Open Packet Processor:
Platform-agnostic Behavioral Forwarding and Stateful Flow Processing at wire speed

Valerio Bruschi, CNIT/University of Rome “Tor Vergata”

Joint work with: G. Bianchi, M. Bonola, S. Pontarelli, A. Capone, C. Cascone, D. Sanvito.

EU support:
Approach proposed

Stateful data plane
**OpenFlow/SDN (2009)**

- **Controller**
- **Switch**

- **Forwarding Rules**
  - Set of match/action packets must match *(STATIC RULES)*

- **API to the data plane** *(e.g., OpenFlow)*

**Dumb switch**: need to ask controller if something changes

**OpenState/SDN (2014)**

- **Controller**
- **Switch**

- **Forwarding Behavior**
  - Set of match/action packets must match
  - How rules should change or adapt to events

**Smart switch**: can dynamically update flow tables
Motivations

- OpenFlow's platform-agnostic programmatic interface permits to dynamically update match/action forwarding rules only via the explicit involvement of an external controller.
- OpenFlow does not permit to deploy forwarding behaviors directly in the switches, i.e. describe how rules should evolve in time as a consequence of packet-level events.
- Such static nature of the OpenFlow forwarding abstraction raises serious concerns regarding:
  - Scalability
  - Latency
  - Security/reliability

Stateless vs. Stateful in SDN

**Stateless data plane model** (e.g. OpenFlow)

- **Controller**
  - Global + local states
  - SMART! but Slow!

- **Switch**
  - Stateless
  - DUMB!

- Event notifications
- Control enforcing

**Stateful data plane model** (e.g. OpenState)

- **Controller**
  - Global states
  - SMART!

- **Switch**
  - Local states
  - SMART!

  *Auto-adaption*

**Signalling & latency**: $O(100 \text{ ms})$

100ms = 30M packets lost @ 100 gbps

**Signalling & latency**: update forwarding rules in 1 packet time – *3 ns* @ 40B x 100 Gbps
Beyond OpenState

Mealy Machine: nice but insufficient!

- State alone is insufficient
- OpenFlow (forwarding) actions are insufficient
- No flow processing

Flow Processing

- Flow processing requires memory, registries, counters, etc
- Flow processing requires operations (compare, add, shift, etc)
- Processing = CPU! cannot afford any ordinary CPUs at ns time scales wire speed!
Open Packet Processor

- From mealy finite state machines (FSM) to **Extended finite state machines (XFSM)**
- An EFSM is a finite state machine in which:
  1. state transitions **depend** also on a set of triggering **conditions** depending on data variables;
  2. state transitions **trigger the update** of data variables
- It also allows **cross-flow** state modification.
- **Hard parts**: use platform agnostic abstractions and make it run at wire speed – *no CPUs!*

---

Open Packet Processor: workflow

Stage 1
- Flow context table
  - FK, state, \( R_0, R_1, \ldots, R_k \)
- Condition block
  - Progr. Boolean circuitry
  - \( c_0, c_1, \ldots, c_m \)
- XFSM table
  - MATCH
    - \( c_0, c_1, \ldots, c_m \)
    - state
  - ACTIONS
    - packet fields
    - next state
    - packet actions
    - update functions

Stage 2
- Global Data Variables
  - \( G_0, G_1, \ldots, G_n \)

Stage 3
- Next_state, update_functions

Stage 4
- Update logic block
  - Array of ALU
- Update key extractor
  - pkt, FK, state

Flow-specific: \( R_0, R_1, \ldots, R_k \)
Global-shared: \( G_0, G_1, \ldots, G_n \)
Registries \( D = R \cup G = < R_0, R_1, \ldots, R_k, G_0, G_1, \ldots, G_n > \)
Open Packet Processor: workflow

**Per flow registers:** programmer-defined
(like variables in a program)
e.g.: custom statistics, traffic features, etc; Updated packet by packet

Global registers: common to multiple flows; Can be updated by multiple flows – like a global variable in a SW program
Open Packet Processor: workflow

User-programmed set of comparators. Compares pairs of quantities among registries, global variables, and packet header fields, using user-selected $>$, $<$, $=$, $\leq$, $\geq$ comparators; **returns 0/1 vector**

Condition results (a 0/1 bit string vector) **can now be used for matching**. wildcard permits to filter condition of interest for different states/events
Open Packet Processor: workflow

Returns microinstructions (of a domain-specific custom ALU instruction set) to be applied

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Instructions</th>
<th>note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic ALU Instruction</td>
<td>NOP, AND, OR,</td>
<td>standard logic operations</td>
</tr>
<tr>
<td></td>
<td>XOR, NOT</td>
<td></td>
</tr>
<tr>
<td>Arithmetic ALU Instruction</td>
<td>ADD,ADC,</td>
<td>standard arithmetic operations</td>
</tr>
<tr>
<td></td>
<td>SUB,SBC,MUL</td>
<td></td>
</tr>
<tr>
<td>Shift/Rotate Instruction</td>
<td>LSL (Logical Shift Left)</td>
<td>performs logic and arithmetic shift/rotate operations</td>
</tr>
<tr>
<td></td>
<td>LSR (Logical Shift Right)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASR (Arithmetic Shift Right)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ROR (Rotate Right)</td>
<td></td>
</tr>
<tr>
<td>pkt/flow specific Instruction</td>
<td>ewma(),avg(), std()</td>
<td>compute specific pkt/flow task</td>
</tr>
</tbody>
</table>

Flow-specific

Global-shared

Registries \( D = R \cup G = \langle R_0, R_1, ..., R_k, G_0, G_1, ..., G_h \rangle \)
Next state & results written back into registers. Note that Update may differ from lookup, for bidirectional flow handling.

Parallel array of ALUs: executes (in 2 clock cycles) all returned microinstructions and updates relevant registers. IN/OUT also written in TCAM output - e.g. ADD(Ri, Gj) → Rk
Overall vision: still “SDN”

Controller still in charge to ‘program’ the network
But can ‘push’ time-critical / localized stateful control tasks down in the switches

Several applications
- Traffic policing
- Classifiers
- DoS mitigation
- Fault tolerance and fast failover
- Data driven routing
- Security/monitoring
- Stateful firewall
NetFPGA prototype

HW proof of concept implementation
Prototype architecture

Implemented in a NetFPGA SUME Virtex 7
Prototype architecture

**nf_10ge_interface:** Four ingress queues collect the packets coming from the ingress ports.

**input_arbiter:** A 4-input 1-output mixer block aggregates the packets using a round robin policy. The output of the mixer is a 320 bits data bus able to provide an overall throughput of 50 Gbps.

A delay queue stores the packet during the time need by the Open Packet Processor tables to operate.
Prototype architecture

The state table is realized by the d-left hash table (4k entries, MHT without moving capability) and a small TCAM (32 entries * 128 bits) and a companion SRAM (configured as dual port RAM).

First TCAM only for static states (e.g. packets belonging to a given subnet).

The look-up and update extractor blocks that build the keys that are used to read/update the state table. The 128 bit output is given as input to the state lookup and update.
Prototype architecture
The XFSM table is realized by the second TCAM/SRAM pair. The TCAM has 128 entries * 160 bits and the RAM store the next state, an action (if any) and a set of ALU INSTRUCTIONS.
Prototype architecture

This block deploys an **array of ALUs** (Arithmetic and Logic Units) which support a specific set of (micro)instructions and which execute in parallel the instructions provided as output of the XFSM Table. The updated registry values are stored in the memory locations (flow registries and/or global registries).

**output queues_v1_0_0:**
The action block applies the selected actions and forward the packet to the output queues.
Prototype architecture

Each component **is memory mapped** in the address space handled by the NetFPGA with the protocol AXI-lite. Thus, prototype is configurable via MicroBlaze or PCIe which **can directly read/write** the content of these components.
TCAM-based packet processing engine!

- Extreme flexibility!
  - XFSM ‘programs’ almost flexible as ordinary programming language
    - can define variables, store and change values, compute features, etc

- Guaranteed wire speed!
  - Fixed time per-packet computational loop
    - 6 clock cycles in our ongoing HW design

- (currently two tech limitations)
  - Only 1 ALU operation per each packet $\rightarrow$ pipelined ALU arrays possible, but would increase processing time and yield more complex configuration
  - ALUs only in update, not in conditions $\rightarrow$ does not permit conditions such as $(R1+R2>100)$
    - Solution (not nice, but workaround): compute $R1+R2 \rightarrow R3$ during previous packet, then use $(R3>100)$
DEMO
LOAD BALANCING, flow-consistent
Demo high level description

Counter: 2
Demo detailed deployment

nina Intel i5 server

nina eth4
web virtual host 1
on 10.0.0.2:80

nina eth5
web virtual host 2
on 10.0.0.3:80

NetFPGA OPP PoC

PowerEdge Intel i5 server

PowerEdge eth4
web client 1

PowerEdge eth5
web client 2
Configuring the NetFPGA
WEB client 1 get http://www.sosr-demo.eu
WEB client 2 get http://www.sosr-demo.eu
You're browsing privately

Firefox won't remember any history for this window.

That includes browsing history, search history, download history, web form history, cookies, and temporary internet files. However, files you download and bookmarks you make will be kept.

While this computer won't have a record of your browsing history, your employer or internet service provider can still track the pages you visit.

Learn More.
Dumping Flow Context table

Insert '1' to dump Flow Context table

1

searching on HT

-------------------------------------

FLOW KEY

<table>
<thead>
<tr>
<th>SRC IP</th>
<th>DST &amp; SRC port</th>
<th>Number of packet forwarded</th>
</tr>
</thead>
<tbody>
<tr>
<td>0200000A</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0200000A</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0200000A</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0200000A</td>
<td>00000000</td>
<td>00000000</td>
</tr>
</tbody>
</table>

Present state

<table>
<thead>
<tr>
<th>LocalRegister: Number of packet forwarded</th>
</tr>
</thead>
<tbody>
<tr>
<td>C00000B1</td>
</tr>
<tr>
<td>C00000B1</td>
</tr>
<tr>
<td>C00000B1</td>
</tr>
<tr>
<td>C00000B1</td>
</tr>
<tr>
<td>C00000AA</td>
</tr>
<tr>
<td>C00000AA</td>
</tr>
<tr>
<td>C00000AA</td>
</tr>
<tr>
<td>C00000AA</td>
</tr>
</tbody>
</table>

Insert '1' to dump Flow Context table
Thank you!

Contact:
- Valerio.Bruschi@students.uniroma2.eu
- Valerio.Bruschi@cnit.it