P4 for Measurement Purposes

Stateful Adventures on Tofino

RoN++, SURFnet, Utrecht
December 3, 2018

Luuk Hendriks,
Jeroen van Ingen

UNIVERSITY
OF TWENTE.

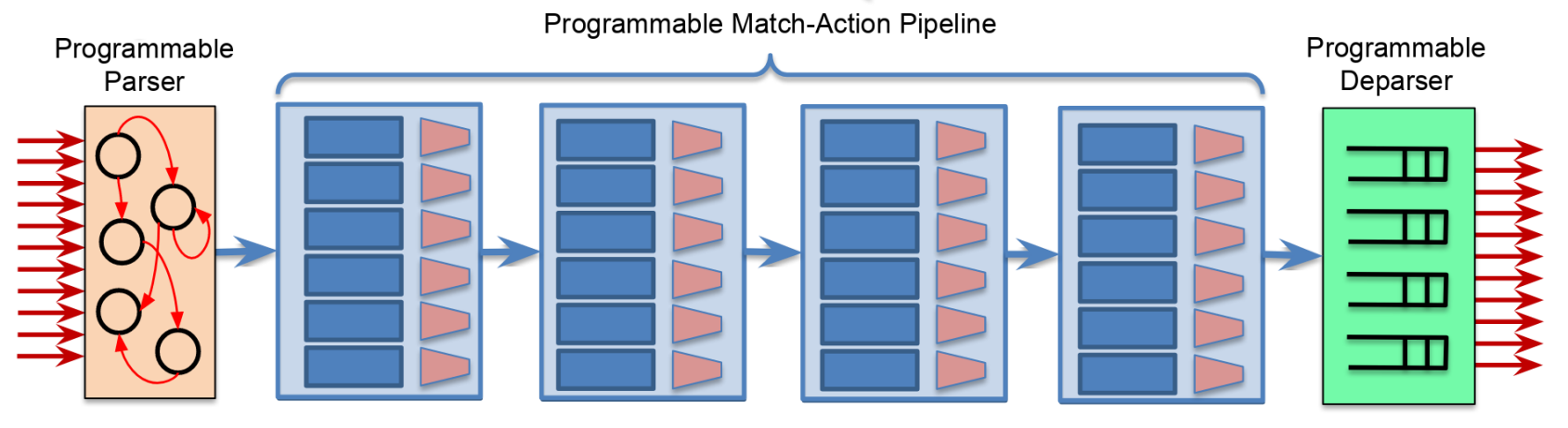
The Plan

- We like flow measurements: NetFlow, IPFIX
- Measurements using OpenFlow: not a success
- Is P4 a viable option?
- Goal: investigate P4's capabilities w.r.t. flow measurements in terms of accuracy and performance

P4 101

- Programmable data plane, instead of closed ASIC
- Describe how a packet should be processed
- P4: the Domain Specific Language (DSL) to do this
- Two flavours: p4_14 and p4_16

P4 pipeline



P4 pipeline architecture, showing the path every packet traverses through a P4-capable device [1]

[1] https://github.com/p4lang/tutorials/blob/master/P4_tutorial.pdf

Testbed

Components

- Barefoot Tofino based Wedge 100B switch (33x100Gbps)
- Netronome 2x10G SmartNIC
- Intel 2x40G NIC
- Generic x86 server to house both NICs

Testbed

Components

- Barefoot Tofino based Wedge 100B switch (33x100Gbps)
- Netronome 2x10G SmartNIC
- Intel 2x40G NIC
- Generic x86 server to house both NICs

Experience

- Hardware (compatibility) issues:
  - Netronome clashed with server
  - Netronome NIC went FUBAR, reflashed with help from Netronome
  - Intel NIC has firmware troubles, not 40G capable anymore
- No other 100G device yet

Testbed

Components

- Barefoot Tofino based Wedge 100B switch (33x100Gbps)
- Netronome 2x10G SmartNIC
- Intel 2x40G NIC
- Generic x86 server to house both NICs

Experience

- Hardware (compatibility) issues:
  - Netronome clashed with server
  - Netronome NIC went FUBAR, reflashed with help from Netronome
  - Intel NIC has firmware troubles, not 40G capable anymore
- No other 100G device yet

$\rightarrow$ focussed on Barefoot Tofino, starting with p4_14

Meanwhile, in BMv2

- Student working on stateful programming using the BMv2 model [2]
- Goal: assess filtering methods for DDoS scenarios
- Implemented History-Based IP filtering (HIF)

[2] https://github.com/JJK96/P4-filtering

Meanwhile, in BMv2

- Student working on stateful programming using the BMv2 model [2]
- Goal: assess filtering methods for DDoS scenarios
- Implemented History-Based IP filtering (HIF)

Takeaways:

- BMv2: 'unlimited' registers, nice for exploring, but not representative for hardware-based solutions
- **Hash collisions** turned out to be a problem in this work, causing inaccuracies

[2] https://github.com/JJK96/P4-filtering

Flow measurements: MA-tables

Concepts

- Define a Match-Action **`table`** where the matching is done based on the desired flow-key (e.g., the classic 5-tuple)
- Define a **`Counter`**, e.g. of type **`packets_and_bytes`**, and connect it to the table

Flow measurements: MA-tables

Concepts

- Define a Match-Action **`table`** where the matching is done based on the desired flow-key (e.g., the classic 5-tuple)
- Define a **`Counter`**, e.g. of type **`packets_and_bytes`**, and connect it to the table
- MA table entries can only be written from the control plane!

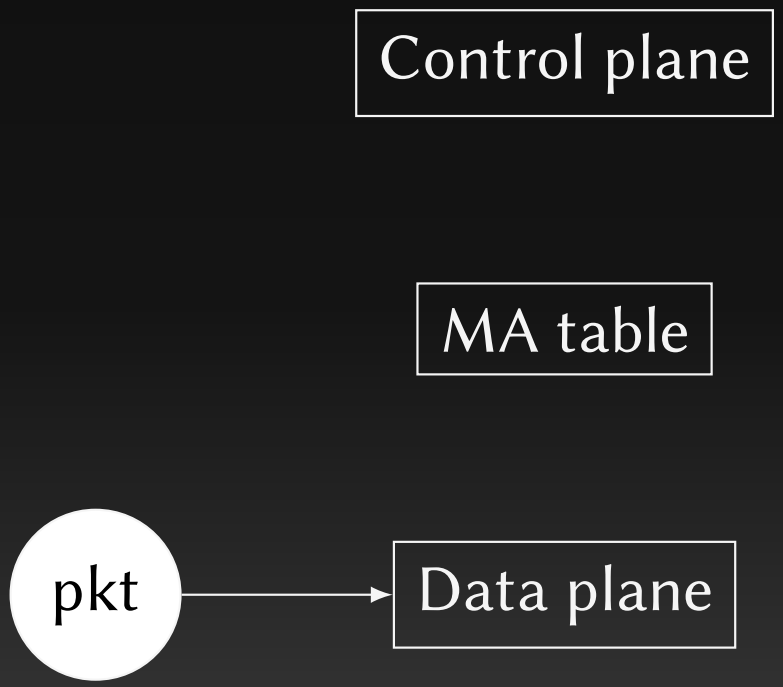
$\Rightarrow$ Leverage **`digest`**s to communicate with the control plane

Flow measurements: MA-tables

Visualisation

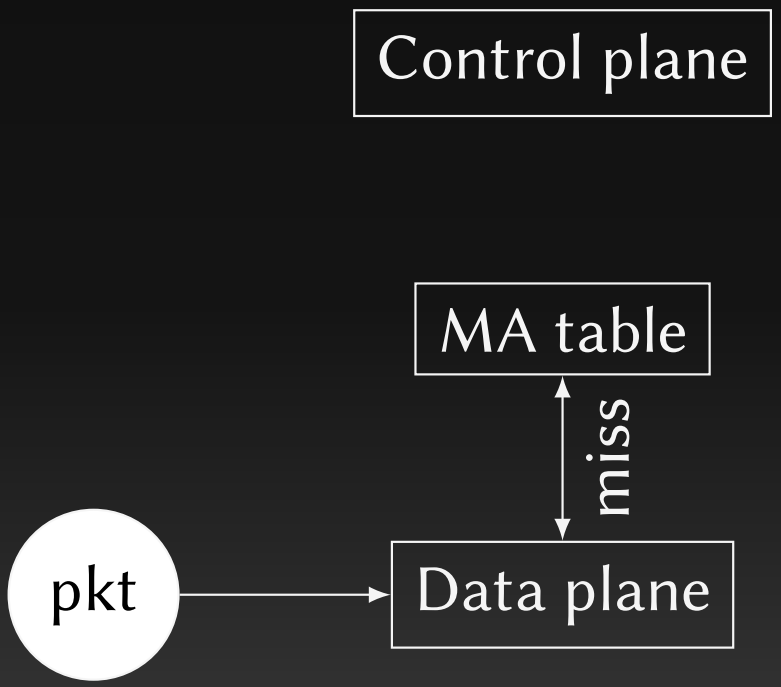
Flow measurements: MA-tables

Visualisation



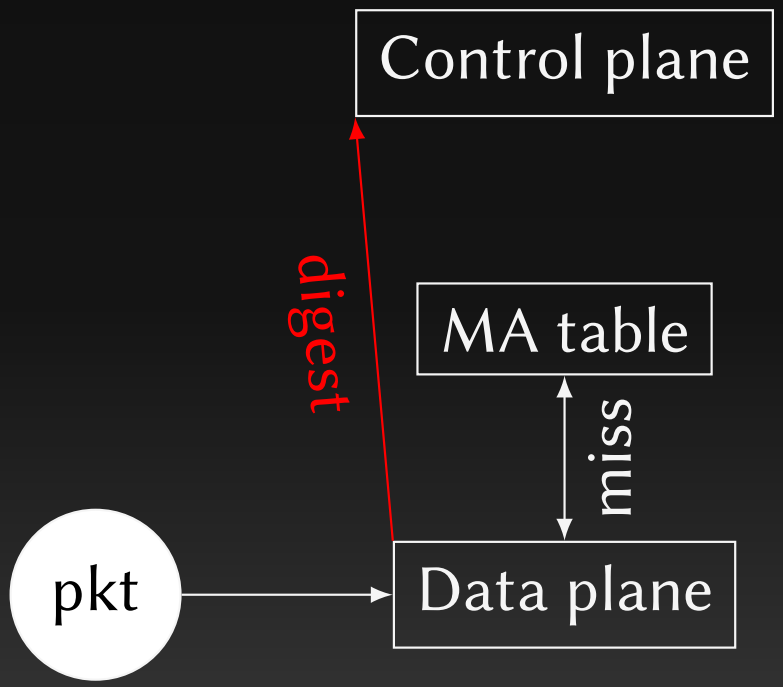
Flow measurements: MA-tables

Visualisation



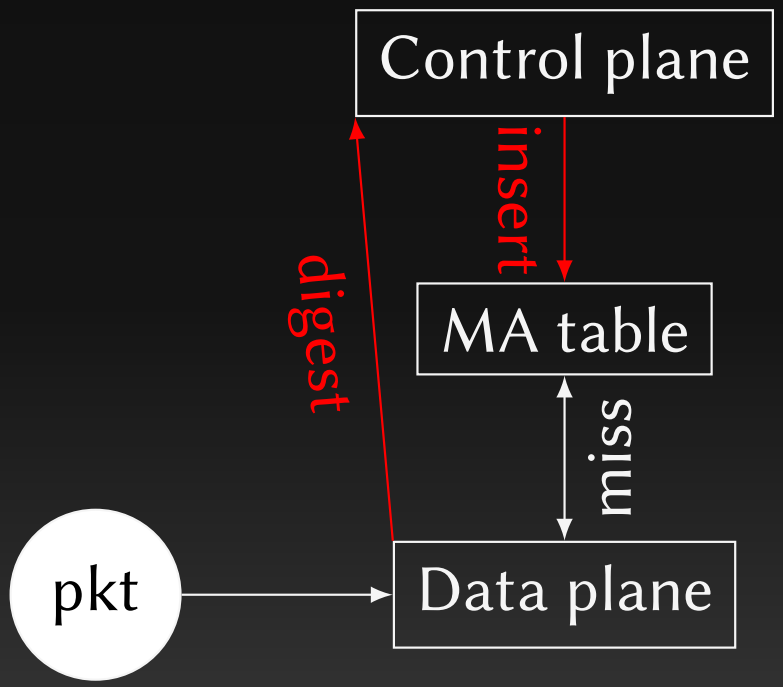
Flow measurements: MA-tables

Visualisation



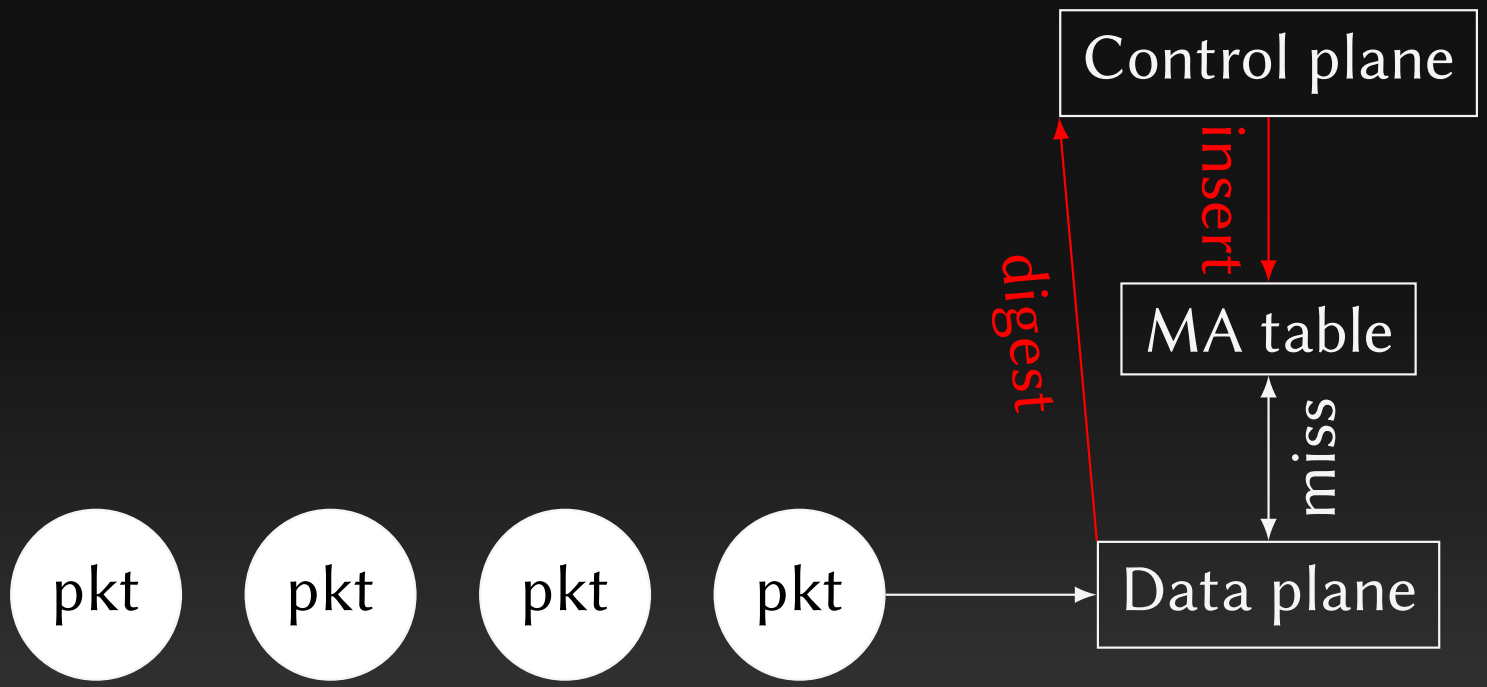
Flow measurements: MA-tables

Visualisation



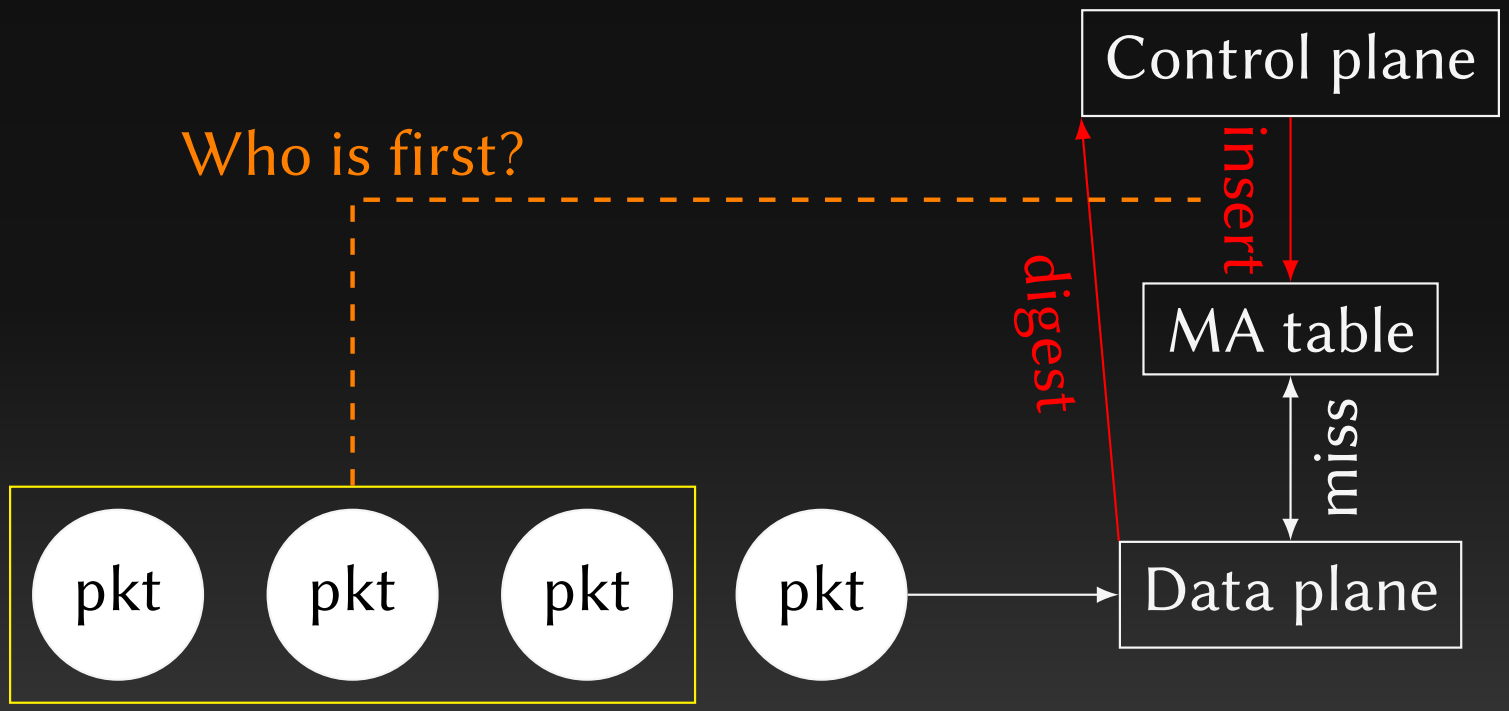
Flow measurements: MA-tables

Visualisation



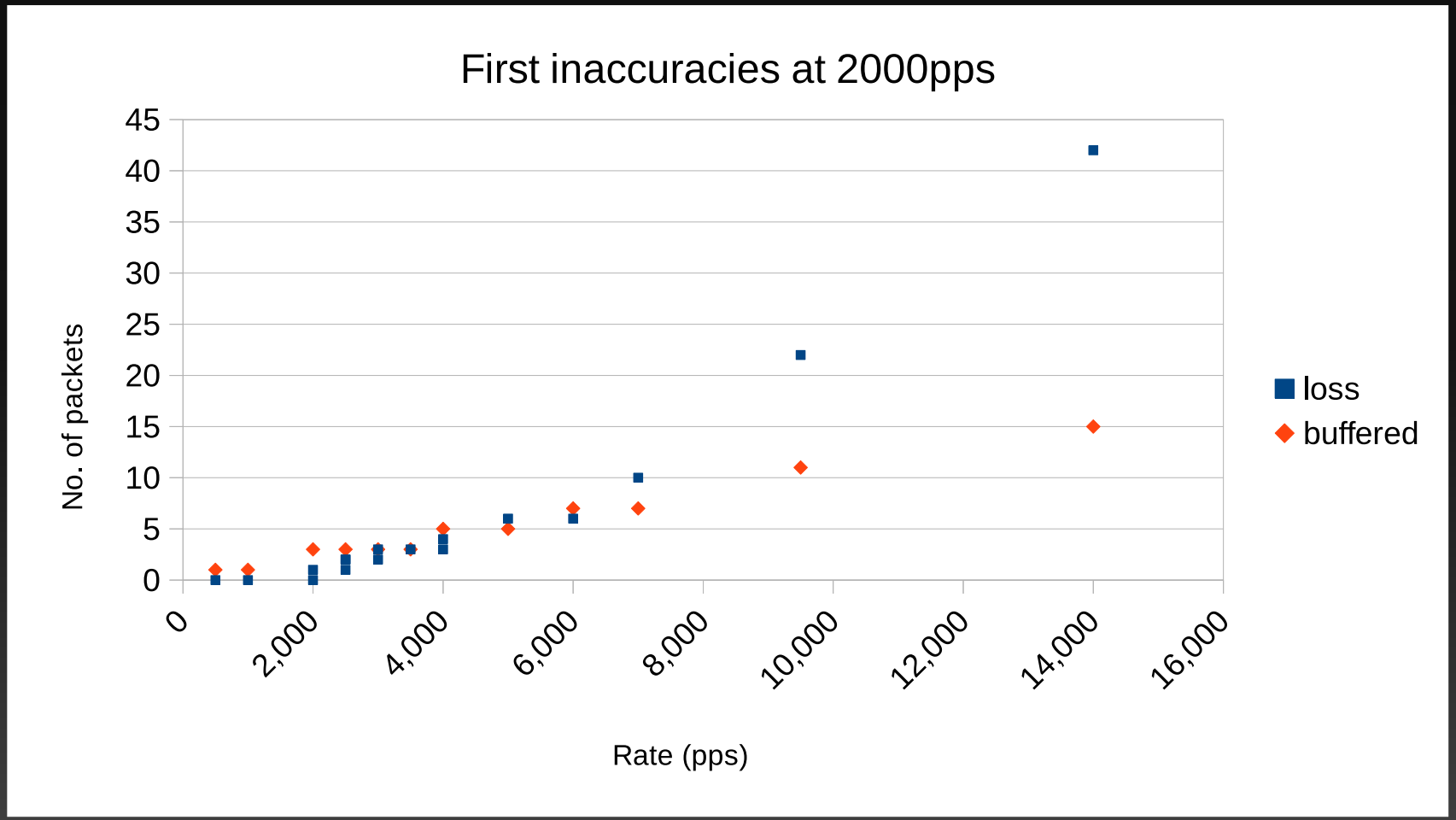
Flow measurements: MA-tables

Visualisation



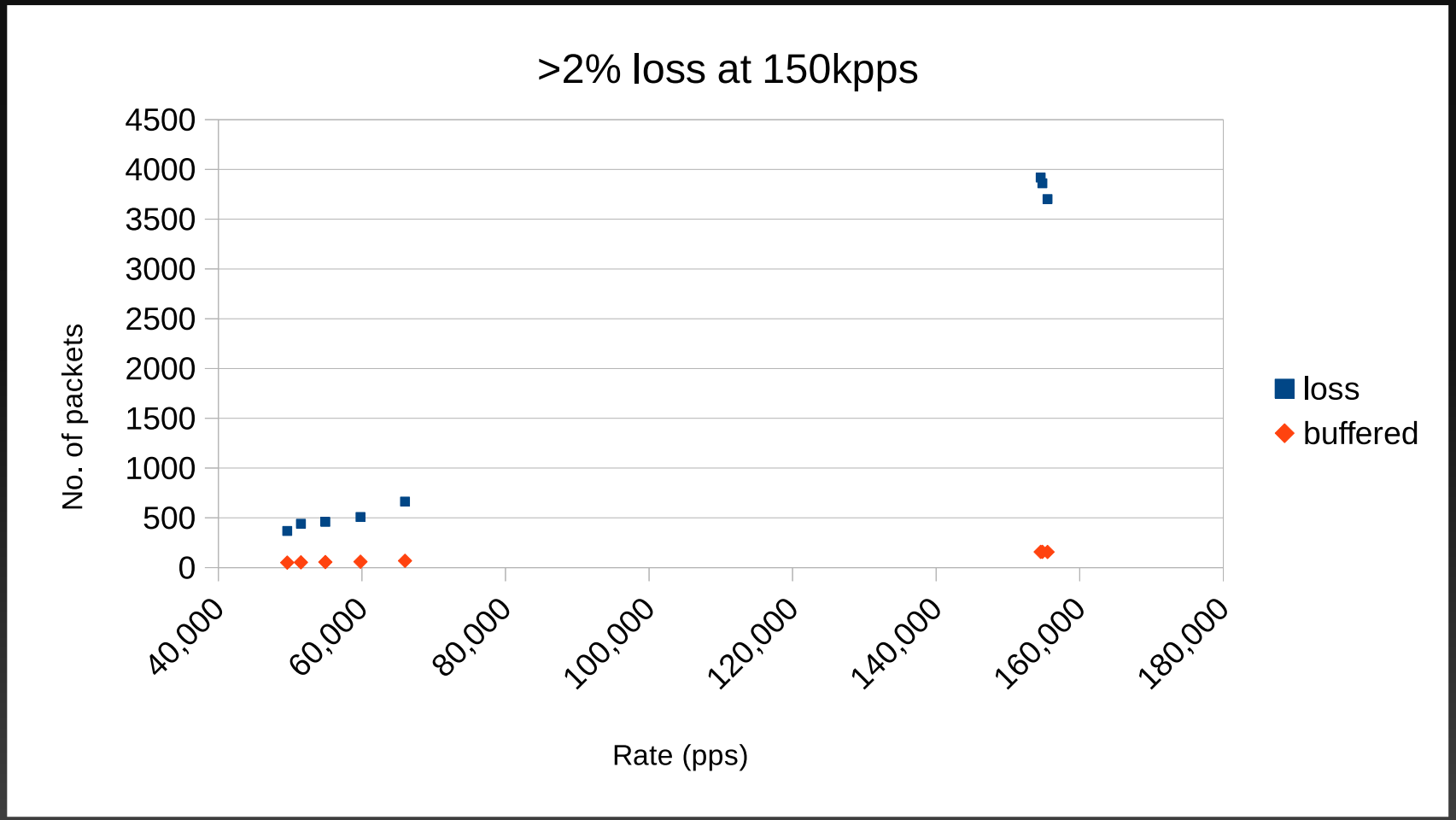
Flow measurements: MA-tables

Evaluation: quantification of ‘lost’ packets



Flow measurements: MA-tables

Evaluation: quantification of ‘lost’ packets



Flow measurements: MA-tables

Evaluation

- Even with a digest buffer in the control plane, we miss a significant number of packets
- Possibly lost digests?
- Improvements: C API instead of Python?
  NOTIFY vs **POLL** for the digests?

Flow measurements: MA-tables

Evaluation

- Even with a digest buffer in the control plane, we miss a significant number of packets
- Possibly lost digests?
- Improvements: C API instead of Python?
  NOTIFY vs **POLL** for the digests?

→ Slow path unlikely to be capable enough to perform line rate measurements, whatever we try

Flow measurements: MA-tables

Evaluation

- Even with a digest buffer in the control plane, we miss a significant number of packets
- Possibly lost digests?
- Improvements: C API instead of Python?
  NOTIFY vs **POLL** for the digests?

→ Slow path unlikely to be capable enough to perform line rate measurements, whatever we try

- Limited number of entries ($\sim$10k) in MA-table

Flow measurements: MA-tables

Conclusions

Pro

- Based purely on p4(_14) specified concepts
- Flexible in terms of flow-keys

Con

- Only packet and bytes
- Needs the control plane:
  - slow
  - vendor-specific API
- Limited no. of flow entries

Flow measurements: registers

Concepts

- No interaction with CPU/control plane: ‘stay in the data plane’

[3] https://github.com/jsonch/p4_code

Flow measurements: registers

Concepts

- No interaction with CPU/control plane:
  'stay in the data plane'
- Keep statistics in registers, indexed based on hashes
- **Embrace the hash collision** to trigger early export[3]

[3] https://github.com/jsonch/p4_code

Flow measurements: registers

Concepts

- No interaction with CPU/control plane: 'stay in the data plane'
- Keep statistics in registers, indexed based on hashes
- **Embrace the hash collision** to trigger early export[3]
- Insert a custom header with the statistics
- Final aggregation done 'externally', i.e. a different machine

→ Reduce PPS and BPS, possible measuring a 100Gbps link via a 10G link

[3] https://github.com/jsonch/p4_code

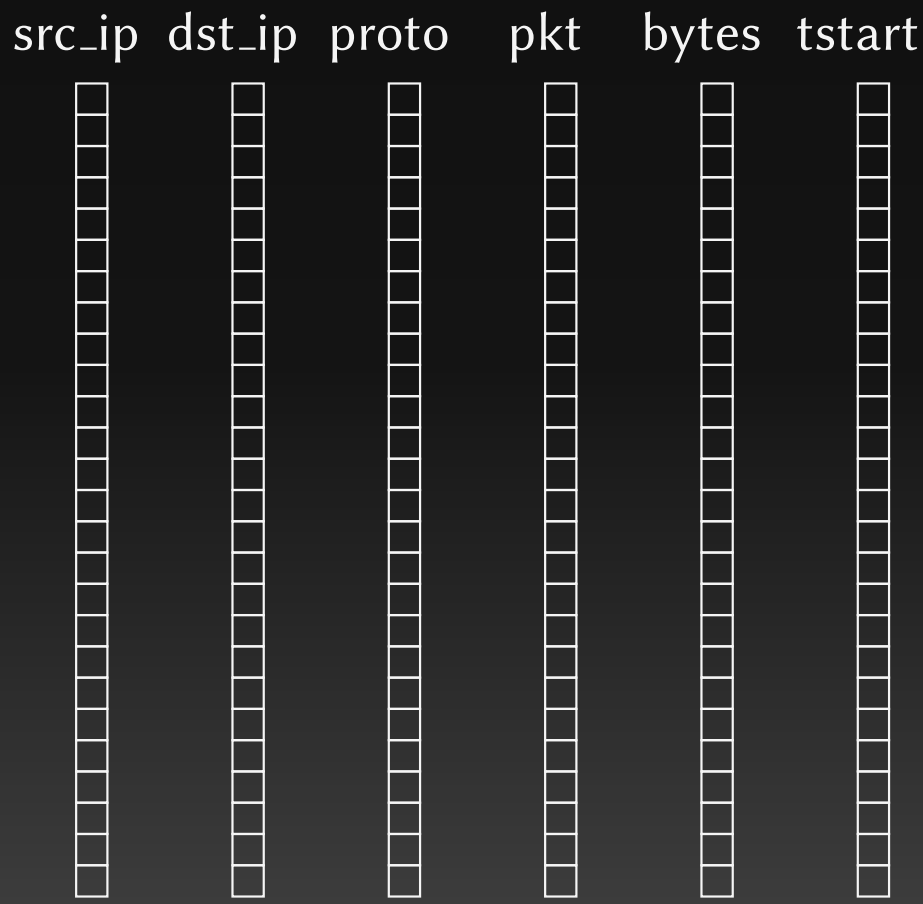
Flow measurements: registers

Hash $\approx$ flow-key

- Calculate hash based on 'flow-key', i.e. 5-tuple
- Update multiple registers using this hash:
  - `reg_srcip`, `reg_dstip`, `reg_proto`, etc
  - `reg_packets`
  - `reg_bytes`
  - `reg_tstart`
- Determine whether a collision occurs via the `srcip` (etc.) registers.
- Collision? Export what is in the registers, start over with the new (colliding) flow.

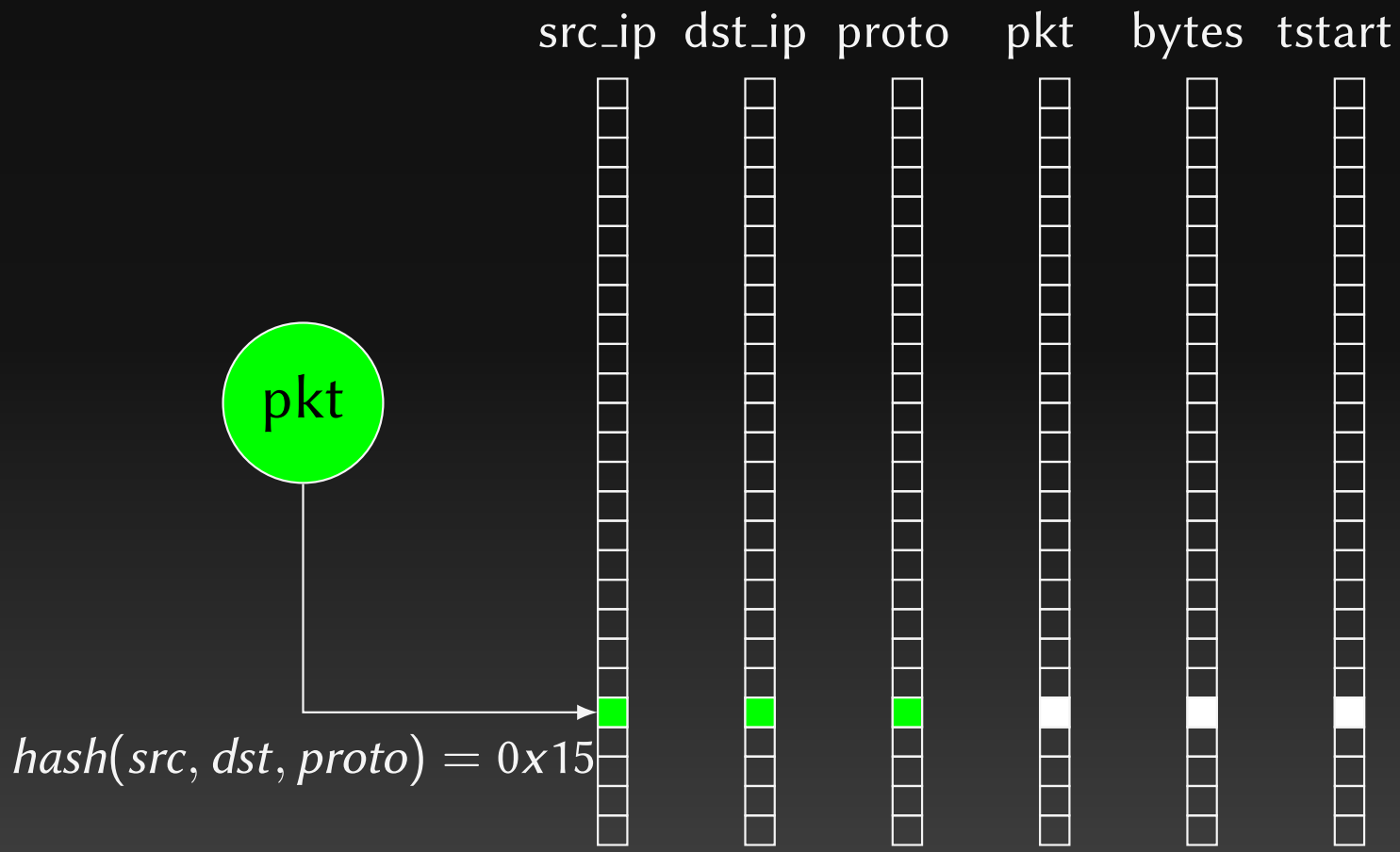
Flow measurements: registers

Visualisation



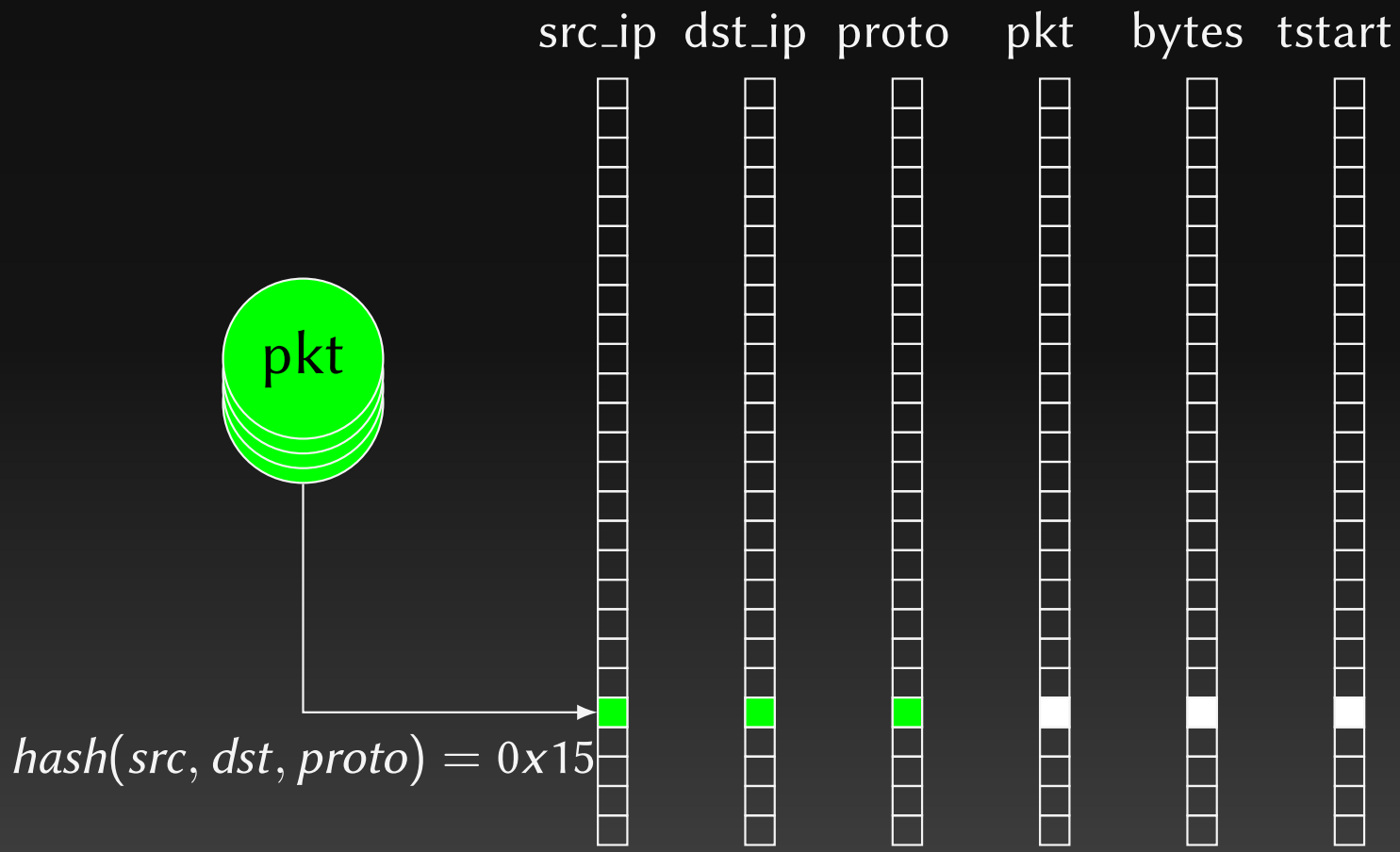
Flow measurements: registers

Visualisation



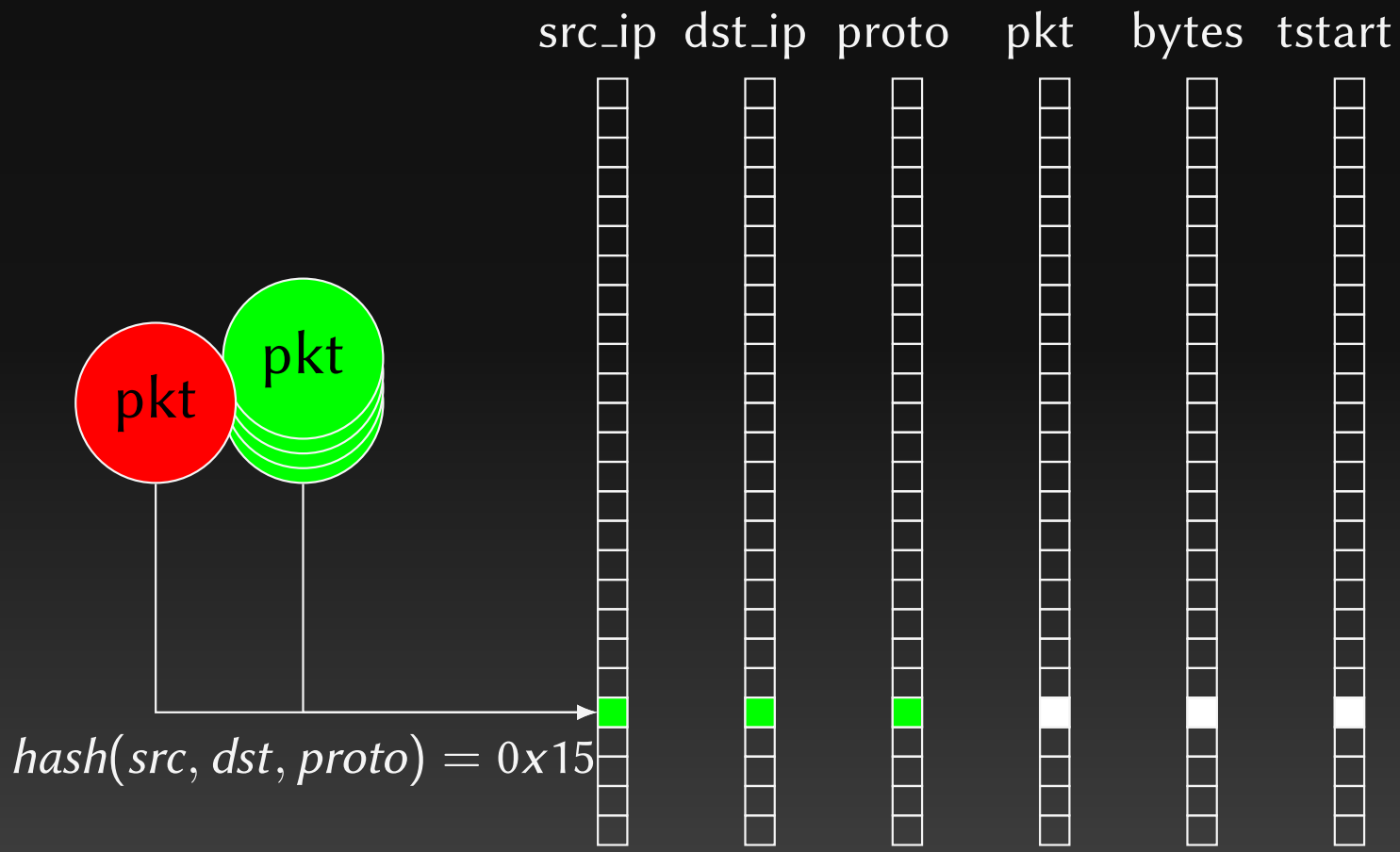
Flow measurements: registers

Visualisation



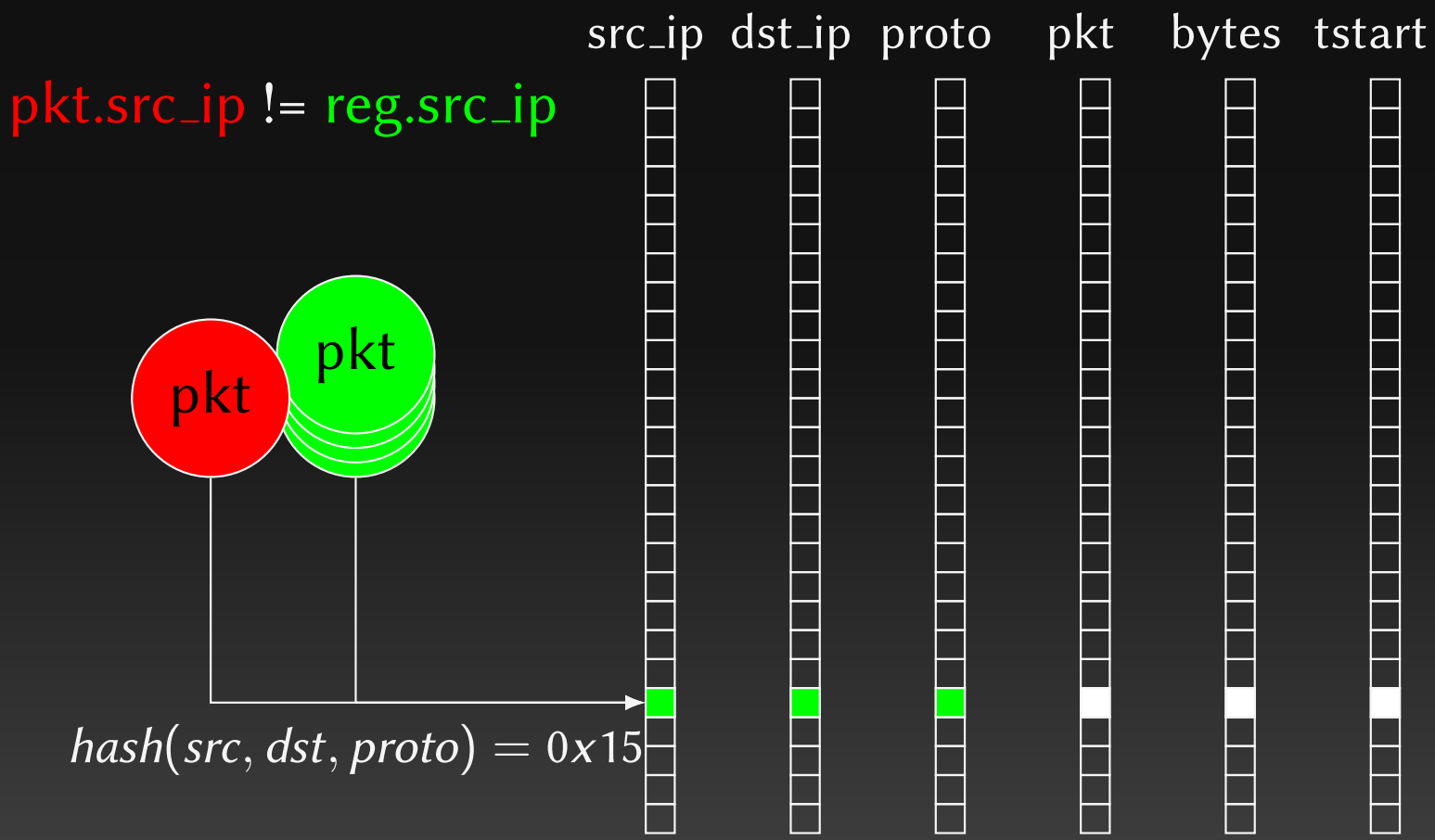
Flow measurements: registers

Visualisation



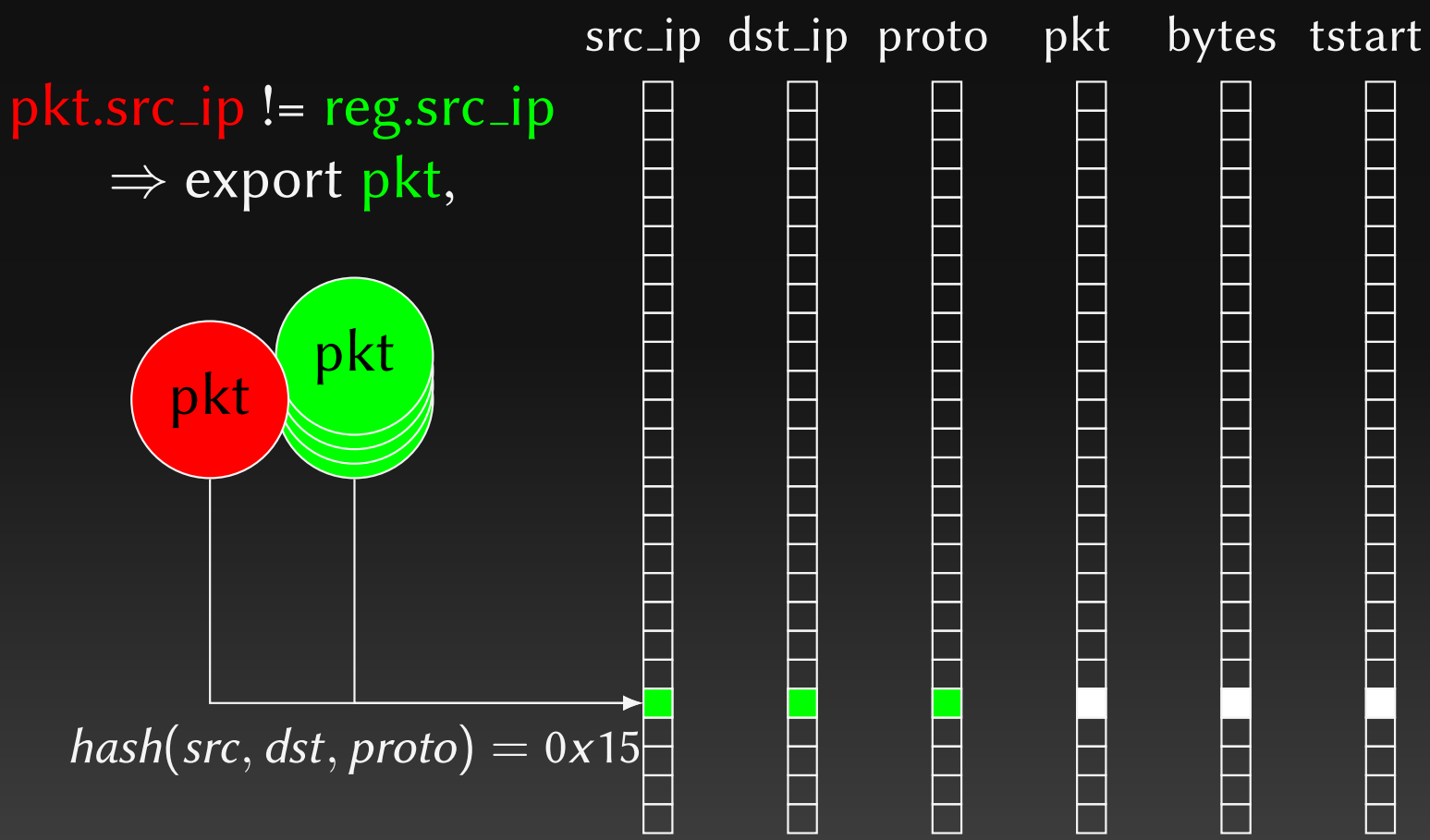
Flow measurements: registers

Visualisation



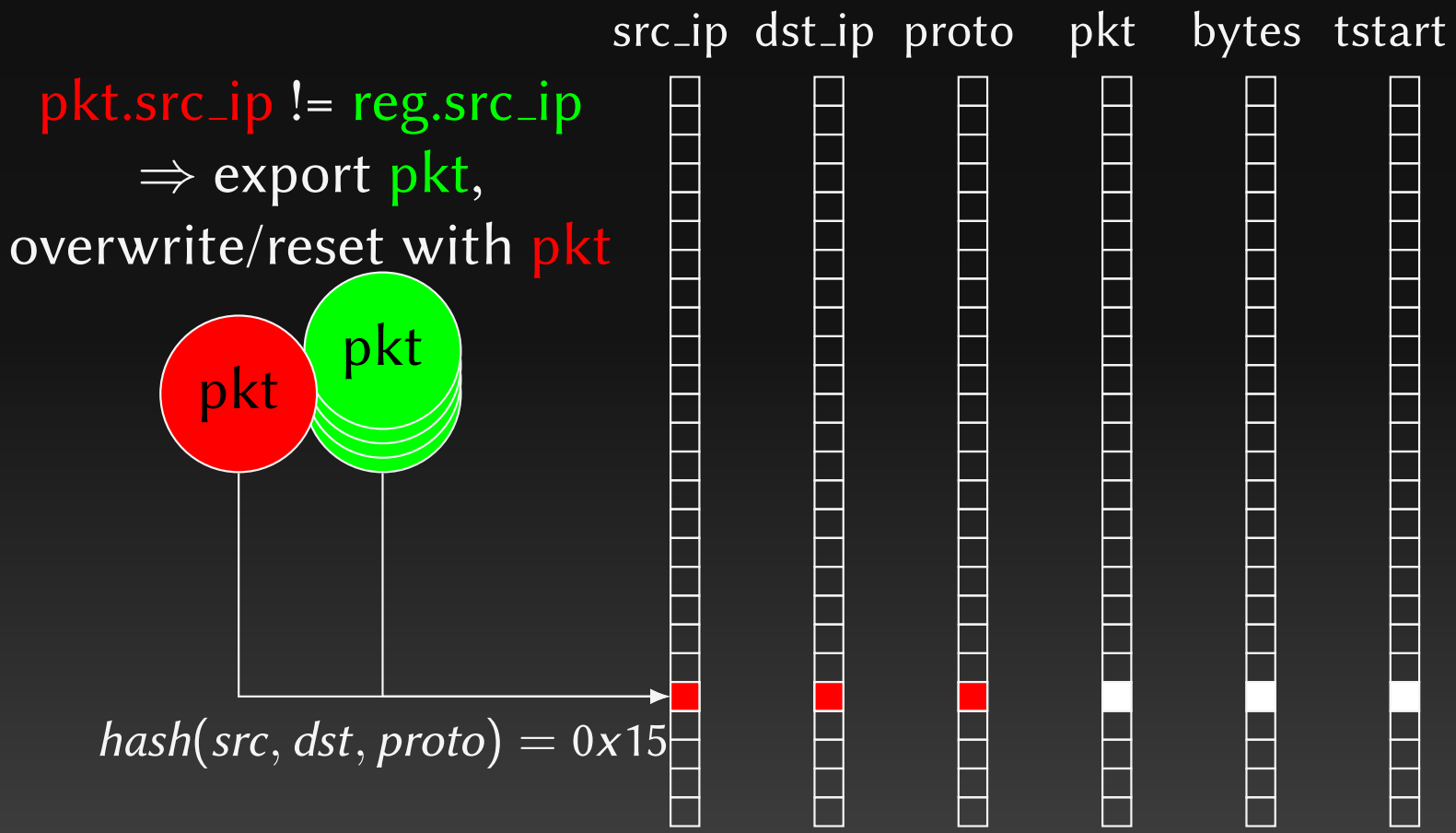
Flow measurements: registers

Visualisation



Flow measurements: registers

Visualisation



Flow measurements: registers

(Looking for) Problems

- Existing work is based on Netronome, in p4_14
- p4_16 support in latest Barefoot SDE!

Flow measurements: registers

(Looking for) Problems

- Existing work is based on Netronome, in p4_14
- p4_16 support in latest Barefoot SDE!

New in p4_16 (w.r.t. registers):

- Abstraction (**`extern`**) seemingly allows easier implementation of the ALU **`blackbox`** operations.
- Unfortunately, as of yet this results in unclear error messages containing references to external/invisible code

Flow measurements: registers

(Looking for) Problems

- Existing work is based on Netronome, in p4_14
- p4_16 support in latest Barefoot SDE!

New in p4_16 (w.r.t. registers):

- Abstraction (**`extern`**) seemingly allows easier implementation of the ALU **`blackbox`** operations.
- Unfortunately, as of yet this results in unclear error messages containing references to external/invisible code

```
error:
Incompatible outputs in RegisterAction: alu_hi
and alu_lo
```

Being put off by 1

```
error: expression too complex for stateful alu
read_pkt = flow.pkt + 1;
```

Flow measurements: registers

Experiences while developing

- Fiddling with **`apply`** / **`table`**s / **`action`**s / **`RegisterAction`**s
- p4c can only compile into ‘single stage actions’
- p4_16 support is really new, hopefully things improve

→ Use the available TNA model while developing!

Flow measurements: registers

Experiences while developing

- Fiddling with **`apply`** / **`table`**s / **`action`**s / **`RegisterAction`**s
- p4c can only compile into ‘single stage actions’
- p4_16 support is really new, hopefully things improve

→ Use the available TNA model while developing!

- Hash collision trick not working with current p4c

Flow measurements: registers

Experiences while developing

- Fiddling with **`apply`** / **`table`**s / **`action`**s / **`RegisterAction`**s
- p4c can only compile into ‘single stage actions’
- p4_16 support is really new, hopefully things improve

→ Use the available TNA model while developing!

- Hash collision trick not working with current p4c

On the positive side:

- Managed to implement packets, bytes, **start/end times**
- Export is easy with the new deparser in p4_16

Flow measurements: registers

(Concessions for) Evaluation

- Export on every third packet:
  - Simply add in our statistics header
- Forward everything out of one interface

$\rightarrow$ Allow both testing of

- actual forwarding (is anything dropped?)
- aggregation on the third packets (complete, accurate statistics?)

Flow measurements: registers

Evaluation

Generated traffic + `tcpreplay`:

- Single flow, 300k packets
  (we need $N \times 3$ packets)
- Max replay speed, approx. 150kpps

Flow measurements: registers

Evaluation

Generated traffic + `tcpreplay`:

- Single flow, 300k packets
  (we need $N \times 3$ packets)
- Max replay speed, approx. 150kpps

Result:

- It works!
- Everything is forwarded.
- Statistics (packet and byte counters) are correct.

Flow measurements: registers

Conclusions

- Promising, but much future work (next slide).
- ‘If it compiles, it works at line rate’ seems to be true.
- If it compiles.

Flow measurements with P4

Conclusions

- Incorporating the control plane (i.e. MA-tables) not scalable (limited table entries, incomplete statistics)
- Register-based is (still) quite cumbersome to implement (on Tofino in p4_16).
- But register-based is the way to go:
  - complete, accurate statistics
  - allows for more stats than just packets and bytes

Flow measurements with P4

Conclusions

- Incorporating the control plane (i.e. MA-tables) not scalable (limited table entries, incomplete statistics)
- Register-based is (still) quite cumbersome to implement (on Tofino in p4_16).
- But register-based is the way to go:
  - complete, accurate statistics
  - allows for more stats than just packets and bytes

Development on Tofino:

- Maybe a bit early to develop using p4_16
- Barefoot support is great (thank you Vladimir!)

Future work

- Proper export based on hash collision
- Actual aggregation on external machine
- Find limits of flow-keys (i.e. fields passed to hash calc function)
- Quantify the no. of flows vs hash collision rate
- Determine how much we can actually reduce PPS/BPS (affected by hash collision rate)

P4 for Measurement Purposes

Stateful Adventures on Tofino

RoN++, SURFnet, Utrecht
December 3, 2018

Luuk Hendriks,
Jeroen van Ingen

UNIVERSITY
OF TWENTE.