BPF IETF BoF

March 2023

- What BPF stands for?
  - The name given to an instruction set (ISA) 30 years ago by Steven McCanne and Van Jacobson.
  - This ISA is now called classic BPF (cBPF)
- extended BPF (eBPF) ISA proposed in 2013
  - first appeared in the kernel as internal BPF (iBPF)

BPF is an universal assembly language

- strictly typed assembly language
- safe for kernel and for HW
- stable instruction set, all extensions are backwards compatible

BPF design goals

- verifiable ISA
- write programs in C and compile into BPF ISA with GCC/LLVM
- Just-In-Time convert to modern 64-bit CPU
- minimal performance overhead:
  - C -> BPF ISA -> native ISA vs C -> native ISA
  - kernel code -> BPF code -> kernel code
- BPF calling convention compatible with modern 64-bit ISAs

Made ISA look familiar to existing ISA to convince kernel maintainers

- Other ISAs in the kernel
  - BPF, iptables, netfilter tables, inet_diag

- Made eBPF similar to cBPF
  - Reuse opcode encoding and 8-byte size of insn
  - Only >, >= operations initially
  - host_to_network instruction instead of bswap
  - unsigned div/mod

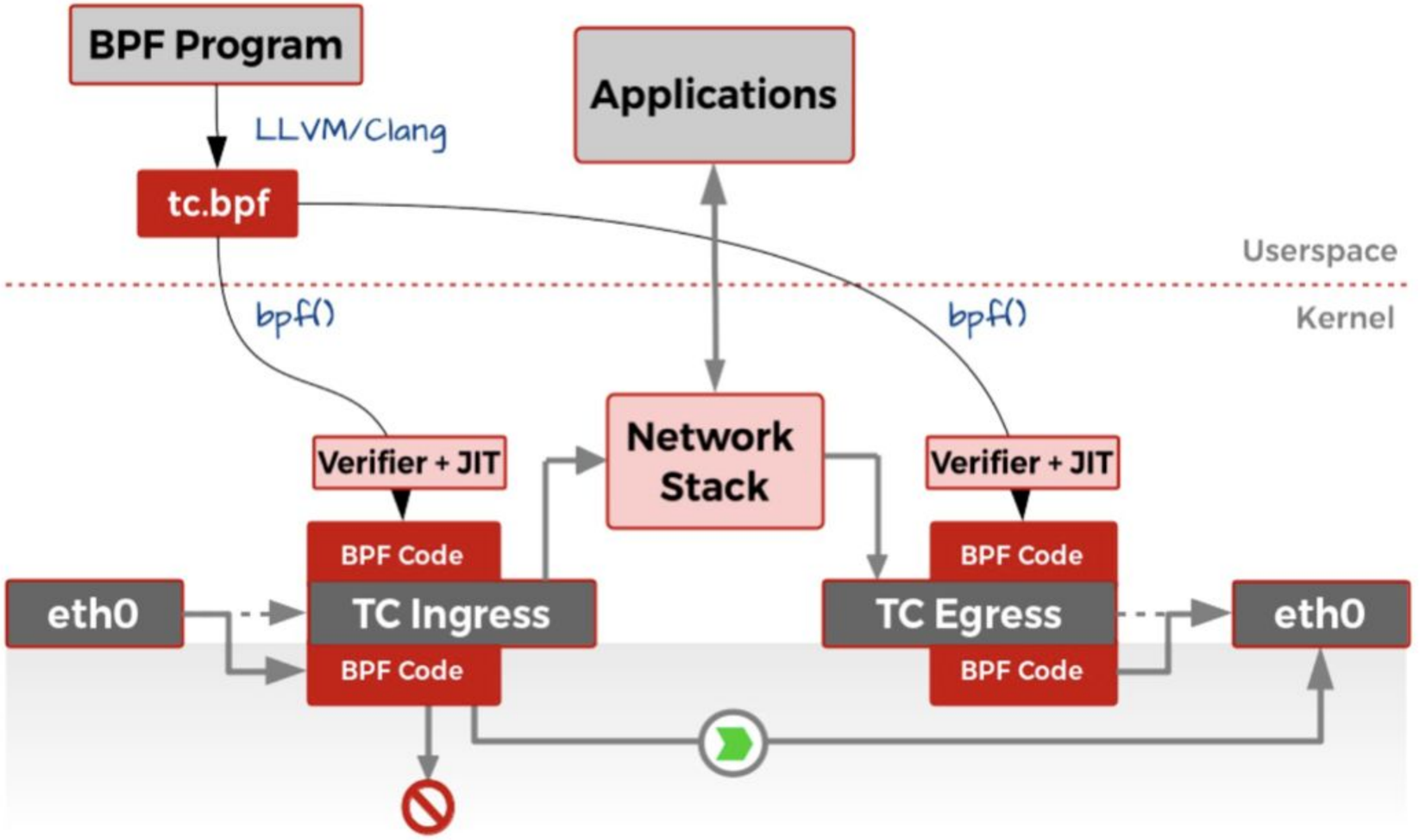
```
struct sock_filter {    /* Filter block */
        __u16   code;   /* Actual filter code */
        __u8    jt;     /* Jump true */
        __u8    jf;     /* Jump false */
        __u32   k;      /* Generic multiuse field */
};
```

```
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 */
};
```

extensions of extended BPF (2014 till now)

. ISA was extended 5 times
  . <, <= instructions
  . 32-bit compare
  . atomics
. LLVM support -mcpu=v1, v2, v3
. -mcpu=v4 is WIP
  . sign extending loads
  . bswap
  . long jmp
  . sdiv/smod (consolidating xBPF and eBPF)

BPF in networking (TC and XDP)



TC with cls_bpf and XDP. Actions on packet data is taken at the driver level in XDP

Katran - production BPF prog written in C

```
SEC("xdp")
int balancer_ingress(struct xdp_md *ctx)
{
        void *data_end = (void *)(long)ctx->data_end;
        void *data = (void *)(long)ctx->data;
        struct eth_hdr *eth = data;
        __u32 eth_proto;
        __u32 nh_off;

        nh_off = sizeof(struct eth_hdr);
        if (data + nh_off > data_end)
                return XDP_DROP;
        eth_proto = eth->eth_proto;
        if (eth_proto == bpf_htons(ETH_P_IP))
                return process_packet(data, nh_off, data_end, false, ctx);
        else if (eth_proto == bpf_htons(ETH_P_IPV6))
                return process_packet(data, nh_off, data_end, true, ctx);
        else
                return XDP_PASS;
}
```

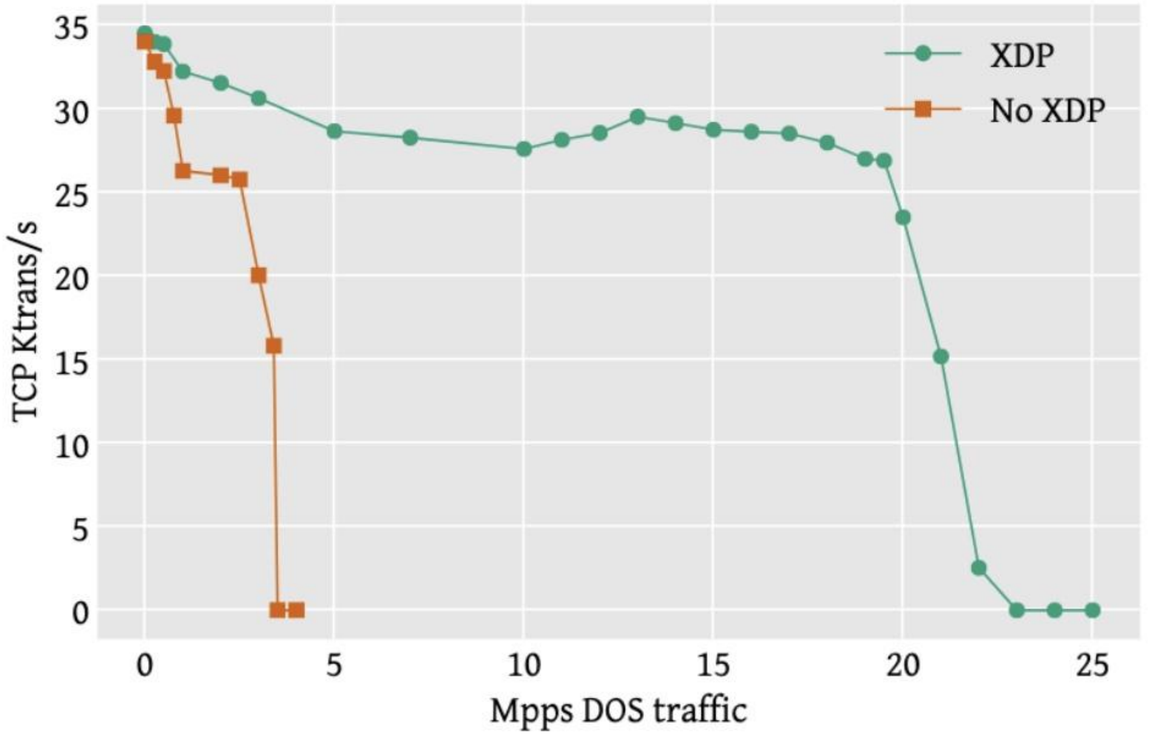
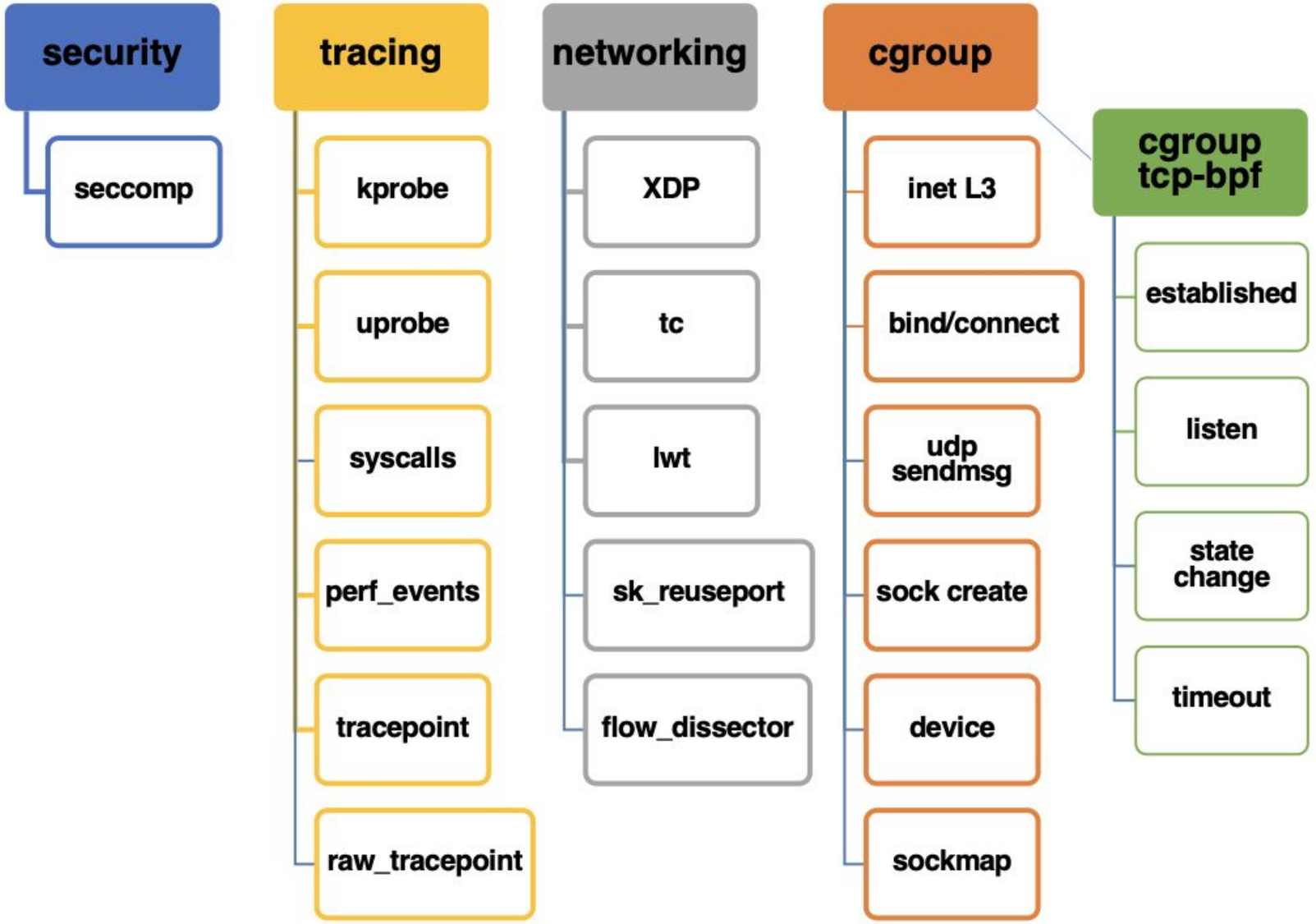


Figure 7: DDoS performance. Number of TCP transactions per second as the level of attack traffic directed at the server increases.

Table 2: Load balancer performance (Mpps).

CPU Cores	1	2	3	4	5	6
XDP (Katran)	5.2	10.1	14.6	19.5	23.4	29.3
Linux (IPVS)	1.2	2.4	3.7	4.8	6.0	7.3

BPF hooks hierarchy



BPF in the datacenter

- network edge
  - XDP firewall
  - XDP L4 loadbalancer (katran)
- inside datacenter
  - socket load balancing for L7 proxy
  - active/active transition
  - TCP/IP CC
  - monitoring/analytics at TCP and flow level

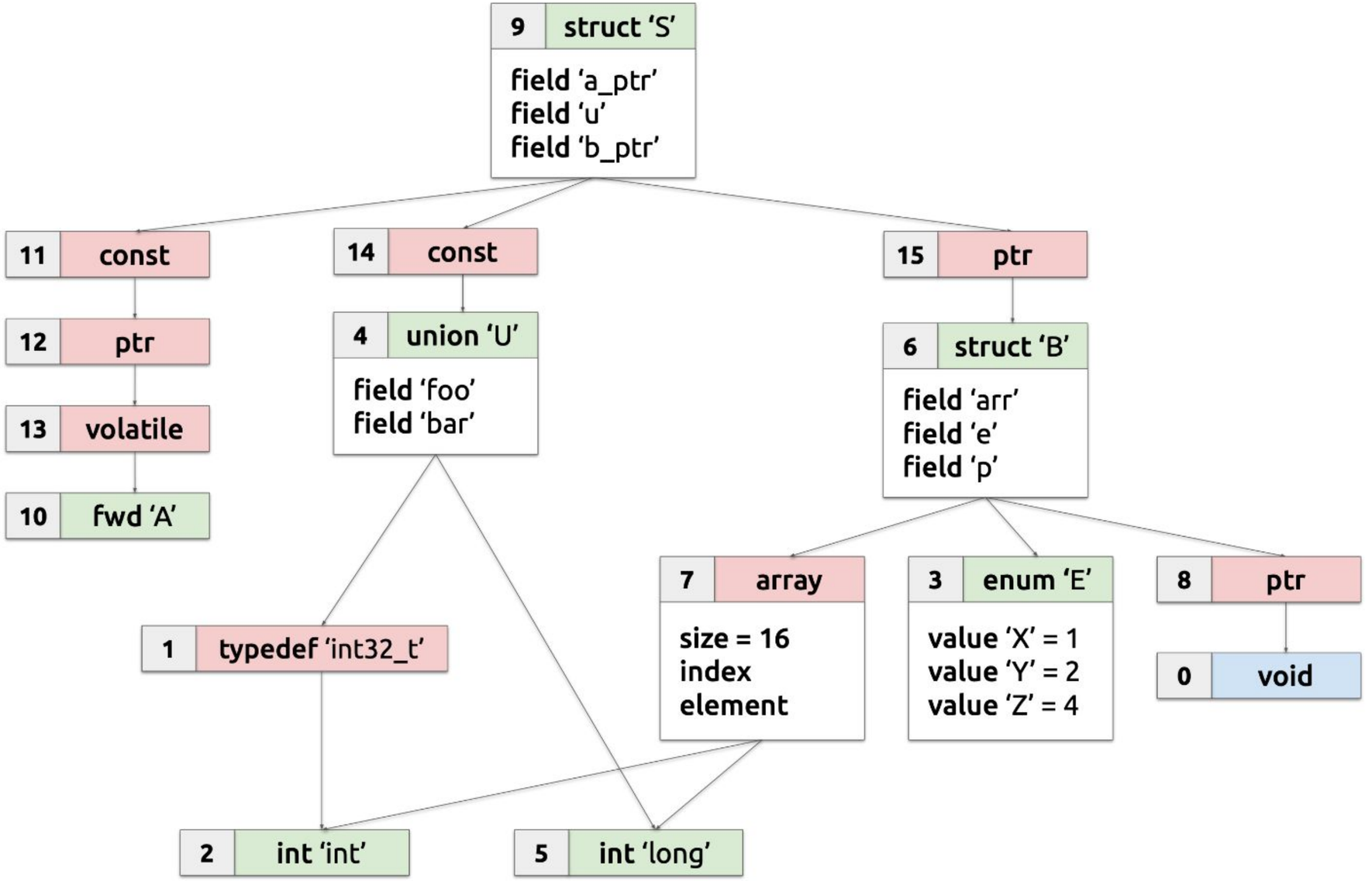
BPF in Netronome (Smart NIC)

- JIT from 64-bit BPF ISA to NFP 32-bit ISA:
  - https://git.kernel.org/pub/scm/linux/kernel/git/bpf/bpf.git/tree/drivers/net/ethernet/netronome/nfp/bpf/jit.c
- Hundreds of cpus, heterogeneous memory
- BPF hashmap -> NFP's custom logic in firmware
- XDP only

- Netronome HW offload contributed 32-bit support in LLVM and in the verifier

BTF (BPF Type Format)

```c
typedef int int32_t;

enum E {
    X = 1,
    Y = 2,
    Z = 4
};

union U {
    int32_t foo;
    long bar;
};

struct A;

struct B {
    long arr[16];
    enum E e;
    void* p;
};

struct S {
    volatile struct A* const a_ptr;
    const union U u;
    struct B* b_ptr;
};

int main() {
    struct S s;
    return 0;
}
```



Standard worthy BPF bits

- eBPF ISA
- Definition of valid program (verifier expectations)
- BTF
  - BTF.ext (Compile Once Run Everywhere)
- psABI
  - calling convention
  - linker requirements
  - relocations
- ELF format
  - section naming convention
  - BTF map definition
- Architecture:
  - programs, maps, links, BTF
  - program types (XDP, ...)
  - map types (hash, ...)