DEVIEW 2020

K8s 에서의 eBPF/XDP 기반 고성능 & 고가용성 NAT 시스템

송인주 NAVER

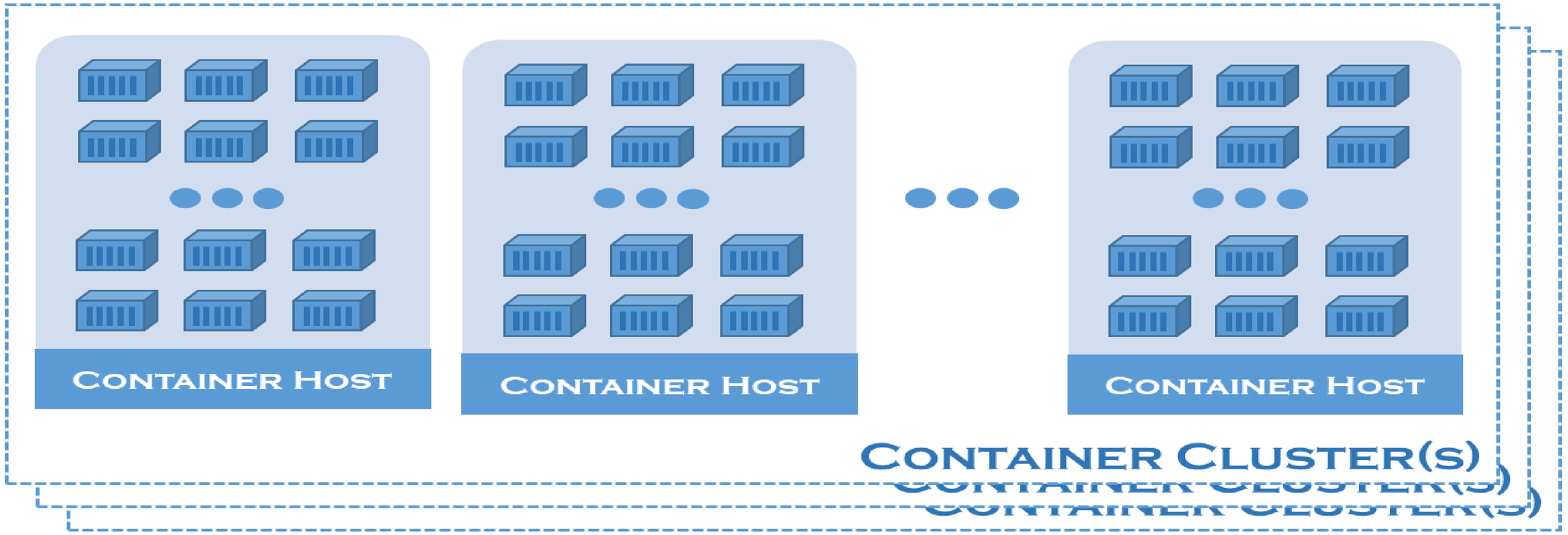
CONTENTS

1. 컨테이너 클러스터에서의 IP ACL 문제

1.1 컨테이너의 IP 할당/해제

컨테이너 클러스터

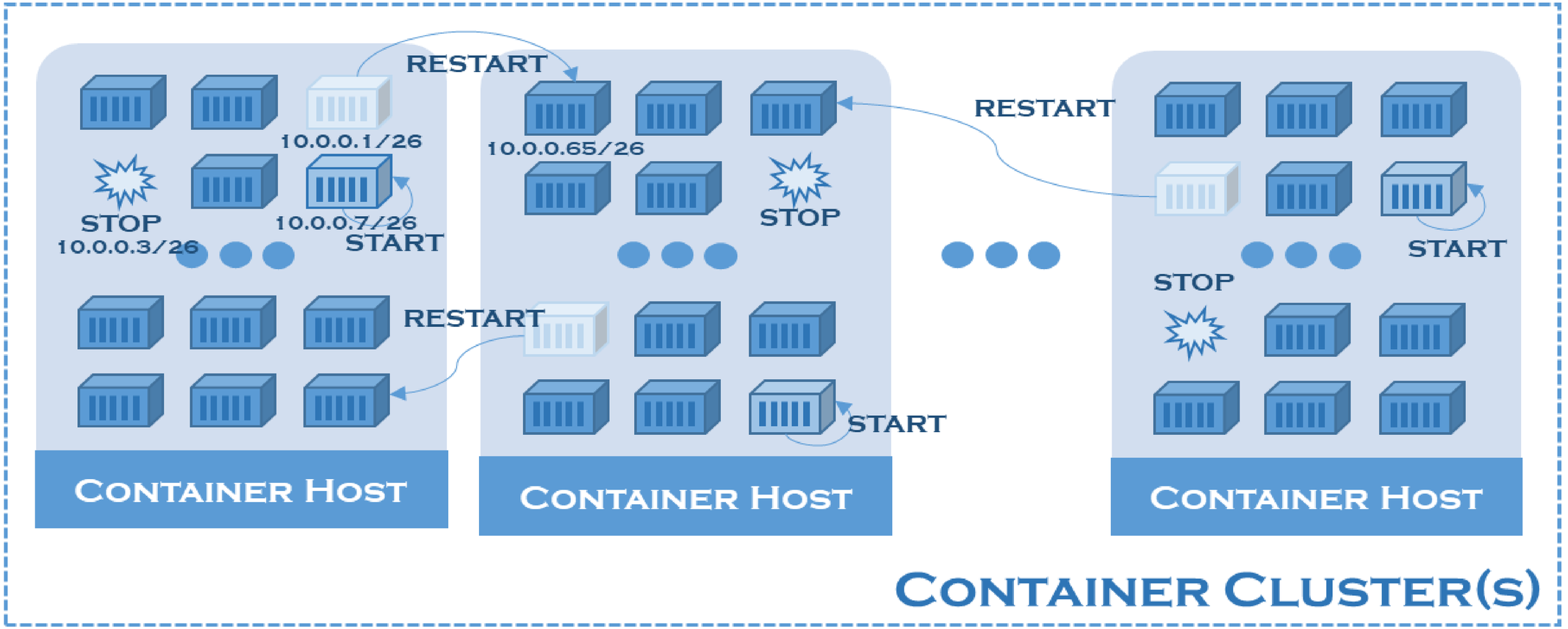
- 약 500 대 이상의 노드들로 구성된 클러스터에서 수천 개 이상의 컨테이너를 운영



1.1 컨테이너의 IP 할당/해제

유동적인 컨테이너의 IP

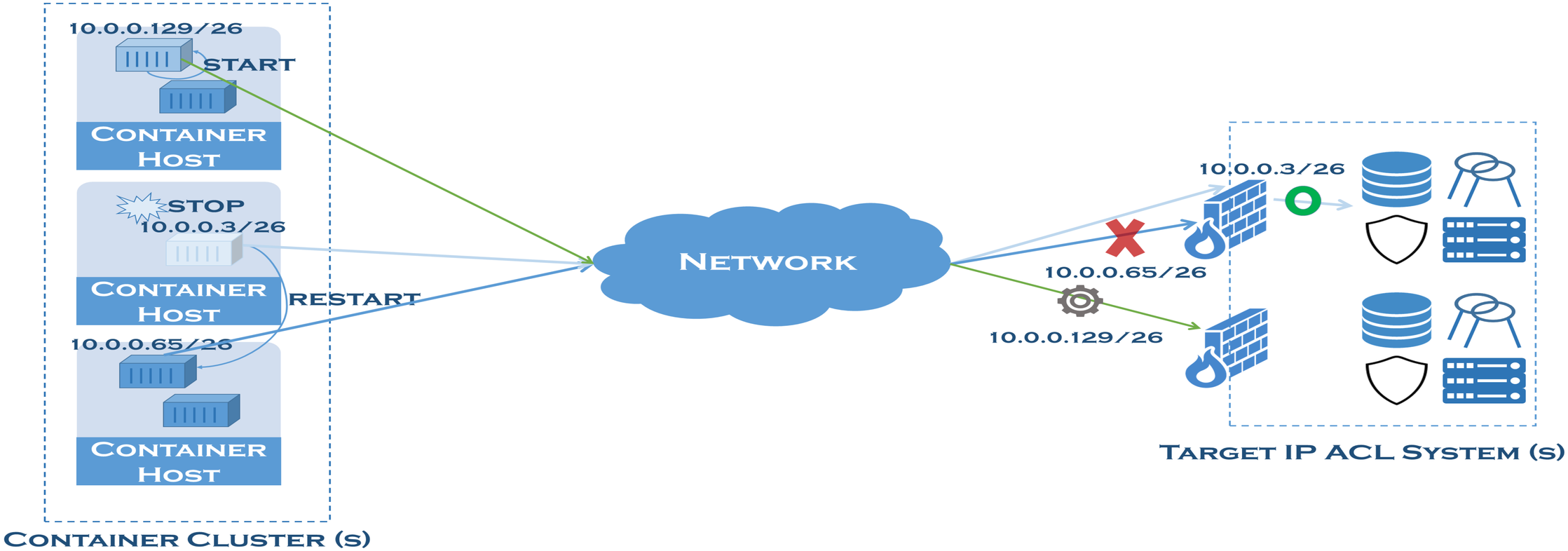
- 대규모 클러스터에서 IP가 고정될 수 없는 컨테이너의 특성



1.2 컨테이너 환경에서 발생하는 IP ACL 문제

컨테이너의 IP 변경

- IP 주소로 ACL 을 관리하는 시스템에 접근하는 경우, 컨테이너 IP 변경으로 인해 접근이 불가능

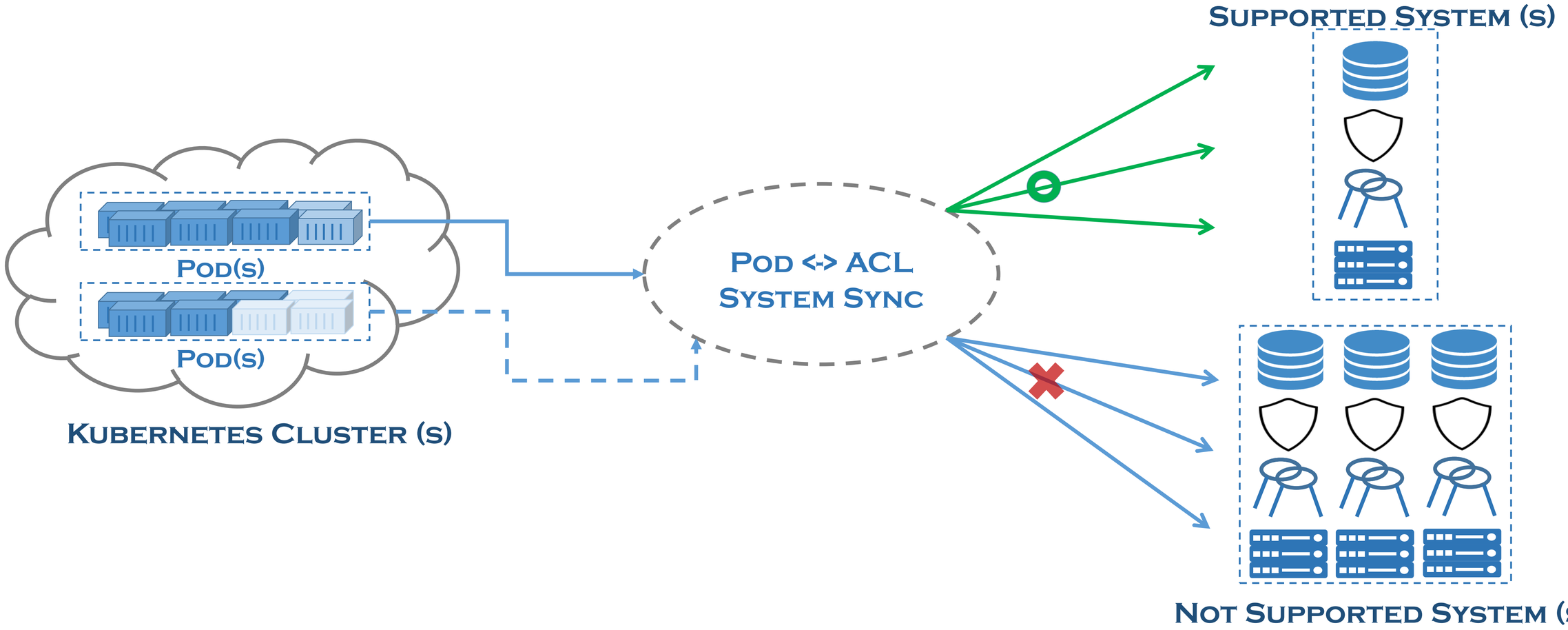


1.2 컨테이너 환경에서 발생하는 IP ACL 문제

DEVIEW 2020

Pod <-> ACL System 간 정보 동기화

- 컨테이너(Pod) IP 변경 정보와 외부 시스템을 연동시키는 방식



1.2 컨테이너 환경에서 발생하는 IP ACL 문제

DEVIEW 2020

Pod <-> ACL System 간 정보 동기화 방식의 한계

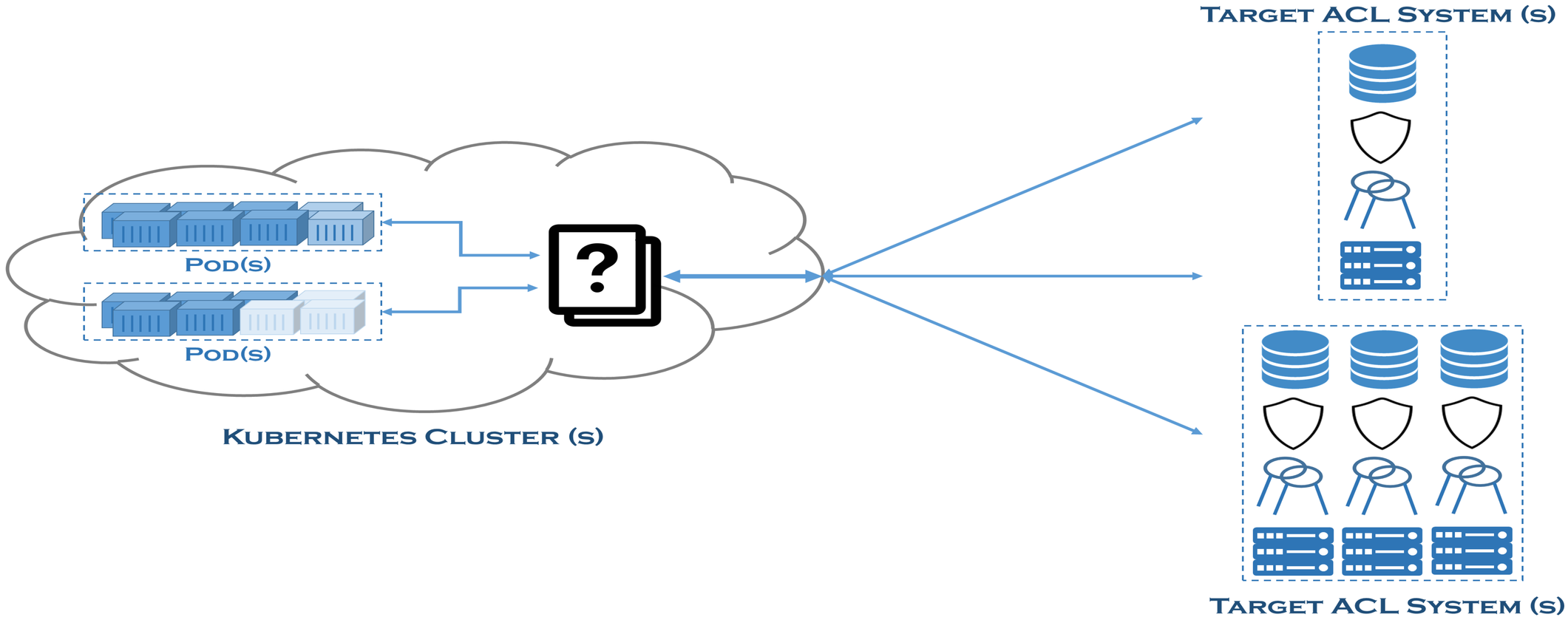
- 사내에 해당 시스템이 있지만, 다수의 Pod * 다수의 타겟 간에 동기화 부담
- 다수의 Pod 으로 구성된 네이버 서비스에 일부 Pod 이라도 ACL 이 전파되지 않으면 장애로 이어질 수 있음.
- 대부분의 네이버 서비스는 일부 Connection/Request 의 유실도 모니터링하면서 발생 시 바로 문의가 오는 상황

1.2 컨테이너 환경에서 발생하는 IP ACL 문제

DEVIEW 2020

IP ACL 해결 솔루션

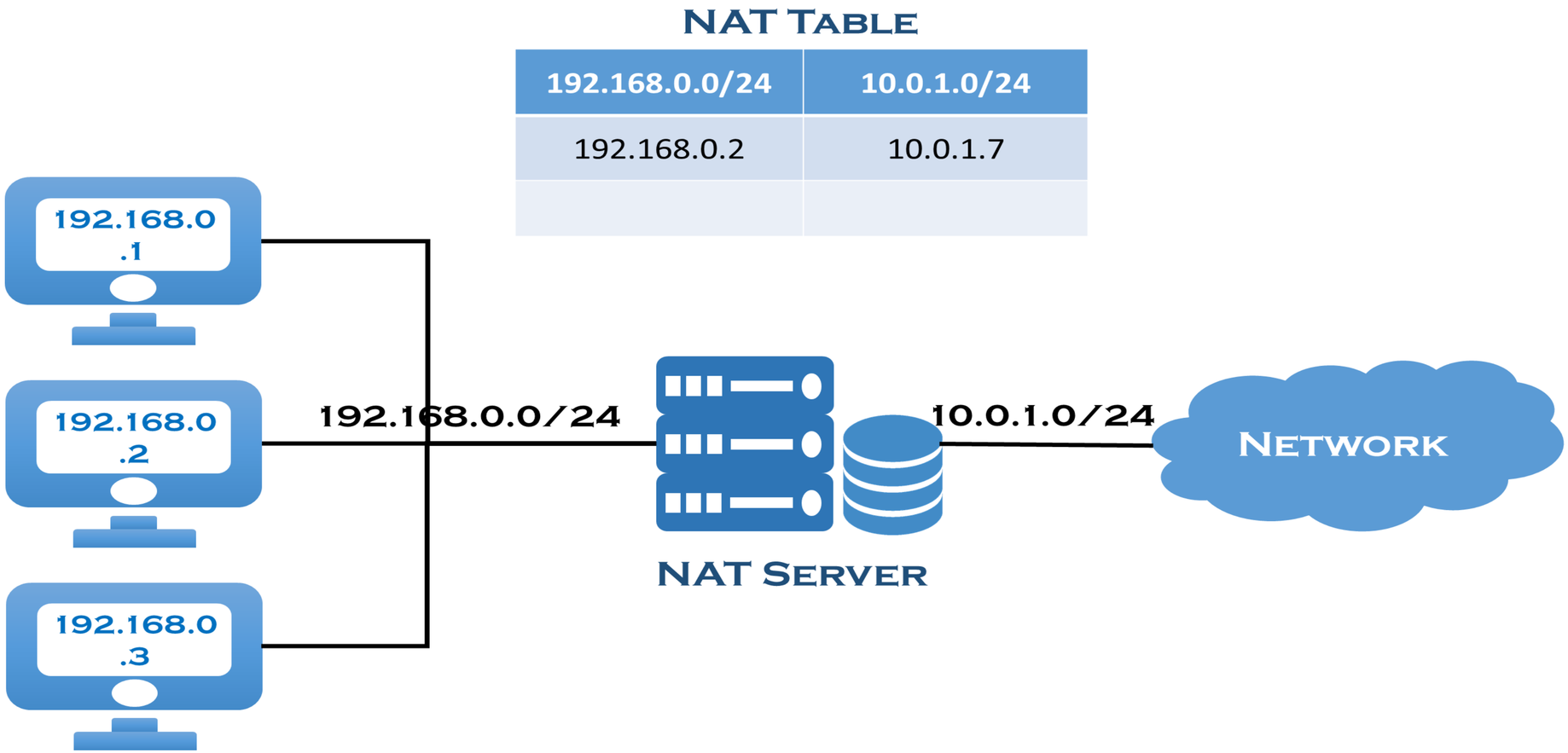
- 컨테이너의 IP 변경, 타겟 시스템과 관계없이 ACL 에 접근이 가능한 시스템 구축 필요



1.3 NAT 를 통한 문제 해결 시도

Network Address Translation

- 네트워크 주소 (IP, Port) 를 변환하는 메커니즘

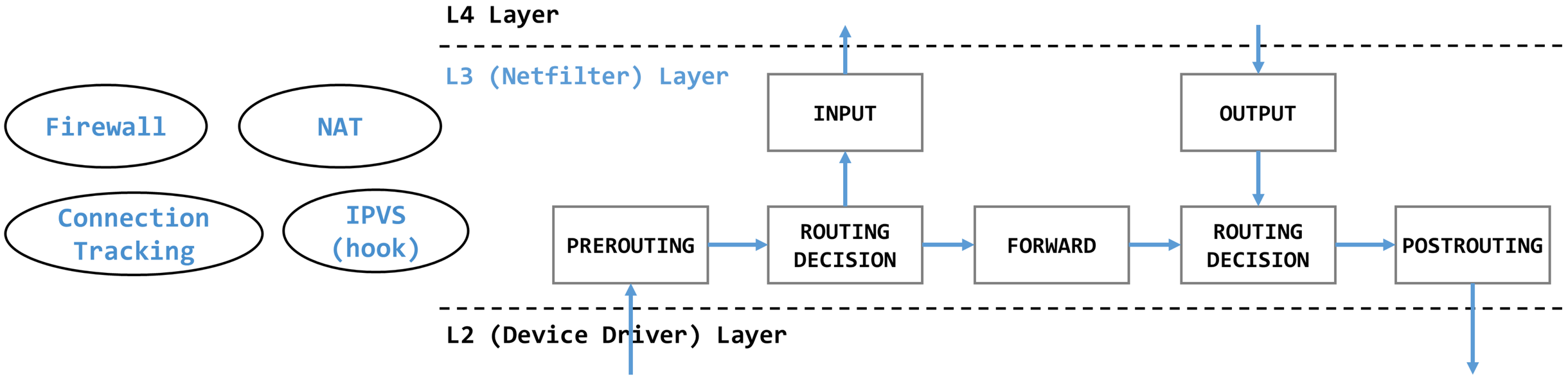


1.3 NAT 를 통한 문제 해결 시도

DEVIEW 2020

Linux Netfilter(iptables)

- Linux 에서 네트워크 패킷을 control 할 수 있는 시스템
- Kernel 의 Networking Stack 에서 동작
- 다양한 Networking 기능 구현 가능



1.3 NAT 를 통한 문제 해결 시도

DEVIEW 2020

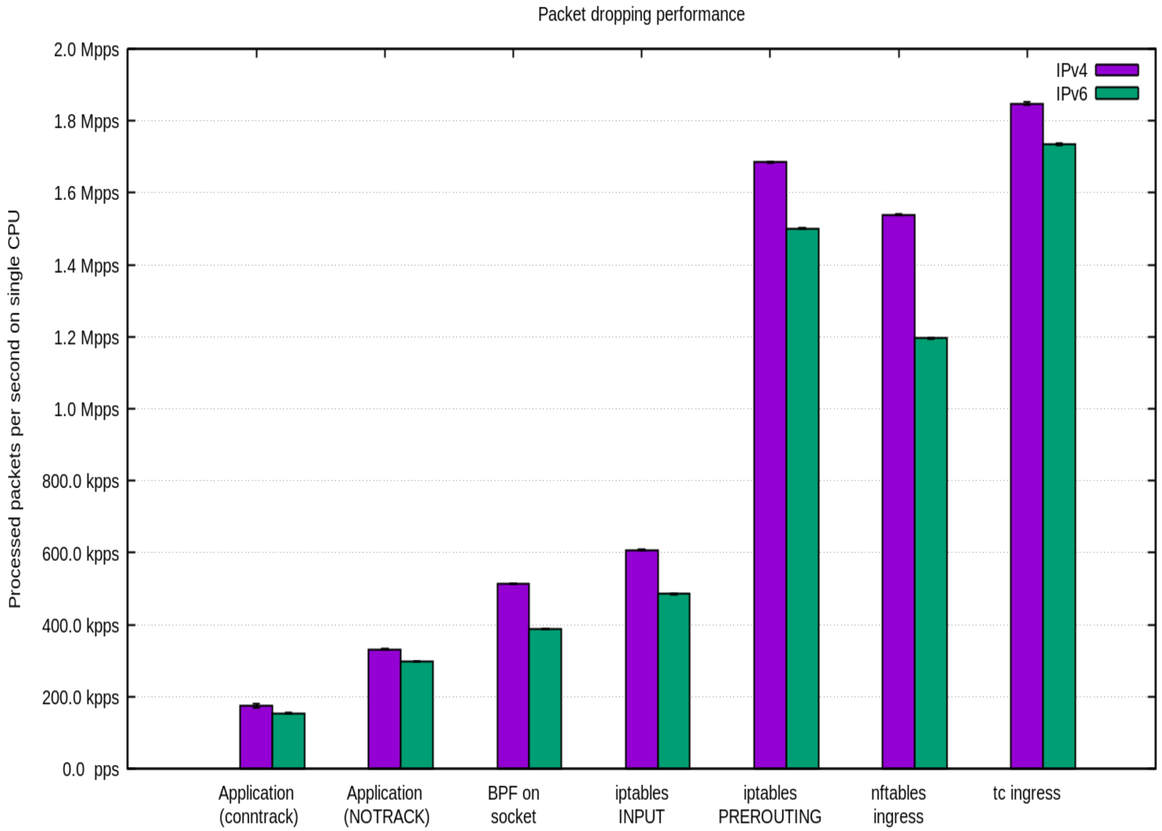
Linux Netfilter(iptables)

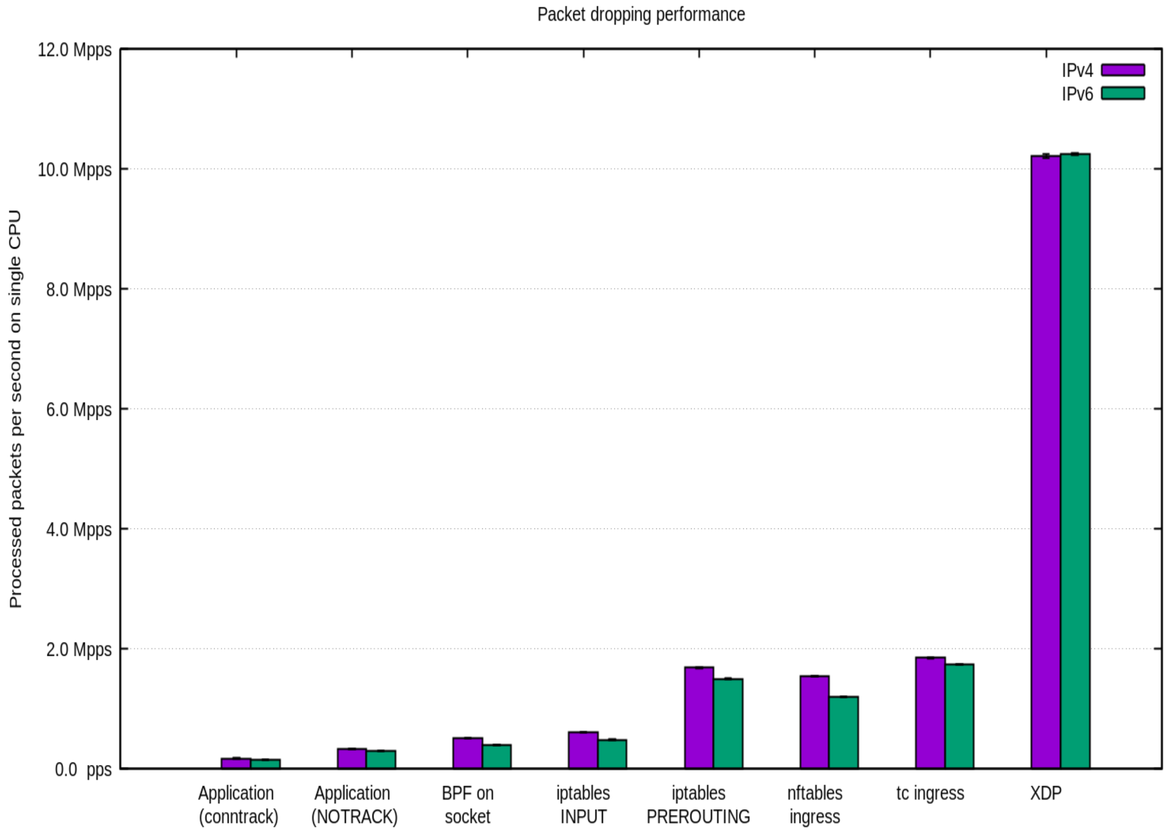
- 이미 linux kernel 의 iptables/ipvs 기술로 SWLB 를 구축한 경험
- 하지만 실제 서비스에서 운영 시 connection table 관리에 어려움이 있었음
- 일부 서비스에서는 connection 과정의 지연에 매우 민감
- NAT 를 iptables 로 제공할 시, 실 서비스를 위한 대용량 트래픽에 대해서 성능이 충분할까?

1.3 NAT 를 통한 문제 해결 시도

Linux iptables vs BPF/XDP

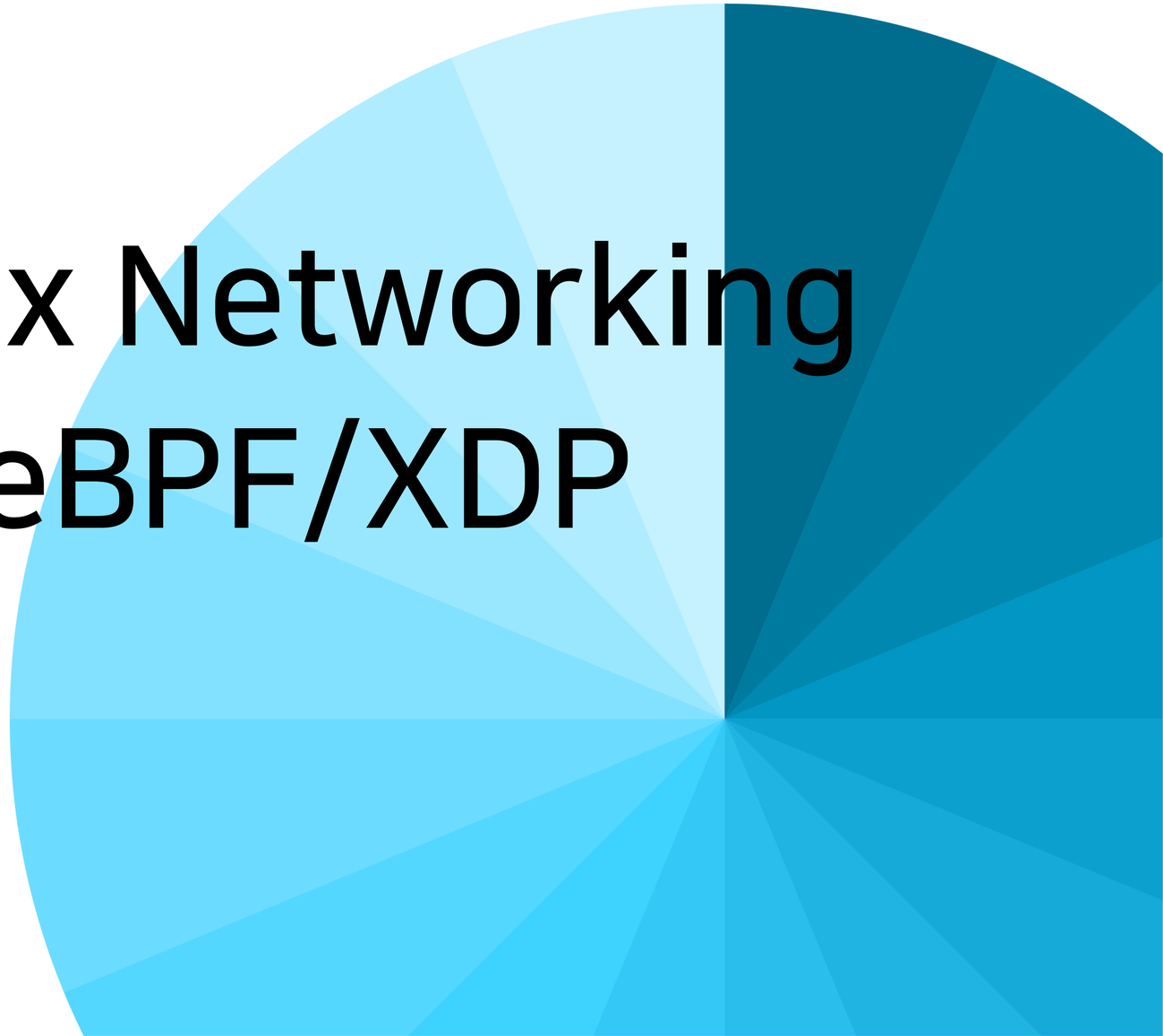
- 더 효율적인 트래픽 처리를 위한 새로운 Networking 기술에 주목





How to drop 10 million packets per second | https://blog.cloudflare.com/how-to-drop-10-million-packets/

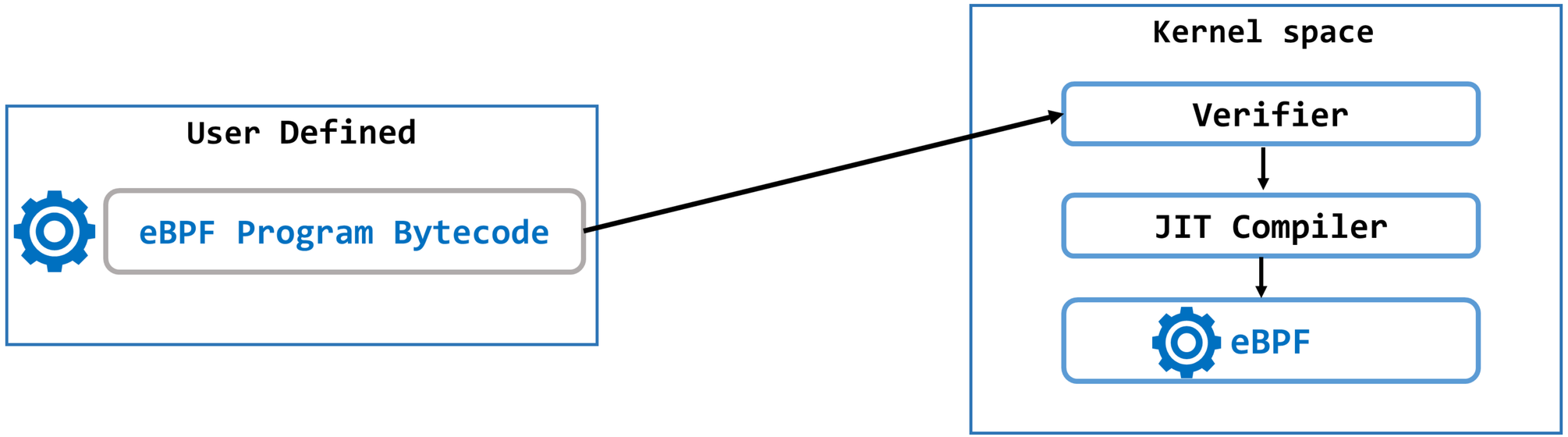
2. 차세대 Linux Networking 메커니즘: eBPF/XDP



2.1 eBPF

extended Berkeley Packet Filter

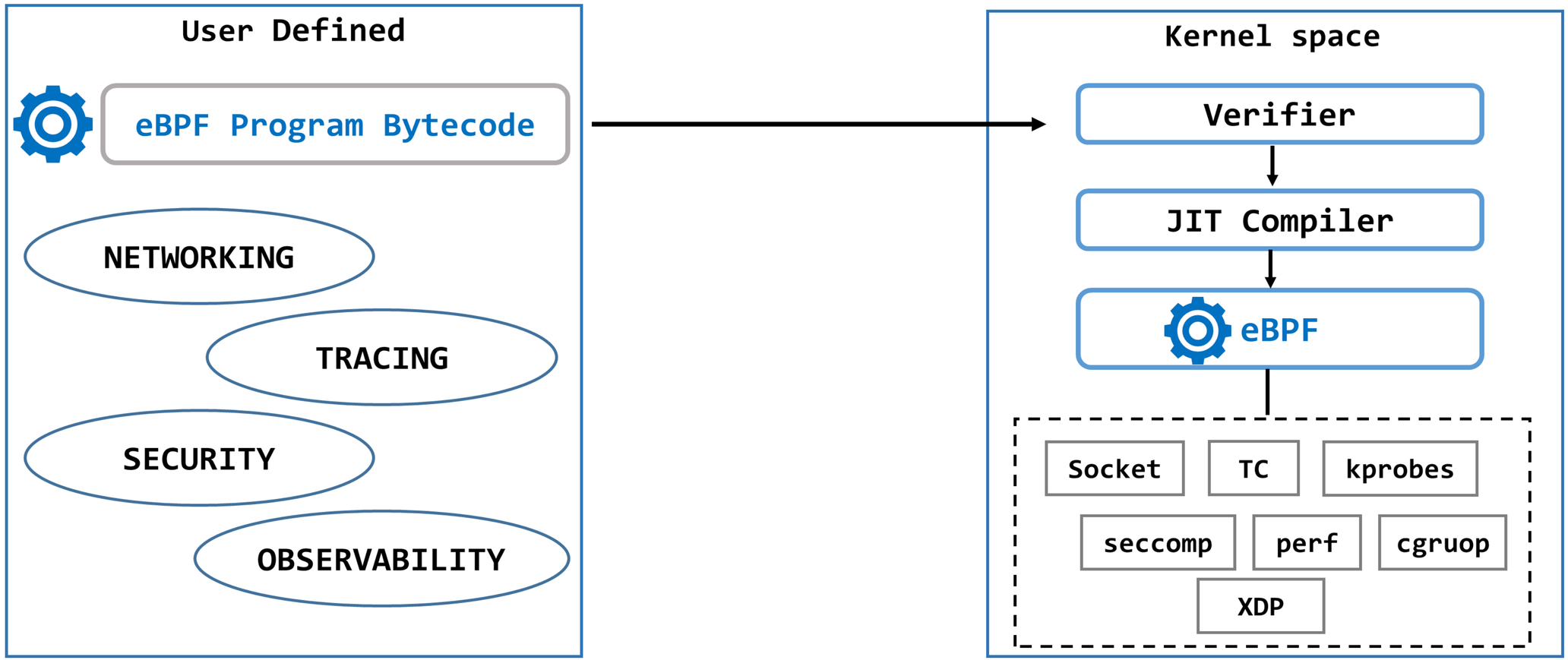
- Kernel Layer 에서 동작하는 Bytecode 를 안전하게 동적으로 Kernel 에 Loading



2.1 eBPF

BPF Program Type

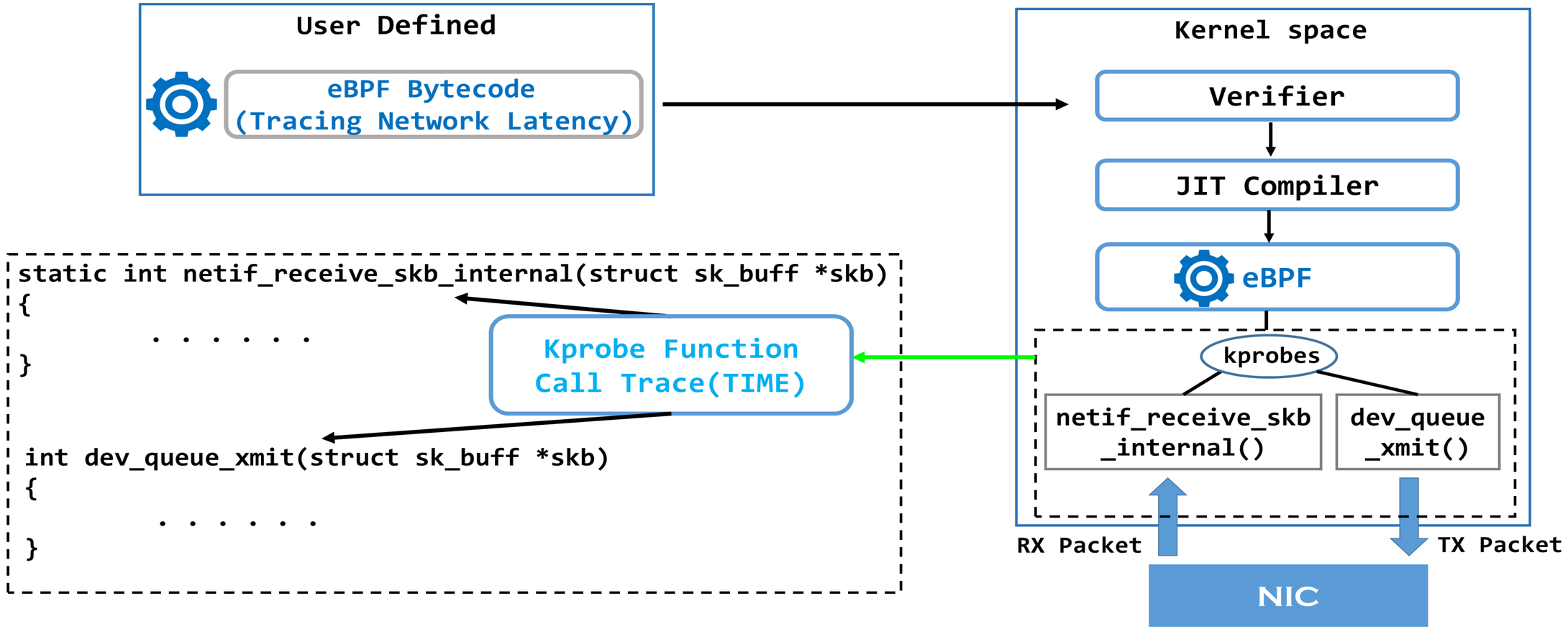
- Kernel Layer 에 동작하는 여러 Type 의 eBPF Program 을 제공



2.1 eBPF

Ex) Tracing Network Latency

- Kernel Networking 에서 Data Link Layer 의 Latency 측정



2.1 eBPF

Ex) Tracing Network Latency

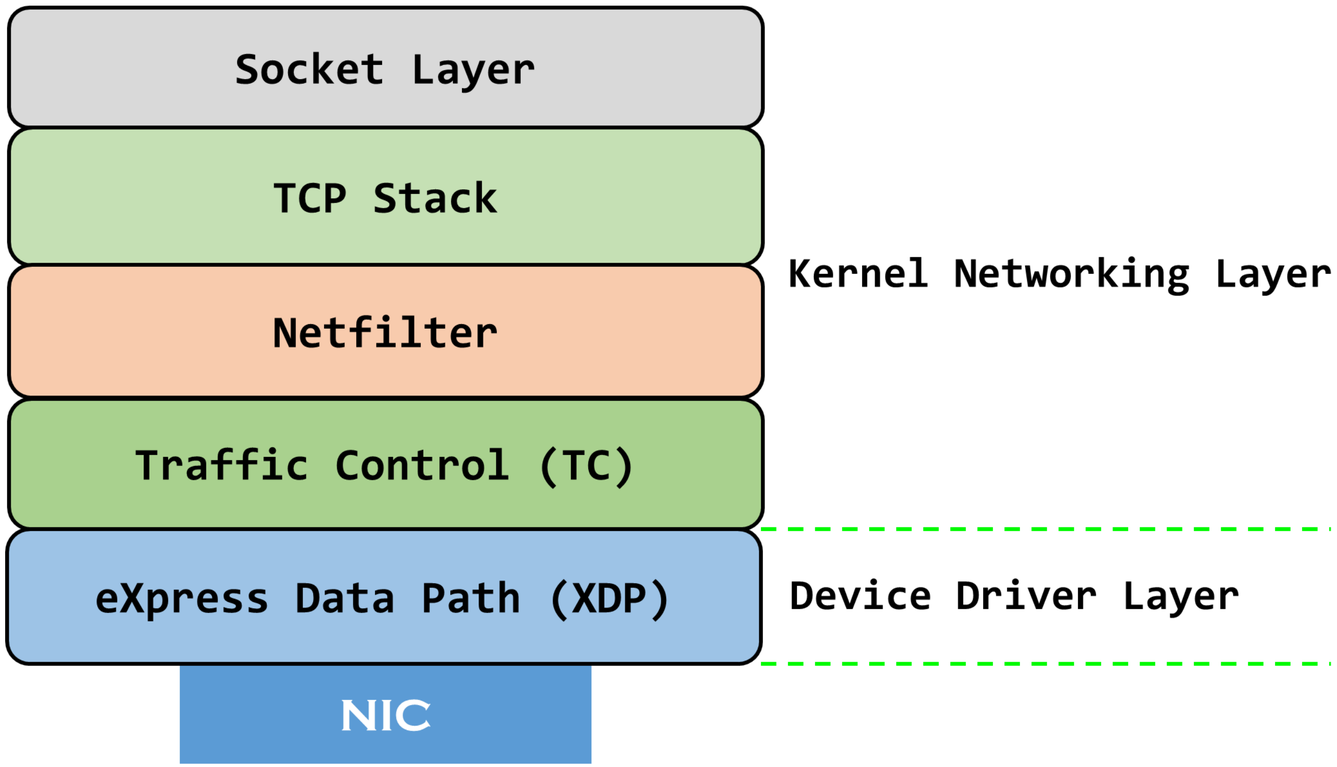
- Kernel Networking 에서 Data Link Layer 의 Latency 측정

```
make -C /var/lib/inju/linux-4.19.93/samples/bpf/../../tools/lib/bpf/ RM='rm -rf' LDFLAGS= srctree=/var/lib/inju/l
inux-4.19.93/samples/bpf/../../ 0=
Warning: Kernel ABI header at 'tools/include/uapi/linux/bpf.h' differs from latest version at 'include/uapi/linux
/bpf.h'
  Building modules, stage 2.
  MODPOST 0 modules
  heatmap of IO latency
   - many events with this latency
   - few events
|1ns      |10ns      |100ns     |1us       |10us      |100us     |1ms       |10ms      |100ms     |1s        |10s
                                                                                                        # 1
                                                                                                        # 1984
                                                                                                        # 1808
                                                                                                        # 2146
                                                                                                        # 2027
                                                                                                        # 2285
                                                                                                        # 2119
                                                                                                        # 2463
```

2.2 XDP

eXpress Data Path

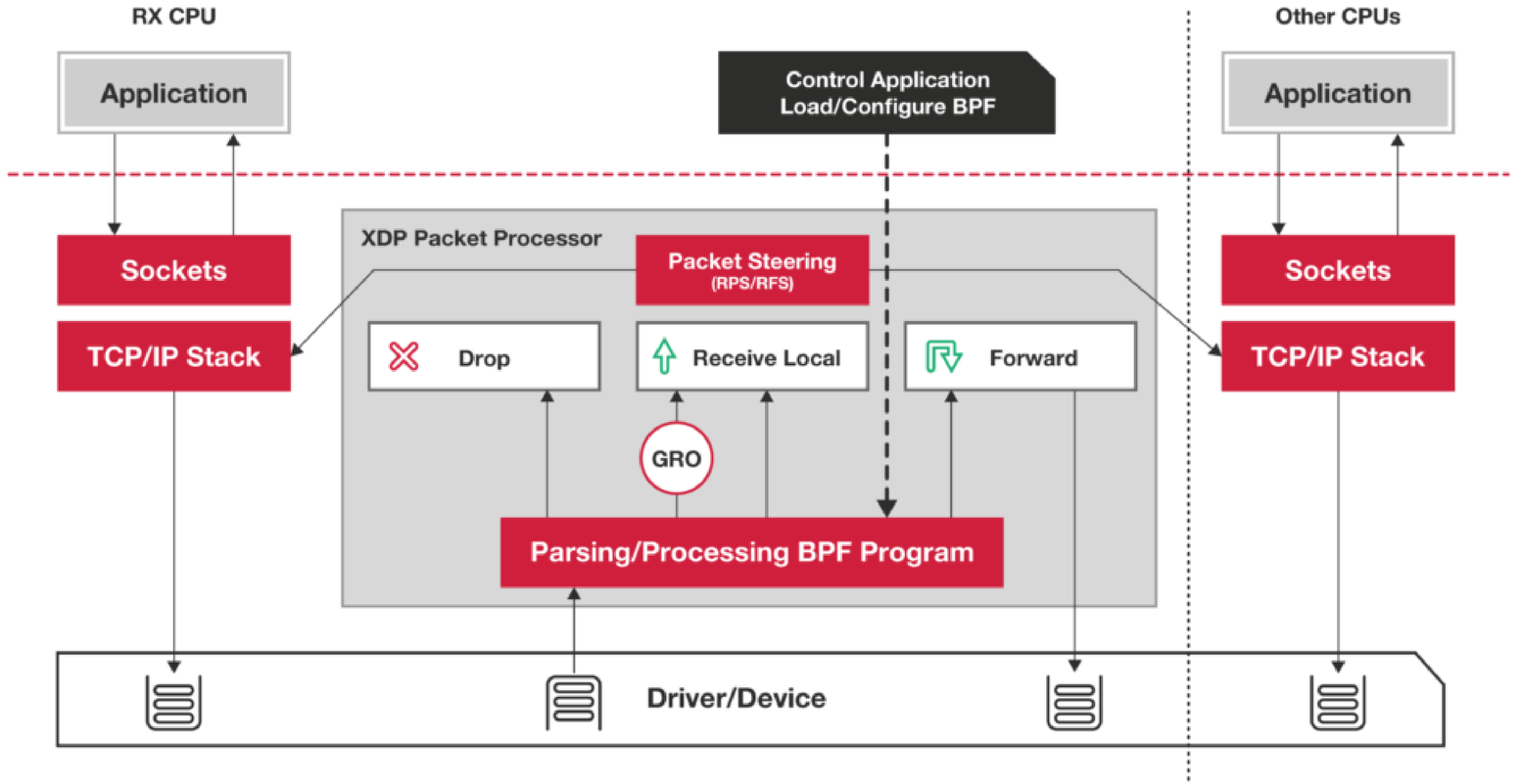
- Kernel Networking Stack 전 Device Driver Level 에서의 Packet 처리



2.2 XDP

eXpress Data Path

- NIC Device Driver Level 에서 동작하는 Packet RX Hooking

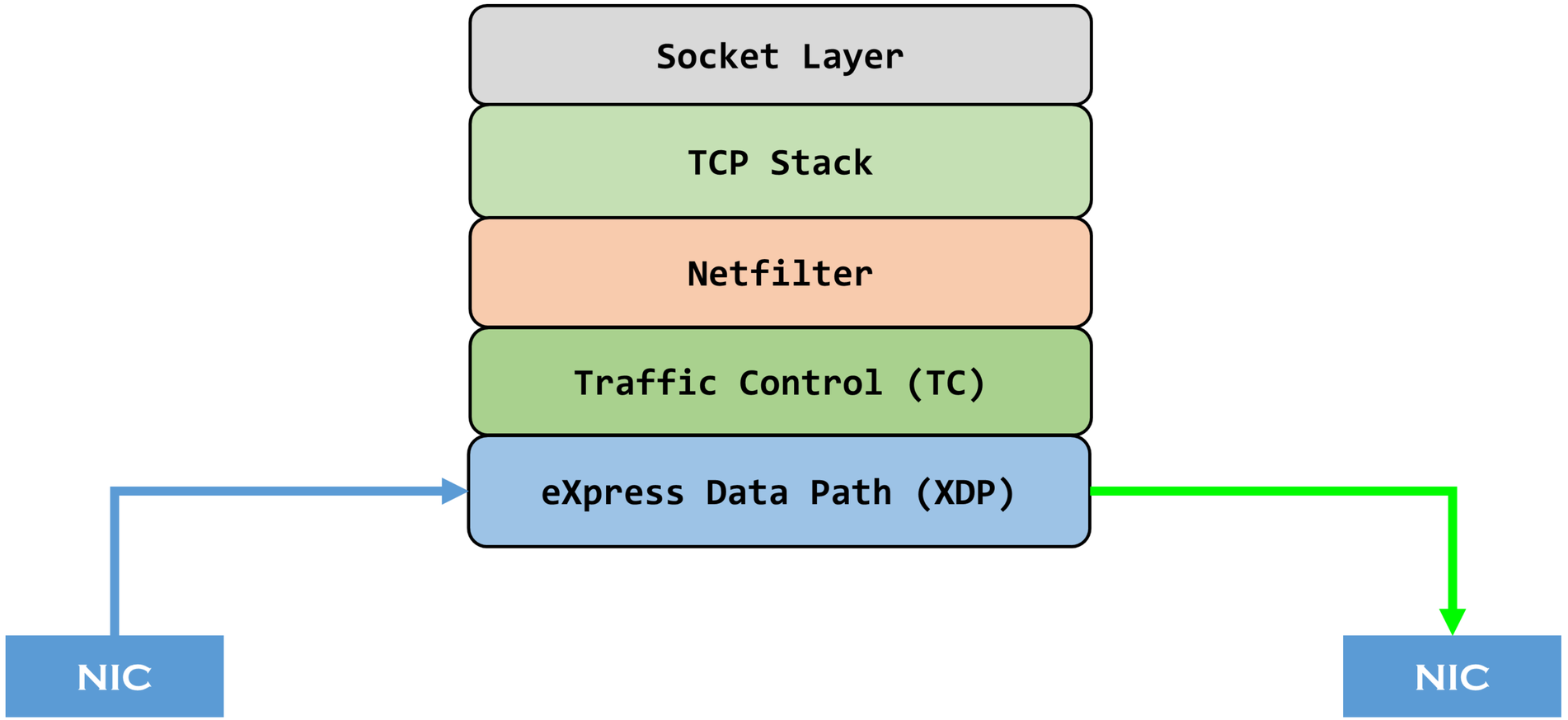


XDP – IO Visor Project | https://www.iovisor.org/technology/xdp

2.2 XDP

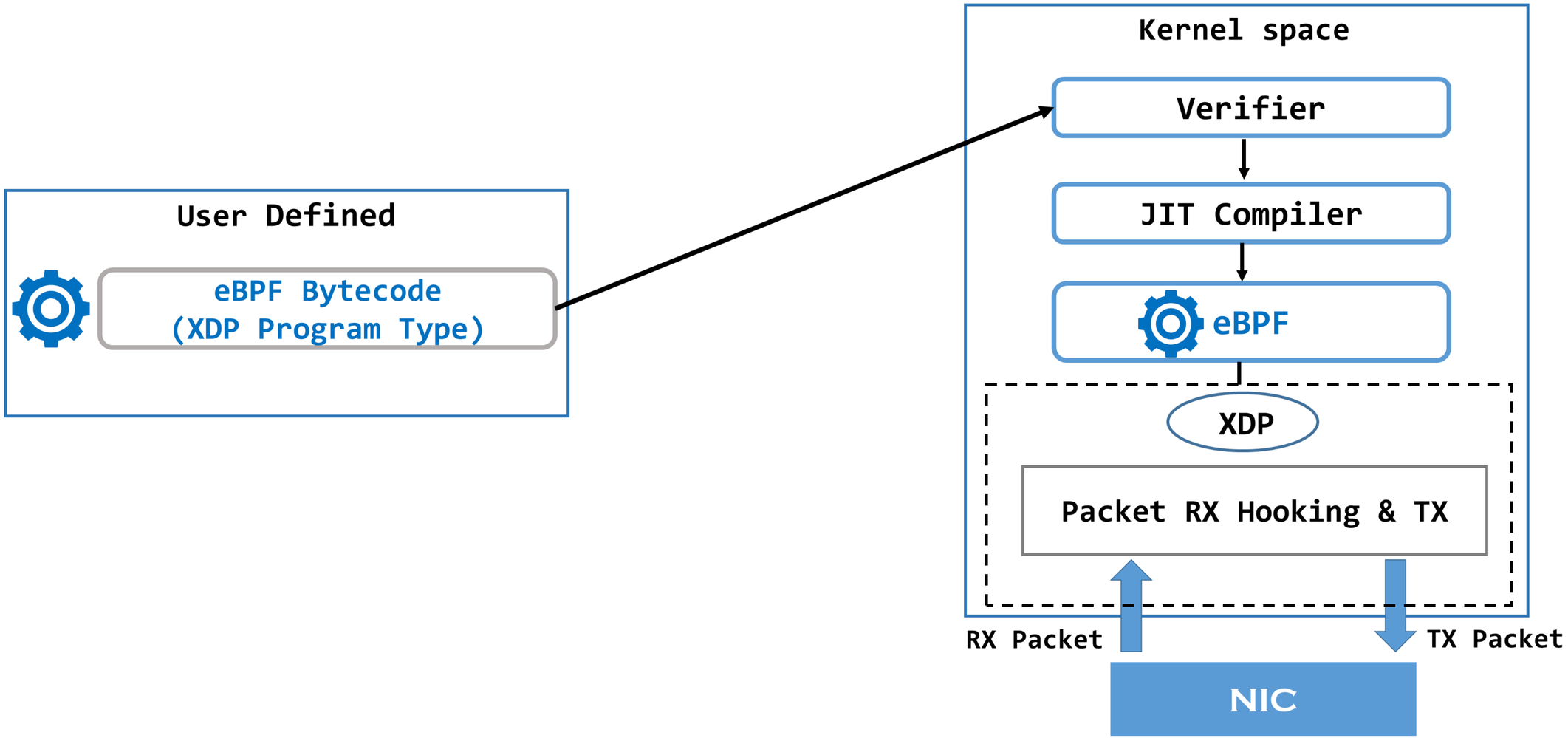
Packet Data Path 최적화를 통한 고성능 Network Traffic Handling

- 병목을 야기하는 Kernel Network Layer 구간과 Structure 할당 우회가 가능



2.3 eBPF & XDP Networking 모듈

BPF/XDP 타입의 Bytecode 를 NIC Driver Layer 에 적재

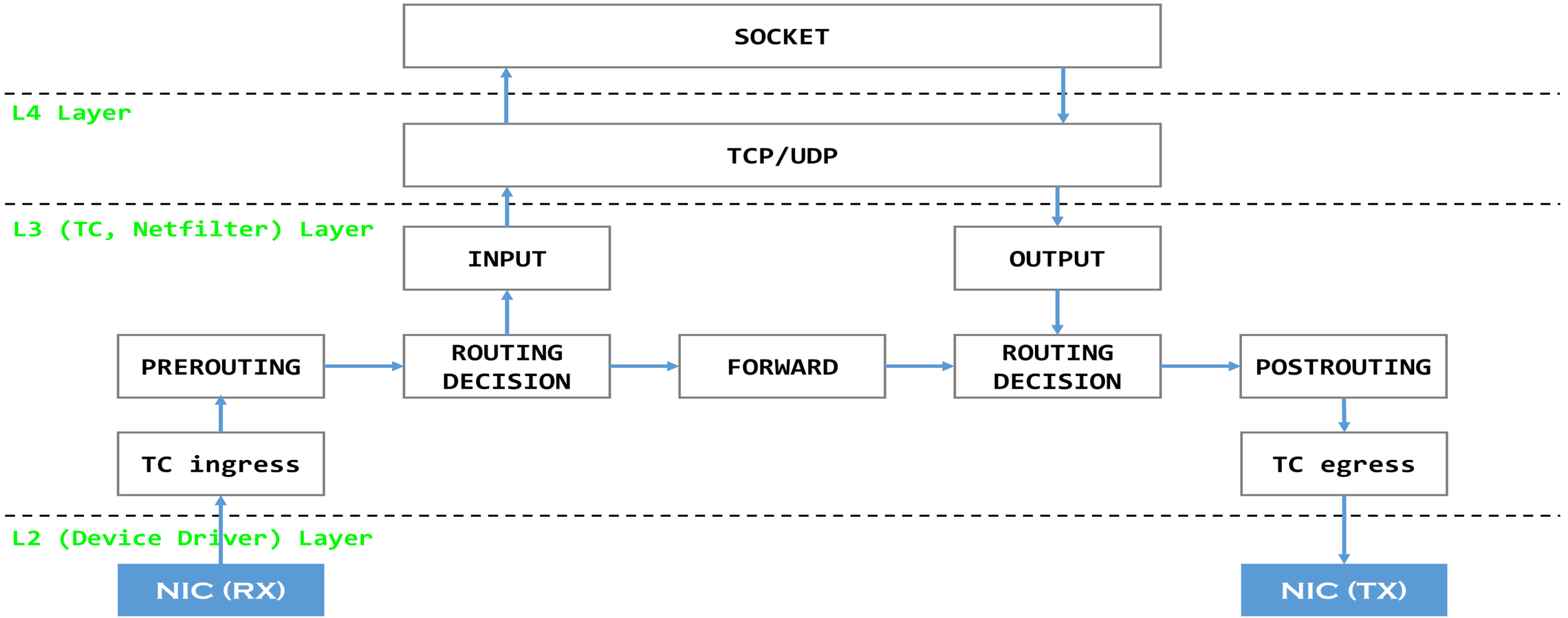


2.3 eBPF & XDP Networking 모듈

DEVIEW 2020

Bypass Kernel Networking Layer

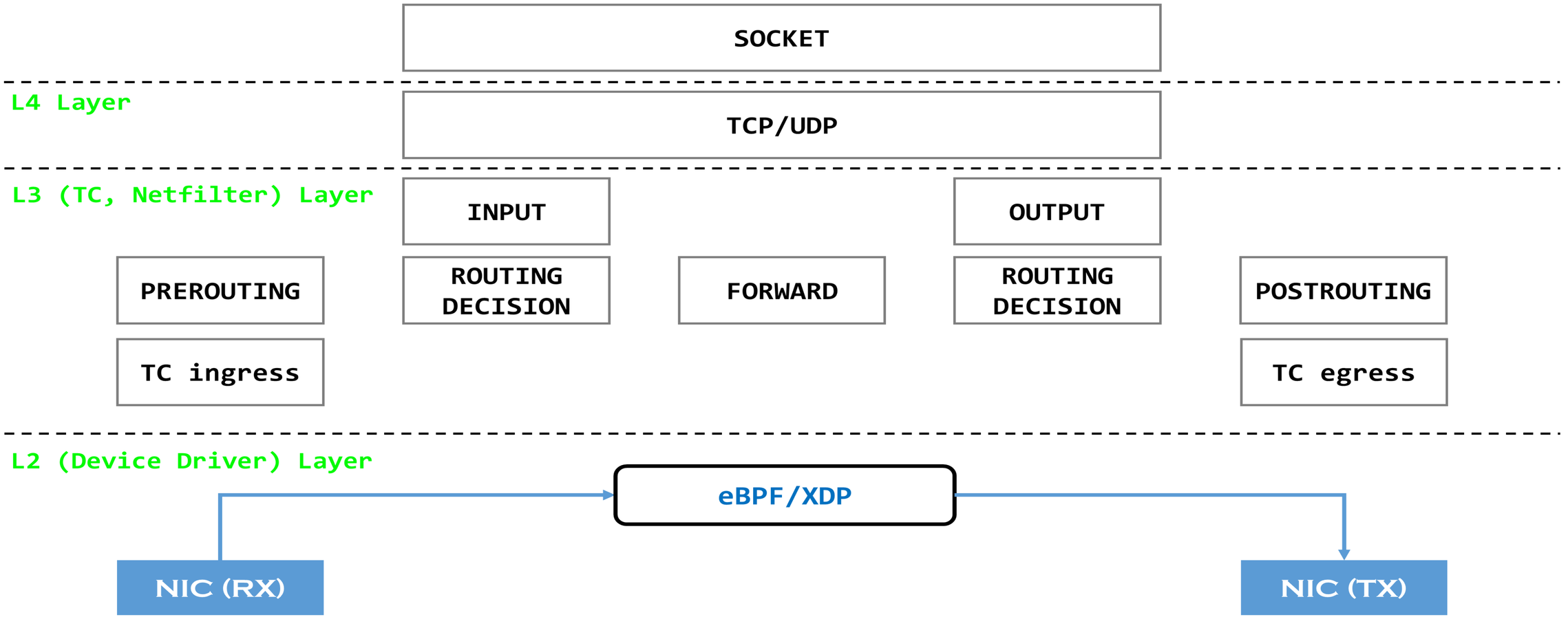
- L3/L4 구간과 Netfilter(iptables), 오버헤드가 큰 Kernel structure 할당을 우회



2.3 eBPF & XDP Networking 모듈

Bypass Kernel Networking Layer

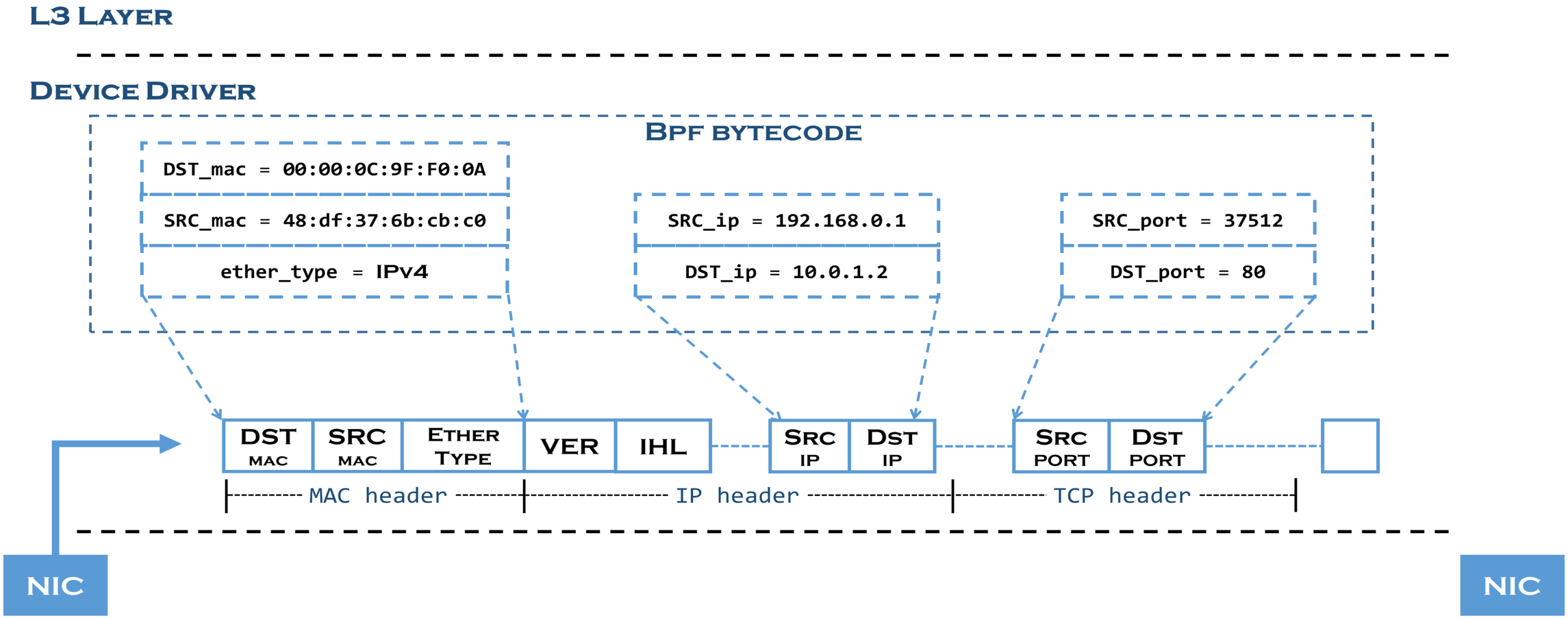
- L3/L4 구간과 Netfilter(iptables), 오버헤드가 큰 Kernel structure 할당을 우회



2.3 eBPF & XDP Networking 모듈

Packet Datagram Level 에서 Data 를 처리

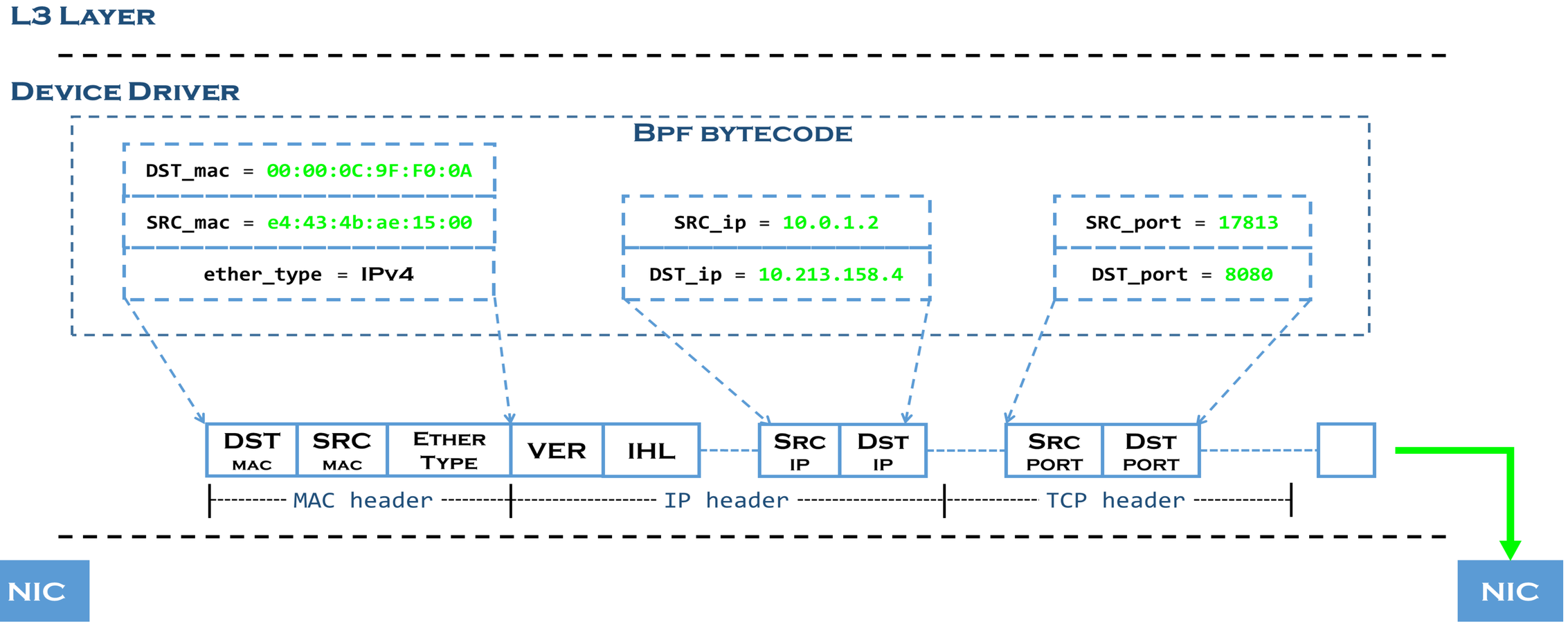
- MAC/IP/TCP Header 등의 Data 를 관련 Kernel structure 할당 없이 직접 제어



2.3 eBPF & XDP Networking 모듈

Packet Datagram Level 에서 Data 를 처리

- MAC/IP/TCP Header 등의 Data 를 관련 Kernel structure 할당 없이 직접 제어



2.3 eBPF & XDP Networking 모듈

eBPF & XDP 로 구현한 Networking 모듈

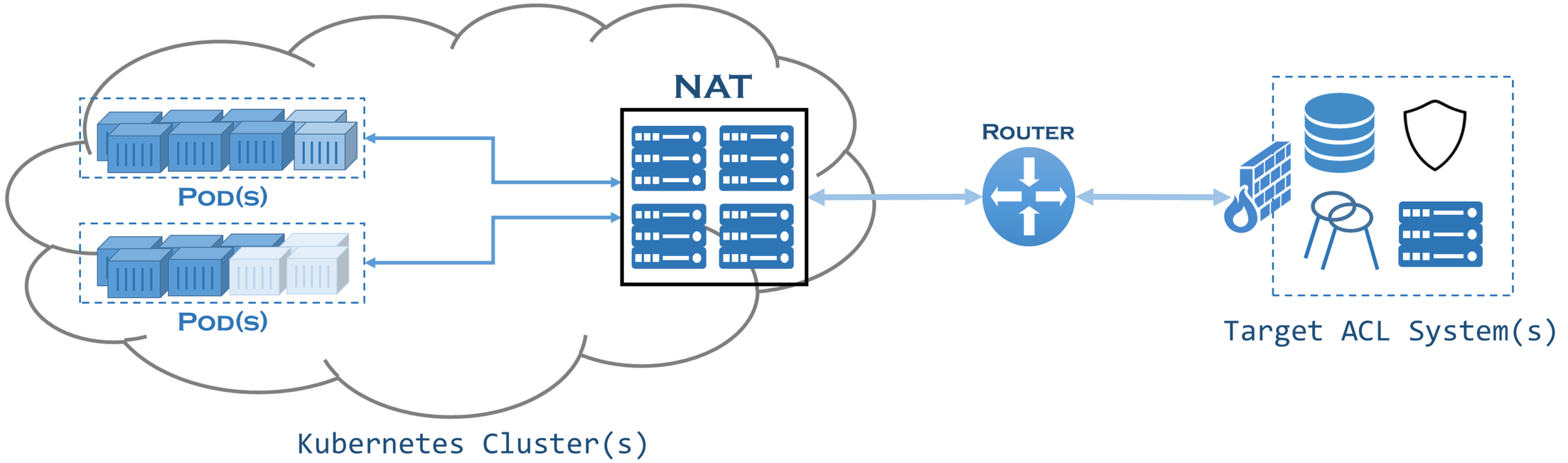
- eBPF Bytecode 는 kernel crash 로 부터 안전하게 동적으로 적재가 가능
- 매우 다양한 Program Type 제공
- Kernel Networking Layer 우회로 높은 성능을 낼 수 있음
- 여러 BPF Function 과 Packet 직접 제어로 대부분의 Networking 연산이 가능함

3. k8s 클러스터 IP ACL 솔루션: eBPF/XDP NAT 시스템

3.1 K8s eBPF/XDP NAT 시스템 소개

Pod <-> eBPF/XDP NAT <-> Target ACL Traffic Flow

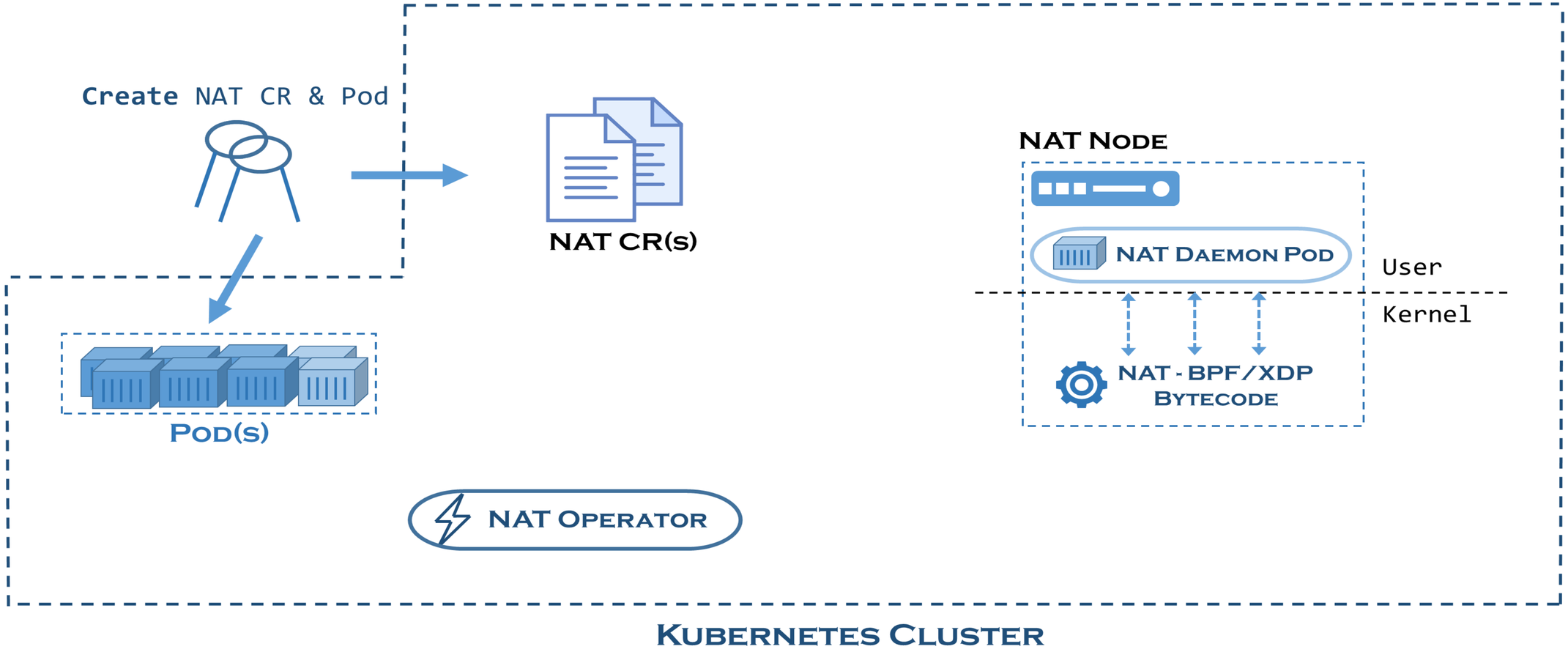
- Pod 으로 부터 오는 Packet 의 Network Address Translation



3.1 K8s eBPF/XDP NAT 시스템 소개

K8s, User Level, Kernel Level Layer 로 구성됨

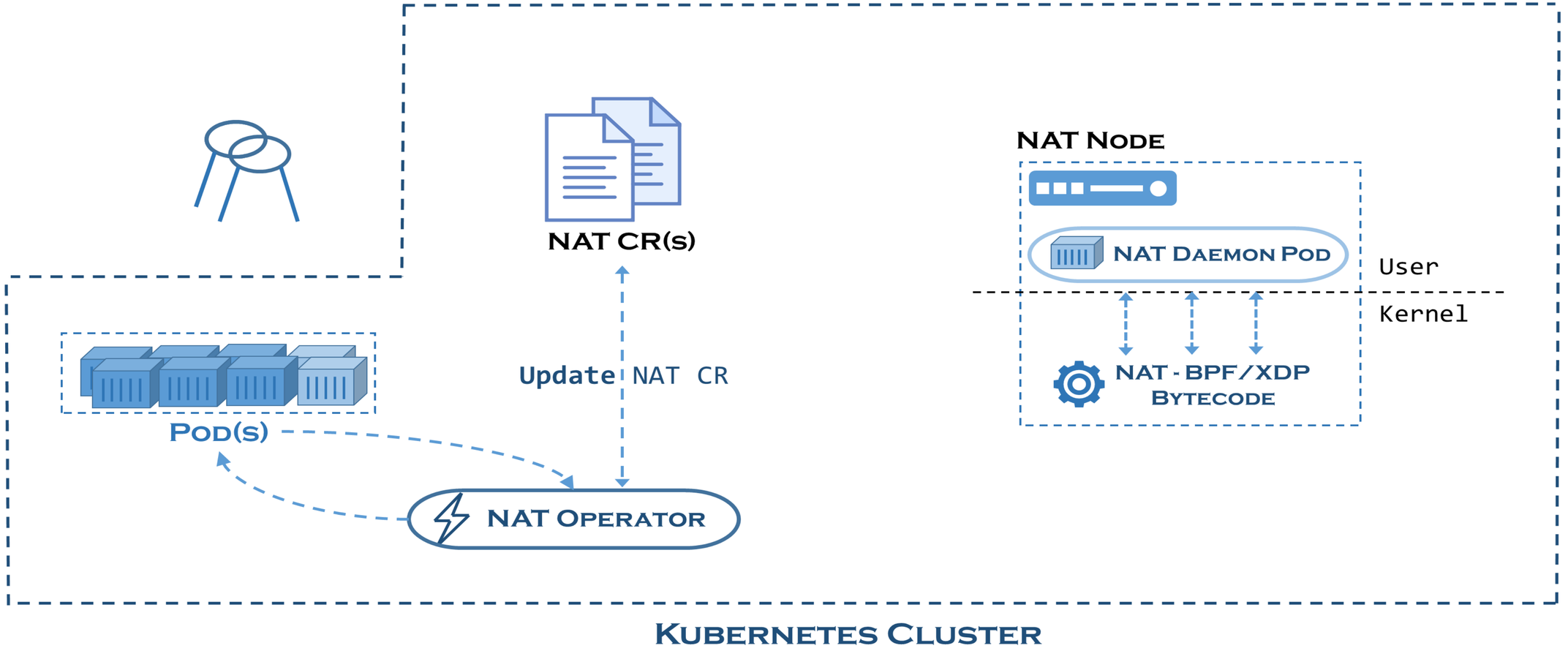
- User 가 NAT Custom Resource 를 생성



3.1 K8s eBPF/XDP NAT 시스템 소개

K8s, User Level, Kernel Level Layer 로 구성됨

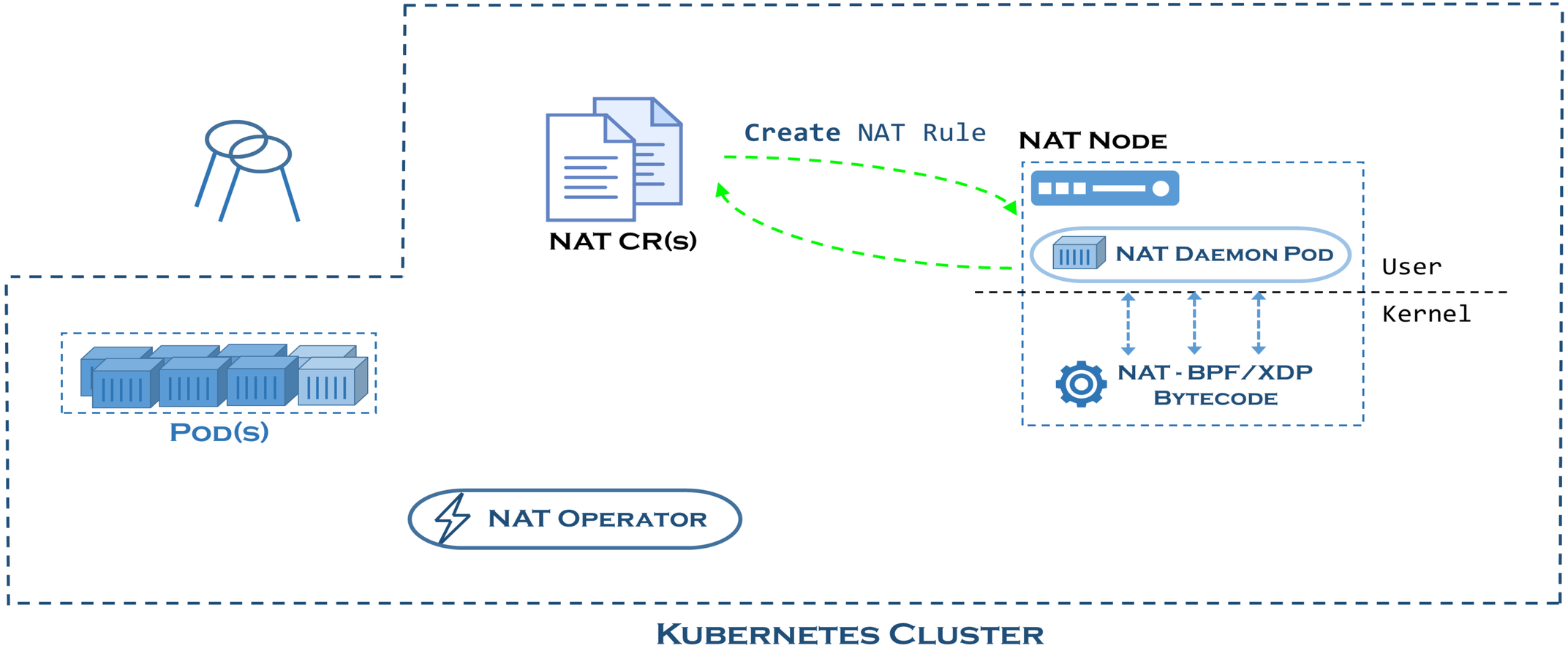
- NAT 를 수행할 Pod 정보를 업데이트



3.1 K8s eBPF/XDP NAT 시스템 소개

K8s, User Level, Kernel Level Layer 로 구성됨

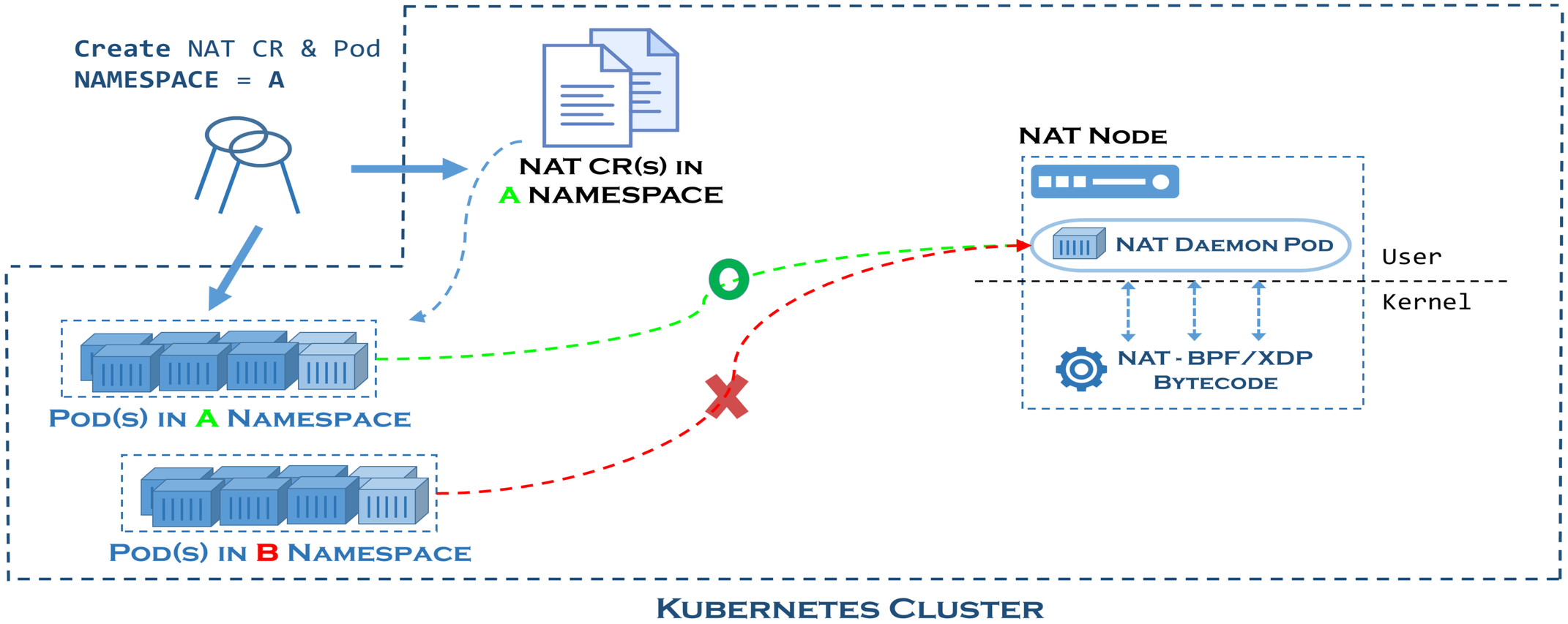
- NAT Daemon Pod 이 해당 Data 를 읽어 eBPF/XDP NAT 에 등록



3.1 K8s eBPF/XDP NAT 시스템 소개

Multi Tenancy 환경에서의 Namespace 별 Mapping 권한 제어

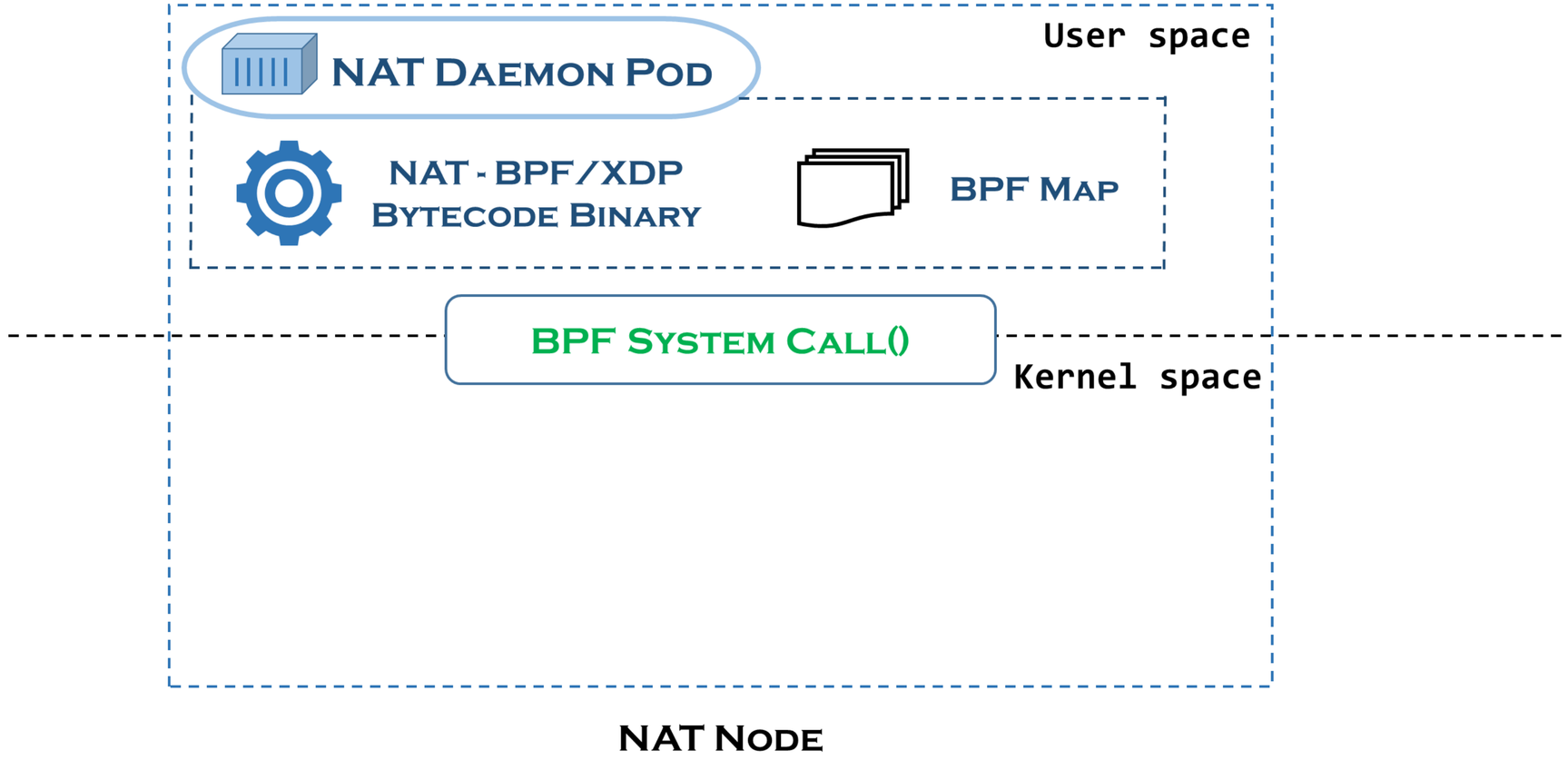
- Pod <-> NAT 간 Mapping 은 같은 K8s Namespace 에서만 가능



3.2 BPF 시스템 콜

User Level 에서의 eBPF system call

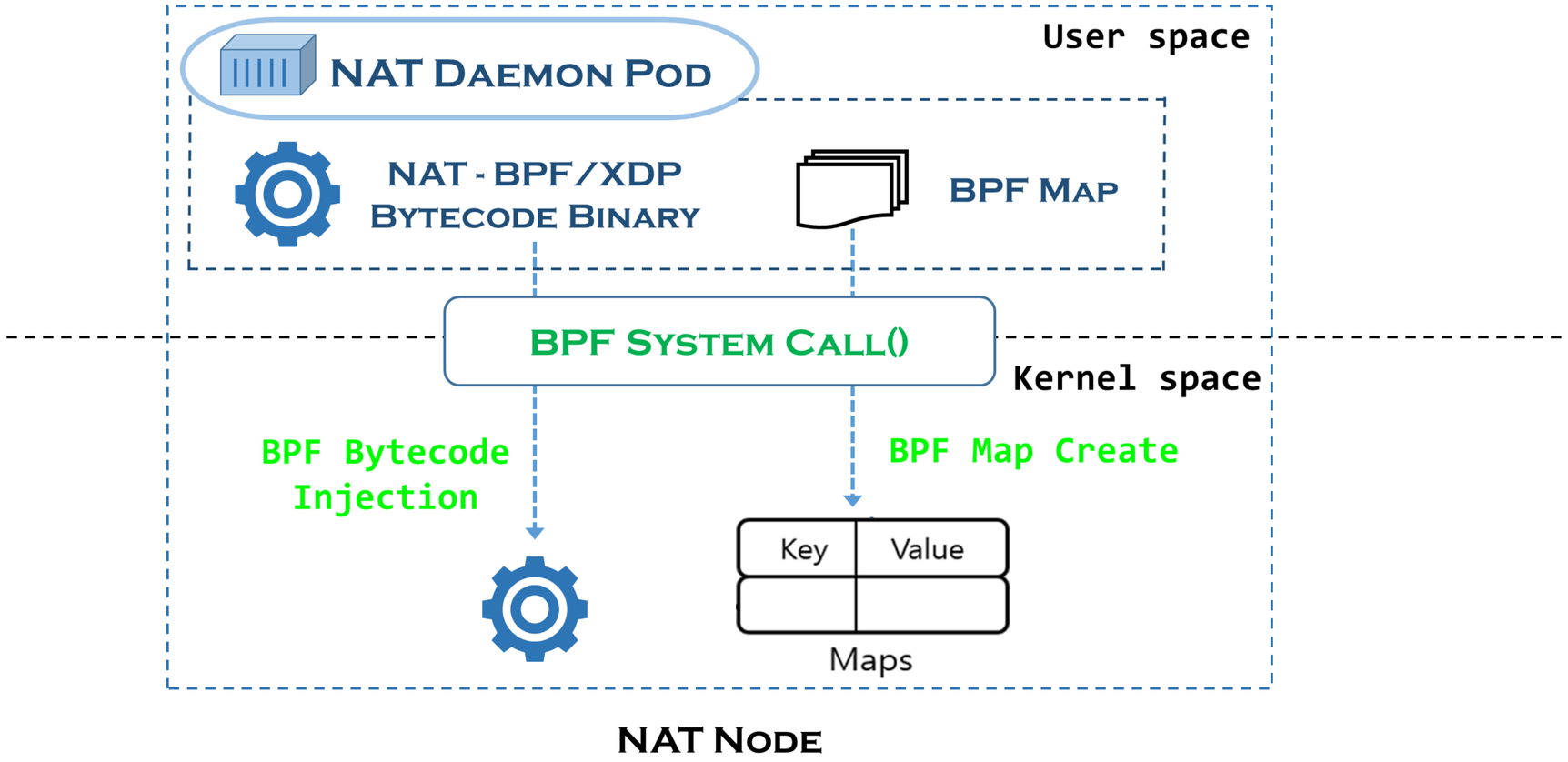
- User <-> Kernel 간 Data 교환을 위한 BPF Map 정의(Hash, Array, Stack 등)



3.2 BPF 시스템 콜

User Level 에서의 eBPF system call

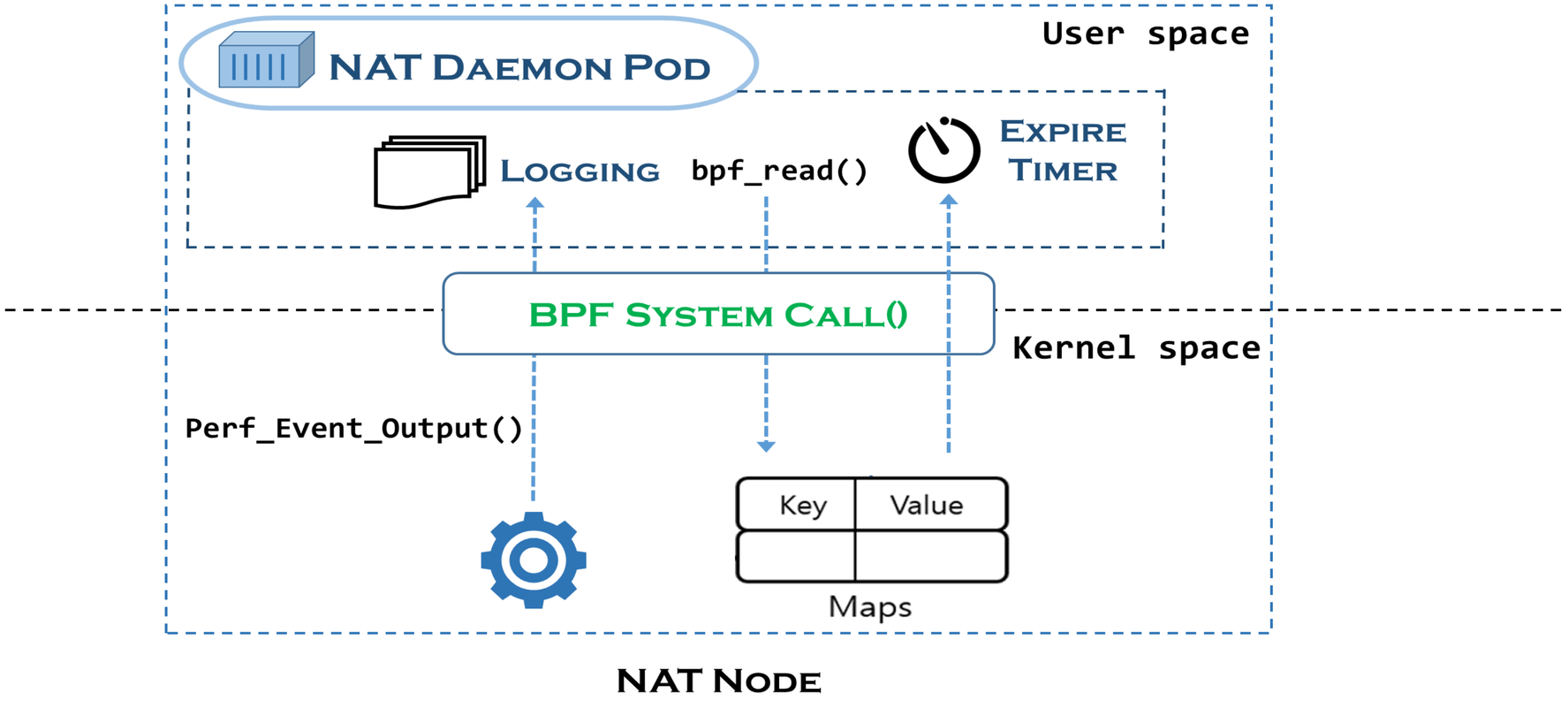
- BPF Kernel Bytecode Injection 및 BPF Map 생성



3.2 BPF 시스템 콜

BPF System Call 을 통한 User <-> Kernel 간 Communication

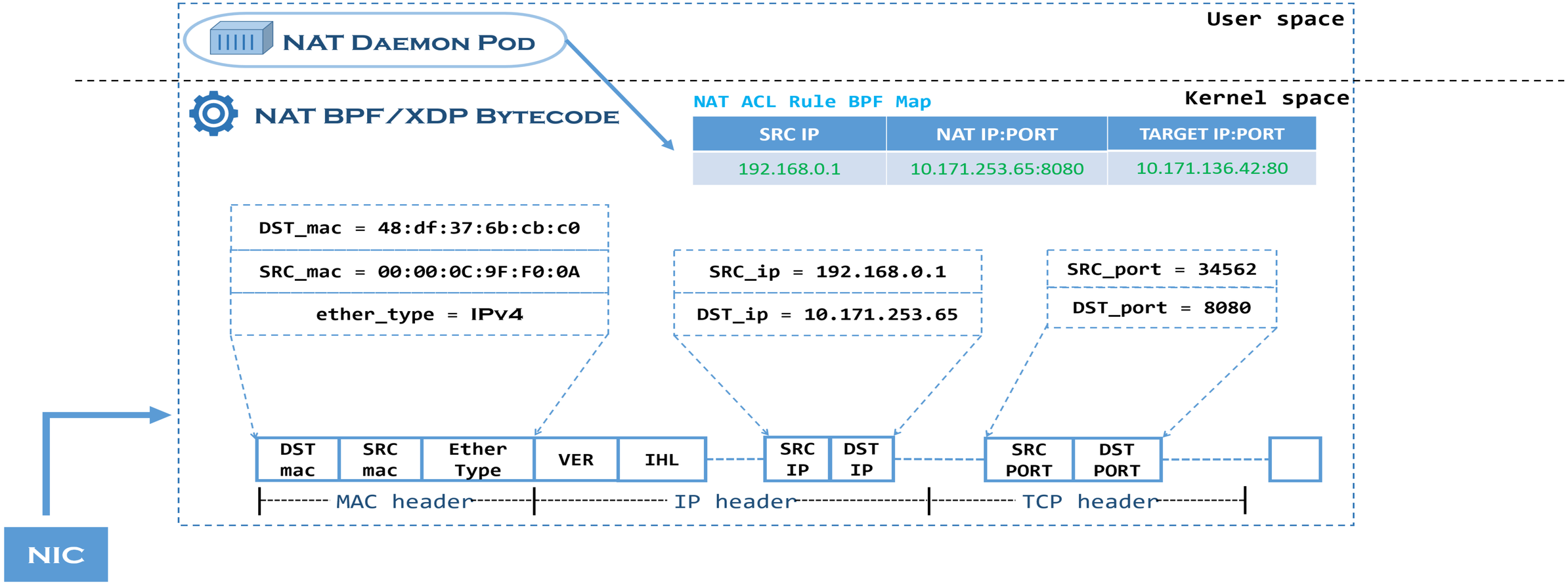
- NAT Connection 정보를 관리하고, 사용이 종료된 연결을 Timer 를 통해 정리 및 Logging



3.3 Network Driver Layer 에서 NAT

Packet 의 MAC/IP/TCP Header 접근 및 변경

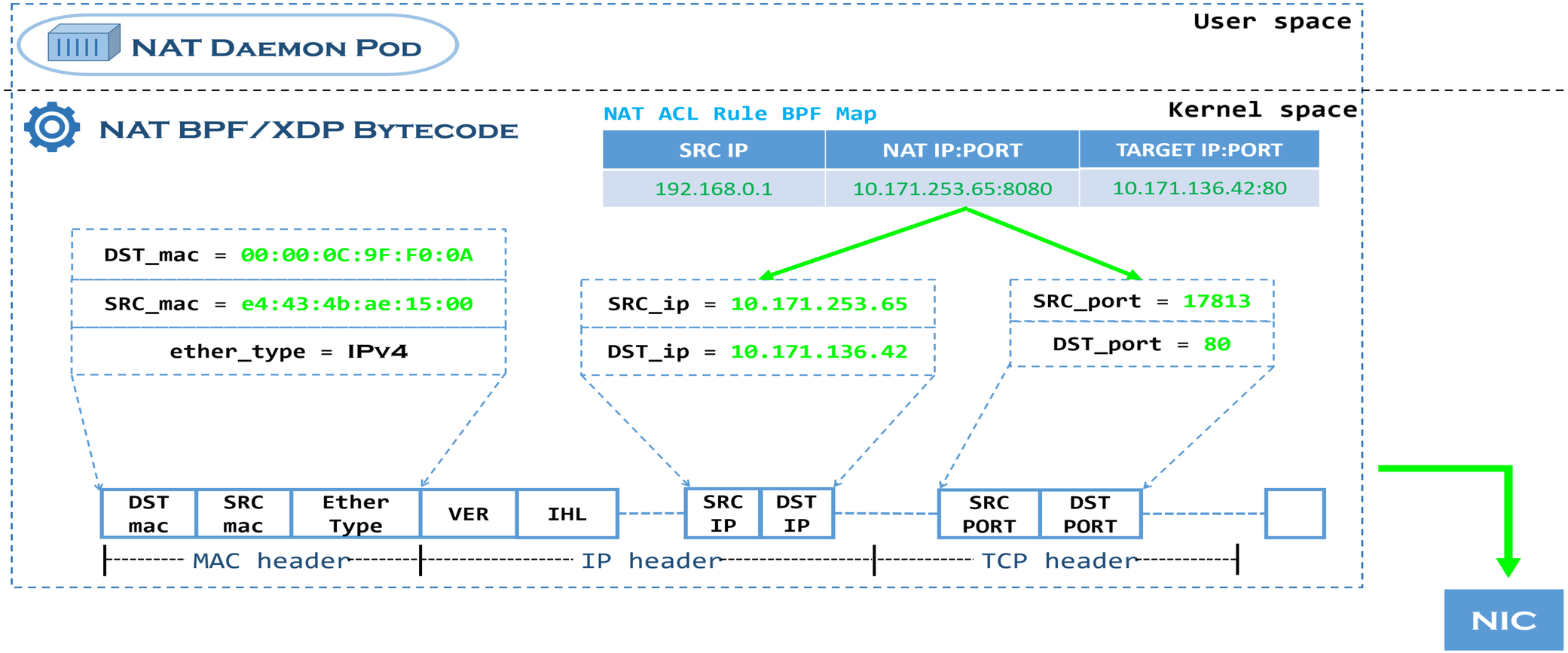
- 각 Header 의 Byte 를 계산하여 Header data 에 접근, IP/Port 값 변경



3.3 Network Driver Layer 에서 NAT

Packet 의 MAC/IP/TCP Header 접근 및 변경

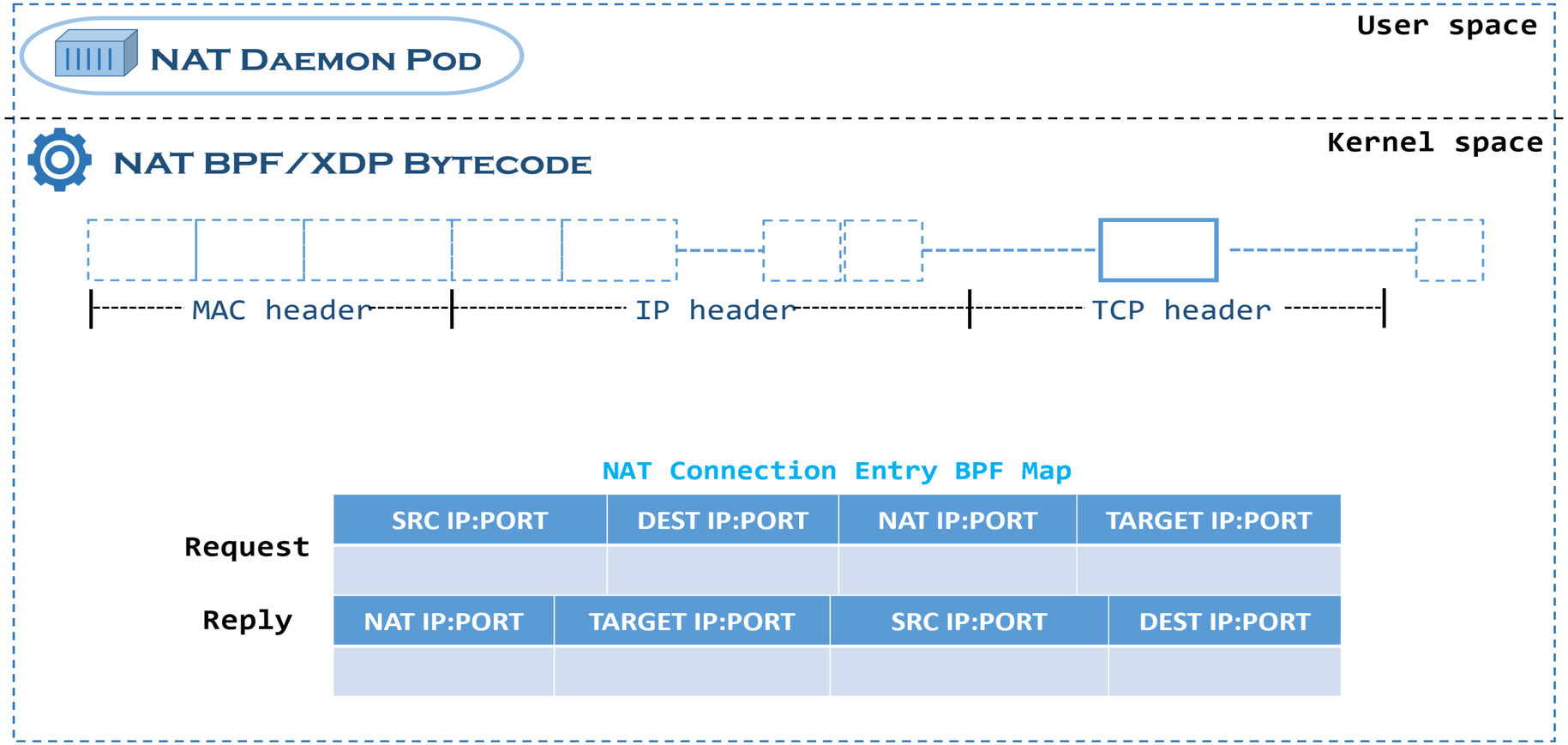
- 각 Header 의 Byte 를 계산하여 Header data 에 접근, IP/Port 값 변경



3.3 Network Driver Layer 에서 NAT

NAT Connection Entry 업데이트

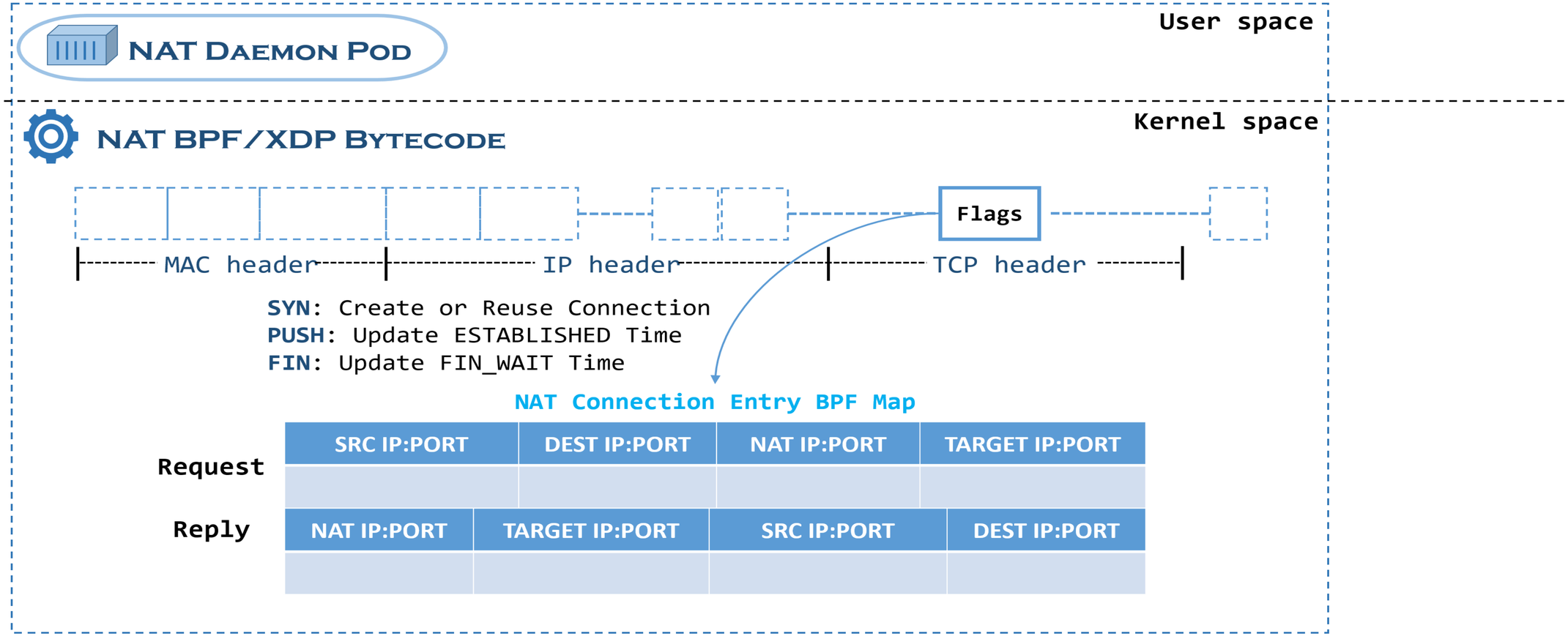
- 매 통신마다 Connection 정보가 담긴 Entry 를 유지하고 상태를 관리



3.3 Network Driver Layer 에서 NAT

NAT Connection Entry 업데이트

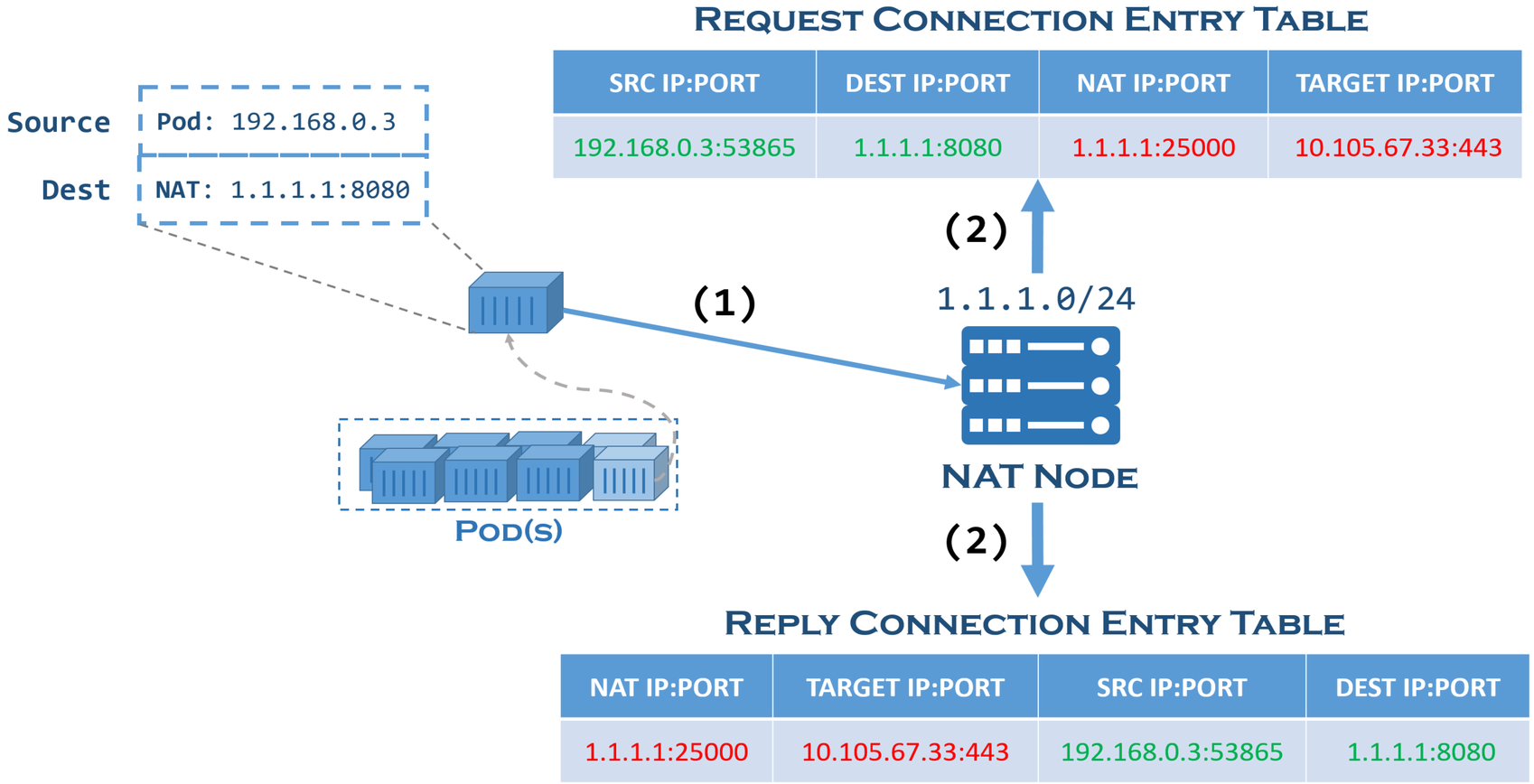
- TCP Packet Flag(SYN/PUSH/FIN) 등으로 Connection Entry 를 업데이트

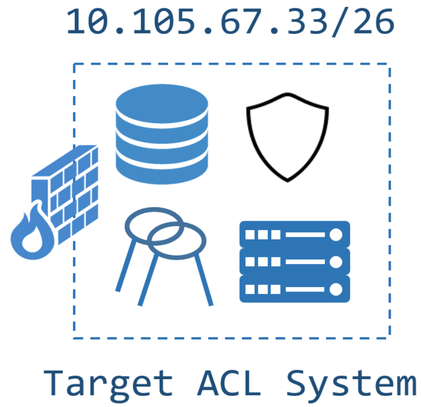


3.4 NAT Connection Entry Table

Connection Entry Table 을 통한 NAT 메커니즘

- Request: Pod -> NAT -> Target

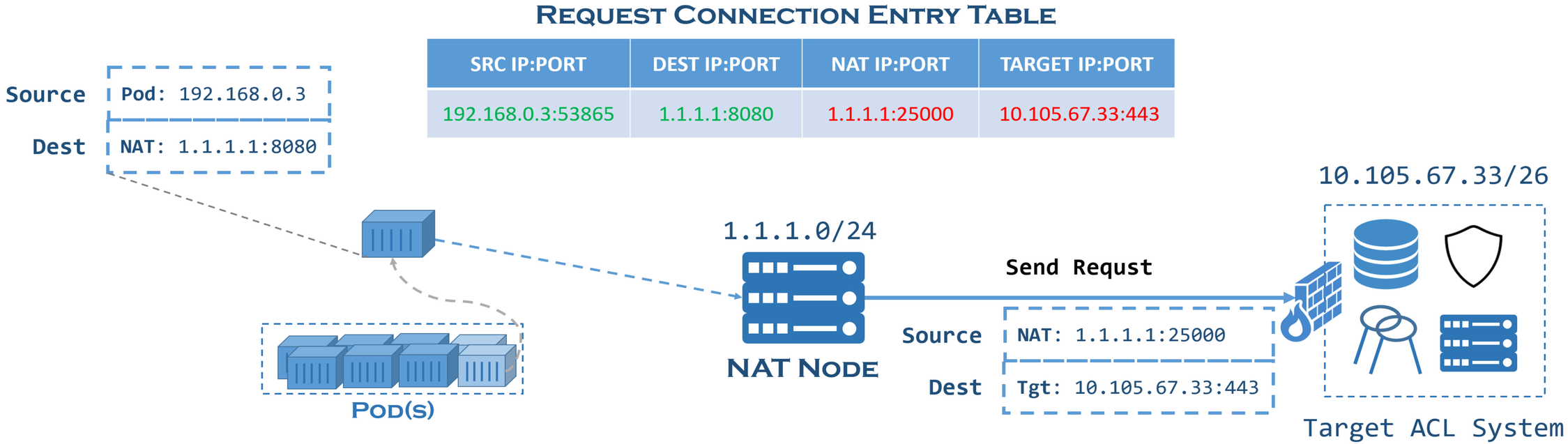




3.4 NAT Connection Entry Table

Connection Entry Table 을 통한 NAT 메커니즘

- Request: Pod -> NAT -> Target



Reply Connection Entry Table

NAT IP:PORT	TARGET IP:PORT	SRC IP:PORT	DEST IP:PORT
1.1.1.1:25000	10.105.67.33:443	192.168.0.3:53865	1.1.1.1:8080

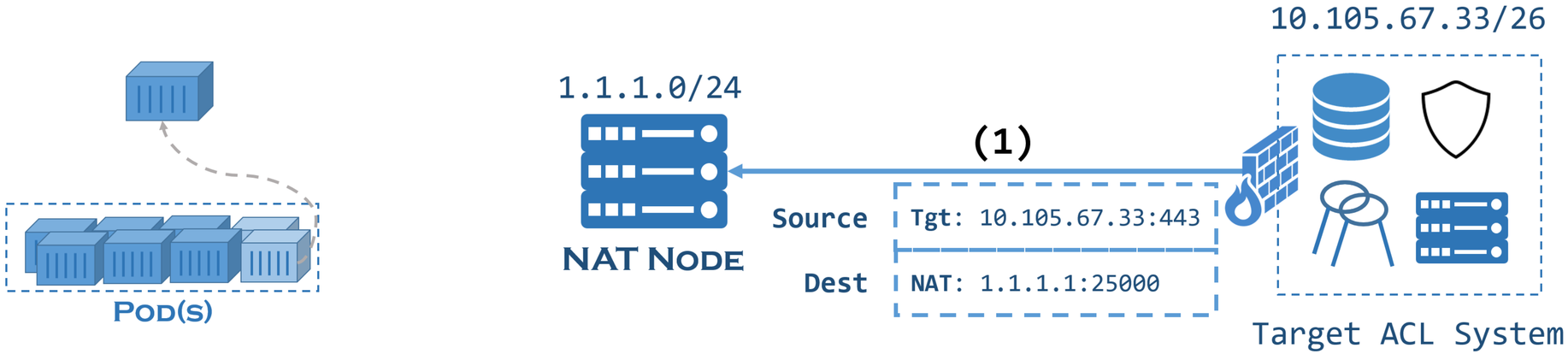
3.4 NAT Connection Entry Table

Connection Entry Table 을 통한 NAT 메커니즘

- Reply: Pod <- NAT <- Target

Request Connection Entry Table

SRC IP:PORT	DEST IP:PORT	NAT IP:PORT	TARGET IP:PORT
192.168.0.3:53865	1.1.1.1:8080	1.1.1.1:25000	10.105.67.33:443



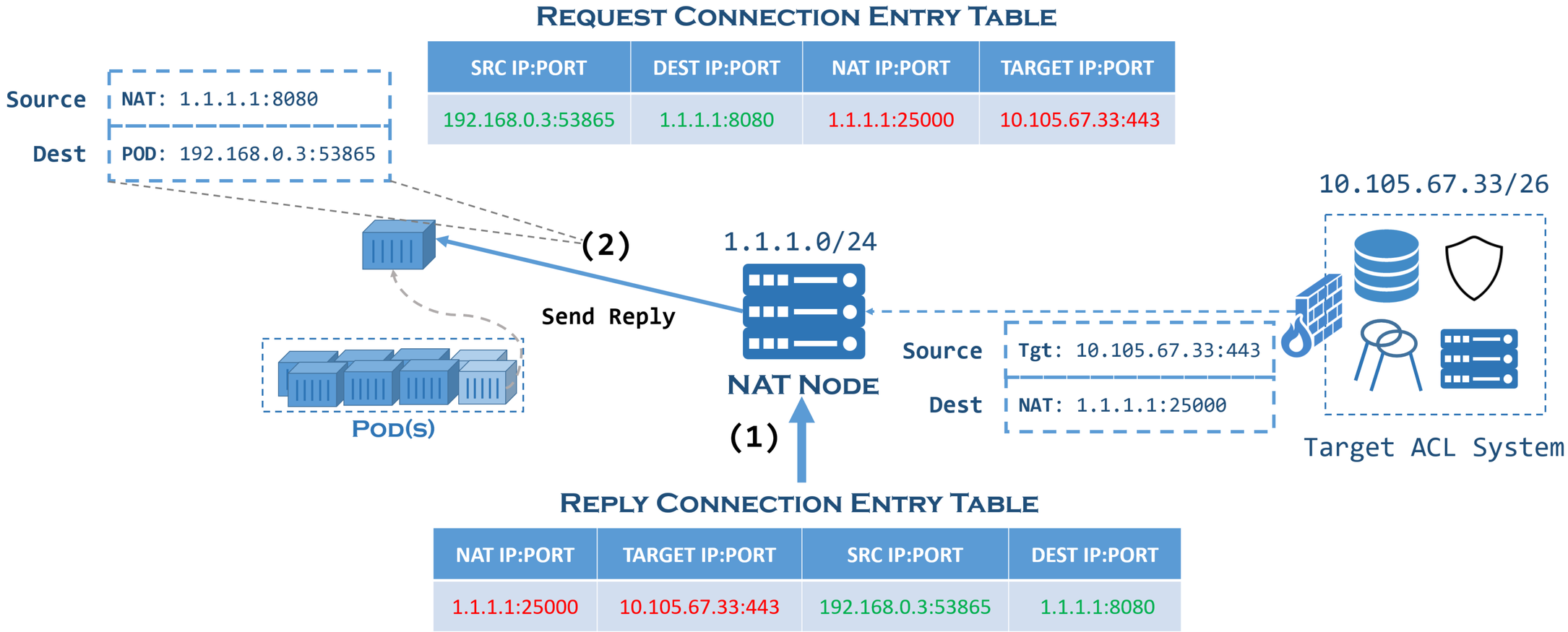
Reply Connection Entry Table

NAT IP:PORT	TARGET IP:PORT	SRC IP:PORT	DEST IP:PORT
1.1.1.1:25000	10.105.67.33:443	192.168.0.3:53865	1.1.1.1:8080

3.4 NAT Connection Entry Table

Connection Entry Table 을 통한 NAT 메커니즘

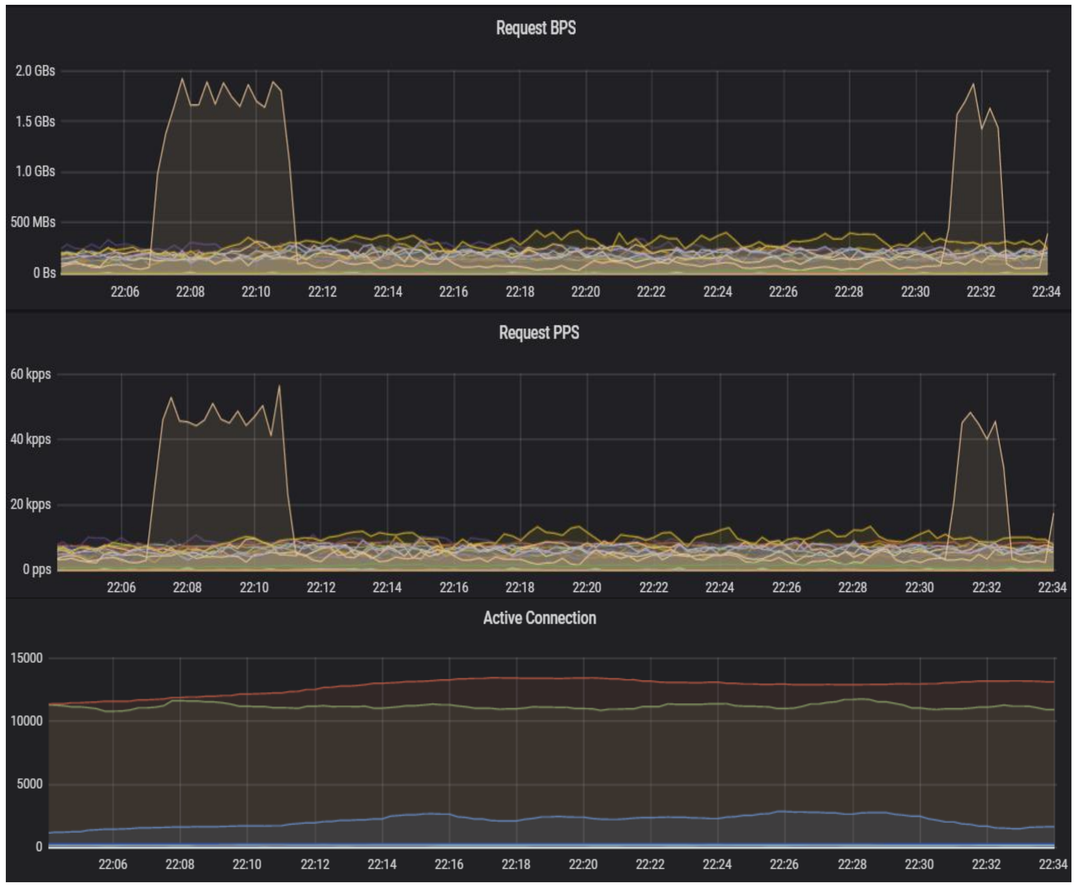
- Reply: Pod <- NAT <- Target

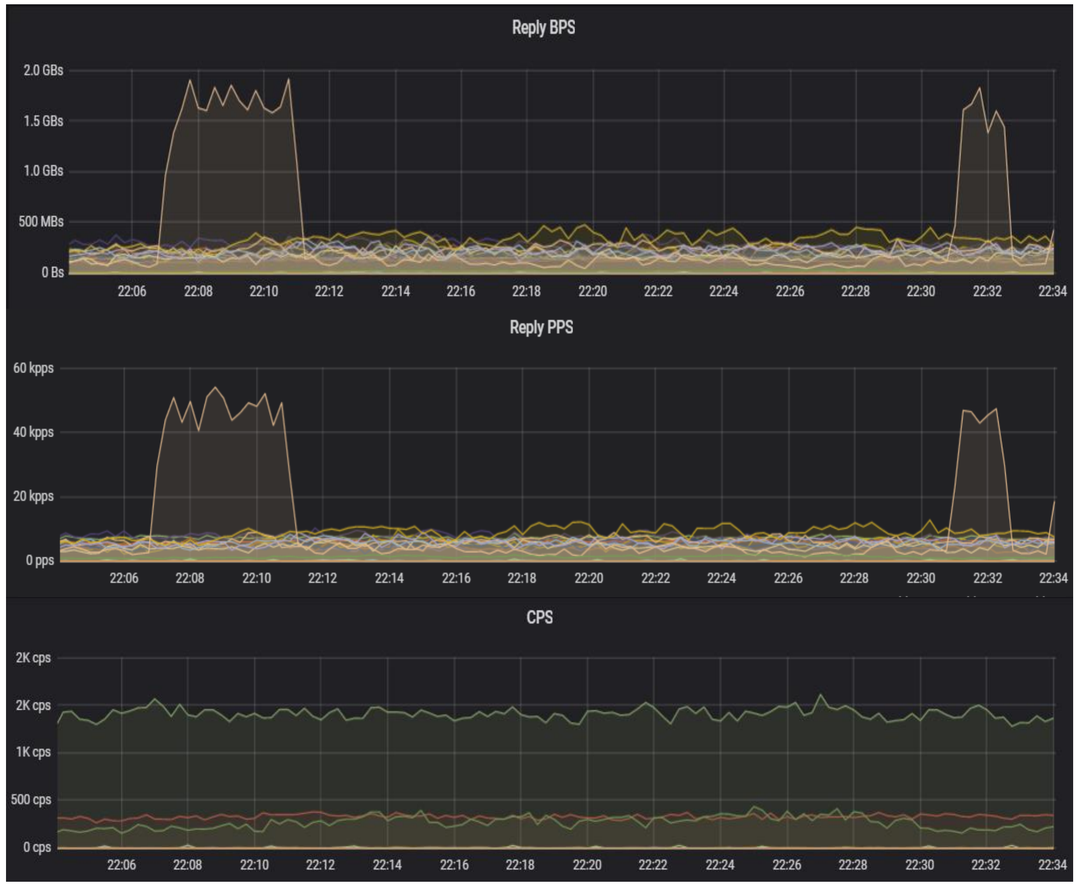


3.5 Prometheus & Grafana 모니터링

NAT Resource Monitoring Dashboard 제공

- NAT Packet RX/TX 시 CPS, BPS/PPS, Active Connection 등의 Resource 현황



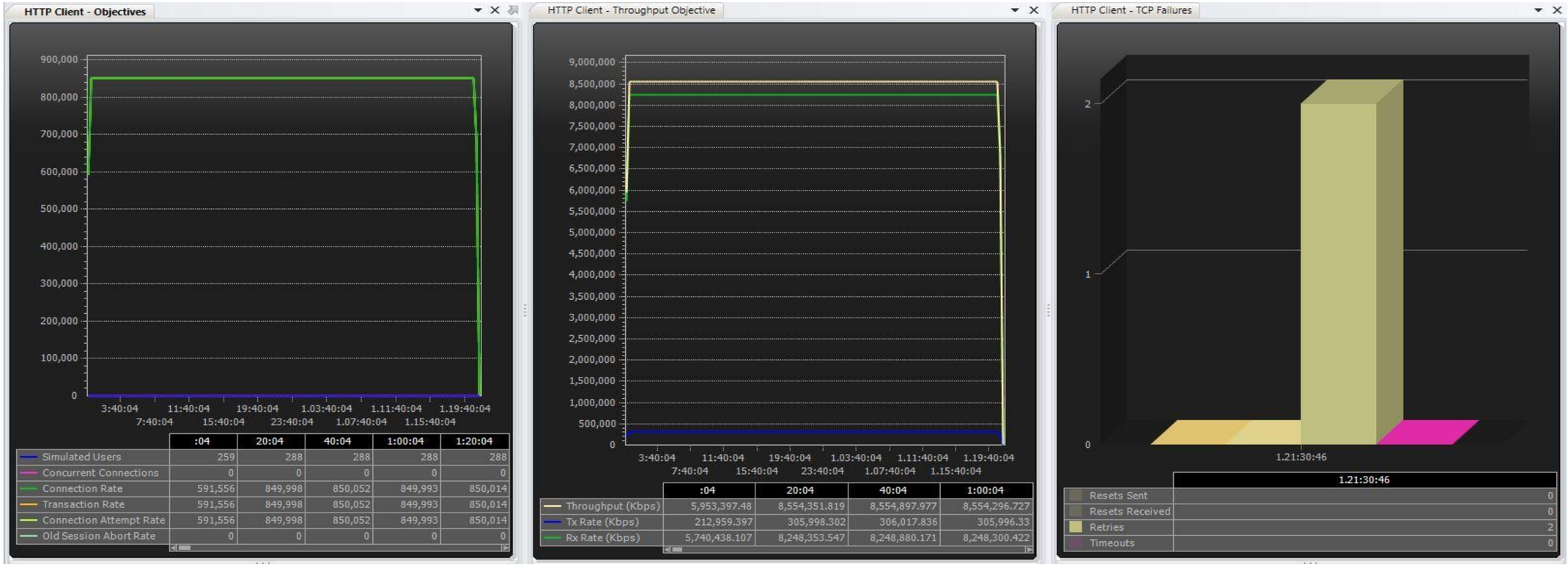


4. 고성능 & 고가용성 NAT 시스템

4.1 eBPF/XDP NAT Performance

NAT 1대 기준 HW 네트워크 계측기를 통한 성능 측정

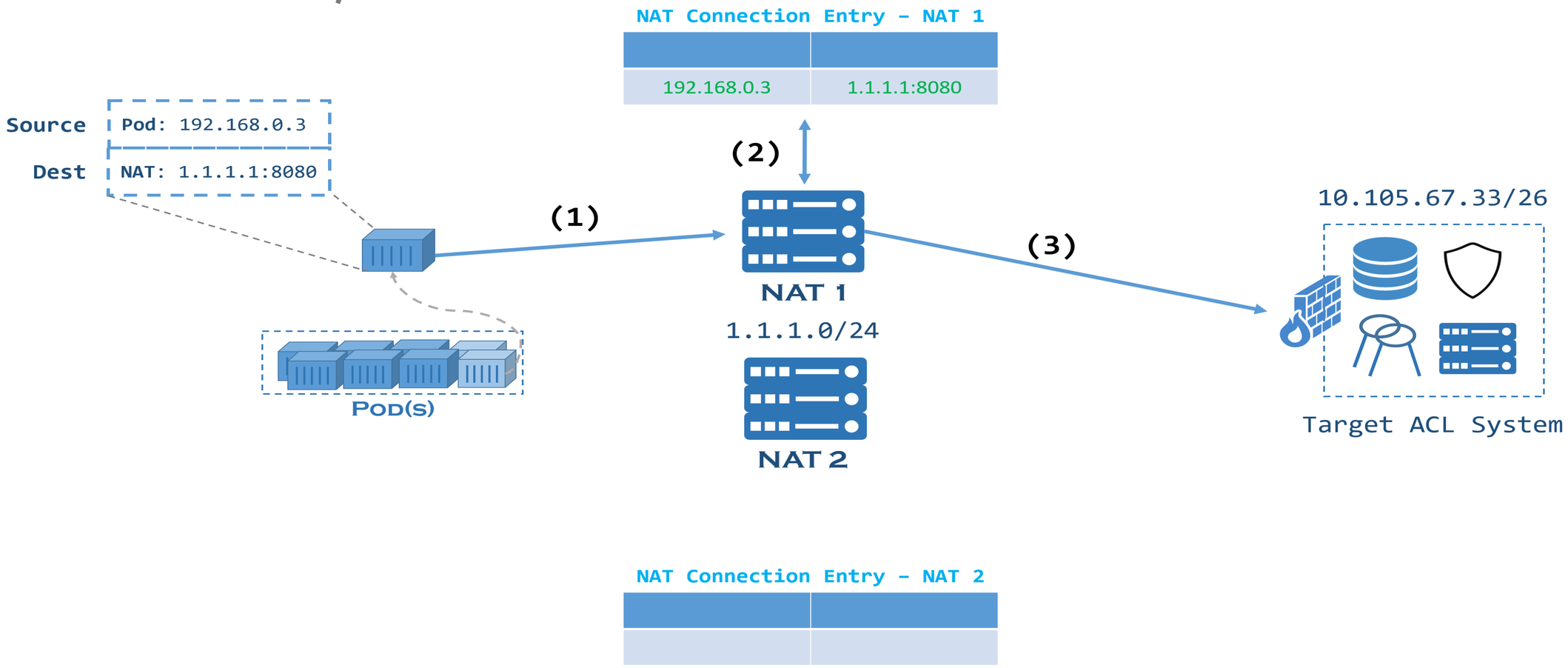
- CPU: 40 Cores, Memory: 256 GB, 10gb NIC
- Payload: 1kbytes, CPS: 850,000 로 측정 결과 에러 없이 약 8.5gbps 의 성능



4.2 Master/Backup NAT 구조

기존 NAT 가 Master/Backup 구조로 운영되는 이유

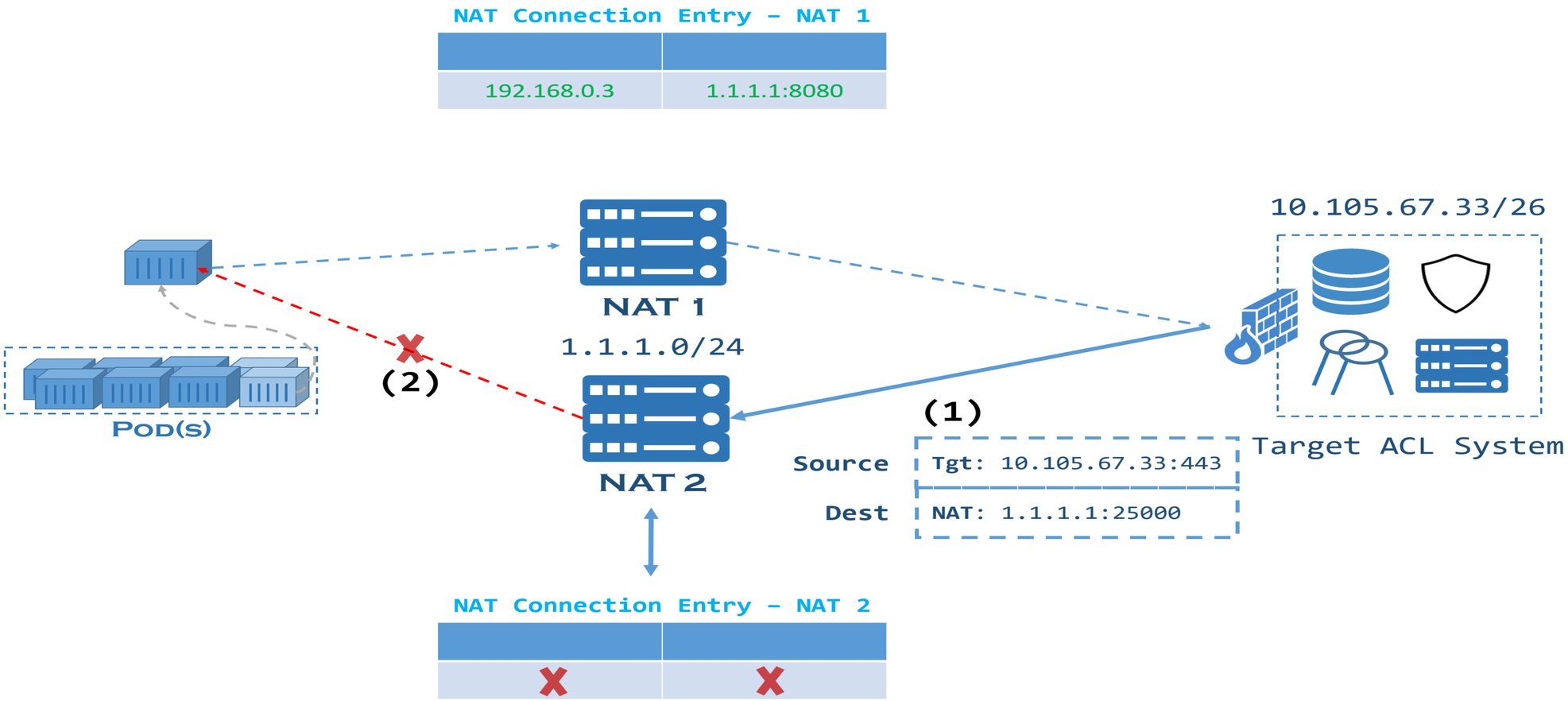
- Connection Entry 동기화 문제



4.2 Master/Backup NAT 구조

기존 NAT 가 Master/Backup 구조로 운영되는 이유

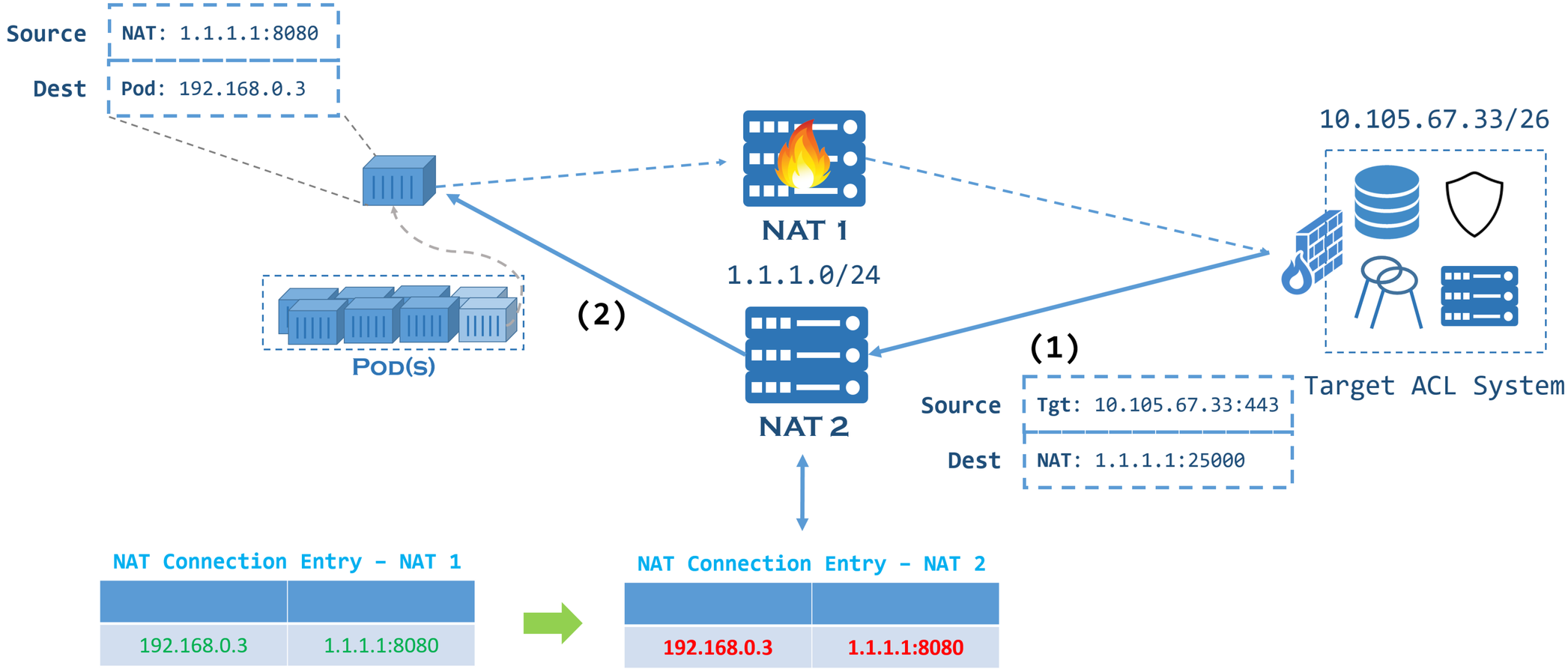
- 서로 다른 NAT Node 로 Packet 이 schedule 될 때 문제가 발생



4.2 Master/Backup NAT 구조

기존 NAT 가 Master/Backup 구조로 운영되는 이유

- Master Down 시 Connection Entry 를 Backup 으로 동기화시키는 방식

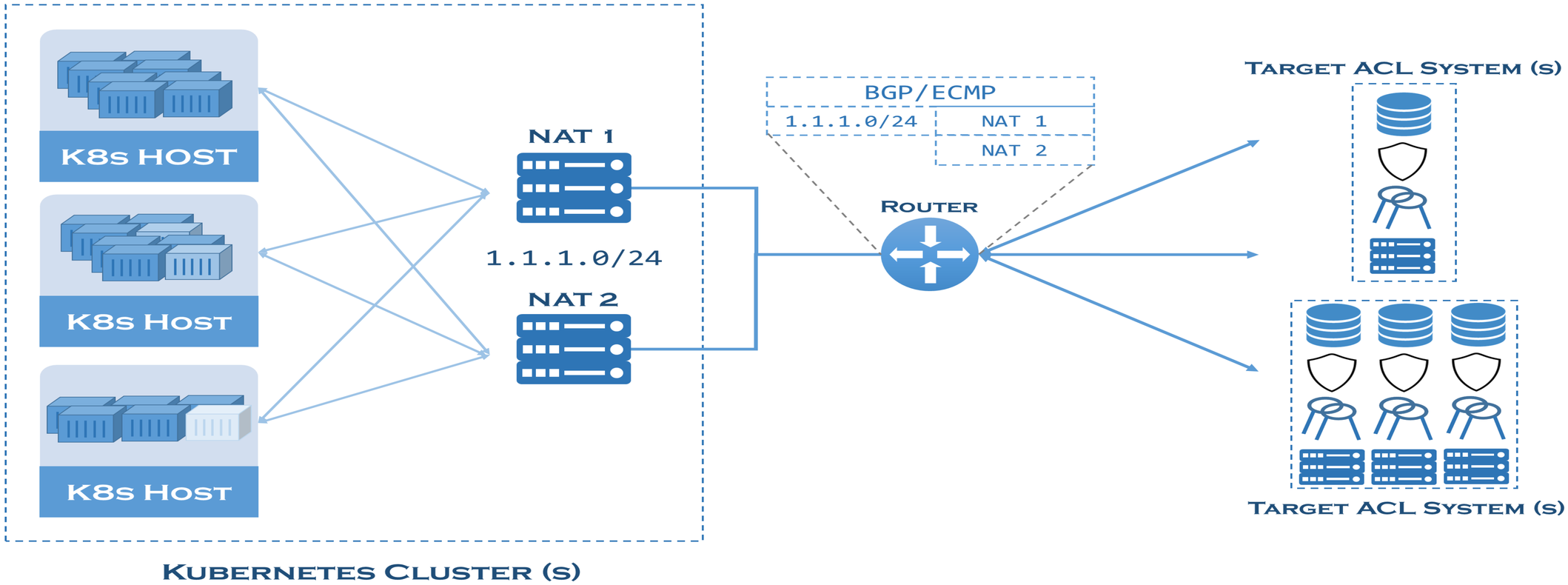


4.3 Active/Active 로 구현한 NAT 시스템

DEVIEW 2020

BGP/ECMP 를 통한 Network Load Balancing

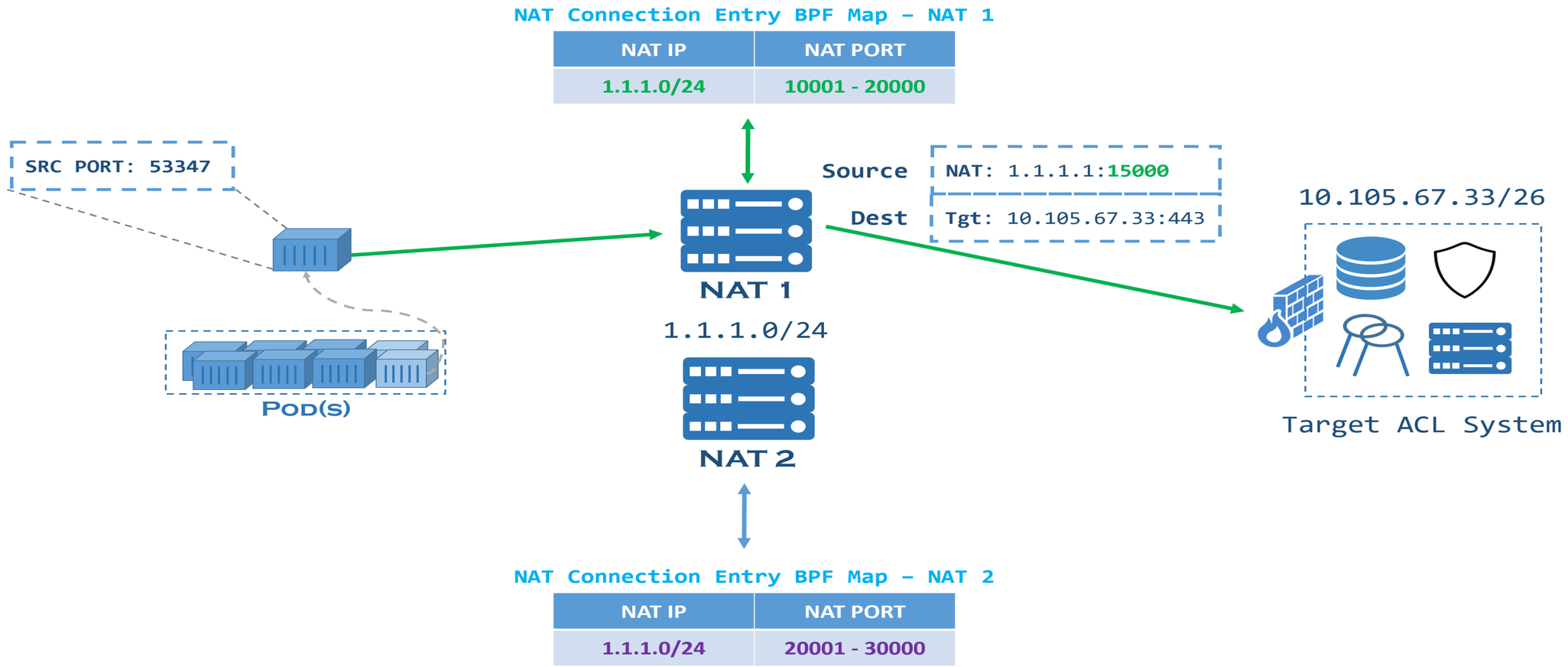
- 각 Router 간 L3 Routing 정보를 동적으로 교환하여 Active/Active 라우팅 구축
- Hash 기반 Scheduling 으로 Network 부하 분산을 제공



4.3 Active/Active 로 구현한 NAT 시스템

NAT Node Port Sharding

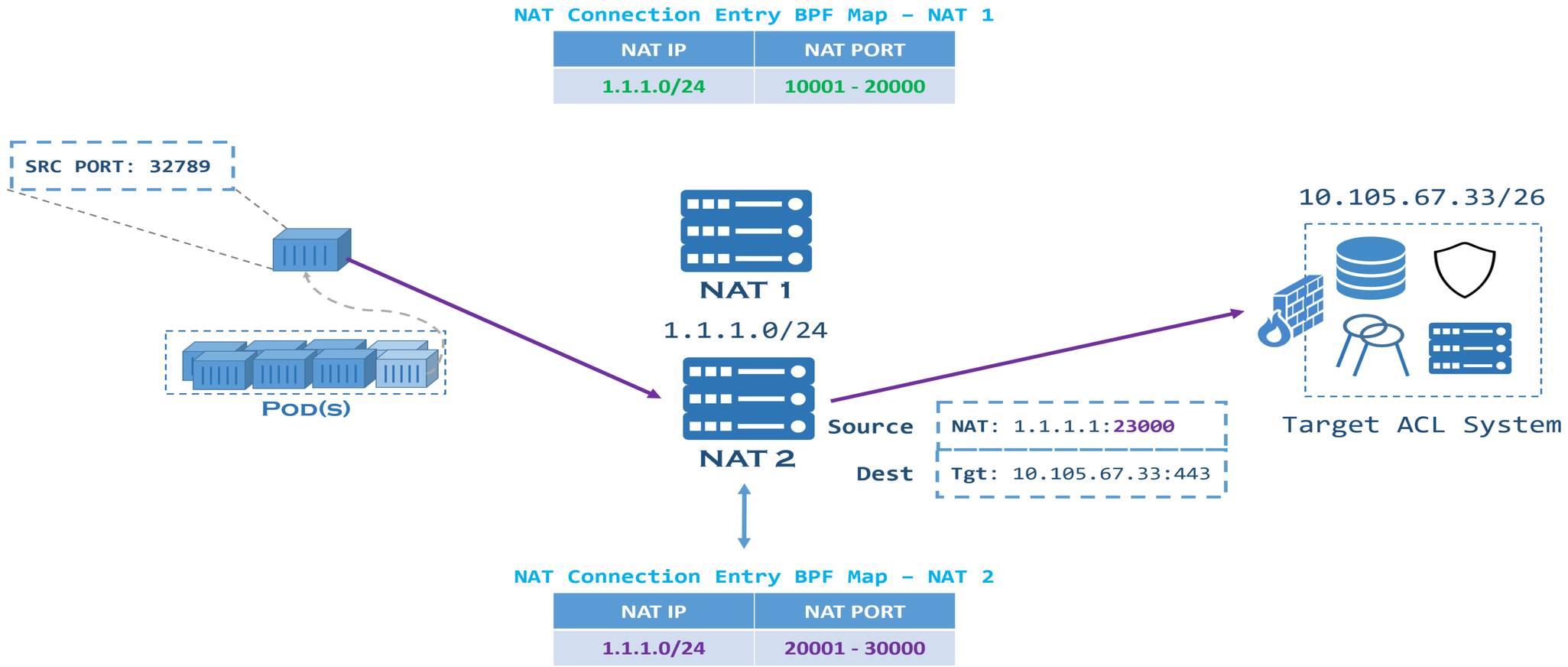
- NAT 에 사용할 수 있는 Port Range 를 각 NAT Node 에 Sharding 시킴



4.3 Active/Active 로 구현한 NAT 시스템

NAT Node Port Sharding

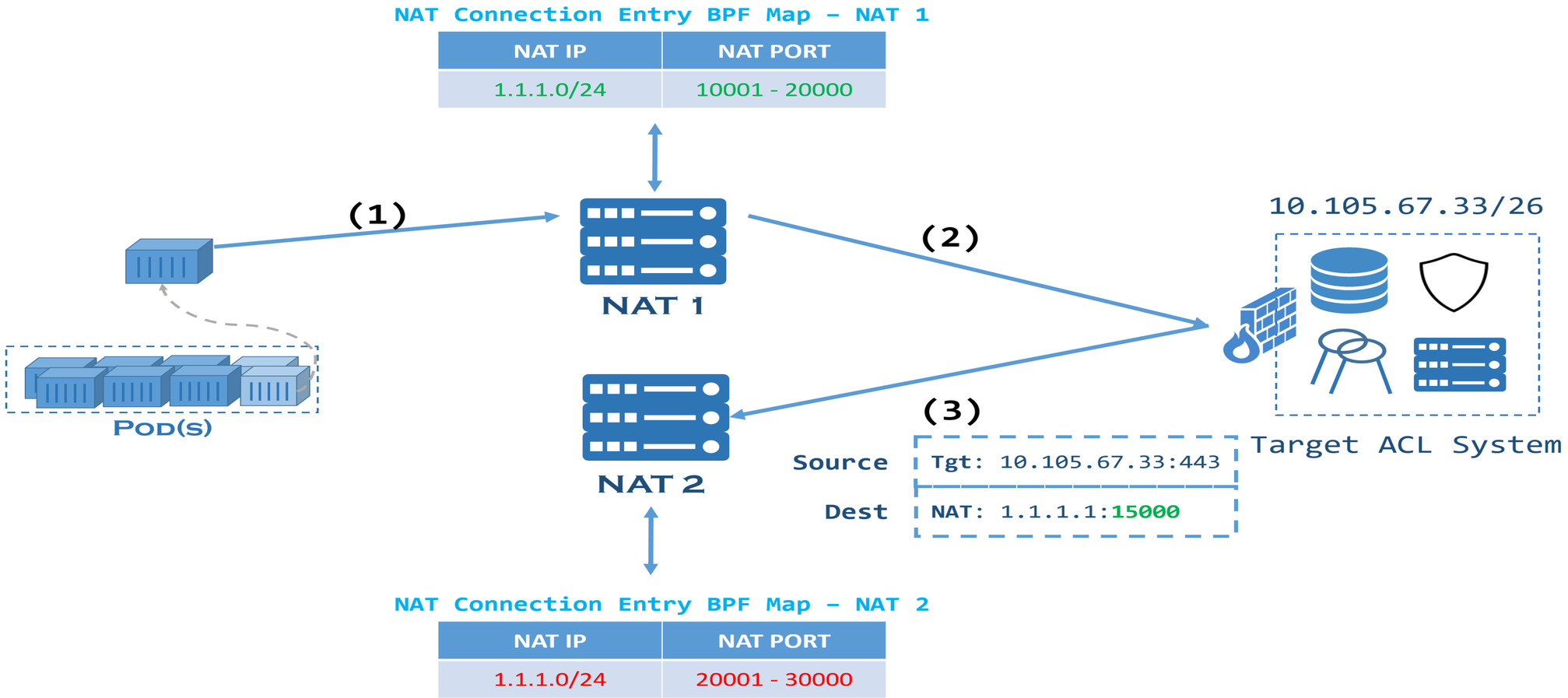
- Connection Table 을 공유하지 못해 발생할 수 있는 Port 충돌 방지



4.3 Active/Active 로 구현한 NAT 시스템

NAT Packet IP Encapsulation & Forwarding

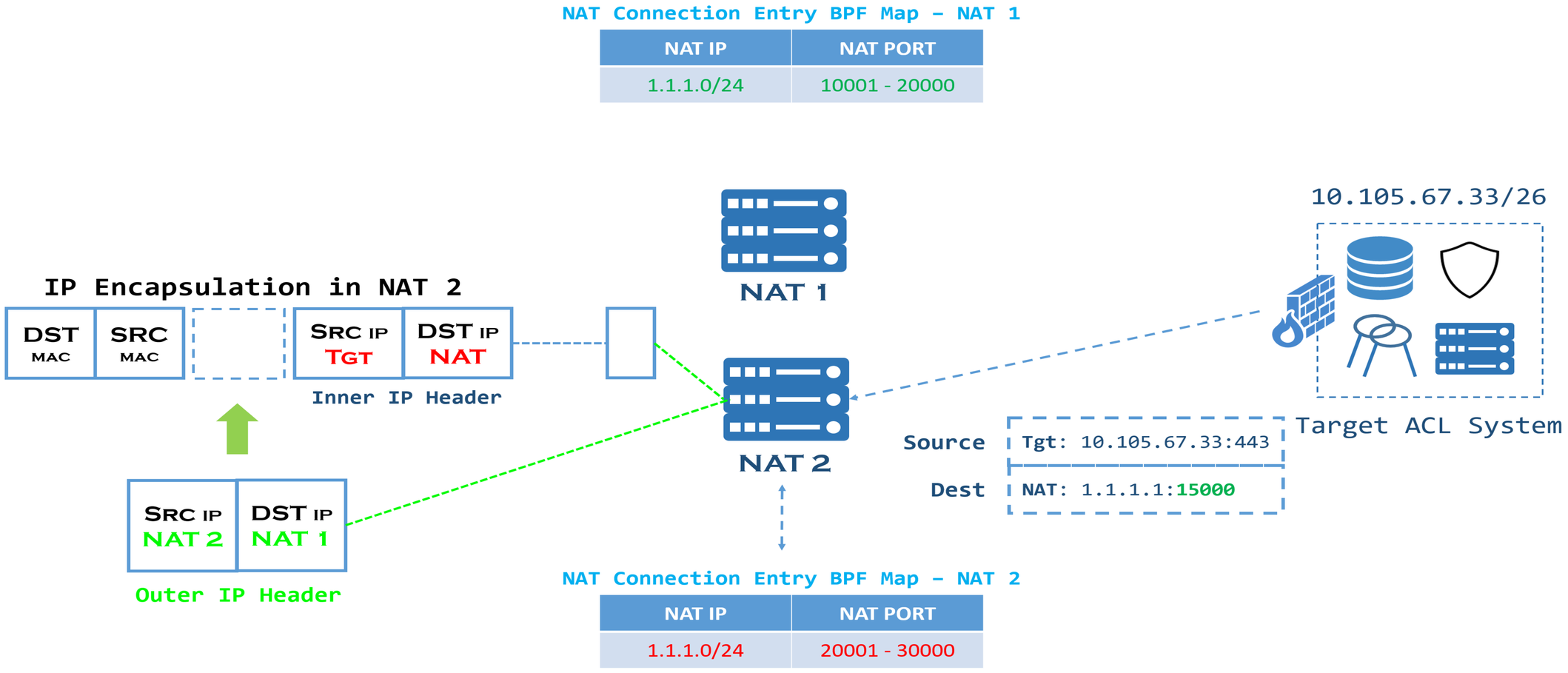
- Connection Entry 가 없는 NAT Node 로 패킷이 scheduling 된 경우



4.3 Active/Active 로 구현한 NAT 시스템

NAT Packet IP Encapsulation & Forwarding

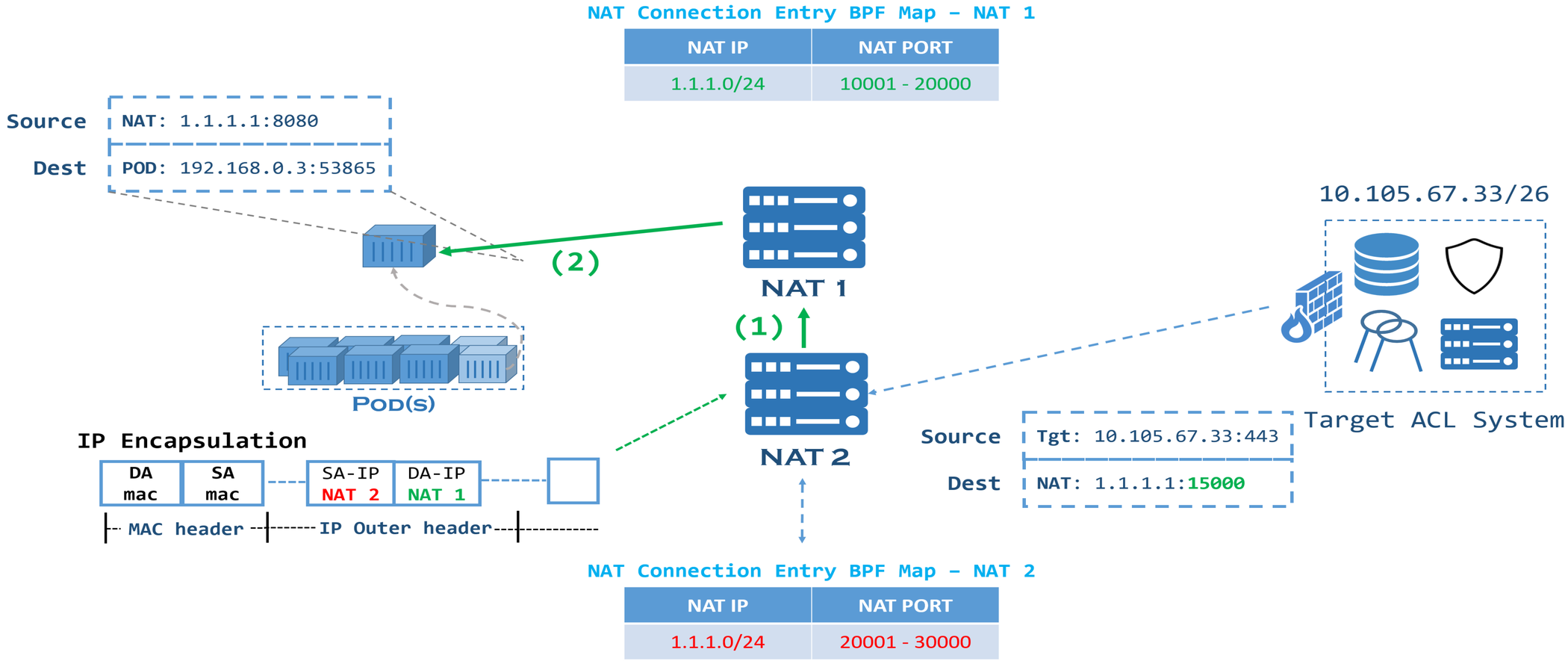
- eBPF/XDP Layer 에서 IP Tunneling 구현, NAT Packet 을 Tunneling 처리



4.3 Active/Active 로 구현한 NAT 시스템

NAT Packet IP Encapsulation & Forwarding

- 해당 Packet 을 처리할 수 있는 Node 로 Forwarding

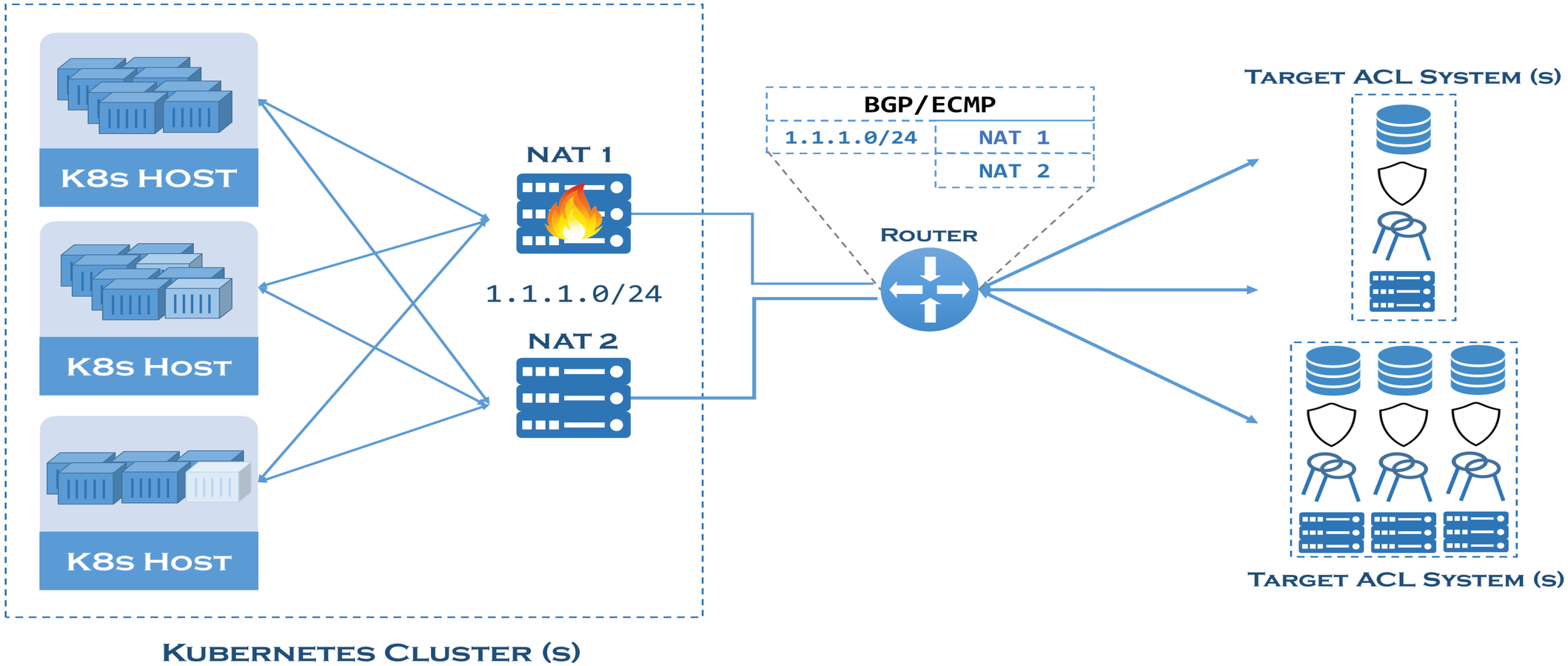


4.4 신뢰할 수 있는 고가용성 NAT 구축하기

DEVIEW 2020

Node Down 시 실시간 서비스 제외 기능 구축

- NAT Node Down

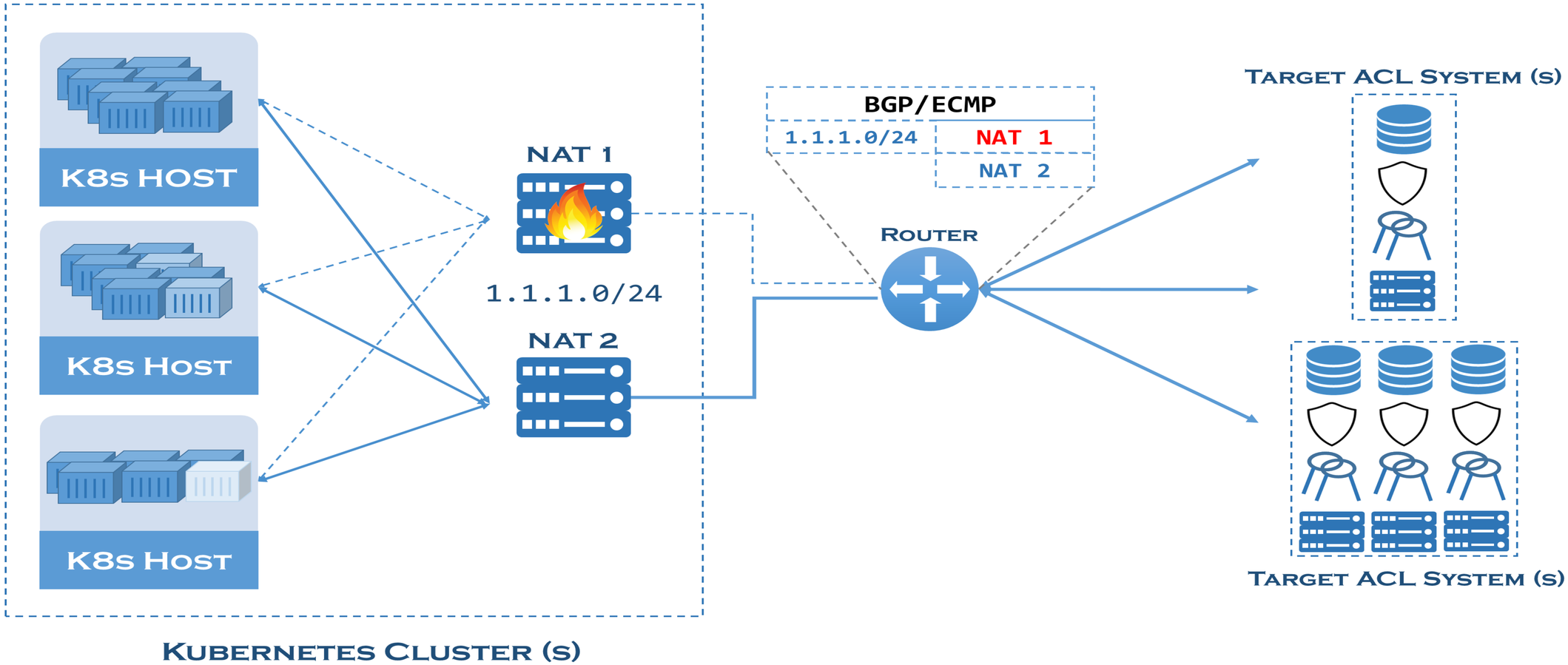


4.4 신뢰할 수 있는 고가용성 NAT 구축하기

DEVIEW 2020

Node Down 시 실시간 서비스 제외 기능 구축

- NAT Node Down -> 수초 내로 BGP/ECMP 에서 해당 Node 제외



4.4 신뢰할 수 있는 고가용성 NAT 구축하기

DEVIEW 2020

BFD Protocol 을 사용한 실시간 서비스 제외 기능 구현

- BFD(양방향 전송 감지) 를 통한 Router <-> NAT Node 간의 시스템 Down 감지

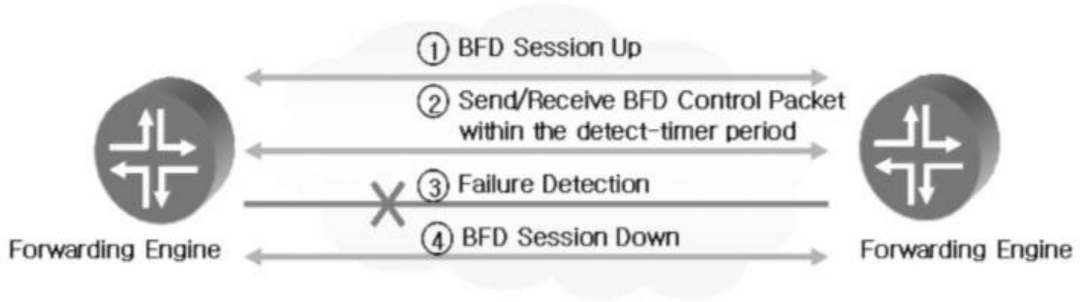


그림 1) BFD Protocol 동작

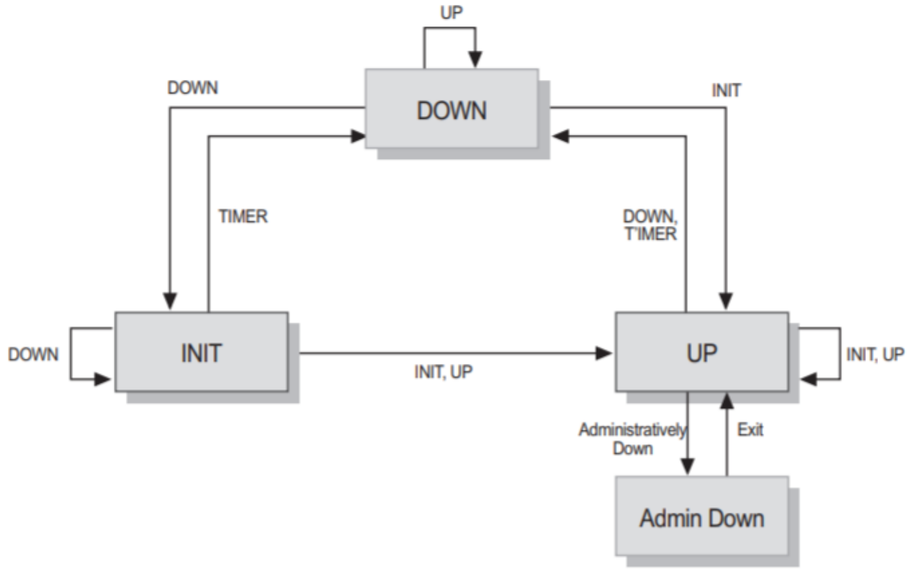
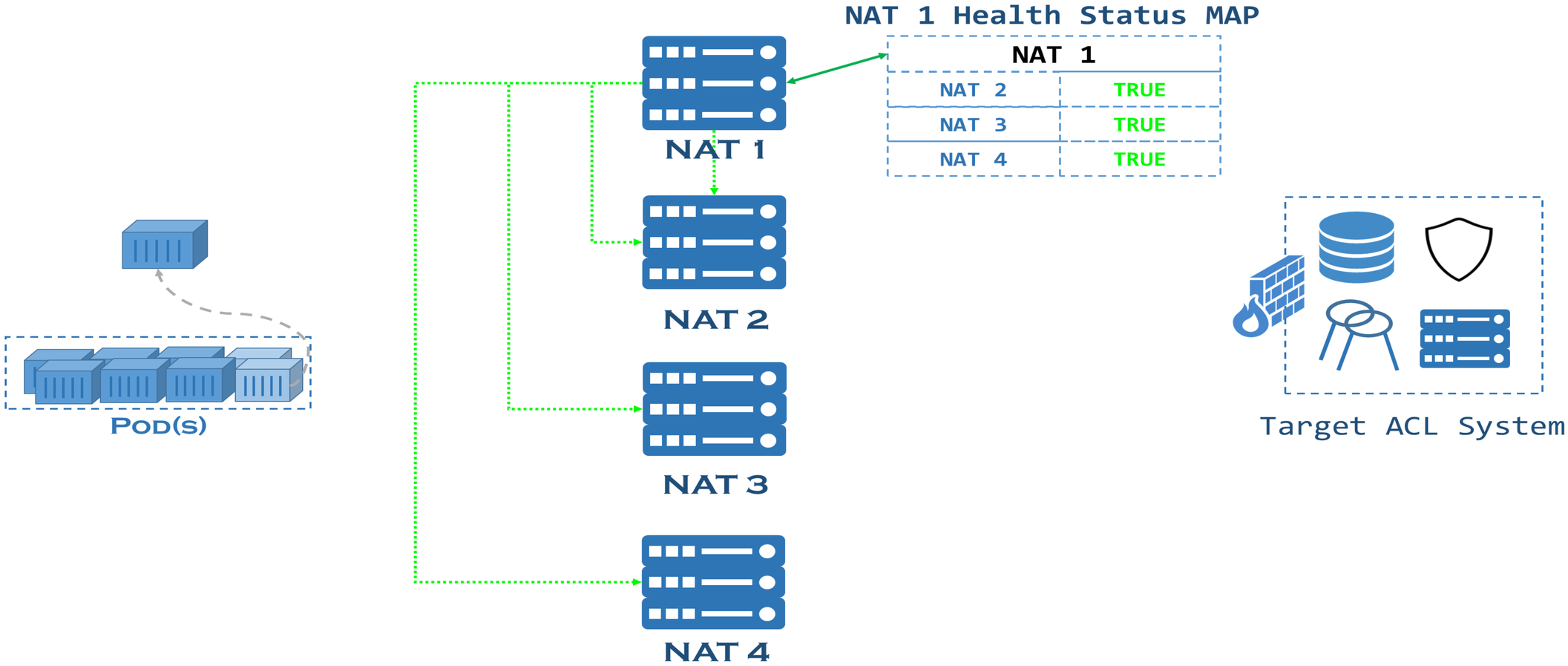


그림 2) BFD State

4.4 신뢰할 수 있는 고가용성 NAT 구축하기

NAT Node 간 Full-Mesh Health Check

- NAT Node 간 Full-Mesh Health Status 를 각 Node 의 BPF Map 에 저장

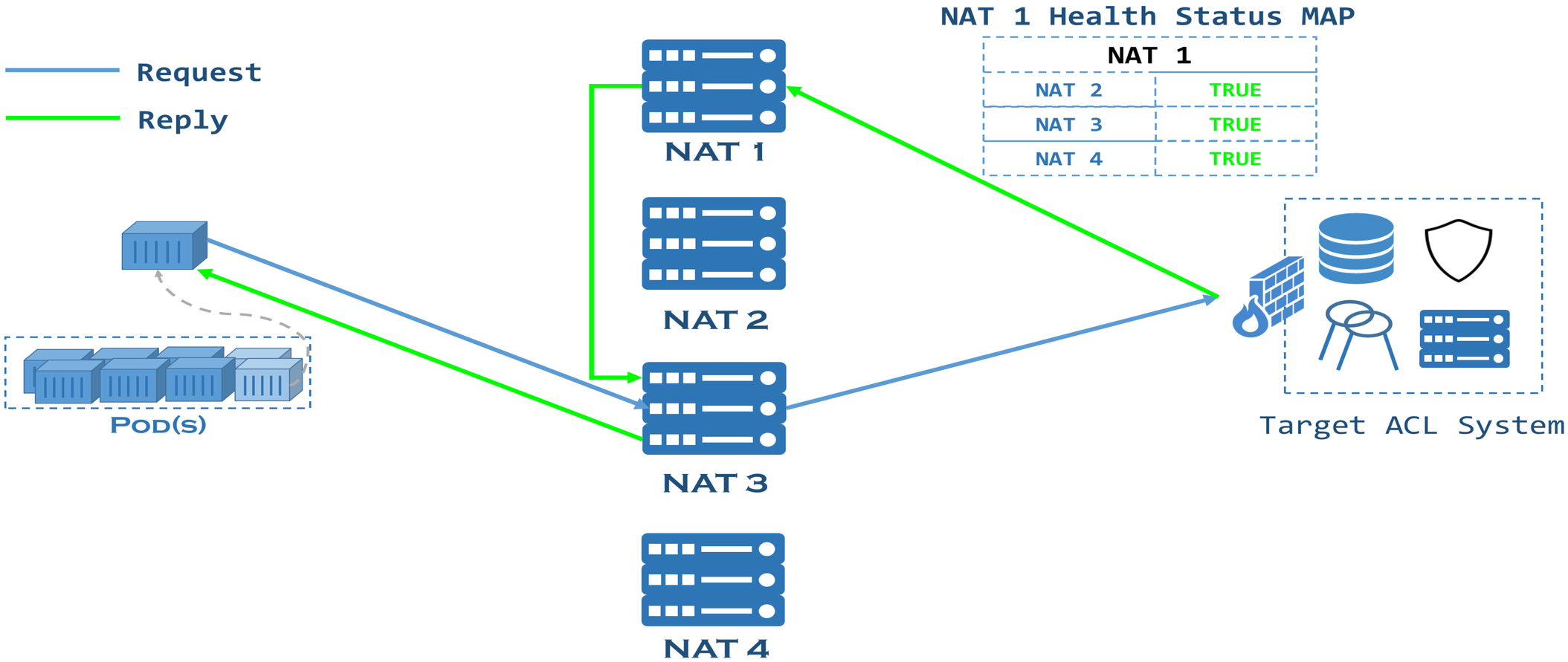


4.4 신뢰할 수 있는 고가용성 NAT 구축하기

DEVIEW 2020

NAT Node 간 Full-Mesh Health Check

- Packet Forwarding 시 Health Status BPF Map 을 참조

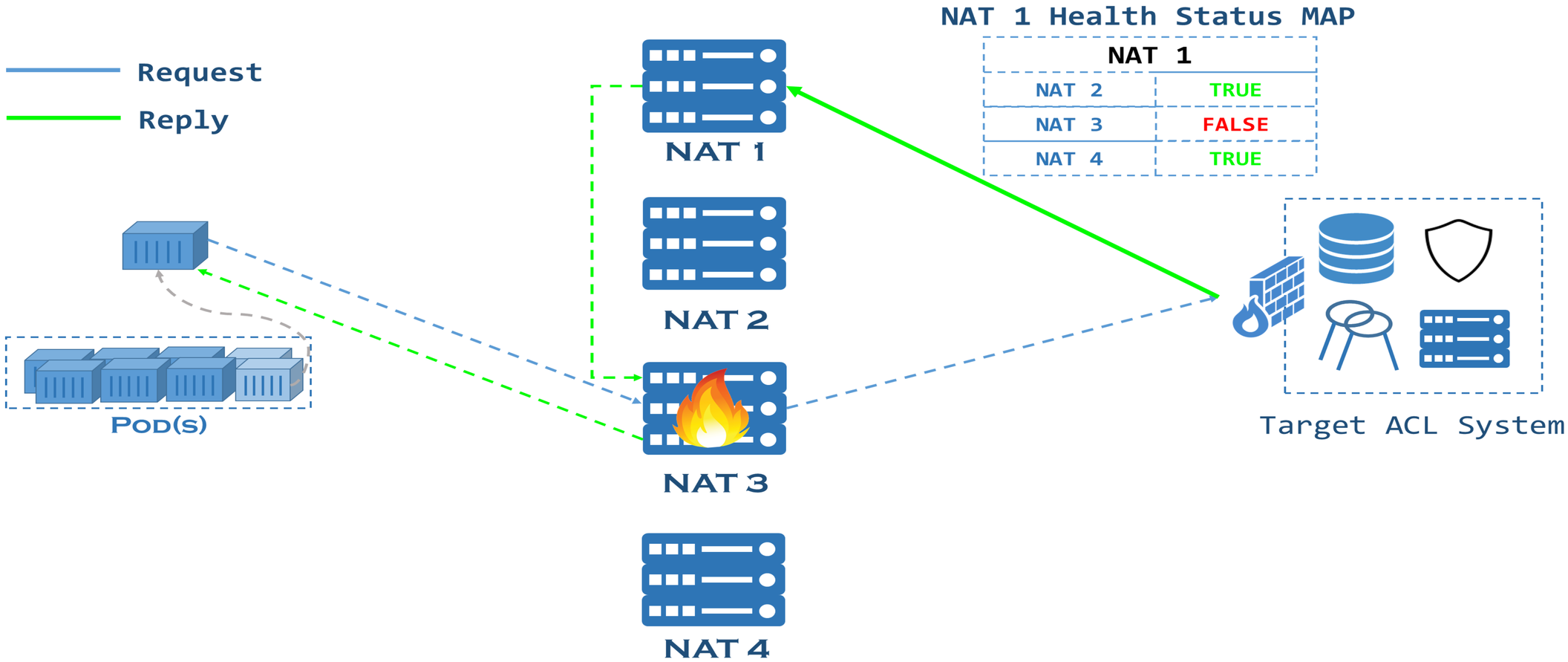


4.4 신뢰할 수 있는 고가용성 NAT 구축하기

DEVIEW 2020

NAT Node 간 Full-Mesh Health Check

- Node Down 등의 이벤트가 발생하면, Health Status 를 False 로 업데이트

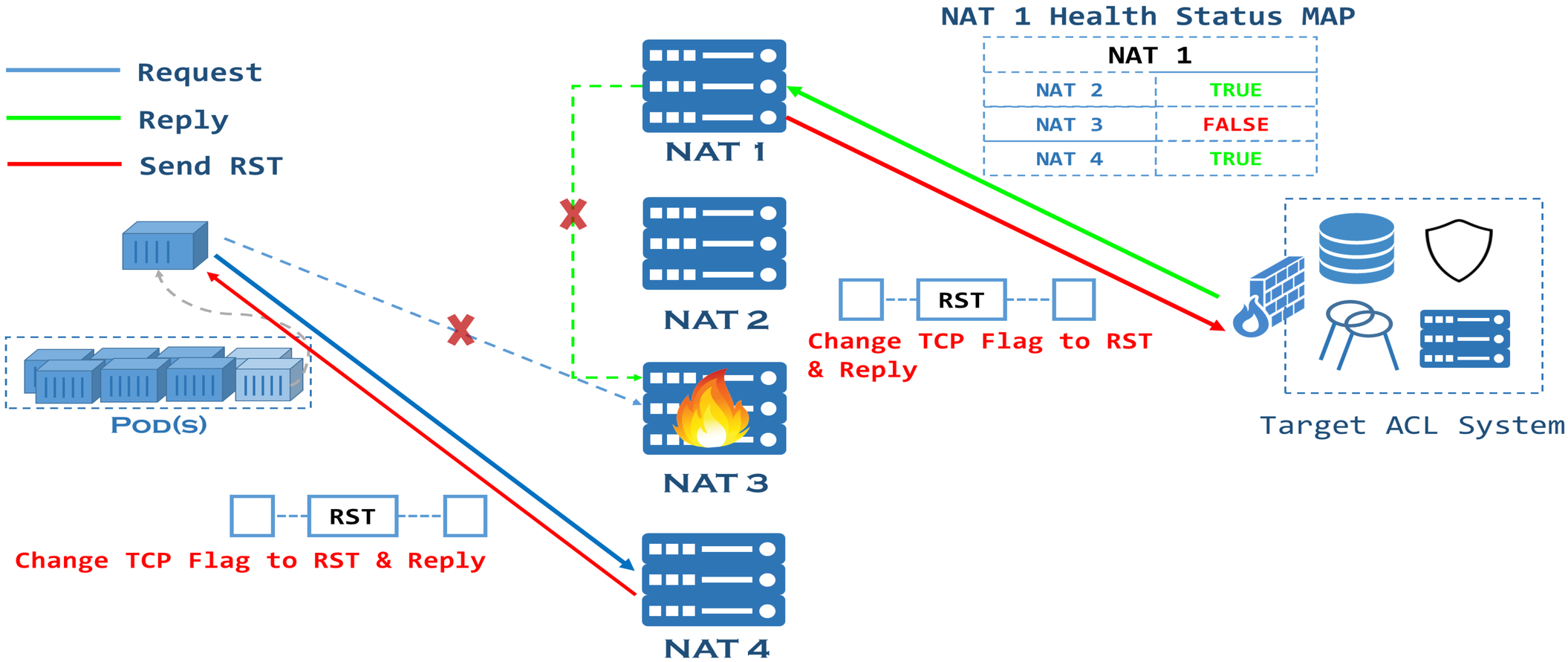


4.4 신뢰할 수 있는 고가용성 NAT 구축하기

DEVIEW 2020

NAT Failover 기능

- Down 된 NAT Node 를 지나던 모든 NAT Connection 에 RST Flag 가 송신됨

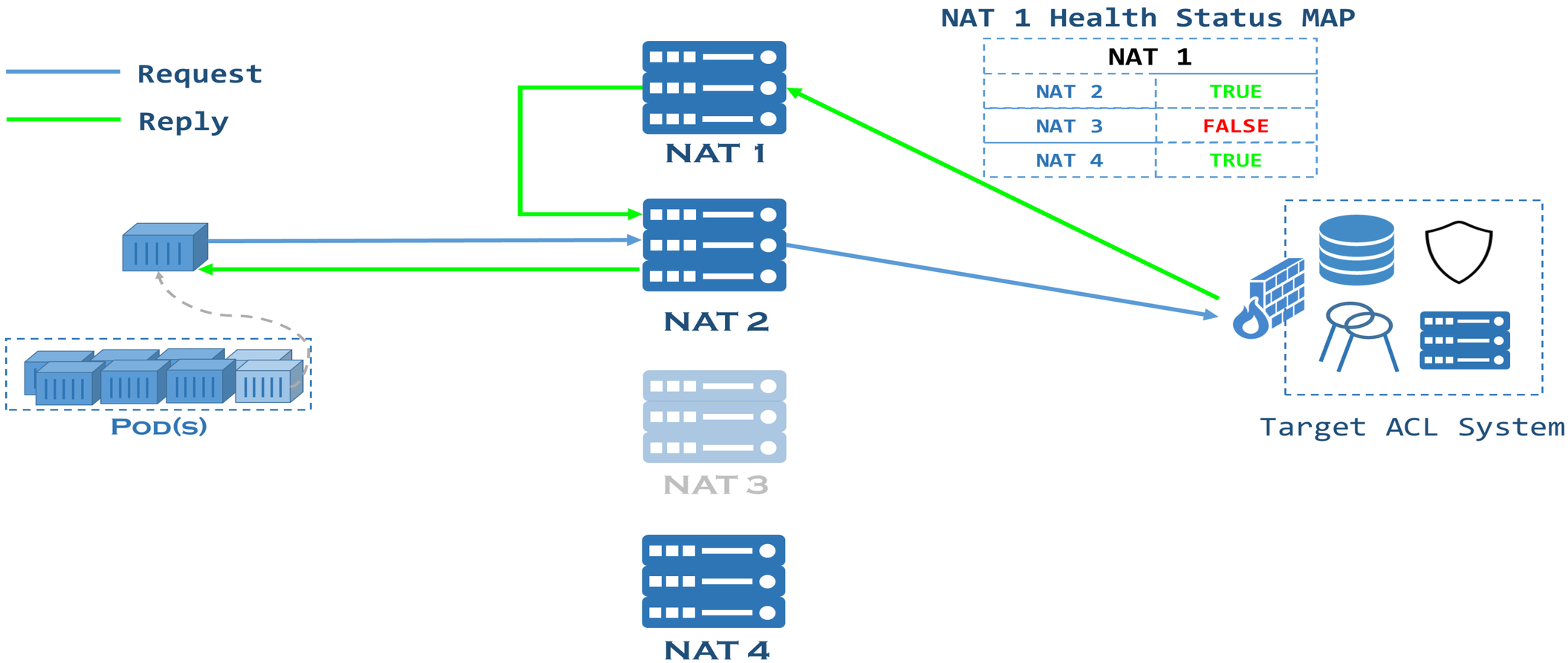


4.4 신뢰할 수 있는 고가용성 NAT 구축하기

DEVIEW 2020

NAT Failover 기능

- RST 을 받은 Pod 은 새로운 NAT Connection 을 열어 통신 재개



4.4 신뢰할 수 있는 고가용성 NAT 구축하기

DEVIEW 2020

High Availability NAT 시스템

- Node Down 감지 후 BFD 에 의해 수초 내로 BGP/ECMP Scheduling 에서 제외
- 해당 Node 로 향하던 Packet 들은 다른 Node 로 Scheduling
- NAT Connection Table 조회 결과에 따라 Reset Packet 을 응답
- Retry 로직을 통해 Failover 후 새로운 Connection 으로 통신 재개
- Client side 에는 RST Flag 수신 시 Retry 로직이 필요

4.5 마치며

K8s 에서의 새로운 NAT 시스템

- 컨테이너 IP 변경에 관계 없이 컨테이너 <-> 타겟 간의 IP ACL 문제 해결
- Multi Tenancy 환경에서의 Namespace & Pod 별 권한 및 접근 제어
- eBPF/XDP 기반의 고성능 Networking Performance 제공
- BGP/ECMP 를 사용해 효율적으로 NAT Traffic 을 Load Balancing
- Active/Active 구현을 위한 NAT Connection Table 관리, Port Sharding, IP Encapsulation & Forwarding 기능
- BFD 및 Failover 기능 개발을 통해 High Availability NAT 시스템 제공

4.5 마치며

eBPF & XDP 기술

- 차세대 Linux Networking 으로 손색 없는 Performance 를 보여줌
- 오래된 기술이지만, 현재까지도 굉장히 가파르게 발전하고 있음
- NAT Kernel Version: v5.4.6
- 여러 기업에서 eBPF 기반의 자체 시스템을 구축 중
- Networking(Load Balancing, NAT, Firewall)
- CPU/MEM/File System/Networking 등 다양한 Kernel Layer 에서의 Latency Tracing, Monitoring 등

4.5 마치며

Further Work

- 컨테이너 클러스터를 위한 NAT 는 eBPF/XDP 기반으로 구현되었지만, Load Balancer 는 아직 IPVS(Netfilter) 기반의 SWLB 로 운영하고 있음
- 검증된 eBPF/XDP 기술로 DSR/Masquerading 이 모두 가능한 LB 를 구현
- End-To-End 뿐만 아니라 Gateway 로서의 NAT 방식 또한 추가적으로 검토 중
- 클러스터 별 시스템이 아닌, IDC Zone 별 Networking 시스템을 구축
- Kubernetes 와 분리된 독립적인 외부 네트워킹 시스템

Thank You

Q & A