Data Plane programmability: CNIT/Uniroma2 ongoing work

Giuseppe Bianchi, Salvatore Pontarelli
CNIT / University of Roma Tor Vergata

Credits to: M. Bonola, A. Capone, C. Cascone, D. Sanvito, F. Gringoli,
M. Spaziani Brunella, V. Bruschi, M. Welzl, M. Faltelli, G. Belocchi
(plus extern colleagues – R. Bifulco and others ...)

EU Support: 5G-PICTURE
Looking forward: «universally» deployable network functions (or even full protocols, e.g. TCP)

- Challenge: programming THE function itself
  - Not the control

- Focus on “universal” portability
  - HW switches
  - Smart NICs

Giuseppe Bianchi
Programming alternatives for network functions?

- P4
- eBPF
- OpenFlow Extensions → XFSM
- Click
- Netbricks
- VMs/Containers/Unikernels (also on smart NICs)
- Custom Network Processor Languages
- Assembly over tailored RISCs
- ...

- Not a clear answer…
  ⇒ (unless answer is critical mass, then… leave it to business)
- Perhaps not a unique answer?
  ⇒ A “protocol” is different from a packet processing function
Some questions worth to ask to candidate Programming Abstraction (PA)

➔ Guaranteed fast path?
  ⇒ *(I mean: by design)*
  ⇒ *with enough thrust, even pigs can fly*

➔ Does PA «forces» modularity?
  ⇒ Avoid spaghetti-code
  ⇒ Avoid implicit dependences

➔ Where is the «state»?
  ⇒ And state of what?
  ⇒ *Clean design → stateless modules.*

➔ What you see (before compiling) is what you only need to see?

➔ Where is «time/buffer management»?
  ⇒ Timers, events, pacing, etc ← any protocol!
  ⇒ *(not just packet forwarding)*
Our take: XFSMs

«if» part

STATE == «wait»
EVENT == TimerExp(A)
CONDITIONS: (A>7) && (Reg1==2)

«then» part

ACTION(s):
TX(pkt B)
SetTimer(B, +10ms)
Pkt C→HashTable

NEXTSTATE == «active»

UPDATE(s):
Reg1+=4
Hash(C) → Reg2
Why we believe in XFSMs?

<table>
<thead>
<tr>
<th>STATE</th>
<th>EVENT</th>
<th>CONDITIONS:</th>
</tr>
</thead>
<tbody>
<tr>
<td>«wait»</td>
<td>TimerExp(A)</td>
<td>(A&gt;7) &amp;&amp; (Reg1==2)</td>
</tr>
</tbody>
</table>

«if» part

<table>
<thead>
<tr>
<th>ACTION(s):</th>
</tr>
</thead>
<tbody>
<tr>
<td>TX(pkt B)</td>
</tr>
<tr>
<td>SetTimer(B, +10ms)</td>
</tr>
<tr>
<td>Pkt C→HashTable</td>
</tr>
</tbody>
</table>

«then» part

<table>
<thead>
<tr>
<th>NEXTSTATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>«active»</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UPDATE(s):</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reg1+=4</td>
</tr>
<tr>
<td>Hash(C) à Reg2</td>
</tr>
</tbody>
</table>

→ THREE «programming levels» in one

→ TOP: Clearcut separation of behavior (state evolution) from «building blocks»
  ⇒ And upfront state – compare to block-based abstractions

→ BOTTOM: domain API
  ⇒ Actions, events, update functions: domain-specific! (e.g. time mgmt)
  ⇒ NOT programmed by XFSM → program as you wish → very clear integration model
  ⇒ Compatible with vendors’ need for closed source

→ MID: Data structure & microprograms!!
  ⇒ In principle unrestricted → Turing-completeness!
  ⇒ Bounded by practical considerations…
Why we believe(d) in OpenFlow++?

- Many “pieces” already specified
  - Actions, packet matching, etc
- Could be smoothly evolved towards state support!
  - Opposed to radical HW (and model) changes of incoming programming models
  - E.g. Openstate → OF1.6....
- And to full XFSM support!
  - XFSM → OF1.xx (versus 2.0)

Indeed: we have already DEMONSTRATED this last step: OPP → FlowBlaze
Flow context retrieval

Tell me what flow the packet belongs to and what is its state (and associated registries)
Open Packet Processor at a glance

Condition verification

Does the flow context respect some (user defined) conditions?
Open Packet Processor at a glance

XFSM execution
Match current status and conditions and retrieve next state and update functions (fetch packet actions)
Open Packet Processor at a glance

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, XOR, NOT</td>
<td>standard logic operations</td>
</tr>
<tr>
<td>Arithmetic ALU Instruction</td>
<td>ADD, ADC, SUB, SBC, MUL</td>
<td>standard arithmetic operations</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>
Open Packet Processor at a glance

Execute $\mu$-instructions
Permits to embed user-defined computation in the pipeline
Open Packet Processor at a glance

And update state and registers for the next packet
Close the “computational loop” – no CPU involved
TCAM as state transition engine and ALUs as processing functions
From flow processing to full protocol modeling:

XTRA – XFSM for TRAnsport
(ongoing work)
From functions to protocol

➔ L4 load balancing, filtering, firewalls: header-based
  ⇒ Easy! (With P4, OF++)

➔ But a full TCP-like protocol...???
  ⇒ Packet forging!
  ⇒ Time events!
  ⇒ Complex storage/buffer (e.g. SACK)
  ⇒ Complex scheduling (e.g. pacing)

➔ Literature: SW abstractions only

Use cases: offloading to FPGA smart NICs, high performance middleboxes (PEPs, proxies, etc)
**XTRA architecture**

→ **Everything is an event**
  ⇒ Tailored «drivers»

→ **Explicit calendar**
  ⇒ Time events/actions

→ **Buffer mgmt**
  ⇒ tailored

→ **Anything else rolls back to XFSMs/OPP**
Table 2: The complete TB-TCP XFSM implementation. States: C=Closed; SyS=SYN Sent; FW(1/2)=FIN-WAIT(1/2); TW=Time Wait; SS=Slow Start; CA=Congestion Avoidance; R=Recovery; PR=Post Recovery

<table>
<thead>
<tr>
<th>#</th>
<th>Event</th>
<th>Conditions</th>
<th>State</th>
<th>Next Actions, Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>connect</td>
<td>any</td>
<td>C</td>
<td>setFlag(SYN), sendEmptyPacket(), setTimer(1, 0.5s)</td>
</tr>
<tr>
<td>2</td>
<td>timeout</td>
<td>any</td>
<td>SyS</td>
<td>same actions as table entry 1</td>
</tr>
<tr>
<td>3</td>
<td>SYNACK</td>
<td>any</td>
<td>SyS</td>
<td>ss removeTimer(1), setFlag(ACK), setTimer(nextTx, 0, '1µs')</td>
</tr>
<tr>
<td>4</td>
<td>any</td>
<td>socketClosed=true</td>
<td>SS, CA, R, FR</td>
<td>FW1 setFlag(FIN), sendEmptyPacket(), setTimer(0, 0, 2*rtt)</td>
</tr>
<tr>
<td>5</td>
<td>timeout</td>
<td>timeoutSeq ≥ lastAck</td>
<td>FW1</td>
<td>FW1 same actions as table entry 4</td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
<td>any</td>
<td>FW1</td>
<td>FW2 nothing to do</td>
</tr>
<tr>
<td>7</td>
<td>FINACK</td>
<td>any</td>
<td>FW1</td>
<td>TW setFlag(ACK), setTimer(120s), sendEmptyPacket()</td>
</tr>
<tr>
<td>8</td>
<td>timeout</td>
<td>any</td>
<td>TW</td>
<td>C closeSocket()</td>
</tr>
<tr>
<td>9</td>
<td>timeout</td>
<td>availWin &gt; 0</td>
<td>SS</td>
<td>setTimer(nextTx, 1.2*rtt), send(nextTx), nextTx++-ms, highTx=mass, setTimer(nextTx, 0, '1µs'), availWin-cwnd+lastAck-highTx</td>
</tr>
<tr>
<td>10</td>
<td>ACK</td>
<td>ackNo ≠ lastAck</td>
<td>SS</td>
<td>cwnd+mass, lastAck=ackNo, setTimer(nextTx, 0, '1µs'), rtt=now-echoRpl, availWin-cwnd+lastAck-highTx</td>
</tr>
<tr>
<td>11</td>
<td>timeout</td>
<td>timeoutSeq ≥ lastAck, refTxCntr &gt; refTxEnd</td>
<td>SS</td>
<td>R prevCwnd=cwnd, prevSyn=now, prevState=0, cwnd=0, refTxRound+=1, pacing=rtt/(cwnd, mass), nextTx=lastAck, setTimer(nextTx, 0, pacing)</td>
</tr>
<tr>
<td>12</td>
<td>timeout</td>
<td>timeoutSeq ≥ lastAck, refTxCntr &gt; refTxEnd</td>
<td>R</td>
<td>R same actions as table entry 11</td>
</tr>
<tr>
<td>13</td>
<td>timeout</td>
<td>nextTx &lt; highTx, nextTx+timeoutSeq</td>
<td>R</td>
<td>R send(nextTx), nextTx+ms, setTimer(timerSeqNo, 0, pacing), setTimer(nextTx, refTxRound+1, 1.2*rtt)</td>
</tr>
<tr>
<td>14</td>
<td>timeout</td>
<td>nextTx ≥ highTx</td>
<td>R</td>
<td>R pktsToPause=(cwnd, mass)-1, setTimer(timerSeqNo, 0, pacing)</td>
</tr>
<tr>
<td>15</td>
<td>ACK/SACK</td>
<td>nextTx ≥ highTx</td>
<td>PR</td>
<td>rtxRound=0, pktsToPause=(cwnd, mass)+1, setTimer(nextTx, 0, pacing)</td>
</tr>
<tr>
<td>16</td>
<td>ACK</td>
<td>nextTx ≥ highTx, ts_echo ≥ ts</td>
<td>PR</td>
<td>R same actions as table entry 15, cwnd=prevCwnd, timeTo=0, spurious=1</td>
</tr>
<tr>
<td>17</td>
<td>SACK</td>
<td>any</td>
<td>R</td>
<td>removeTimers(sackBlock0, sackBlock1, sackBlock2)</td>
</tr>
<tr>
<td>18</td>
<td>timeout</td>
<td>availWin &gt; 0, nextTx = timeoutSeq</td>
<td>FR</td>
<td>PR send(nextTx), nextTx+ms, highTx=mass, pktsToPause=1, pacing=rtt/(pktsToPause, setTimer(nextTx, 0, pacing)), availWin-cwnd+lastAck-highTx</td>
</tr>
<tr>
<td>19</td>
<td>ACK</td>
<td>ackNo ≠ lastAck</td>
<td>FR</td>
<td>PR lastAck=ackNo, availWin-cwnd+lastAck-highTx, rtt=now-echoRpl</td>
</tr>
<tr>
<td>20</td>
<td>timeout</td>
<td>spurious=1, prevState=0</td>
<td>FR</td>
<td>CA spurious=0</td>
</tr>
<tr>
<td>21</td>
<td>timeout</td>
<td>spurious=1, prevState=1</td>
<td>FR</td>
<td>SS spurious=0</td>
</tr>
<tr>
<td>22</td>
<td>ACK</td>
<td>ackNo ≠ lastAck</td>
<td>CA</td>
<td>cwnd=mass/cwnd, lastAck=ackNo, setTimer(nextTx, refTxRound+1, 1.2*rtt), send(nextTx), rtt=now-tsEcho, nextTx+ms, highTx+ms, setTimer(nextTx, 0, '1µs'), availWin-cwnd+lastAck-highTx, prevState=1</td>
</tr>
<tr>
<td>23</td>
<td>timeout</td>
<td>availWin ≥ mss</td>
<td>CA</td>
<td>CA spurious=0</td>
</tr>
<tr>
<td>24</td>
<td>timeout</td>
<td>timeoutSeq ≥ lastAck, refTxCntr &gt; refTxEnd</td>
<td>CA</td>
<td>CA same actions as table entry 11</td>
</tr>
</tbody>
</table>
**HW implementation**

➔ **Proof of concept on NetFPGA SUME**

➔ **Calendar:** this block provides the timer events that are sent to the XFSM.

➔ **Packet parser:** this block receives the packets from the network and extracts the relevant information to provide to the XFSM. In particular it provides the events related to the arrival of an ACK packet.

➔ **XFSM executor:** this block executes the XFSM.

➔ **Packet generator:** this block generates the TCP packets to transmit when the XFSM launches the `send_packet` action.
Timer accuracy of HW calendar

Figure 6: HW Timer accuracy (exponential pacing): absolute (right y-axis, nanosecond) and relative (left y-axis) errors versus packet #.