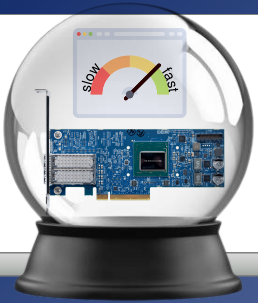


Determination of Throughput Guarantees for Processor-based SmartNICs

Johannes Krude, Jan Rüth,
Daniel Schemmel, Felix Rath,
Iohannes-Heorh Folbort, Klaus Wehrle

DFG Collaborative Research Centre 1053 – MAKI
Multi Mechanism Adaptation for the Future Internet

https://comsys.rwth-aachen.de/

CoNEXT ’21, December 7–10, 2021

COM SYS | RWTH AACHEN UNIVERSITY

Data-plane programmability needs predictable throughput

⚠ **packet drops and congestion** **testing with traffic traces gives no guarantee**

Match-Action-based Programmable Switches

+ **If a program compiles, it runs at ~1 pkt/cycle**
– **Difficult to program** (subset of P4, no loops, few sequential operations)

Data-plane programmability needs predictable throughput

⚠ **packet drops and congestion** **testing with traffic traces gives no guarantee**

Match-Action-based Programmable Switches

+ **If a program compiles, it runs at ~1 pkt/cycle**
– **Difficult to program** (subset of P4, no loops, few sequential operations)

Processor-based SmartNICs

+ **Are easier and more freely programmable** (C, BPF/XDP)
– **Performance varies and is not obvious** [our Netsoft2019 paper]

Data-plane programmability needs predictable throughput

⚠ **packet drops and congestion** **testing with traffic traces gives no guarantee**

Match-Action-based Programmable Switches

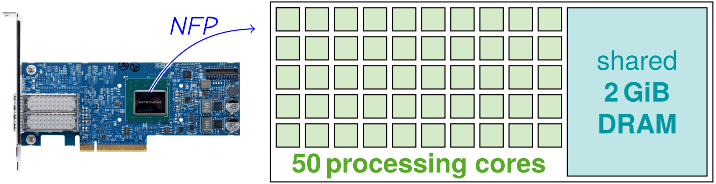
+ **If a program compiles, it runs at ~1 pkt/cycle**
– **Difficult to program** (subset of P4, no loops, few sequential operations)

Processor-based SmartNICs

+ **Are easier and more freely programmable** (C, BPF/XDP)
– ~~**Performance varies and is not obvious** [our Netsoft2019 paper]~~

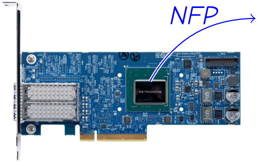
Our Contribution: A tool to calculate the throughput of a program
…while developing a program. …as part of regression tests.

- **Netronome Agilio CX 2x40 GbE**
  - Netronome Flow Processor (NFP)
  - BPF/XDP programs compiled to NFP bytecode

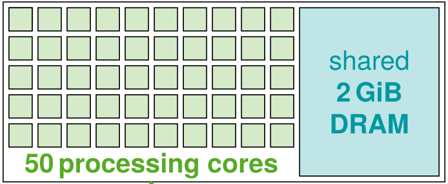


Throughput Analysis

- **Netronome Agilio CX 2x40 GbE**
  - Netronome Flow Processor (NFP)
  - BPF/XDP programs compiled to NFP bytecode



NFP

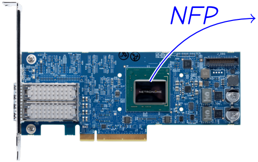


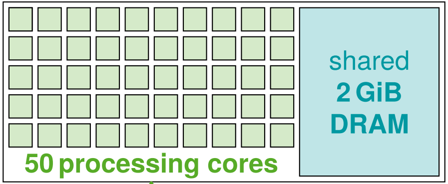

```
if (pkt.size < 100)
  return DROP;
if (pkt[ethtype] == IPV4)
  ip4_counter++;
if (pkt[ethtype] == IPV6)
  for (...) ...;
return PASS;
```

example program

Throughput Analysis

- **Netronome Agilio CX 2x40 GbE**
  - ▶ Netronome Flow Processor (NFP)
  - ▶ BPF/XDP programs compiled to NFP bytecode





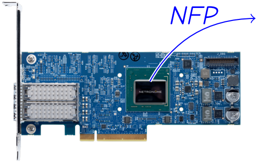

```
if (pkt.size < 100)
  return DROP;
if (pkt[ethtype] == IPV4)
  ip4_counter++;
if (pkt[ethtype] == IPV6)
  for (...) ...;
return PASS;
```

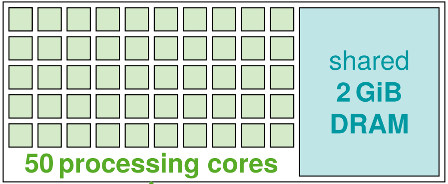
example program

- **Challenges**
  - ▶ The throughput depends on the executed instructions

Throughput Analysis

- **Netronome Agilio CX 2x40 GbE**
  - Netronome Flow Processor (NFP)
  - BPF/XDP programs compiled to NFP bytecode





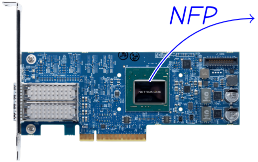

```
if (pkt.size < 100)
  return DROP;
if (pkt[ethtype] == IPV4)
  ip4_counter++;
if (pkt[ethtype] == IPV6)
  for (...) ...;
return PASS;
```

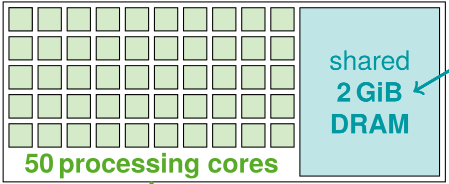
example program

- **Challenges**
  - The throughput depends on the executed instructions

Throughput Analysis

- **Netronome Agilio CX 2x40 GbE**
  - Netronome Flow Processor (NFP)
  - BPF/XDP programs compiled to NFP bytecode





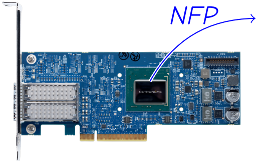

```
if (pkt.size < 100)
  return DROP;
if (pkt[ethtype] == IPV4)
  ip4_counter++;
if (pkt[ethtype] == IPV6)
  for (...) ...;
return PASS;
```

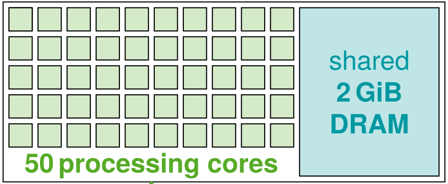
example program

- **Challenges**
  - The throughput depends on the executed instructions
  - DRAM access can be a bottleneck

Throughput Analysis

- **Netronome Agilio CX 2x40 GbE**
  - Netronome Flow Processor (NFP)
  - BPF/XDP programs compiled to NFP bytecode






```
if (pkt.size < 100)
  return DROP;
if (pkt[ethtype] == IPV4)
  ip4_counter++;
if (pkt[ethtype] == IPV6)
  for (...) ...;
return PASS;
```

example program

- **Challenges**
  - The throughput depends on the executed instructions
  - DRAM access can be a bottleneck
  - Packet sizes influence throughput

Throughput depends on executed instructions

- **Different paths cause different throughput**
  - ⇒ The slowest path establishes a lower bound

```
if (pkt.size < 100)
  return DROP;
if (pkt[ethtype] == IPV4)
  ip4_counter++;
if (pkt[ethtype] == IPV6)
  for (...) ...;
return PASS;
```

Throughput depends on executed instructions

- **Different paths cause different throughput**
  - ⇒ The slowest path establishes a lower bound
- **Not every path can be triggered by a packet**
  - ▶ Perhaps a tighter lower bound is possible
  - ⇒ Check each path for contradictions

```
if (pkt.size < 100)
  return DROP;
if (pkt[ethtype] == IPV4)
  ip4_counter++;
if (pkt[ethtype] == IPV6)
  for (...) ...;
return PASS;
```

Throughput depends on executed instructions

- **Different paths cause different throughput**
  - ⇒ The slowest path establishes a lower bound
- **Not every path can be triggered by a packet**
  - ▶ Perhaps a tighter lower bound is possible
  - ⇒ Check each path for contradictions
- **Too many paths (path explosion)**
  - ▶ Up to 2^n paths for n `if`s
  - ⇒ Only analyze the slowest paths

```
if (pkt.size < 100)
    return DROP;
if (pkt[ethtype] == IPV4)
    ip4_counter++;
if (pkt[ethtype] == IPV6)
    for (...) ...;
return PASS;
```

Throughput depends on executed instructions

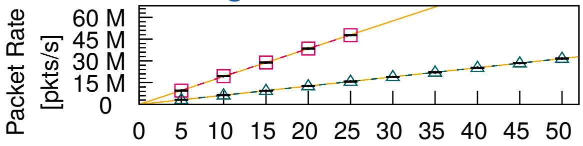
- **Different paths cause different throughput**
  - ⇒ The slowest path establishes a lower bound
- **Not every path can be triggered by a packet**
  - ▶ Perhaps a tighter lower bound is possible
  - ⇒ Check each path for contradictions
- **Too many paths (path explosion)**
  - ▶ Up to 2^n paths for *n* `ifs`
  - ⇒ Only analyze the slowest paths

```
if (pkt.size < 100)
  return DROP;
if (pkt[ethtype] == IPV4)
  ip4_counter++;
if (pkt[ethtype] == IPV6)
  for (...) ...;
return PASS;
```

How do individual instructions influence the throughput?

Processing Cost

- **Linear scaling over 50 cores**

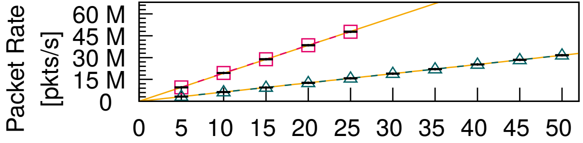


- **Most NFP instructions take 1 cycle**
  - ▶ Branches take 2-3 cycles when taken
  - ▶ Deterministic DRAM access times when not overloaded

Instruction Costs

Processing Cost

- **Linear scaling over 50 cores**



- **Most NFP instructions take 1 cycle**
  - Branches take 2-3 cycles when taken
  - Deterministic DRAM access times when not overloaded

DRAM Cost

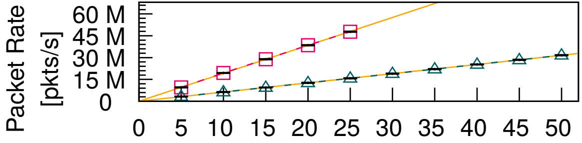
- **Too many memory instructions overload the DRAM**
- **DRAM throughput varies up to $\times 4$**
  - $\Rightarrow$ Lower bound

Atomic Inc:	$\geq$ 248M ops/s	$cost = \frac{50}{rate}$
Read32:	$\geq$ 197M ops/s	

Instruction Costs

Processing Cost

- **Linear scaling over 50 cores**



- **Most NFP instructions take 1 cycle**
  - Branches take 2-3 cycles when taken
  - Deterministic DRAM access times when not overloaded

DRAM Cost

- **Too many memory instructions overload the DRAM**
- **DRAM throughput varies up to $\times 4$**

  $\Rightarrow$ Lower bound

Atomic Inc:	$\geq$ 248M ops/s	$cost = \frac{50}{rate}$
Read32:	$\geq$ 197M ops/s	

A program path is either processing or DRAM bottlenecked

- Per instruction cost tuple: (processing cost; DRAM cost)
- Per path bottleneck: maximum over processing & DRAM cost

Analysing only the slowest paths

- **Can not enumerate all paths:** Incremental longest path algorithm [Kundu, 1994]
- **Check paths for contradictions:** Use SMT solver (similar to symbolic execution)

Analysing only the slowest paths

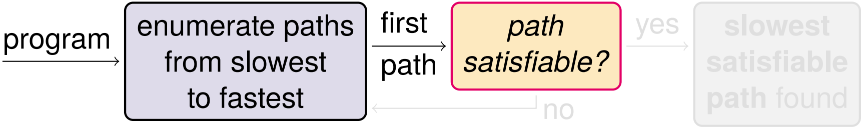
- **Can not enumerate all paths:** Incremental longest path algorithm [Kundu, 1994]
- **Check paths for contradictions:** Use SMT solver (similar to symbolic execution)

Analysing only the slowest paths

- **Can not enumerate all paths:** Incremental longest path algorithm [Kundu, 1994]
- **Check paths for contradictions:** Use SMT solver (similar to symbolic execution)

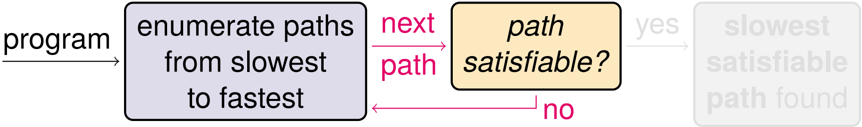
Analysing only the slowest paths

- **Can not enumerate all paths:** Incremental longest path algorithm [Kundu, 1994]
- **Check paths for contradictions:** Use SMT solver (similar to symbolic execution)



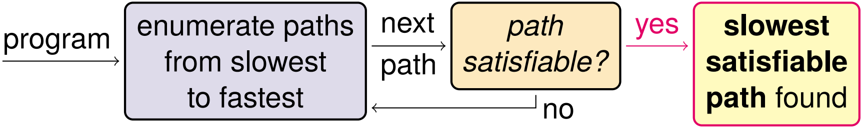
Analysing only the slowest paths

- **Can not enumerate all paths:** Incremental longest path algorithm [Kundu, 1994]
- **Check paths for contradictions:** Use SMT solver (similar to symbolic execution)



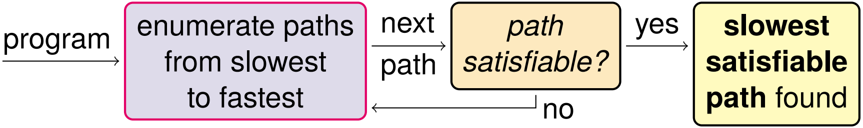
Analysing only the slowest paths

- **Can not enumerate all paths:** Incremental longest path algorithm [Kundu, 1994]
- **Check paths for contradictions:** Use SMT solver (similar to symbolic execution)



Analysing only the slowest paths

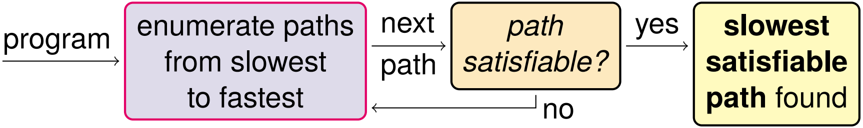
- **Can not enumerate all paths:** Incremental longest path algorithm [Kundu, 1994]
- **Check paths for contradictions:** Use SMT solver (similar to symbolic execution)



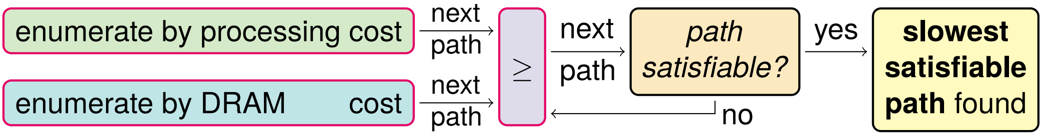
- **Two separate cost functions (processing cost; DRAM cost)**

Analysing only the slowest paths

- **Can not enumerate all paths:** Incremental longest path algorithm [Kundu, 1994]
- **Check paths for contradictions:** Use SMT solver (similar to symbolic execution)

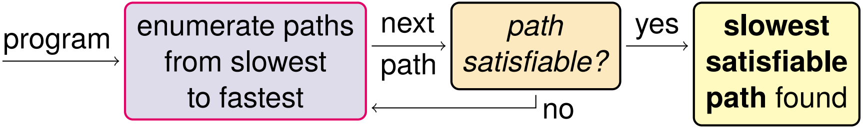


- **Two separate cost functions (processing cost; DRAM cost)**

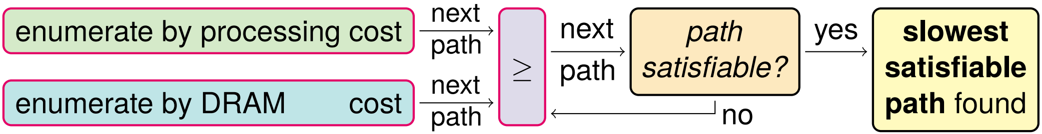


Analysing only the slowest paths

- **Can not enumerate all paths:** Incremental longest path algorithm [Kundu, 1994]
- **Check paths for contradictions:** Use SMT solver (similar to symbolic execution)



- **Two separate cost functions (processing cost; DRAM cost)**



Determines packet-rate guarantees

Packet- vs. Bit-Rate Guarantees

- **Bit-Rate throughput influenced by**
  - ▶ The time it takes to process a packet
  - ▶ The size of the packet

Programs which process fixed sized headers

- **Processing mostly independent from packet size**

  $\rightarrow$ Packet-rate guarantees

Programs which loop over multiple headers and payload

- **Processing depends on packet sizes**

  $\rightarrow$ Bit-rate guarantees

- **Static program analyis to identify packet size classes**

Example Program

```
if (pkt.size < 100)
  return DROP;
if (...) ...;
if (...) ...;
return PASS;
```

Bit-Rate Guarantees & Packet Sizes

- **Static program analyis to identify packet size classes**

Example Program

```
if (pkt.size < 100)
    return DROP;
if (...) ...;
if (...) ...;
return PASS;
```

Bit-Rate Guarantees & Packet Sizes

- **Static program analyis to identify packet size classes**

Example Program	pkt.size: 60-99	pkt.size $\geq$ 100
`if (pkt.size < 100)` `return DROP;` `if (...) ...;` `if (...) ...;` `return PASS;`		

Bit-Rate Guarantees & Packet Sizes

- **Static program analyis to identify packet size classes**

Example Program	pkt.size: 60-99	pkt.size $\geq$ 100

`if (pkt.size < 100)`
`  return DROP;`
`if (...) ...;`
`if (...) ...;`
`return PASS;`
	`if (pkt.size < 100)`
	`  return DROP;`	

Bit-Rate Guarantees & Packet Sizes

- **Static program analyis to identify packet size classes**

<table>
<tr><th>Example Program</th><th>pkt.size: 60-99</th><th>pkt.size $\geq$ 100</th></tr>
<tr><td><pre>if (pkt.size < 100)
  return DROP;
if (...) ...;
if (...) ...;
return PASS;</pre></td><td><pre>if (pkt.size < 100)
  return DROP;</pre></td><td><pre>if (pkt.size < 100)

if (...) ...;
if (...) ...;
return PASS;</pre></td></tr>
</table>

Bit-Rate Guarantees & Packet Sizes

- **Static program analyis to identify packet size classes**

Example Program	pkt.size: 60-99	pkt.size $\geq$ 100

`if (pkt.size < 100)`
`  return DROP;`
`if (...) ...;`
`if (...) ...;`
`return PASS;`
	`if (pkt.size < 100)`
	`  return DROP;`
		`if (pkt.size < 100)`
		
		`if (...) ...;`
		`if (...) ...;`
		`return PASS;`

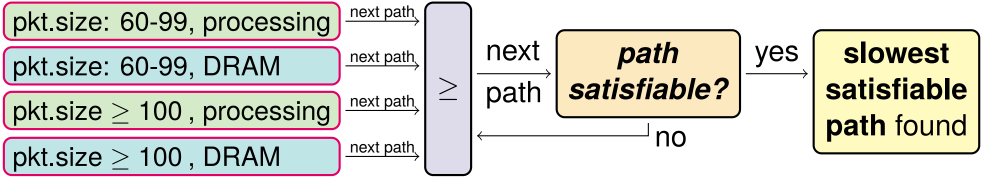
Bit-Rate Guarantees & Packet Sizes

- **Static program analyis to identify packet size classes**

Example Program	pkt.size: 60-99	pkt.size $\geq$ 100

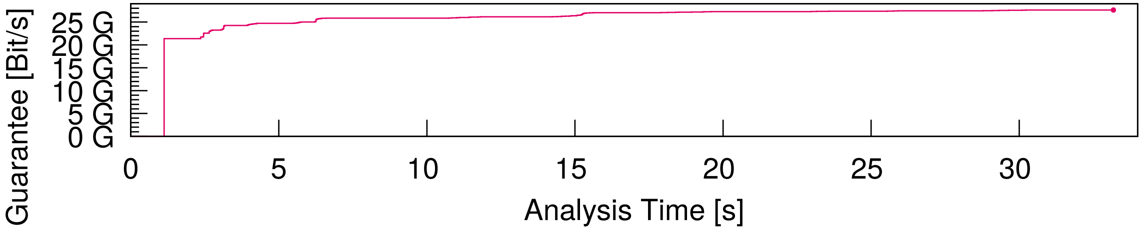
`if (pkt.size < 100)`
`  return DROP;`
`if (...) ...;`
`if (...) ...;`
`return PASS;`
	`if (pkt.size < 100)`
	`  return DROP;`
		`if (pkt.size < 100)`
		
		`if (...) ...;`
		`if (...) ...;`
		`return PASS;`

- **Enumerate paths ordered by bit-rate**



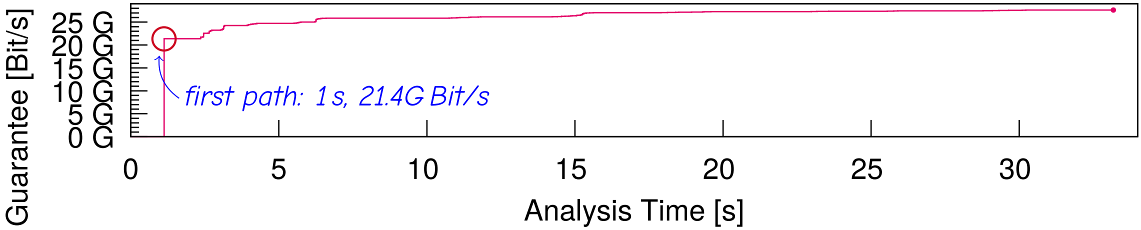
Evaluation: Analysis Time

Example Program: QUIC Load Balancer with IPv6 Option parsing



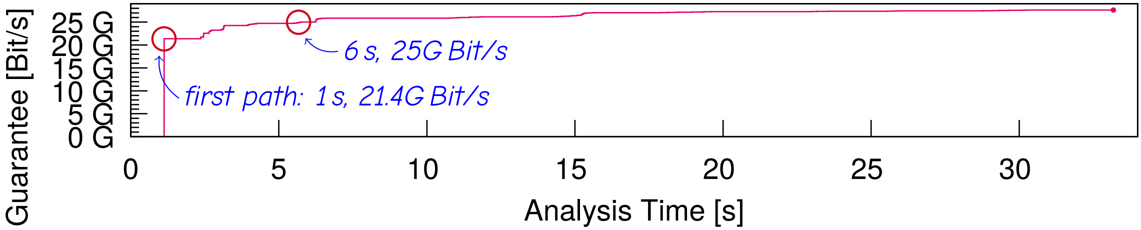
Evaluation: Analysis Time

Example Program: QUIC Load Balancer with IPv6 Option parsing



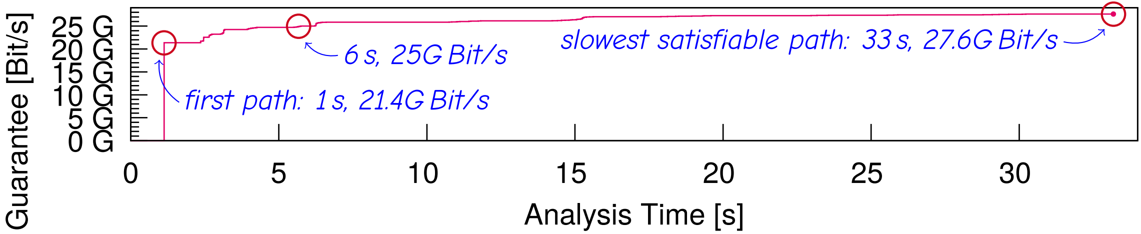
Evaluation: Analysis Time

Example Program: QUIC Load Balancer with IPv6 Option parsing

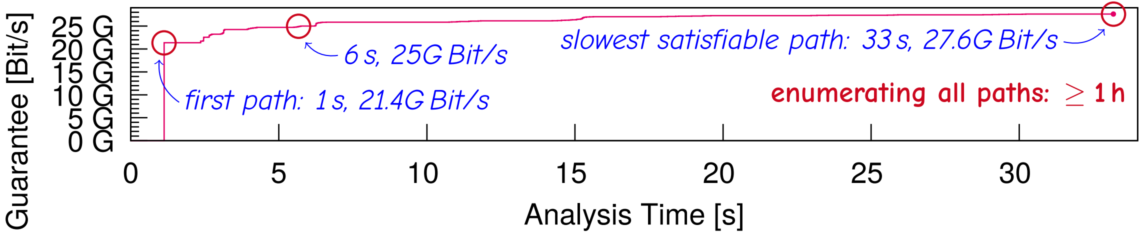


Evaluation: Analysis Time

Example Program: QUIC Load Balancer with IPv6 Option parsing

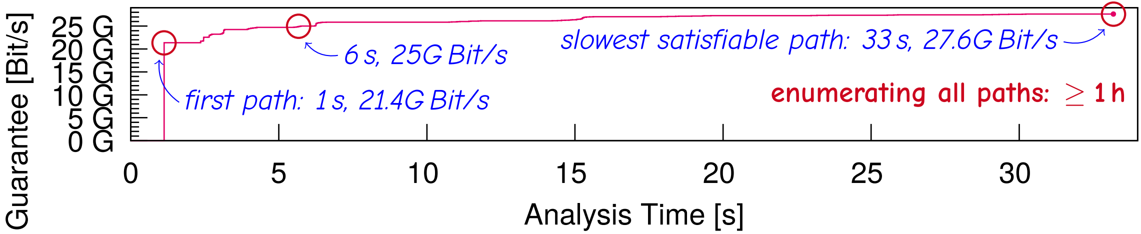


Example Program: QUIC Load Balancer with IPv6 Option parsing



Evaluation: Analysis Time

Example Program: QUIC Load Balancer with IPv6 Option parsing



- **Analyzed 9 real XDP/BPF programs on a desktop PC (Core i7, 16 GiB)**
  - Up to 102 s analysis time (except maliciously crafted example)
  - SMT checking improved throughput guarantees by up to 44%

Is fast enough to be part of the regular development cycle

Evaluation: Estimation Accuracy

- **Measured throughput of 21k paths through all analyzed programs**
  - Minimally sized example packets produced by the SMT solver

Evaluation: Estimation Accuracy

- **Measured throughput of 21k paths through all analyzed programs**
  - Minimally sized example packets produced by the SMT solver

Program	*Slowest Path*
switch.p4 (parser)	✓
Cloudflare DoS	✓
QUIC LB (IPv4)	✓
QUIC LB (IPv6)	✓
RTP a→µ-law	✓
RTP a→µ-law (opt)	✓
DNS Cache	**7^{th}**
Count-Min (5)	✓
Count-Min (20)	✓

Evaluation: Estimation Accuracy

- **Measured throughput of 21k paths through all analyzed programs**
  - Minimally sized example packets produced by the SMT solver

Program	*Slowest Path*	*Estimated*	*Measured*	*Accuracy*
switch.p4 (parser)	✓	24.7G Bit/s	25.8G Bit/s	+4.1%
Cloudflare DoS	✓	35.2G Bit/s	34.7G Bit/s	**-1.0%**
QUIC LB (IPv4)	✓	22.8G Bit/s	23.5G Bit/s	+2.9%
QUIC LB (IPv6)	✓	27.6G Bit/s	28.5G Bit/s	+2.9%
RTP a→µ-law	✓	2.97G Bit/s	2.97G Bit/s	✓
RTP a→µ-law (opt)	✓	4.70G Bit/s	4.70G Bit/s	✓
DNS Cache	**7th**	9.0G Bit/s	10.4G Bit/s	**+13.1%**
Count-Min (5)	✓	21.6G Bit/s	22.2G Bit/s	+2.4%
Count-Min (20)	✓	6.0G Bit/s	6.0G Bit/s	✓

not a lower bound

worst underestimation

Evaluation: Estimation Accuracy

- **Measured throughput of 21k paths through all analyzed programs**
  - Minimally sized example packets produced by the SMT solver

Program	*Slowest Path*	*Estimated*	*Measured*	*Accuracy*
switch.p4 (parser)	✓	24.7G Bit/s	25.8G Bit/s	+4.1%
Cloudflare DoS	✓	35.2G Bit/s	34.7G Bit/s	**-1.0%**
QUIC LB (IPv4)	✓	22.8G Bit/s	23.5G Bit/s	+2.9%
QUIC LB (IPv6)	✓	27.6G Bit/s	28.5G Bit/s	+2.9%
RTP a→μ-law	✓	2.97G Bit/s	2.97G Bit/s	✓
RTP a→μ-law (opt)	✓	4.70G Bit/s	4.70G Bit/s	✓
DNS Cache	**7th**	9.0G Bit/s	10.4G Bit/s	**+13.1%**
Count-Min (5)	✓	21.6G Bit/s	22.2G Bit/s	+2.4%
Count-Min (20)	✓	6.0G Bit/s	6.0G Bit/s	✓

not a lower bound

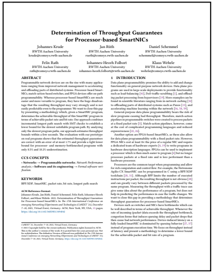
worst underestimation

Accurate throughput guarantees

Conclusion

Throughput Guarantees for Processor-based SmartNICs

- Which packet-rate or bit-rate is achievable by a given program?
- Similar determinism as match-action pipelines and FPGAs



Determination of Throughput Guarantees for Processor-based SmartNICs

Johannes Krude, RWTH Aachen University, krude@comsys.rwth-aachen.de
Jan Rüth, RWTH Aachen University, rueth@comsys.rwth-aachen.de
Daniel Schemmel, RWTH Aachen University, schemmel@comsys.rwth-aachen.de
Felix Rath, RWTH Aachen University, rath@comsys.rwth-aachen.de
Iohannes-Heorh Folbort, RWTH Aachen University, iohannes-heorh.folbort@rwth-aachen.de
Klaus Wehrle, RWTH Aachen University, wehrle@comsys.rwth-aachen.de

ABSTRACT

1 INTRODUCTION

- **Our approach is fast**
  - ▶ Because, we combine incremental ordered enumeration with SMT checks
- **Our approach is accurate**
  - ▶ Because, we model the SmartNIC performance characteristics

all the details

Full Implementation & All Measurement Data

https://github.com/johannes-krude/nfp-pred-artifacts