

# Linux Kernel - BPF / XDP

KossLab 유태희, 송태웅

# BPF 란 ?

1. Berkeley Packet Filter since 1992
2. Kernel Infrastructure

# BPF 란 ?

1. Berkeley Packet Filter since 1992

## 2. Kernel Infrastructure

- **Interpreter** in-kernel virtual machine
- **Hook points** in-kernel callback point
- **Map**
- **Helper**

# BPF 란 ?

“Safe dynamic programs and tools”



"런타임중 안전하게 커널코드를 삽입하는 기술"

# BPF Infrastructure:

## 안전한 code injection 작전

- 1) Native 머신코드 대신 BPF instruction 을 활용하자
- 2) Verifier 를 통해 위험요소를 미리검사하자
- 3) (기존)커널함수가 필요할때 Helper 함수를 통해서만 호출하자

# BPF Infrastructure:

## 안전한 code injection 작전

- 1) Native 머신코드 대신 **BPF instruction** 을 활용하자

# BPF Infrastructure:

## 안전한 code injection 작전

2) Verifier 를 통해 위험요소를 미리검사하자

# BPF Infrastructure:

## 안전한 code injection 작전

3) (기존) 커널 함수가 필요할 때 Helper 함수를 통해서만 호출하자



# BPF Infrastructure:

## 안전한 code injection 작전

- Kernel** += **BPF Interpreter** in-kernel virtual machine
- + **Verifier**
  - + **BPF Helper 함수 추가** leveraging kernel func
  - + **BPF syscall** prog/map: loading & attaching 등

# BPF Instruction set:

1) BPF = classic BPF:10% + x86:70% + arm64:25% + risc:5%

2) Instruction encoding 사이즈 고정

(for high interpreter speed)

3) x86 주니어 Instruction set 'simplified x86'

(참고: PLUMgrind의 x86 bytecode verifier 실패)

4) 간소화 -> 위험을 예측하고 예방하기 수월

(Verifier를 통한 loop, memory access 범위 점검 등)

5) Architecture-independent

# BPF Instruction set:

immediate:32	offset:16	src:4	dst:4	opcode:8
--------------	-----------	-------	-------	----------

```
$ cat include/uapi/linux/bpf.h
```

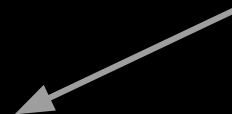
```
[...]
```

```
struct bpf_insn {  
    __u8    code;           /* opcode */  
    __u8    dst_reg:4;      /* dest register */  
    __u8    src_reg:4;      /* source register */  
    __s16   off;            /* signed offset */  
    __s32   imm;            /* signed immediate constant */  
};
```

```
[...]
```

# BPF Instruction set:

immediate:32	offset:16	src:4	dst:4	opcode:8
--------------	-----------	-------	-------	----------



eBPF: include/uapi/linux/bpf.h

cBPF: include/uapi/linux/bpf\_common.h

# BPF Instruction set:

immediate:32	offset:16	src:4	dst:4	opcode:8
--------------	-----------	-------	-------	----------

LD/ST 계열:  
0x00 ~ 0x03

ALU/JMP 계열:  
0x04 ~ 0x07

class:4 + LD/ST fields:4  
+ ALU/JUM fields:4

eBPF: include/uapi/linux/bpf.h

cBPF: include/uapi/linux/bpf\_common.h

# BPF Instruction set:

immediate:32	offset:16	src:4	dst:4	opcode:8
--------------	-----------	-------	-------	----------

LD/ST 계열:  
0x00 ~ 0x03

ALU/JMP 계열:  
0x04 ~ 0x07

class:4 + LD/ST fields:4  
+ ALU/JUM fields:4

eBPF: include/uapi/linux/bpf.h  
cBPF: include/uapi/linux/bpf\_common.h

# BPF Instruction set:

```
struct bpf_insn prog[] = {
    BPF_MOV64_REG(BPF_REG_6, BPF_REG_1),
    BPF_LD_ABS(BPF_B, ETH_HLEN + offsetof(struct iphdr, protocol) /* R0 = ip->proto */),
    BPF_STX_MEM(BPF_W, BPF_REG_10, BPF_REG_0, -4), /* *(u32 *) (fp - 4) = r0 */
    BPF_MOV64_REG(BPF_REG_2, BPF_REG_10),
    BPF_ALU64_IMM(BPF_ADD, BPF_REG_2, -4), /* r2 = fp - 4 */
    BPF_LD_MAP_FD(BPF_REG_1, map_fd),
    BPF_RAW_INSN(BPF_JMP | BPF_CALL, 0, 0, 0, BPF_FUNC_map_lookup_elem),
    BPF_JMP_IMM(BPF_JEQ, BPF_REG_0, 0, 2),
    BPF_MOV64_IMM(BPF_REG_1, 1), /* r1 = 1 */
    BPF_RAW_INSN(BPF_STX | BPF_XADD | BPF_DW, BPF_REG_0, BPF_REG_1, 0, 0), /* xadd r0 += r1 */
    BPF_MOV64_IMM(BPF_REG_0, 0), /* r0 = 0 */
    BPF_EXIT_INSN(),
};
```

[https://git.kernel.org/pub/scm/linux/kernel/git/bpf/bpf.git/tree/samples/bpf/sock\\_example.c](https://git.kernel.org/pub/scm/linux/kernel/git/bpf/bpf.git/tree/samples/bpf/sock_example.c)

# BPF Helper 함수:

```
$ grep BPF_CALL
```

```
kernel/bpf/helpers.c:
```

```
BPF_CALL_2(bpf_map_lookup_elem, struct bpf_map *, map, void *, key)
```

```
BPF_CALL_4(bpf_map_update_elem, struct bpf_map *, map, void *, key,
```

```
[...]
```

```
kernel/trace/bpf_trace.c:
```

```
BPF_CALL_2(bpf_override_return, struct pt_regs *, regs, unsigned long, rc)
```

```
BPF_CALL_3(bpf_probe_read, void *, dst, u32, size, const void *, unsafe_ptr)
```

```
BPF_CALL_3(bpf_probe_write_user, void *, unsafe_ptr, const void *, src,
```

```
BPF_CALL_5(bpf_trace_printk, char *, fmt, u32, fmt_size, u64, arg1,
```

```
[...]
```

```
net/core/filter.c:
```

```
BPF_CALL_1(bpf_skb_get_pay_offset, struct sk_buff *, skb)
```

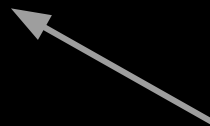
```
BPF_CALL_3(bpf_skb_get_nlattr, struct sk_buff *, skb, u32, a, u32, x)
```

```
[...]
```



# BPF as a kernel subproject

**“Safe dynamic programs and tools”**



```
$ cat MAINTAINERS | grep -A 3 BPF
BPF (Safe dynamic programs and tools)
M:   Alexei Starovoitov <ast@kernel.org>
M:   Daniel Borkmann <daniel@iogearbox.net>
L:   netdev@vger.kernel.org
[...]
```

# BPF as a kernel subproject

## “Safe dynamic programs and tools”

```
$ cat MAINTAINERS | grep -A 27 BPF
BPF (Safe dynamic programs and tools)
[...]
F:   arch/x86/net/bpf_jit*
[...]
F:   kernel/bpf/
F:   kernel/trace/bpf_trace.c
[...]
F:   net/core/filter.c
F:   net/sched/act_bpf.c
F:   net/sched/cls_bpf.c
[...]
F:   samples/bpf/
F:   tools/bpf/
F:   tools/lib/bpf/
F:   tools/testing/selftests/bpf/
```

# BPF as a kernel subproject

## “Safe dynamic programs and tools”

JIT 지원 arch:

x86,  
arm, arm64  
sparc,  
s390,  
powerpc, mips

```
$ cat MAINTAINERS | grep -A 27 BPF
BPF (Safe dynamic programs and tools)
[...]
F: arch/x86/net/bpf_jit*
[...]
F: kernel/bpf/
F: kernel/trace/bpf_trace.c
[...]
F: net/core/filter.c
F: net/sched/act_bpf.c
F: net/sched/cls_bpf.c
[...]
F: samples/bpf/
F: tools/bpf/
F: tools/lib/bpf/
F: tools/testing/selftests/bpf/
```


# BPF as a kernel subproject

## “Safe dynamic programs and tools”

**BPF core:**

**Syscall,  
Interpreter,  
Verifier,  
Generic Helpers,  
Maps,  
...**

```
$ cat MAINTAINERS | grep -A 27 BPF
BPF (Safe dynamic programs and tools)
[...]
F:  arch/x86/net/bpf_jit*
[...]
F:  kernel/bpf/
F:  kernel/trace/bpf_trace.c
[...]
F:  net/core/filter.c
F:  net/sched/act_bpf.c
F:  net/sched/cls_bpf.c
[...]
[...]
F:  samples/bpf/
F:  tools/bpf/
F:  tools/lib/bpf/
F:  tools/testing/selftests/bpf/
```



# BPF as a kernel subproject

## “Safe dynamic programs and tools”

```
$ cat MAINTAINERS | grep -A 27 BPF
BPF (Safe dynamic programs and tools)
[...]
F:  arch/x86/net/bpf_jit*
[...]
F:  kernel/bpf/
F:  kernel/trace/bpf_trace.c
[...]
F:  net/core/filter.c
F:  net/sched/act_bpf.c
F:  net/sched/cls_bpf.c
[...]
[...]
F:  samples/bpf/
F:  tools/bpf/
F:  tools/lib/bpf/
F:  tools/testing/selftests/bpf/
```

Hook points,  
Specific Helpers

...

For cBPF, ...

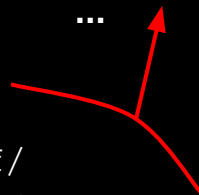
# BPF as a kernel subproject

## “Safe dynamic programs and tools”

```
$ cat MAINTAINERS | grep -A 27 BPF
BPF (Safe dynamic programs and tools)
[...]
F:   arch/x86/net/bpf_jit*
[...]
F:   kernel/bpf/
F:   kernel/trace/bpf_trace.c
[...]
F:   net/core/filter.c
F:   net/sched/act_bpf.c
F:   net/sched/cls_bpf.c
[...]
```

```
[...]
F:   samples/bpf/
F:   tools/bpf/
F:   tools/lib/bpf/
F:   tools/testing/selftests/bpf/
```

**bpf loading(lib),  
bpf tool,  
test codes,  
samples,  
...**

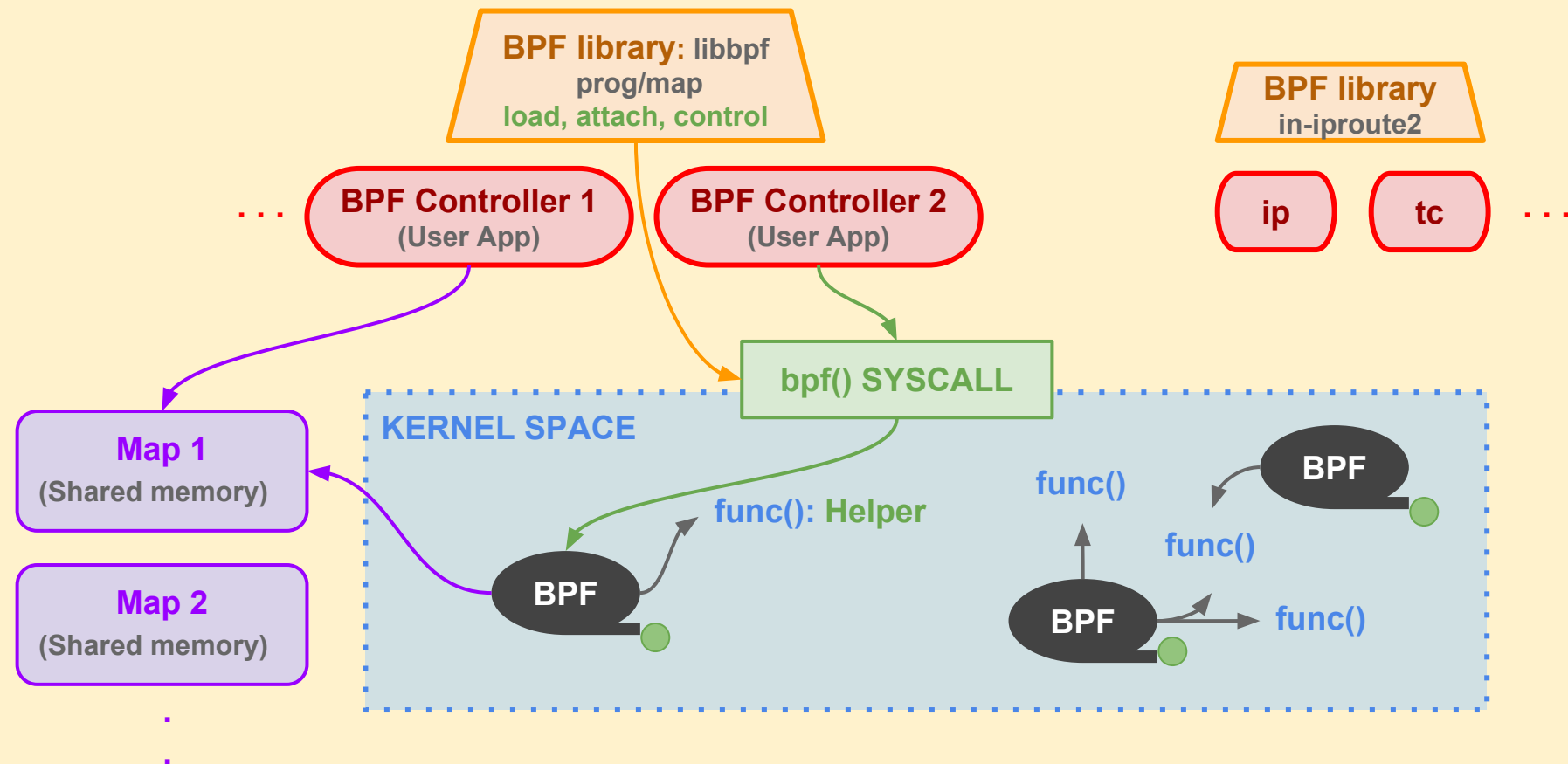


# BPF Infrastructure:

## BPF 프로그램 활용을 위한 지원

- 1) Hook points in-kernel callback point
- 2) Map user-to-kernel shared memory
- 3) helper를 통한 커널함수호출 leveraging
- 4) Object pinning `/sys/fs/bpf/...`

# BPF Architecture:





**XDP**

**iptables**는 충분히 빠른가요?

**iptables**는 왜 느릴까요?

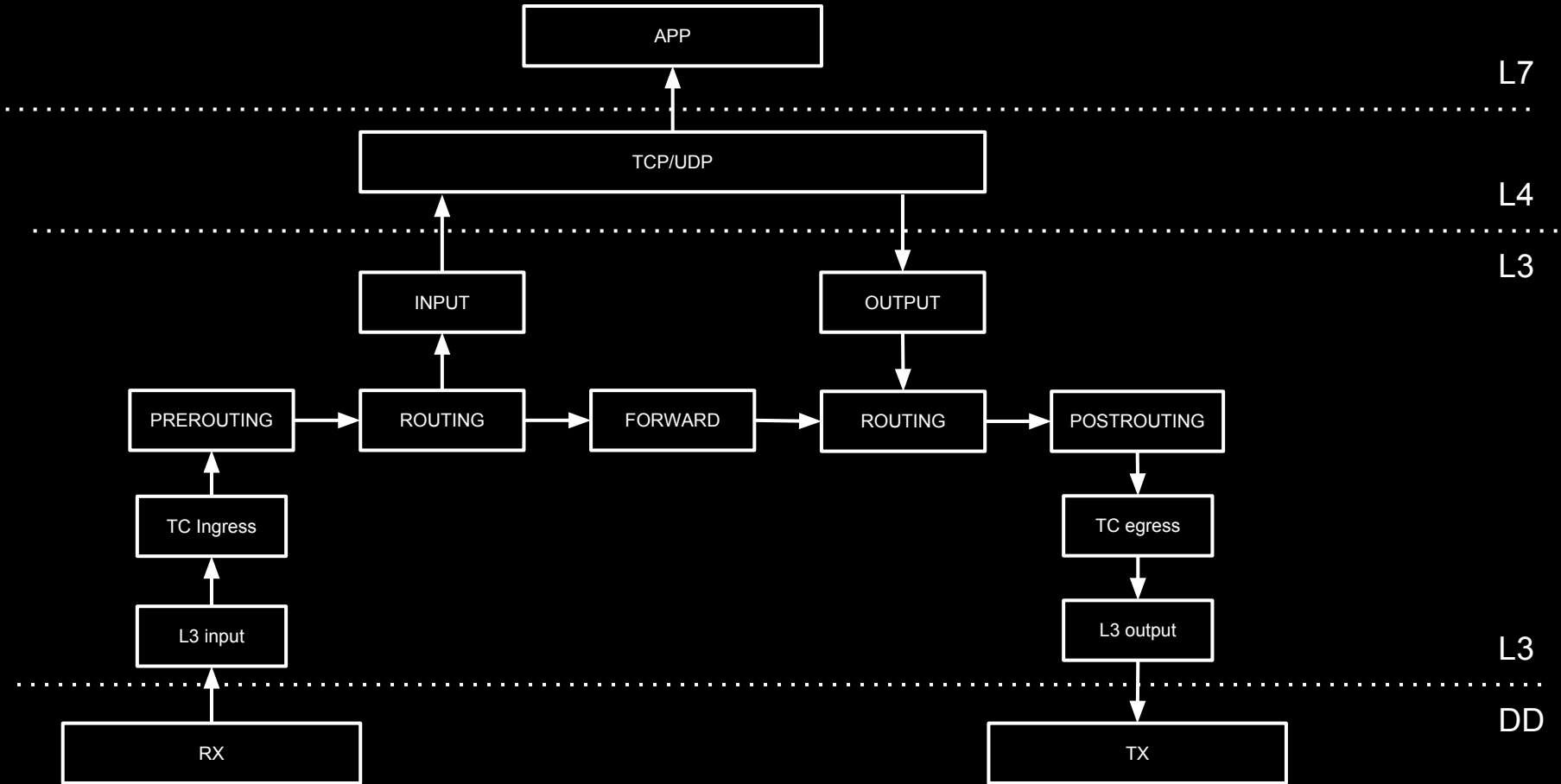
iptables의 정책을 튜닝해본적 있으신가요?

**XDP**

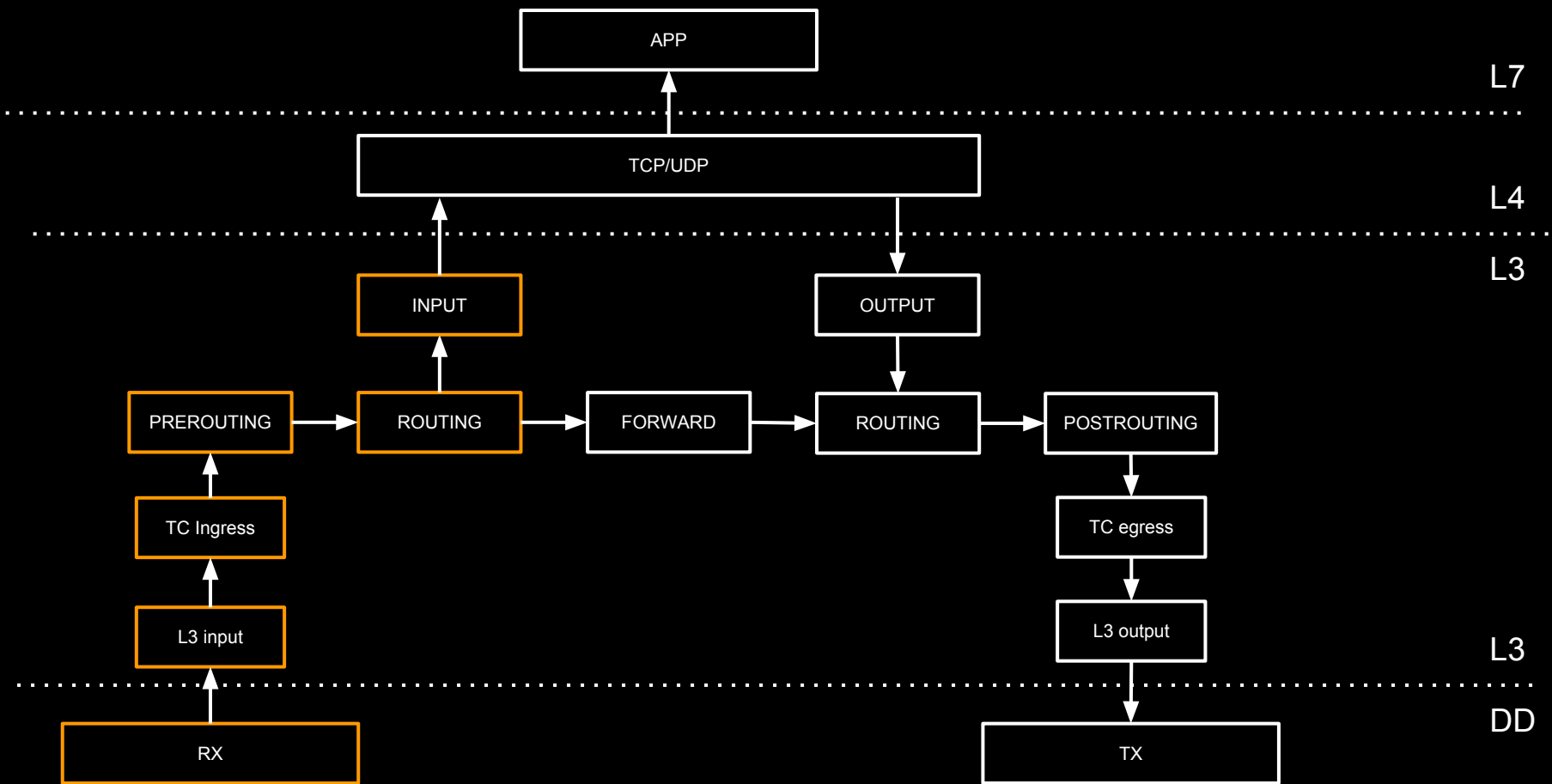
**(eXpress Data Path)**

**XDP == FAST PATH**

# NORMAL PATH

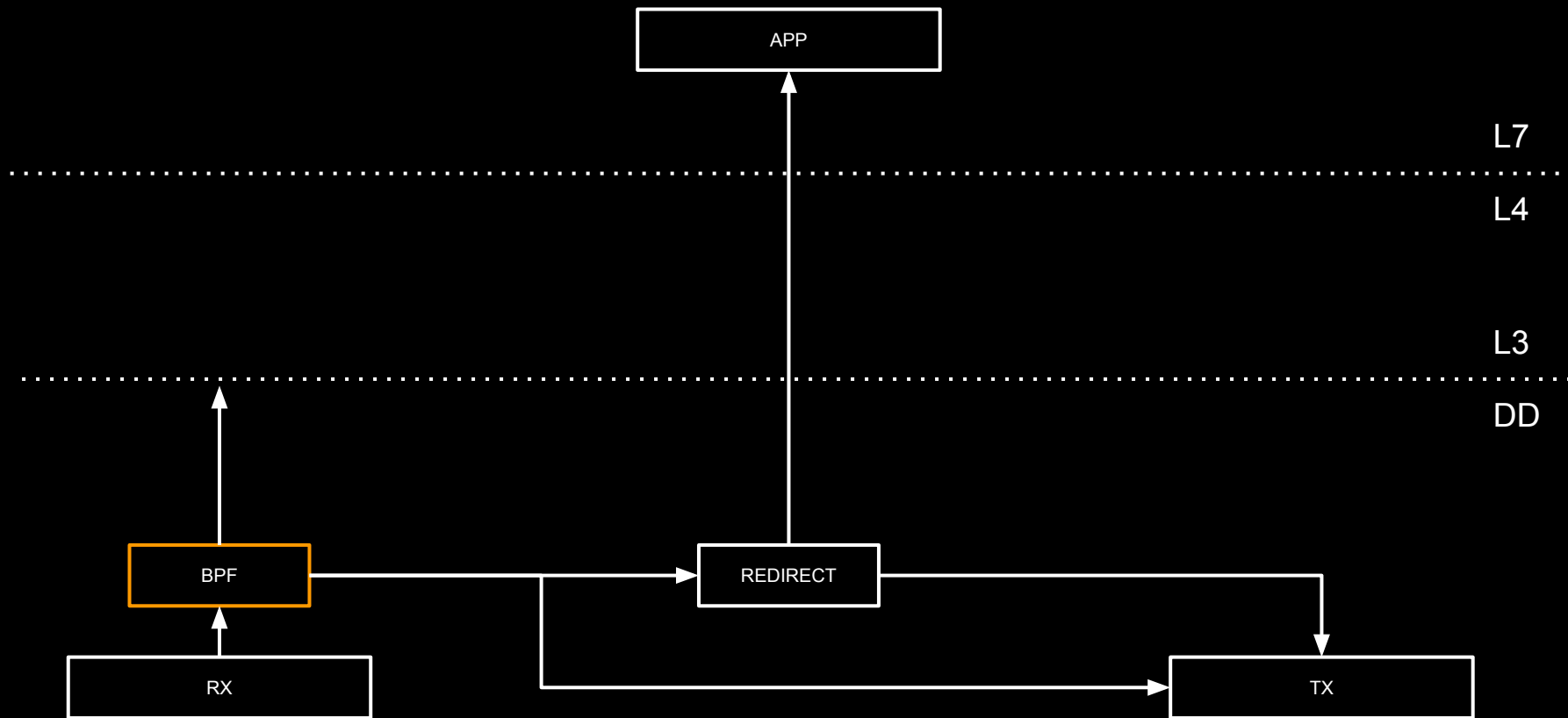


# NORMAL PATH

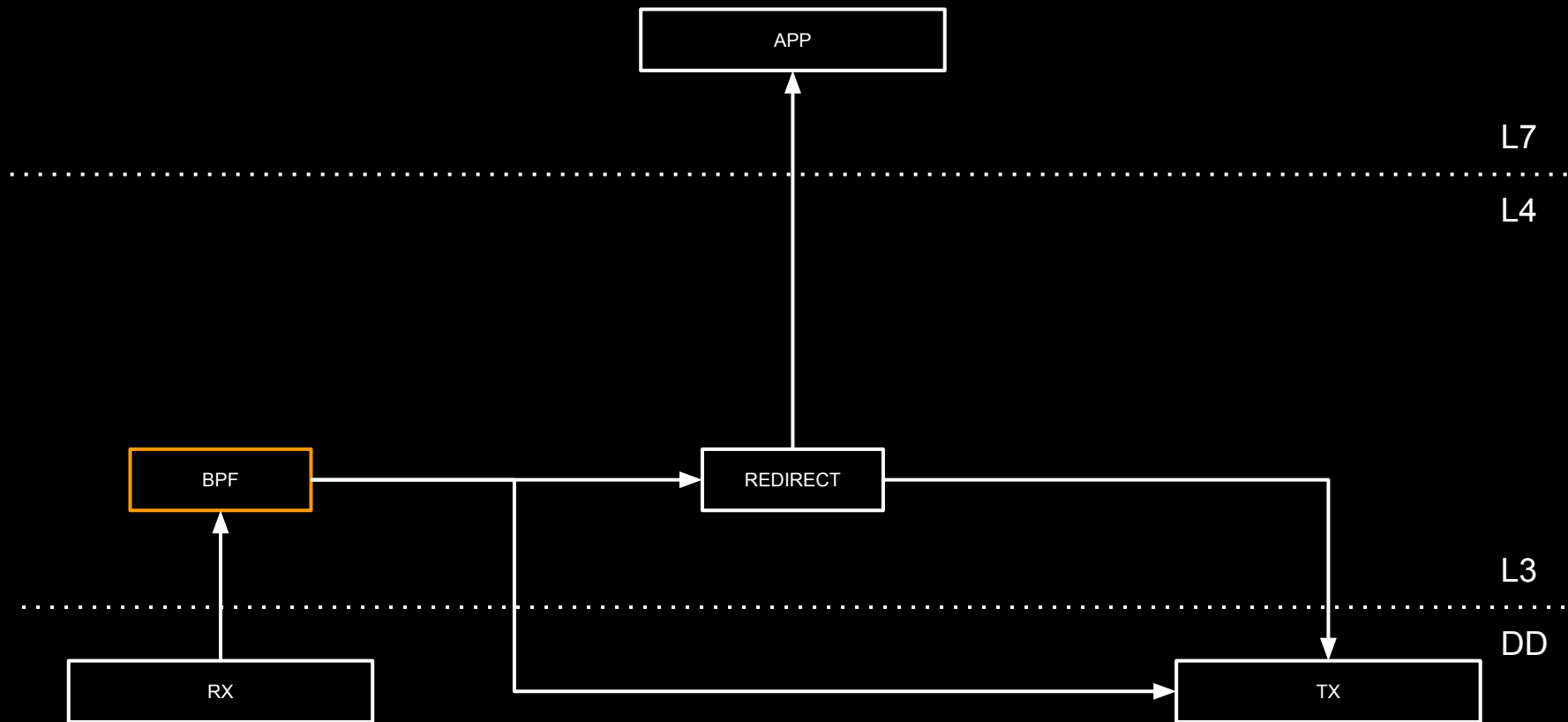




# XDP FAST PATH



# XDP GENERIC PATH



# Tutorial

## 준비물

1. 컴파일 컴퓨터 1대
2. 테스트 컴퓨터 1대(x86추천)
3. 커널 소스코드
4. clang + llvm(컴파일러)
5. bpftool(bpf 프로그램 로더)
6. bpf를 지원하는 iproute2 패키지

컴파일러

**clang + llvm**

커널 소스코드

**git.kernel.org 의 bpf tree**

<https://git.kernel.org/pub/scm/linux/kernel/git/bpf/bpf.git>

## BPF 프로그램 로더

**bpftool**

<https://git.kernel.org/pub/scm/linux/kernel/git/bpf/bpf.git/tree/tools/bpf/bpftool>

## XDP 설정도구

### **iproute2**

<https://git.kernel.org/pub/scm/linux/kernel/git/dborkman/iproute2.git>



예제

**kernel source code 및 bpf sample code**

**samples/bpf**

예제 (xdp\_rxq\_info\_kern.c)

**kernel소스 내 sample code 분석**

**samples/bpf**

컴파일

**BPF 프로그램 컴파일 실습**

**samples/bpf**

## 프로그램 로드

```
$ bpftool prog load ./xdp_rxq_info_kern.o /sys/fs/bpf/xdp
```

## 프로그램 확인

```
$ ls /sys/fs/bpf/
```

```
$ ./bpftool prog list
```

```
$ ./bpftool prog dump xlated id X
```

```
jited
```

## XDP 프로그램 설정

```
$ ip link set dev lo xdp pin /sys/fs/bpf/xdp
```

## XDP프로그램 설정 확인

```
$ ip link show dev lo
```

## XDP 프로그램 설정 제거

```
$ ip link set dev lo xdp off
```

```
$ rm /sys/fs/bpf/xdp
```