FOSDEM’17
Brussels, 2017-02-04

Stateful packet processing with eBPF: An implementation of OpenState interface

Quentin Monnet
<quentin.monnet@6wind.com>
@qeole

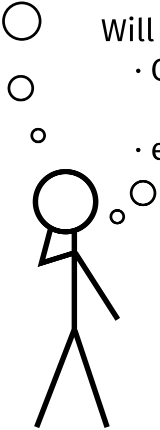
6WIND

Agenda

I'll speak at FOSDEM

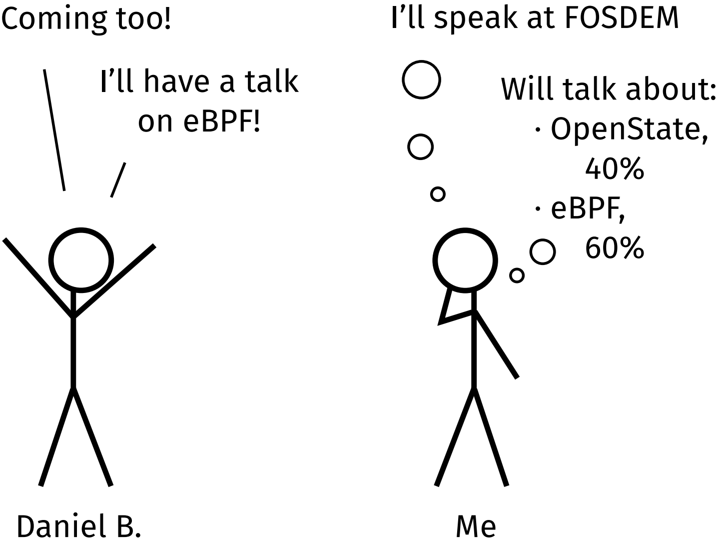
Will talk about:

- OpenState, 40%
- eBPF, 60%

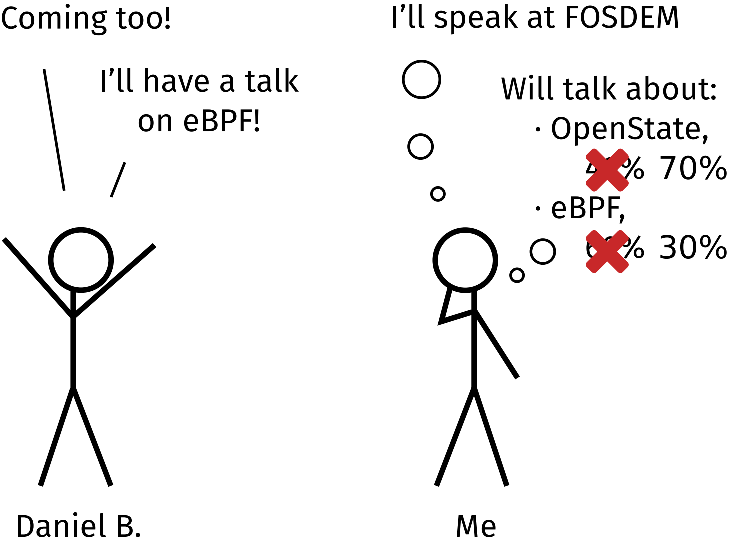


Me

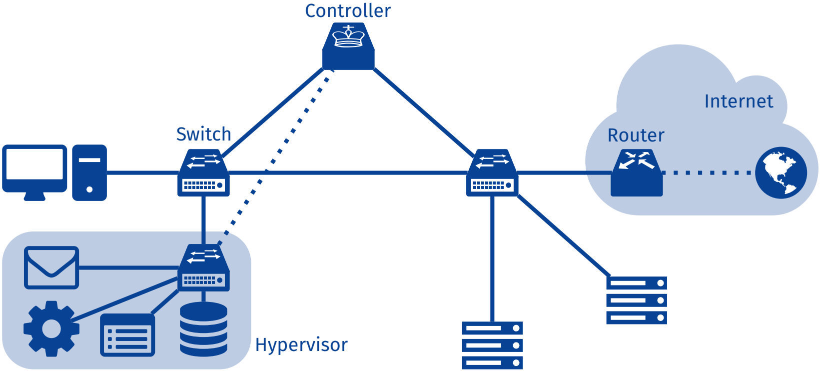
Agenda


Coming too!
I'll have a talk on eBPF!
I'll speak at FOSDEM
Will talk about:
· OpenState, 40%
· eBPF, 60%
Daniel B.
Me

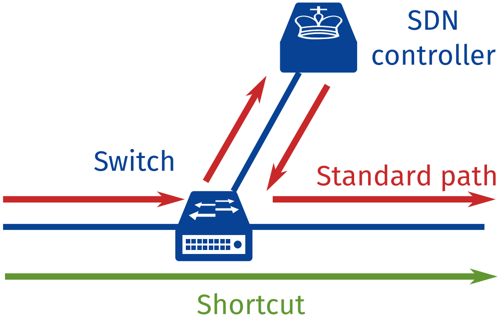
Agenda


Coming too!
I'll have a talk on eBPF!
I'll speak at FOSDEM
Will talk about:
· OpenState, 70%
· eBPF, 30%
Daniel B.
Me

SDN: hosts, VMs, programmable switches, controllers...

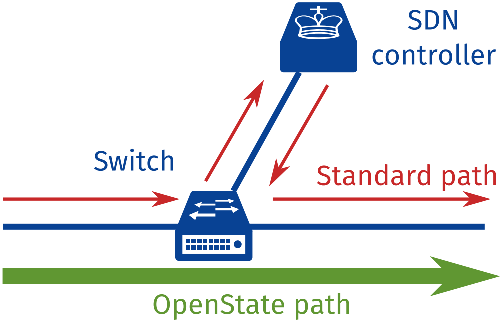


Two paths for dataplane



- Most packets goes through the “shortcut” dataplane
- Some packets are sent as exceptions—this generally includes stateful processing

What about: bringing back some control into the switch?



- Can we make the switch “smarter”, without loosing SDN benefits?
- How could we abstract stateful packet processing, in such a way the controller can easily set up the switches?

Beba
BEhavioural BAsed forwarding

European Commission
Horizon 2020

Objectives:

- Wire-speed-reactive control/processing tasks inside the switches
- Centralized control
- Scalability
- Platform-independent

From January 2015 to March 2017 (27 months)

More info at http://www.beba-project.eu/

BEBA: Who?

NEC

cnit consorzio nazionale interuniversitario per le telecomunicazioni

Beba

BEhavioural BAsed forwarding

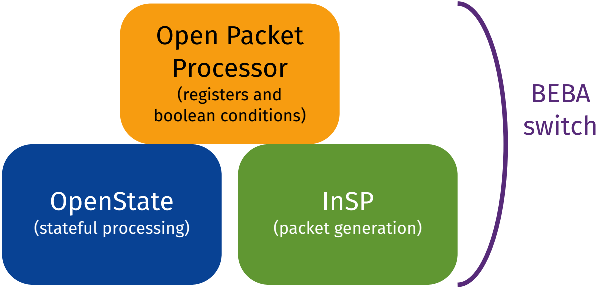
THALES

KTH VETENSKAP OCH KONST

CESNET

6WIND

BEBA switch

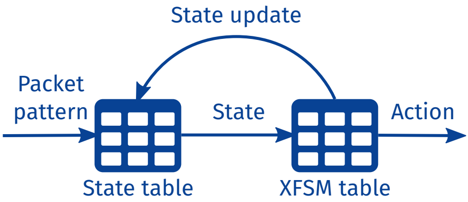

Open Packet Processor
(registers and boolean conditions)
OpenState
(stateful processing)
InSP
(packet generation)
BEBA switch

OpenState: stateful packet processing

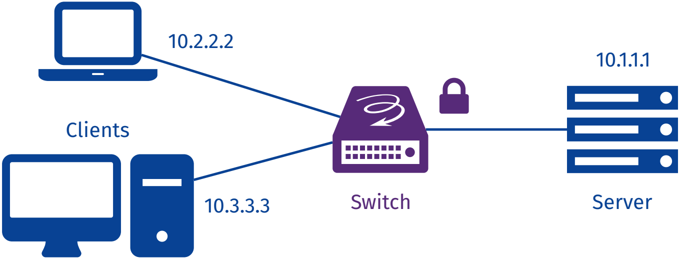
Forwarding depends on traffic previously observed

1. **Lookup** for flow **state**
2. **Lookup** for **action** associated to flow state, perform action
3. **Update state** to new value

So we need two tables: a **state table** and a table for actions: **XFSM table** (eXtended Finite State Machine)



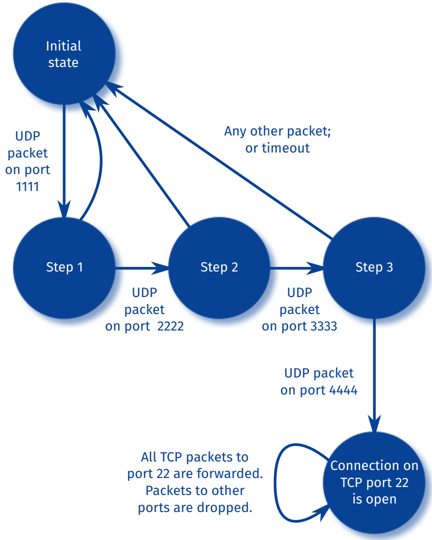
Case study: port knocking



- Clients see port 22 of the server as closed
- To access port 22, they first have to send a secret packet sequence to that port

Our example secret sequence: UDP packet on port 1111, 2222, 3333 then 4444

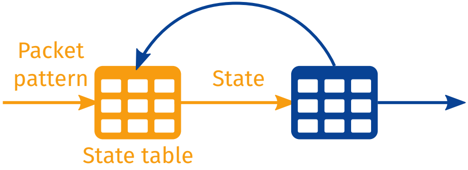
Case study: port knocking



State table

- Tracks current state for each flow

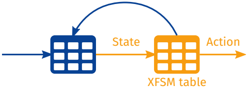
Flow matching pattern	State
...	...
IP src = any	DEFAULT



XFSM table

- To state and “event” pattern, associates action and “next state”

Flow matching pattern		Actions	
State	**Event**	**Action**	**Next state**
...	...	...	...
DEFAULT	UDP dst port = 1111	Drop	STEP_1
STEP_1	UDP dst port = 2222	Drop	STEP_2
STEP_2	UDP dst port = 3333	Drop	STEP_3
STEP_3	UDP dst port = 4444	Drop	OPEN
OPEN	TCP dst port 22	Forward	OPEN
OPEN	Port = *	Drop	OPEN
...	...	...	...
*	Port = *	Drop	DEFAULT

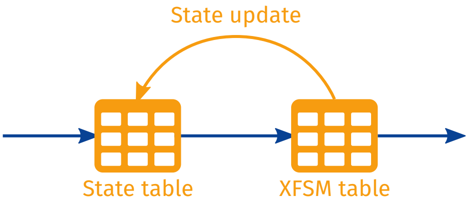


- “Next state” is used to update the entry for this flow in the state table

State table update

- The state of the flow is updated for each packet, thus unrolling the port knocking sequence

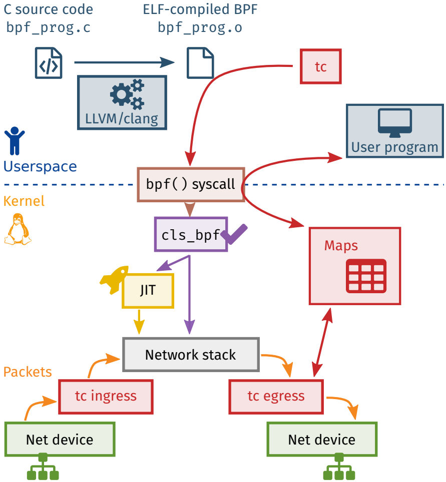
Flow matching pattern	State
...	...
IP src = 10.3.3.3, IP dst = 10.1.1.1	STEP_1
...	...
IP src = any	DEFAULT



Can we implement that with eBPF?

eBPF ~ extended Berkeley Packet Filter

- Assembly-like language, based on cBPF (packet filtering)
- Programs come from user space, run in the kernel



Stateful eBPF

- Default behavior: program is run to process a packet, no state preserved on exit
- However: eBPF Maps (kernel 3.18+):
  - Memory area accessible from eBPF program through specific kernel helpers
  - Arrays, hash maps (and several other kinds)
  - **Persistent** across multiple runs of an eBPF program
  - Can be shared with other eBPF programs
  - Can be shared with userspace applications

→ Let's use hash maps for OpenState tables!

(`https://github.com/qmonnet/pkpoc-bpf`)

openstate.h

```c
/* State table */

struct StateTableKey {
    uint16_t ether_type;
    uint32_t ip_src;
    uint32_t ip_dst;
};

struct StateTableVal {
    int32_t state;
};

/* XFSM table */

struct XFSMTableKey {
    int32_t  state;
    uint8_t  l4_proto;
    uint16_t src_port;
    uint16_t dst_port;
};

struct XFSMTableVal {
    int32_t action;
    int32_t next_state;
};
```

portknocking.c: State table lookup

```c
/* [Truncated]
 * Parse headers and make sure we have an IP packet, extract src and dst
 * addresses; since we will need it at next step, also extract UDP src and dst
 * ports.
 */

state_idx.ether_type = ntohs(ethernet->type);
struct StateTableKey state_idx;
state_idx.ip_src = ntohl(ip->src);
state_idx.ip_dst = ntohl(ip->dst);

/* State table lookup */

struct StateTableVal *state_val = map_lookup_elem(&state_table, &state_idx);

if (state_val) {
    current_state = state_val->state;
    /* If we found a state, go on and search XFSM table for this state and
     * for current event.
     */
    goto xfsmlookup;
}
goto end_of_program;
```

portknocking.c: XFSM table lookup, state table update, action

```
/* Set up the key */
xfsm_idx.state    = current_state;
xfsm_idx.l4_proto = ip->next_protocol;
xfsm_idx.src_port = 0;
xfsm_idx.dst_port = dst_port;

/* Lookup */
struct XFSMTableVal *xfsm_val = map_lookup_elem(&xfsm_table, &xfsm_idx);

if (xfsm_val) {

    /* Update state table entry with new state value */
    struct StateTableVal new_state = { xfsm_val->next_state };
    map_update_elem(&state_table, &state_idx, &new_state, BPF_ANY);

    /* Return action code */
    switch (xfsm_val->action) {
        case ACTION_DROP:
            return TC_ACT_SHOT;
        case ACTION_FORWARD:
            return TC_ACT_OK;
        default:
            return TC_ACT_UNSPEC;
    }
}
```

Compile and run

- One would compile the complete program into eBPF with:

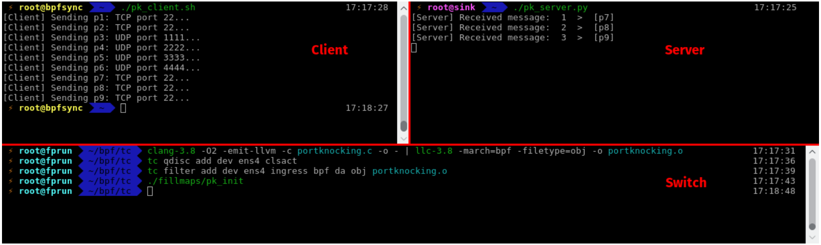
```
$ clang -O2 -emit-llvm -c openstate.c -o - | \
    llc -march=bpf -filetype=obj -o openstate.o
```

- ... and attach it with, for example:

```
# tc qdisc add dev eth0 clsact
# tc filter add dev eth0 ingress bpf da obj openstate.o
```

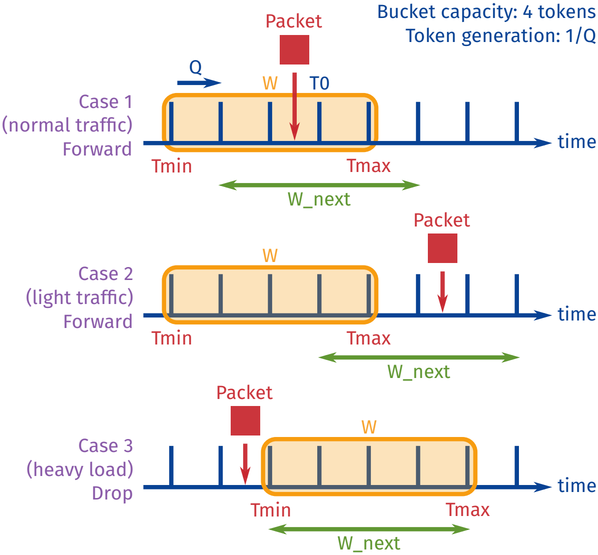
- One more thing: initialize the maps (user-space program with `bpf()` syscall)
- Alternative method: bcc's Python wrappers provide an easier way to initialize maps, to compile and to inject programs

Result


root@bpfsync ~ ./pk_client.sh 17:17:28
[Client] Sending p1: TCP port 22...
[Client] Sending p2: TCP port 22...
[Client] Sending p3: UDP port 1111...
[Client] Sending p4: UDP port 2222...
[Client] Sending p5: UDP port 3333...
[Client] Sending p6: UDP port 4444...
[Client] Sending p7: TCP port 22...
[Client] Sending p8: TCP port 22...
[Client] Sending p9: TCP port 22...
root@bpfsync ~ 17:18:27
Client
root@sink ~ ./pk_server.py 17:17:25
[Server] Received message: 1 > [p7]
[Server] Received message: 2 > [p8]
[Server] Received message: 3 > [p9]
Server
root@fprun ~/bpf/tc clang-3.8 -O2 -emit-llvm -c portknocking.c -o - | llc-3.8 -march=bpf -filetype=obj -o portknocking.o 17:17:31
root@fprun ~/bpf/tc tc qdisc add dev ens4 clsact 17:17:36
root@fprun ~/bpf/tc tc filter add dev ens4 ingress bpf da obj portknocking.o 17:17:39
root@fprun ~/bpf/tc ./fillmaps/pk_init 17:17:43
root@fprun ~/bpf/tc 17:18:48
Switch

Second case study: token bucket

Token bucket algorithm

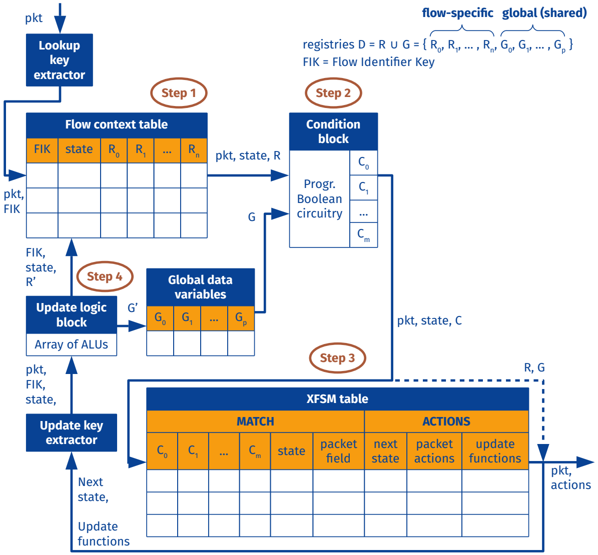


Model side: Open Packet Processor

- Extension of OpenState:
  - With global and per-flow registers
  - Registers evaluated with a set of conditions
  - XFSM table lookup must also match on conditions
- For token bucket: registers for `Tmin` and `Tmax`, then evaluate conditions:
  - `cond1 = (t ≥ Tmin);  cond2 = (t ≤ Tmax)`
  - `cond1 == true && cond2 == true` → Case 1
  - `cond1 == true && cond2 == false` → Case 2
  - `cond1 == false` → Case 3

(https://github.com/qmonnet/tbpoc-bpf)

OPP: architecture



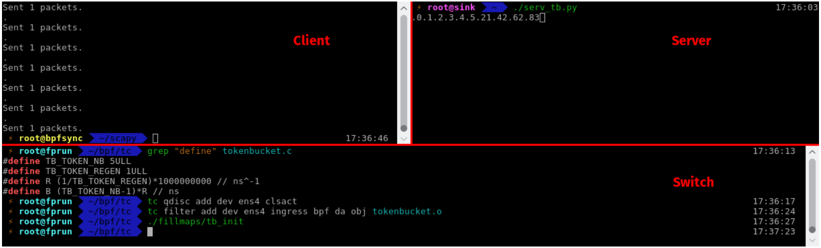
eBPF side

- Arrival time of the packet: there is a helper (`ktime_get_ns()`)
- Conditions: can be defined in each program, we need to encode the result to store it in the tables

```
uint64_t tnow = ktime_get_ns();

/* State table lookup */
state_val = map_lookup_elem(&state_table, &state_idx);

current_state = state_val->state;
tmin          = state_val->r1;
tmax          = state_val->r2;

/* Evaluate conditions */
int32_t cond1 = check_condition(GE, tnow, tmin);
int32_t cond2 = check_condition(LE, tnow, tmax);
if (cond1 == ERROR || cond2 == ERROR)
  goto error;

/* XFSM table lookup */
xfsm_idx.state = current_state;
xfsm_idx.cond1 = cond1;
xfsm_idx.cond2 = cond2;
xfsm_val = map_lookup_elem(&xfsm_table, &xfsm_idx);
```

- Tables: just add the registers, we have everything else already

Result


Sent 1 packets.
Sent 1 packets.
Sent 1 packets.
Sent 1 packets.
Sent 1 packets.
Sent 1 packets.
Sent 1 packets.
root@bpfsync ~/scapy
17:36:46
Client
root@sink ~ ./serv_tb.py
17:36:03
.0.1.2.3.4.5.21.42.62.83
Server
root@fprun ~/bpf/tc grep "define" tokenbucket.c
17:36:13
#define TB_TOKEN_NB 5ULL
#define TB_TOKEN_REGEN 1ULL
#define R (1/TB_TOKEN_REGEN)*1000000000 // ns^-1
#define B (TB_TOKEN_NB-1)*R // ns
root@fprun ~/bpf/tc tc qdisc add dev ens4 clsact
17:36:17
root@fprun ~/bpf/tc tc filter add dev ens4 ingress bpf da obj tokenbucket.o
17:36:24
root@fprun ~/bpf/tc ./fillmaps/tb_init
17:36:27
root@fprun ~/bpf/tc
17:37:23
Switch

Additional use cases for OpenState and OPP

- QoS, load balancer
- DDoS detection and mitigation as middle box application or at network level
- In-switch ARP handling in datacenter
- Forwarding consistency
- Failure detection and recovery
- ...

Conclusion

- eBPF makes a nice target for BEBA architecture (OpenState, Open Packet Processor)
- Some limitations:
  - no wildcard mechanism for map lookup (yet)
  - locks for concurrent access?
- Next step:
  - With XDP (hook in the driver instead of tc interface)?
  - High-level description language to generate the program
- More implementations, by other project partners:
  - Reference implementation: ofsoftswitch
  - Acceleration with PFQ (controller: Ryu)
  - Acceleration with DPDK, running on a FPGA

Thank you!

Resources

BEBA project web page
http://www.beba-project.eu/

BEBA repositories: reference implementations of BEBA switch and controller
https://github.com/beba-eu

OpenState article (SIGCOMM 2014)
http://openstate-sdn.org/pub/openstate-ccr.pdf

Open Packet Processor article (TBP)
https://arxiv.org/abs/1605.01977

Code for the port knocking proof-of-concept in eBPF
https://github.com/qmonnet/pkpoc-bpf

Code for the token bucket proof-of-concept in eBPF
https://github.com/qmonnet/tbpoc-bpf

Resources on BPF — *Dive into BPF: a list of reading material*
https://qmonnet.github.io/whirl-offload/2016/09/01/dive-into-bpf/

GitHub repository of the IO Visor project (bcc tools, documentation, and more)
https://github.com/iovisor/