제점은 리스닝 소켓에 물려있는 커넥션 큐를 여러 쓰레드가 공유하기 때문에, 커넥션

큐에 대한 작업이 직렬화 (serialize)된다는 데 있다. 즉, 코어가 많이 있고, 쓰레드가 병렬로 동작할 수 있더라도, 커넥션 큐에 새로운 TCP 연결을 넣고 빼는 동작은 직렬화되어 (serialize) 커널에서 accept()를 처리하는 작업은 lock을 통해 할 수 밖에 없는 문제가 있다. 따라서 많은 CPU 코어가 있더라도, 큐 접근 오버헤드 때문에 실제 연결 성능은 하나의 코어를 쓰는 것보다 더 낮게 나올 수 있다. 이 외에도 POSIX 호환 file descriptor 생성에의 제약으로 오는 직렬화 및 작은 패킷을 빨리 처리하지 못하는 커널 구현의 비효율성도 버클리 소켓을 통한 미들박스 성능의 저하를 가져 올 수 있다.

최근 연구들은 문제점들을 극복하기 위한 몇 가지 방식을 제시하고 있다[2][3][4]. 리눅스 커널 버전 3.9.4에 들어간 SO_REUSEPORT옵션은 코어별로 로컬 커넥션 큐를 만들게 하여, 하나의 커넥션 큐를 여러 쓰레드/프로세스가 공유하지 않게 할 수 있게 하였고, MegaPipe[3]에서는 CPU코어별 file descriptor 스페이스를 추가해 file descriptor 생성에서도 병렬성을 지원할 수 있게 하였다. 또한 작은 데이터를 주고 받는 시스템 콜을 모아서 한꺼번에 처리할 수 있게 하여, 유저-커널간 모드 변환에서 오는 CPU 캐시의 비효율적 사용을 줄여, 전체 성능을 크게 증가시킬 수 있다는 점을 보여 주었다. mTCP[4]는 TCP 스택 자체를 유저 레벨에서 구현하여 여러 개의 패킷 또는 플로별 이벤트를 한꺼번에 묶어 처리하는 배치 프로세싱을 패킷 입출력부터 애플리케이션 처리에 이르는 모든 레이어에서 지원할 경우 기존 커널 기반 스택보다 3배에서25배까지의 성능향상을 할 수 있다는 점을 보여 주었다. TCP 프로세싱을 유저레벨에 구현하여 디버깅 및 유지 보수가 쉬운 스택을 선보였고, 리눅스 epoll() 시스템 콜을 지원하여 이벤트 기반 네트워크 프로그램을 쉽게 포팅할 수 있도록 하였다.

2. 플로 처리 전용 스택의 필요성

애플리케이션 레벨의 처리가 아닌, 패킷 레벨 또는 TCP/UDP 플로 처리를 겸하는 미들박스는 대개가 pcap[5]등의 별도 패킷 캡쳐 라이브러리를 이용해 패킷들을 읽어 들이고, TCP 플로 처리를 각각의 미들박스 시스템별로 새로 구현하여 쓰는 형태를 취하고 있다. Snort, Suricata 같은 침입탐지시스템, 네트워크 트래픽의 내용을 보고 애플리케이션을 탐지하는 DPI시스템, 플로 레벨 방화벽, FTP등의 페이로드를 처리해 주는 NAT등 애플리케이션 계층 아래에서 플로 처리를 해야 하는 미들박스의 종류는 다양하다.

이런 미들박스들이 플로 처리 모듈을 자신의 요구에 맞게 각각 따로 제작하는 현재 상황은 여러모로 낭비적인 요소를 만들게 된다. 첫째, 똑같은 기능을 구현하더라도 서로 다른 시스템에서 호환이 되지 않아 하나의 시스템에서 만든 모듈을 다른 시스템에서 재활용할 수 없는 문제가 있다. 앞서 서두에서 말한 바와 같이 같은 IDS/IPS의 기능을 제공하는 공개 소프트웨어인 Snort와 Suricata도 플로 처리 모듈을 각각 따라 구현하고 있다. 이는 두 시스템이 공통으로 쓸 수 있는 일반적인 플로 처리 모듈이 존재하지 않기에 필요에 따라 새로 구현한 경우이다. 하지만 이런 커스텀 모듈들은 다른 시스템에서의 호환성을 고려하지 않고 구현되기 때문에, Snort에서 쓰는 플로 처리 모듈을 활용해 플로 레벨 방화벽 같은 다른 미들박스 시스템을 만들기가 굉장히 어려운 문제점을 만들어 낸다. 둘째, 이런 개별 모듈을 따로 구현하게 되면 공통적인 구현 또는 취약점 이슈도 따로 처리해야 하는 문제가 생긴다. Suricata는 Snort의 플로 처리 모듈을 재활용 하지 않기 때문에, Snort에서 해결했던 여러 구현 및 운영상의 버그들을 따로 구현해야 한다. 즉, Snort에 대해선 통하지 않는 공격들이 Suricata에는 통하는 경우가 생기는 것이다[1]. 마지막으로, 개발자의 입장에서는 미들박스 서비스의 본질적이고 핵심적인 부분에 대한 설계 및 구현뿐 아니라, 이미 잘 표준화 되어 있는 TCP 프로토콜 자체 구현에 대해서도 세심하게 신경 써야 하는 큰 부담이 생긴다. TCP 프로토콜은 비교적 간단하게 시작하였지만 최초의 TCP표준문서인 RFC 793이후에 여러 RFC가 추가 정의 되어 있어, 표준을 따르는 상용 수준으로의 개발은 상당한 시간이 소요된다. 현재는 각 플로 기반 미들박스들이 따로 구현하는 형태를 택하기 때문에 각 미들박스 개발자들이 표준화 이슈들을 다 공부해야 하는 문제가 있다.

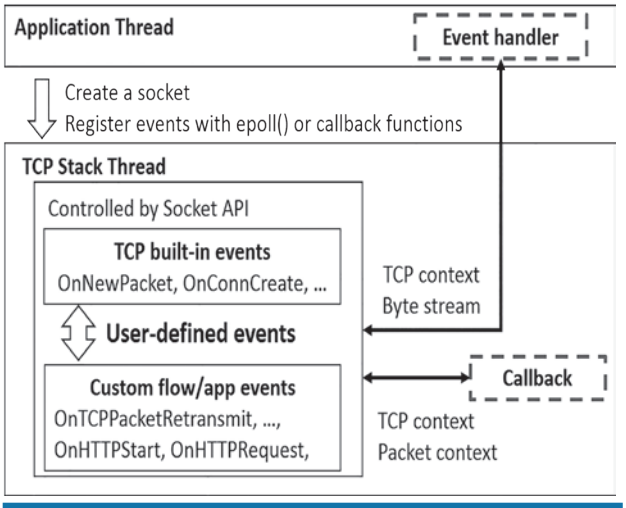

그림 1. 유저 정의 이벤트

Ⅲ. 미들박스를 제작을 위한 전용 소프트웨어 플랫폼

본 논문의 기본 주장은 엔드 노드에서 활용하는 버클리 소켓과 같은 표준 인터페이스를, 패킷 또는 플로 처리를 주로 하는 네트워크 미들박스에도 적용하자는 데 있다. 공통 네트워킹 및 자원관리 인터페이스와 공유 가능한 모듈들의 조합을 통해 복잡한 미들박스도 쉽게 만들 수 있도록 하고, 이미 알려진 최적화 기법을 공통으로 적용해 최소한의 노력으로 고성능 미들박스 시스템을 만들 수 있는 플랫폼을 지원하는 것이 본고의 핵심 주장이다. 이번 섹션에서는 미들박스 플랫폼에서 어떤 기능을 지원해야 하는지 알아 본다.

1. 미들박스 전용 네트워킹 스택

미들박스를 위한 전용 네트워킹 스택은 미들박스가 필요로 하는 패킷 또는 플로 레벨의 네트워킹 연산을 기존의 버클리 소켓처럼 잘 정립된 인터페이스와 함수로 수행할 수 있는 소프트웨어 스택을 의미한다. pcap라이브러리와 같은 간단한 인터페이스를 지녔지만 패킷 처리에만 쓸 수 있는 스택이 아니라, 플로 레벨의 여러 기능을 지정한 설정에 따라 자동으로 처리해 주기도 하고, 때로는 플로 레벨 처리 중간에 쉽게 미들박스 특화 처리를 끼워 넣을 수 있는 유연한 스택을 얘기하는 것이다. 여기에 더하여 멀티 코어, NUMA 아키텍쳐 등의 최신 컴퓨팅 하드웨어의 특성을 잘 활용해 미들박스 개발자가 네트워킹 스택에 대한 특별한 최적화를 하지 않더라도 높은 성능을 낼 수 있어야 한다.

이런 미들박스 전용 네트워킹 스택의 요구사항으로 생각할 수 있는 점들을 정리해 보면, 첫째, 패킷 처리 또는 플로 처리의 여러 과정 중 어디에나 미들박스 프로세싱을 넣었다 뺄 수 있는 유연성, 둘째, 전용 스택과 그 스택을 이용하는 애플리케이션과의 효율적인 통신을 담당하는 잘 정립된 인터페이스, 셋째 여러 미들박스 서비스가 공존할 때 공통적인 모듈 또는 이벤트를 공유할 수 있도록 하여 중복 연산을 막을 수 있어야 한다. 이에 대한 구체적인 내용을 다음 몇 섹션에 걸쳐 설명하고자 한다.

2. 유저 정의 이벤트 프로세싱

플로 레벨 미들박스 시스템은 다양한 플로 레벨 이벤트에 맞춰서 프로세싱을 달리 해야 하는 요구사항이 있다. 예를 들어, IDS의 경우 TCP 플로의 상태 (state)를 매 패킷 처리마다 업데이트해야 하고, 새로운 연결이 이루어지거나 연결이 끊어지는 이벤트마다 다른 프로세싱을 할 수 있어야 한다. 또 이동통신망 데이터를 공짜로 쓰게 하는 TCP 재전송을 통한 과금 시스템 공격을 방어하는 Abacus[7]와 같은 미들박스 시스템은 재전송된 TCP 패킷의 내용이 기존의 내용과 같은 지를 확인할 수 있어야 하기 때문에 매 TCP 재전송 이벤트마다 미들박스 특화 프로세싱을 수행할 수 있어야 한다. 또, Monbot[15]과 같이 네트워크 트래픽 간의 내용 중복도를 실시간 계산하는 시스템은 TCP 플로 재조합 후에 일정 크기의 연속적인 세그먼트가 만들어지면 SHA1같은 해시함수를 돌릴 수 있도록 해야 한다.

이런 다양한 이벤트에 맞춰 미들박스 프로세싱을 지원하기 위해 본고에서는 미들박스 를 위한 유저 정의 이벤트라는 개념을 제안한다. 기존 select() 또는 epoll()과 같은 함수를 사용하는 이벤트 기반 소켓 프로그래밍에서의 이벤트는 read, write, exception[1]등과 같이 관심이 있는file descriptor와 연관되어 있는 TCP buffer의 상태 변화 (readable or writable) 이벤트 (혹은 각종 에러 상태)에만 초점을 맞춘다. 애플리케이션 계층에서만 동작하면 되는 미들박스 또는 엔드 노드를 위한 네트워크 서버/클라이언트의 경우는 이 정도의 이벤트만으로도 충분히 블락킹이 없는 (non-blocking) 효율적인 프로그래밍을 할 수 있지만, 플로 레벨 미들박스의 경우에는 플로 프로세싱의 다양한 상태변화에 맞게 알맞은 이벤트가 생성되어야 한다. 유저 정의 이벤트는 이와 같은 요구에 부응하기 위한 개념으로, 사용자가 이벤트 생성에 합당한 조건을 제시하면, 그 조건을 만족하는 상태가 되면 이벤트를 생성해 주는 식으로 동작한다.

유저 정의 이벤트는 TCP 연결이 특정 상태로 변화할 때 발생되는 빌트인 이벤트 (built-in event)와 사용자가 이벤트의 조건을 함수로 표현하는 특화 이벤트 (custom event)로 나뉜다. 특화 이벤트는 사용자가 원하는 조건의 성립을 함수 수행으로 판별할 수 있도록 하여 이벤트 정의의 유연성을 극대화한다. 이벤트 처리도 두 가지 경우로 나눌 수 있다. 첫째, 이벤트 처리 함수를 인라인 (inline)으로 등록하면 그 이벤트 발생시 지연 없이 핸들러를 수행하여 이벤트 발생과 처리와의 시간을 최소화 할 수 있다. 둘째로, epoll()과 같은 이벤트 함수를 통해 애플리케이션쪽으로 전달하게 하면 여러 이벤트를 한꺼번에 모아 처리하여, 일괄 처리를 통한 효율적인 프로세싱을 꾀할 수 있다. 이처럼 유저 정의 이벤트는 플로 프로세싱 모듈을 일반화하여 다양한 미들박스의 요구사항을 만족시켜 줄 수 있는 메커니즘을 제공한다. 유저 정의 이벤트의 궁극적인 목표는 플로 프로세싱과 미들박스 처리를 분리하여 개발자로 하여금 플로 프로세싱 보다는 미들박스 서비스 고유의 연산들에 초점을 맞추도록 함에 있다. 즉 공유할 수 있는 모듈들을 최대한으로 늘여 여러 미들박스 지원을 하는 데 목적이 있다.

1 Exception 이벤트는 TCP의 out-of-band 데이터 전송 이벤트등에 사용될 수 있지만 실제 사용빈도는 낮다.

3. 전용 미들박스 스택 구조

미들박스를 위한 전용 네트워킹 스택의 구현은 멀티코어 및 NUMA와 같은 최신 하드웨어 기법에 최적화하여, 최대한의 프로세싱 병렬성을 이끌어 내고, 메모리 대역폭의 손실이 없는 방향으로 진행되어야 한다. 또한 될 수 있으면 커널 프로그래밍을 지양해, 인텔의 Data Plane Development Kit (DPDK)[8], PacketShader IOEngine[9], netmap[10], PF_RING[11] 과 같은 멀티코어를 잘 활용하여 수십 Gbps의 고성능 패킷 입출력이 가능한 라이브러리를 사용해, 유저 레벨에서 구현하는 것이 개발, 디버깅, 유지/보수를 위해 좋을 것이다. 전용 미들박스 네트워킹 스택은 mTCP[4]의 경우처럼 각 미들 박스 애플리케이션이 각기 사용하는 유저 레벨 쓰레드로 구현될 수 있다. 가장 간단하게는 미들박스 애플리케이션 별로 따로 네트워킹 스택을 두고, 이 스택은 유저 레벨 쓰레드로 구현해, 네트워킹 스택의 동작과 애플리케이션 로직을 분리하게 하는 것이다. 물론 이런 경우 애플리케이션과 네트워킹 스택의 통신이 쓰레드간 통신(inter-thread communication)이 되기 때문에 컨텍스트 스위칭 오버헤드가 통신량에 비해 상대적으로 커지는 문제가 생긴다. 이는 일괄처리(batch processing)를 적절히 함으로써 여러 이벤트를 하나로 묶어 처리하게 하면 오버헤드 감소를 꾀할 수 있다. 실제 mTCP에서는 한번의 쓰레드 통신에, 2000여 개의 이벤트를 묶어 보내게 처리함으로써 쓰레드 컨텍스트 스위칭 오버헤드를 크게 낮춘바 있다. 비슷한 테크닉을 적용하면 미들박스 네트워킹 스택을 애플리케이션에서 분리함과 동시에 고성능을 기대할 수 있을 것이다.

4. 중복 연산 제거를 통한 모듈

최근 미들박스 서비스는 여러 개의 서비스를 하나의 물리적 머신에서 제공하는 통합 추세를 보이고 있다. 예를 들어, 차세대 방화벽 (Next Generation Firewall, NGFW)은 기존의 방화벽, IDS/IPS, VPN같은 여러 미들박스 기능을 하나의 물리적 머신에서 통합 제공하고 있다. 이처럼 여러 미들박스 서비스를 하나의 머신에서 제공하는 이유는 첫째, 서비스별로 머신을 두게 되면 관리해야 할 장비수가 늘고, 장비별로 설정하는 방식이 달라 결과적으로 관리비용이 커질 수 있고, 둘째, 미들박스 서비스를 수행하는 성능의 관점에서도 하나의 패킷이 들어와 n개의 서로 다른 장비를 거치려면 여러 번 패킷이 포워딩 되어야 하고, 이 때문에 처리 지연이 생길 수 있기 때문이다. 또 서비스별로 플로 관리와 같은 연산을 반복적으로 수행할 수 있어, 중복 연산으로 오는 자원 낭비도 심각해 질 수 있다.

CoMb[12]는 여러 미들박스 서비스를 하나의 물리적 머신에서 구현해 같은 기능을 갖는 모듈들을 여러 서비스에서 공유했을 때, 불필요한 중복 연산 제거를 통해 전체 시스템 성능이 올라 갈 수 있음을 보였다. 또 xOMB[13]는 패킷 프로세싱을 위한 모듈공유에서 한발 더 나가 플로별로 프로세싱 파이프라인을 구성할 수 있도록 하였다. 플로별로 여러 서비스를 유연하게 적용할 수 있게, 비동기적 모듈 프로세싱 (asynchronous module processing)과 개별 메시지 메타데이터(per-message metadata)를 지원하여 서비스간의 데이터 공유를 쉽게 할 수 있는 플랫폼을 제시했다.

서비스별 중복 연산을 줄이기 위해서는 여러 서비스를 하나의 네트워크 스택에서 관리해, 여러 서비스들이 메타데이터 (예: TCP 컨텍스트 정보, 수행 결과)를 효율적으로 공유할 수 있는 구조를 제시해야 한다. 즉, 각 서비스별로 관심이 있는 플로를 제시할 수 있는 시스템 함수를 지원하고, 이 함수를 통해 모니터링 되는 공통 플로의 TCP 컨텍스트는 각 패킷 이벤트마다 한 번만 업데이트하되 여러 서비스에서 공통으로 참조할 수 있게 구현하면 된다. 또 서비스 적용의 선*후 관계를 서비스 설정에서 명확하게 구별하여 TCP 컨텍스트 또는 이에 수반되는 이벤트의 처리 순서를 명확하게 정할 필요가 있다. 이렇게 되지 않을 경우 여러 서비스가 동시에 TCP 컨텍스트에 접근하고, 내용을 비결정적으로 (non-deterministically) 수정하게 되면, 동시 접근 때문에 디버깅이 아주 어렵게 된다.

하나의 머신에서 여러 서비스를 운영하다 보면 머신의 물리적 용량 제한 때문에 성능이 떨어지는 문제가 생길 수 있다. 따라서 일정 수준의 서비스 통합 후에는 물리적 머신의 수를 늘여서 원하는 용량을 맞출 수 있도록 수평적 용량 증대(horizontal scaling)도 지원해야 한다. 미들박스 서비스 체인이 물리적 머신을 뛰어넘어 다른 장비로 확장되는 경우에도 중복 연산 제거를 위해 이미 계산된 플로 정보 또는 서비스 수행 중간 결과는 다른 물리적 머신에서 돌아가는 서비스로 전달 될 수 있는 통신 프로토콜도 지원해야 한다.

VI. 미들박스 전용 네트워킹 스택과 NFV

본고에서 논의했던 미들박스 전용 네트워킹 스택은 SDN과 NFV지원 장비 개발 측면에서 중요한 의미를 지닌다. SDN은 네트워크 플로레벨 라우팅을 프로그래밍 인터페이스로 조절하는 기능을 가져 일부 대형 데이터센터에서 네트워크 가상화 구현에 필수적인 요소로 잡았고, NFV는 미들박스 기능을 포함한 네트워크의 여러 기능을 기능의 구현 장소와 처리 용량에 구애받지 않게 가상화 하여 최근 큰 관심을 받고 있다. 본고에서 논

의한 미들박스 전용 네트워킹 스택, 그리고 더 나아가 미들박스 서비스를 위한 전용 운영체제의 지원은 SDN과 NFV를 하나로 묶어 네트워크 운영 전반을 유연한 소프트웨어 스택으로 제어할 수 있는 발판을 마련할 것으로 보인다. 또한 이런 시스템 지원이 오픈 소스로 활용된다면 앞으로의 네트워크 기능 구현 및 관리를 통일시킬 수 있는 중요한 계기를 마련하리라 본다.

1. 미들박스 지원 소프트웨어 스위치

미들박스 운영체제가 개발되면 SDN을 지원하는 소프트웨어 기반 고속 스위치에 NFV를 탑재한 미들박스 서비스를 추가 지원할 수 있을 것으로 예상할 수 있다. 미들박스 운영체제의 호환성에 힘입어 여러 미들박스 서비스를 포팅하여 하나의 박스에서 자원을 공유하면서 동작하는 걸 생각해 볼 수 있다. 소프트웨어기반 라우터는 이미 하나의 서버 박스에서 40Gbps정도의 성능을 보였고[9], 소프트웨어기반 전용 IDS[16]및 SSL 리버스 프락시[17]도 하나의 머신에서 수십 Gbps의 성능을 보이고 있다. 네트워크 스위칭 기능과 미들박스를 한데 모은다면 불필요한 포워딩으로 인한 자원낭비 및 지연을 최소한으로 줄이고, 여러 관련 기능을 하나의 박스에서 순차적으로 수행함으로써 (e.g., SSL 복호화 후 IDS 수행) 불필요한 중복 연산을 줄여서 효율을 높일 수 있을 것이라 판단된다.

2. SDN과 NFV를 동시에 지원하는 통합 컨트롤러

SDN은 SDN을 지원하는 네트워크 장비들을 제어하는 SDN 컨트롤러에 의해 네트워크 플로들을 중앙집중적으로 관리할 수 있다. 미들박스 운영체제가 본격적으로 자리를 잡게 되면 SDN 스위치에 미들박스 기능을 탑재할 수 있고, 이런 기능들도 SDN + NFV 통합 컨트롤러에서 제어할 수 있을 것으로 판단된다. 하나의 컨트롤러에서 플로 레벨 라우팅 뿐만 아니라 플로별 미들박스 체인을 달아주거나 없애주는 일까지 동시에 할 수 있어, 네트워크 플로를 세밀하게 처리할 수 있을 것으로 판단된다. 또한 통합 컨트롤러 자체도 미들박스 운영체제 스택으로 구현하여 하나의 코드 베이스로 전체를 관리할 수 있는 시스템을 구축할 수 있는 장점도 생기지 않을까 한다.

V. 결 론

소프트웨어 기반 미들박스는 소비자 하드웨어의 대량 생산/대량 소비에서 오는 가격 경쟁력과 소프트웨어를 통한 구현에서 오는 높은 기능 유연성 때문에 미래 인터넷 트래픽 관리에 필수적인 요소로 자리 잡고 있다. 하지만 현재의 미들박스 시스템 설계/구현은 각각 수직적으로 분리되어 있어 같은 기능도 서로 다르게 구현되고, 하나의 시스템에서 해결된 문제도 다른 시스템으로 쉽게 전파하지 못하는 문제점이 있다. 이에 본 논문은 소프트웨어 기반 미들박스가 공통으로 쓸 수 있는 전용 미들박스 네트워킹 스택을 만들 것을 주장하고, 이에 대한 요구 사항을 정리해 보았다. 미들박스 전용 네트워킹 스택은 버클리 소켓과 같이 잘 정리된 API를 제공하며, 이벤트 기반 프로그래밍을 효율적으로 지원하여야 한다. 다양한 미들박스 서비스의 요구를 반영하기 위해선 프로그래밍을 위한 이벤트를 개발자가 쉽게 정의할 수 있어야 한다. 이에 본고는 유저 정의 이벤트를 제시하여 사용자로 하여금 원하는 이벤트의 조건을 직접 정의하게끔 하였으며, 이를 동기적 또는 비동기적으로 처리할 수 있는 프레임워크를 제시하였다. 또 전용 미들박스 스택과 애플리케이션의 통신을 효율적으로 진행하는 방법과 플로 처리시 여러 서비스에서 공통으로 수행 해야 하는 연산을 한번만 수행하게 하여 중복 연산으로 오는 자원 낭비 및 성능 저하를 막을 것을 주장하였다.

미래 인터넷은 여러 네트워크의 기능(Network functions)들을 소프트웨어 정의 네트워킹 (SDN)로 제어하는 형태로 트래픽 관리를 할 것으로 예상된다. 이런 경향에서 네트워크 기능을 구현을 담당하는 소프트웨어 기반 미들박스의 역할은 날로 증대될 것이다.

참 고 문 헌

[1] J. Novak., "Suricata TCP Evasions," Personal Blog, 2010 (http://www.packetstan.com/2010/09/suricata-tcp-evasions.html).

[2] A. Pesterev, J. Strauss, N. Zeldovich, and R. T. Morris. "Improving network connection locality on multi-core systems," In Proceedings of the ACM European conference on Computer Systems (EuroSys), 2012.

[3] S. Han, S. Marshall, B.-G. Chun, and S. Ratnasamy. "MegaPipe: a new programming interface for scalable network I/O," In Proceedings of the USENIX conference on Operating Systems Design and Implementation (OSDI), 2012.

[4] E. Jung, S. Woo, M. Jamshed, H. Jung, S. Ihm, D. Han, K. Park. "mTCP: A Highly Scalable User-level

TCP Stack for Multicore Systems," In Proceedings of the USENIX Symposium on Networked System Design and Implementation (NSDI), 2014.
[5] TCPDUMP. http://www.tcpdump.org.
[6] M. Roesch. "Snort – Lightweight Intrusion Detection for Networks," In Proceedings of the USENIX Systems Administration Conference (LISA), 1999.
[7] Y. Go, J. Won, D. Kune, E. Jung, Y. Kim, K. Park. "Gaining Control of Cellular Traffic Accounting by Spurious TCP Retransmission," In Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS), 2014.
[8] Intel DPDK: Data Plane Development Kit. http://dpdk.org/.
[9] S. Han, K. Jang, K. Park, and S. Moon. "PacketShader: a GPU-accelerated Software Router," In Proceedings of ACM SIGCOMM, 2010.
[10] L. Rizzo. "netmap: a novel framework for fast packet I/O,"In Proceedings of the USENIX conference on Annual Technical Conference (ATC), 2012.
[11] L. Deri. "Improving Passive Packet Capture: Beyond Device Polling," In Proceedings of the International System Administration and Network Engineering Conference (SANE), 2004.
[12] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi. "Design and Implementation of a Consolidated Middlebox Architecture,"In Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2012.
[13] J. W. Anderson, R. Braud, R. Kapoor, G. Porter, and A. Vahdat. "xOMB: Extensible Open Middleboxes with Commodity Servers,"In Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems (ANCS), 2012.
[14] Suricata Intrusion Detection System. http://www.openinfosecfoundation.org/index.php/download-suricata.
[15] S. Woo, E. Jeong, S. Park, J. Lee, S. Ihm, and K.Park. "Comparison of caching strategies in modern cellular backhaul networks," In Proceedings of the Annual International Conference on Mobile Systems, Applications, and Services (MobiSys), 2013.
[16] M. Jamshed, J. Lee, S. Moon, I. Yun, D. Kim, S. Lee, Y. Yi, and K. Park. "Kargus: A Highly-scalable Software-based Intrusion Detection System,"In Proceedings of the 19th ACM Conference on Computer and Communications Security (CCS '12), 2012.
[17] K. Jang, S. Han, S. Han, S. Moon, and K. Park. "SSLShader: Cheap SSL Acceleration with Commodity Processors," In Proceedings of the 8th USENIX conference on Networked Systems Design and Implementation (NSDI), 2011.

약 력

박 경 수

1997년 서울대학교 계산통계학과 학사
2004년 프린스턴대학교 전산학과 석사
2007년 프린스턴대학교 전산학과 박사
2007년~2008년 프린스턴 대학교 전산학과 Associate Research Scholar
2009년~2009년 피츠버그 대학교 전산학과 조교수
2010년~현재 카이스트 전기 및 전자공학과 부교수
관심분야: 고성능 네트워크 컴퓨팅 시스템 설계 및 구현, 컴퓨터 네트워킹 및 운영체제