Stateful packet processing with eBPF: An implementation of OpenState interface

Quentin Monnet
<quentin.monnet@6wind.com>
@qeole
I’ll speak at FOSDEM

- Will talk about:
  - OpenState, 40%
  - eBPF, 60%

Me
I’ll speak at FOSDEM
Will talk about:
  ⋅ OpenState,  40%
  ⋅ eBPF,  60%

Daniel B.
Coming too!
I’ll have a talk on eBPF!

Me
I’ll speak at FOSDEM
Will talk about:
- OpenState, 40%
- eBPF, 60%

Coming too!
I’ll have a talk on eBPF!

Daniel B.

Me
SDN: hosts, VMs, programmable switches, controllers...
Two paths for dataplane

- Most packets go 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?
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?
BEBA switch

Open Packet Processor
(stateful processing)

OpenState
InSP
(packet generation)
(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

UDP packet on port 1111

Any other packet; or timeout

UDP packet on port 2222

UDP packet on port 3333

UDP packet on port 4444

All TCP packets to port 22 are forwarded. Packets to other ports are dropped.

Connection on TCP port 22 is open

Initial state

Step 1

Step 2

Step 3

Quentin Monnet (6WIND) | BEBA — OpenState and eBPF
State table

- Tracks current state for each flow

<table>
<thead>
<tr>
<th>Flow matching pattern</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>IP src = any</td>
<td>DEFAULT</td>
</tr>
</tbody>
</table>

 Quentin Monnet (6WIND) | BEBA — OpenState and eBPF
XFSM table

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

<table>
<thead>
<tr>
<th>State</th>
<th>Event</th>
<th>Action</th>
<th>Next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT</td>
<td>UDP dst port = 1111</td>
<td>Drop</td>
<td>STEP_1</td>
</tr>
<tr>
<td>STEP_1</td>
<td>UDP dst port = 2222</td>
<td>Drop</td>
<td>STEP_2</td>
</tr>
<tr>
<td>STEP_2</td>
<td>UDP dst port = 3333</td>
<td>Drop</td>
<td>STEP_3</td>
</tr>
<tr>
<td>STEP_3</td>
<td>UDP dst port = 4444</td>
<td>Drop</td>
<td>OPEN</td>
</tr>
<tr>
<td>OPEN</td>
<td>TCP dst port 22</td>
<td>Forward</td>
<td>OPEN</td>
</tr>
<tr>
<td>OPEN</td>
<td>Port = *</td>
<td>Drop</td>
<td>OPEN</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>*</td>
<td>Port = *</td>
<td>Drop</td>
<td>DEFAULT</td>
</tr>
</tbody>
</table>

“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.

<table>
<thead>
<tr>
<th>Flow matching pattern</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>IP src = 10.3.3.3, IP dst = 10.1.1.1</td>
<td>STEP_1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>IP src = any</td>
<td>DEFAULT</td>
</tr>
</tbody>
</table>

State update

State table update

State table | XFSM table
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

/* 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;
};
/* [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;
/* 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 XFSMTTableVal *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
Second case study: token bucket
Token bucket algorithm

Case 1 (normal traffic)
Forward

Case 2 (light traffic)
Forward

Case 3 (heavy load)
Drop

Bucket capacity: 4 tokens
Token generation: 1/Q

W_next
Packet

T0

W
Packet

Tmin
Tmax

W_next
Packet

Tmin
Tmax

W_next
Packet

Tmin
Tmax

W
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 $T_{min}$ and $T_{max}$, then evaluate conditions:

- $\text{cond}_1 = (t \geq T_{min})$;
- $\text{cond}_2 = (t \leq T_{max})$
- $\text{cond}_1 == true$ && $\text{cond}_2 == true$ → Case 1
- $\text{cond}_1 == true$ && $\text{cond}_2 == false$ → Case 2
- $\text{cond}_1 == false$ → Case 3

(https://github.com/qmonnet/tb poc-bpf)
**OPP: architecture**

- **Step 1:** Flow context table
  - FIK
  - state
  - R₀, R₁, ..., Rₙ

- **Step 2:** Condition block
  - Boolean circuitry
  - C₀, C₁, ..., Cₘ

- **Step 3:** Global data variables
  - G₀, G₁, ..., Gₚ
  - pkt, state, R

- **Step 4:** XFSM table
  - MATCH
  - ACTIONS
  - C₀, C₁, ..., Cₘ
  - state
  - packet field
  - packet actions
  - update functions

**Flow context table**

<table>
<thead>
<tr>
<th>FIK</th>
<th>state</th>
<th>R₀</th>
<th>R₁</th>
<th>...</th>
<th>Rₙ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Condition block**

<table>
<thead>
<tr>
<th>Progr. Boolean circuitry</th>
</tr>
</thead>
<tbody>
<tr>
<td>C₀</td>
</tr>
<tr>
<td>C₁</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Cₘ</td>
</tr>
</tbody>
</table>

**Global data variables**

<table>
<thead>
<tr>
<th>G₀</th>
<th>G₁</th>
<th>...</th>
<th>Gₚ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**XFSM table**

<table>
<thead>
<tr>
<th>MATCH</th>
<th>ACTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>C₀, C₁, ..., Cₘ</td>
<td>state, packet field, next state, packet actions, update functions</td>
</tr>
</tbody>
</table>

**Update logic block**

- Array of ALUs

**Update key extractor**

- pkt, FIK, state, R'

**Next state, Update functions**

**(pkt, state, R')**
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

```c
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

Client

Server

Switch

 Quentin Monnet (6WIND) | BEBA — OpenState and eBPF
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
- ...

Quentin Monnet (6WIND) | BEBA — OpenState and eBPF
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)

Open Packet Processor article (TBP)

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/