Hardware-Accelerated Network Control Planes

Edgar Costa Molero\textsuperscript{(1)},
Stefano Vissicchio\textsuperscript{(2)}, Laurent Vanbever\textsuperscript{(1)}

\textsuperscript{(1)} ETH Zürich
\textsuperscript{(2)} UCL
Modern networks architectures are split in (at least) two planes
Modern networks architectures are split in (at least) two planes

- data plane
- control plane
Network planes can be implemented in both software or hardware.
Existing data plane implementations cover the entire software/hardware **spectrum**
What about the control plane?
Control plane implementations make seldom use of the **hardware** resources
Do we care?
Even state-of-the-art software control planes have room for **improvement**
Even state-of-the-art software control planes have room for improvement.

1. React

It can take up to a minute to detect normal failures.
Even state-of-the-art software control planes have room for **improvement**

1. React

2. Compute

~1.5 minutes to converge the control plane of an IXP route server
Even state-of-the-art software control planes have room for improvement

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>React</td>
</tr>
<tr>
<td>2</td>
<td>Compute</td>
</tr>
<tr>
<td>3</td>
<td>Update</td>
</tr>
</tbody>
</table>
Do we care?
Can we do something about it?
Modern programmable devices can perform computations on billions of packets per second
Modern programmable devices can perform computations on billions of packets per second

- Read & modify packet headers
e.g. to update network state

- Basic operations
  e.g. min & max

- Add or remove custom headers
  e.g. to carry routing information

- Keep state
  e.g. to save best paths
Can we do something about it?
Main **tasks** to compute forwarding state are...

1. **Sensing**

   monitors network to detect changes
Main **tasks** to compute forwarding state are...

1. Sensing

2. Notification
   - exchanges with network devices all the information learnt
Main **tasks** to compute forwarding state are...

1. Sensing
   - Computes forwarding paths when network changes are detected

2. Notification
   - Updates the data plane accordingly

3. Computation
Hardware-based network sensing

Goal

Challenges
Hardware-based network sensing

Goal

Detect both hard and gray failures

Challenges
Hardware-based network sensing

Goal
Detect both hard and gray failures
   e.g. random drop, TCAM bit flips

Challenges
Hardware-based network sensing

Goal
- Detect both hard and **gray** failures
  - e.g. random drop, TCAM bit flips

Challenges
- Basic hello-based mechanisms are not enough
Switches *synchronously* exchange packet counts
Switches *synchronously* exchange packet counts
Upstream switch starts probing campaigns

- # received & forwarded packets
- # sent packets
- detection state stored in registers
Traffic for some prefixes gets dropped

![Diagram showing traffic flow and packet counts between nodes A and B. The diagram includes symbols for start and stop counting, and destinations with counts for received and forwarded packets, as well as sent packets. Red packets get dropped.]
Downstream switch sends counters to upstream

- Start counting
- Traffic
- Destination
- # received & forwarded packets
- Send counters & compare
- Destination
- Stop counting
- # sent packets
- Detection state stored in registers
Upstream switch detects the failure by comparing counters

A

destination

# received & forwarded packets

send counters & compare

traffic

start counting

stop counting

B

# sent packets

destination

detection state

stored in registers

Upstream switch detects the failure by comparing counters
Hardware-based notifications

Goal

Challenges
Hardware-based notifications

Goal

Implement a broadcast notification mechanism in hardware

Challenges
Hardware-based notifications

Goal
Implement a broadcast notification mechanism in hardware

Challenges
Avoid broadcast storms
Require reliable communication
Hardware-based notifications

Avoid broadcast storms

- Use per switch broadcast sequence numbers
Hardware-based notifications

Avoid broadcast storms
  ▸ Use per switch broadcast sequence numbers

Require reliable communication
  ▸ Send notification duplicates
  ▸ Use maximum priority queues
Hardware-based computation

Goal

Challenges
Hardware-based computation

Goal
Run distributed routing algorithms in **hardware**
e.g. path vector

Challenges
## Hardware-based computation

| Goal          | Run distributed routing algorithms in **hardware**  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>e.g. path vector</td>
</tr>
<tr>
<td>Challenges</td>
<td><strong>Computation logic is limited</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Resources are heavily limited</strong></td>
</tr>
</tbody>
</table>
A prefix-to-index lookup is used to determine the output port for a given path. The forwarding state is stored in registers and consists of a prefix-to-index table and link cost information. In the diagram, A is connected to B with a cost of 1, B to C with a cost of 1, and C to D with a cost of 1. The output port from C to D is 0. The path [A B C D] has a prefix-to-index of 50 and a cost of 3.
Statically configured tables map prefixes to registers in memory.
Registers store best paths and its attributes

destination network

prefix-to-index

link cost

port  cost  path

50  1  3  [A B C D]

maps prefixes to registers

statically configured

only store the best path and its attributes

forwarding state

stored in registers

Registers store best paths and its attributes.
Switches periodically advertise vectors to neighbors.
Switches periodically advertise vectors to neighbors

Switches periodically advertise vectors to neighbors.
Switches periodically advertise vectors to neighbors

Switches periodically advertise vectors to neighbors.
Computing new forwarding state after a link failure

prefix-to-index

link cost

statically configured

forwarding state stored in registers
Computing new forwarding state after a link failure

- **Destination path:**
  - A: 1
  - B: 1
  - C: 1
  - D: 0

- **Cost:**
  - A → B: 10
  - B → C: 1
  - C → D: 50

- **Link failure:** Between B and C.

- **Forwarding state:**
  - Statically configured:
    - Prefix-to-index
    - Link cost
  - Dynamically computed:
    - Forwarding state stored in registers

- **Example:**
  - Prefix: A
    - Port: 50
    - Cost: 10
    - Path: [A B C D]

- **Output port:**
  - A: 1
  - B: ∞
  - C: Ø
  - D: 0

- **Computation:**
  - After a link failure, compute a new forwarding state.
Computing new forwarding state after a link failure

- Data-plane-generated path-vector
- Link failure
- Output port
- Prefix-to-index
- Link cost
- Statically configured
- Dynamically computed
- If \((10 + 0) < \infty\)
- Forwarding state stored in registers
Does it actually work?
Does it actually work?
Yes!
Hardware-Accelerated P4 prototype

Implementation

- Implemented in P4\textsubscript{16}
- Compiled it to bmv2
- 2000 lines of P4 code

Capabilities

- Intra-domain destinations
  - path-vector routing
- Inter-domain destinations
  - BGP-like route selection
We tested our implementation in a real case study
Only the internal switches run the hardware-based control plane
Each switch is connected to an external peer or customer.
We generate two TCP flows from AS1 and AS2
Monitor traffic before the failure

Traffic S1 - AS3

Bandwidth [Mbps]

0 4.8 15 25

0 2 4 6 8 10

time [s]
Internal link fails and triggers the path-vector algorithm

Traffic S1- AS3

(1) internal Link failure

S2 to S3 link failure
Internal link fails and triggers the path-vector algorithm
External link failure triggers a prefix withdrawal

Traffic S1-AS3

Bandwidth [Mbps]

S2 to S3 link failure

With withdrawal

Time [s]
Network computes new egress and applies new policies

Traffic S5-AS5

Bandwidth [Mbps]

withdrawal

Bandwidth [Mbps]

withdrawal

Network computes new egress and applies new policies

Traffic S5-AS5
is it a good idea?

hardware based CP
Programmable hardware is great but… not limitless
Programmable hardware is great but... not **limitless**

Some tasks **cannot** be offloaded
Others might not be even **desirable**!
Programmable hardware is great but... not **limitless**

Some tasks **cannot** be offloaded  
Others might not be even **desirable**!

- **Reliable protocols**  
  e.g. TCP would require too many resources!

- **Poor scalability of control plane tasks**  
  hardware memory is scare and expensive
Can we have the best of both worlds?
Can we have the best of both worlds?

HW/SW codesign
Hardware-software codesign

Specification

Optimization

Synthesis

functions

$Cost_i(\cdot) \quad \forall i: \quad \text{pred}(i) < 100$

Performance$_i(\cdot)$
Hardware-software codesign

Specification

- Problem graph
- Mapping set
- Architecture graph

- Functions
  - $Cost_i(.)$
  - $Performance_i(.)$

- Constraints
  - $\forall i: pred(i) < 100$

Optimization

- $\min \sum_{i=1}^{n} Cost_i(.)$
- $\max \sum_{i=1}^{n} Performance_i(.)$

Synthesis

- $\text{cost}(x): 120$
- $\text{perf}(x): 200$

- $\text{cost}(y): 80$
- $\text{perf}(y): 200$
Hardware-software codesign

**Specification**

- Problem graph
- Mapping set
- Architecture graph

- Functions
  - Cost_i(.)
  - Performance_i(.)

- Constraints
  - \( \forall i: \text{pred}(i) < 100 \)

**Optimization**

- Cost(x): 120
- Perf(x): 200
- Cost(y): 80
- Perf(y): 200

- \( \min \sum_{i=1}^{n} \text{Cost}_i(.) \)
- \( \max \sum_{i=1}^{n} \text{Performance}_i(.) \)

**Synthesis**

- Configurations
- C/C++
- Software
- Hardware
- Runtime API
- P4 code
- Configurations
- Hardware
We identified an unexploited opportunity
We identified an unexploited opportunity

We showed that programmable data planes can run control plane tasks
We identified an unexploited opportunity

We showed that programmable data planes can run control plane tasks

We plan on leveraging HW/SW codesign to explore design tradeoffs