Accelerator-Aware In-Network Load Balancing for Improved Application Performance

Hesam Tajbakhsh, Ricardo Parizotto, Miguel Neves, Alberto Schaeffer-Filho, Israat Haque

PINet
Programmable and Intelligent Networking

UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

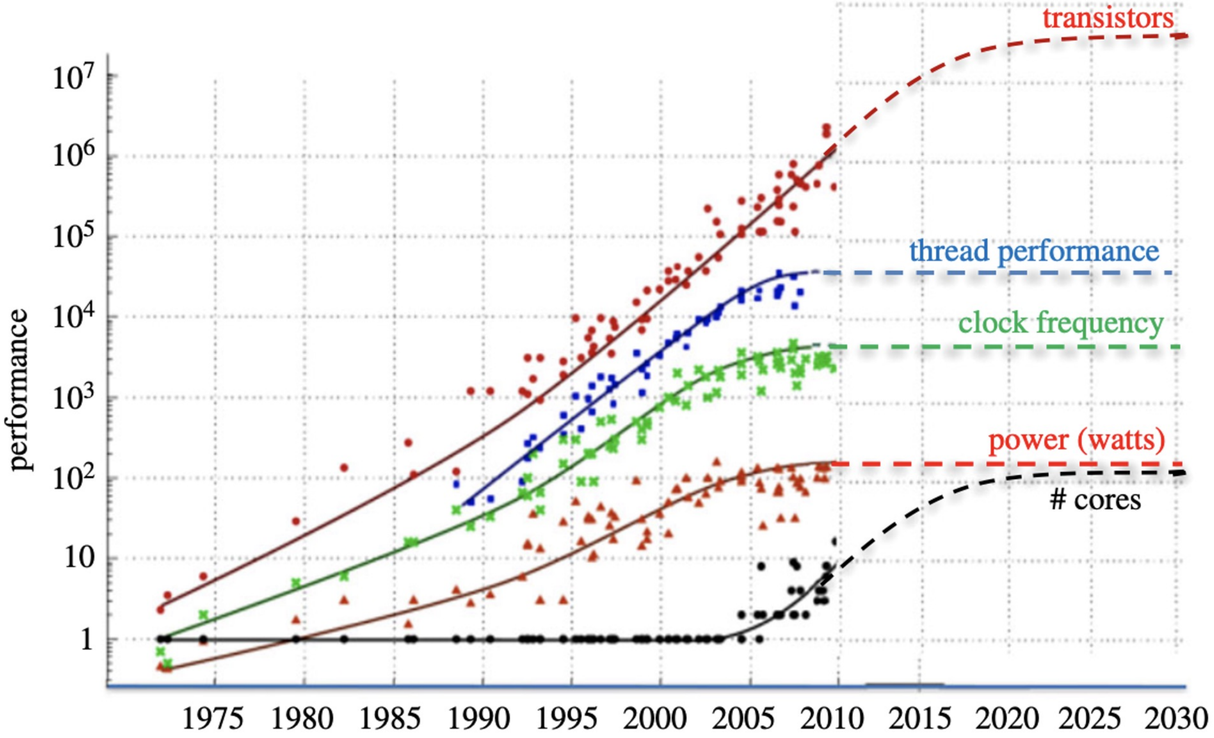
IFIP Networking Conference 2022

Agenda

- Context and motivation
- Background and related work
- P4Mite design
- Implementation and evaluation
- Conclusions

Context and Motivation

Moore's law



Computer performance is reaching a plateau due to the end of Moore's law.

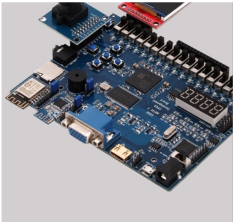
Source: https://tinyurl.com/5xtpd2m5

How to tackle the shortage of CPU advancement?

SmartNICs for networks

Programmable SSDs for storage systems

GPUs for graphical computations



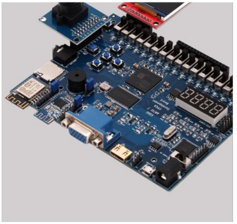
TPUs and FPGAs for miscellaneous tasks

SmartNICs as additional computation resource

SmartNICs for networks

Programmable SSDs for storage systems

GPUs for graphical computations



TPUs and FPGAs for miscellaneous tasks

Computing ability of SmartNICs

Benchmark using *roofline*

Device	Specs	GFLOP/s
CPU (x86)	Xeon(R) Silver 4210R CPU 2.4GHz, 10 cores	91.0
SmartNIC (ARM)	NVIDIA BlueField (ARMv8 A72, 800 MHz, 16 cores)	17.6

- *SmartNIC can add 18% computing support*.
- Also, there can be multiple SmartNICs per server (E3 USENIX'19).

Load balancing approach

- Existing LBs work at a per-server granularity —> The performance is affected drastically because accelerators have
  - limited resources
  - different architectures
- ***There is a need for an LB that operates at a finer granularity —> We introduce per-accelerator granularity in the proposed LB system.***

Load balancer deployment

- On servers or accelerators
  - Usually, run along with other applications

- On programmable switches
  - Process packets at line rates
  - Servers and their accelerators can use their resources for other applications

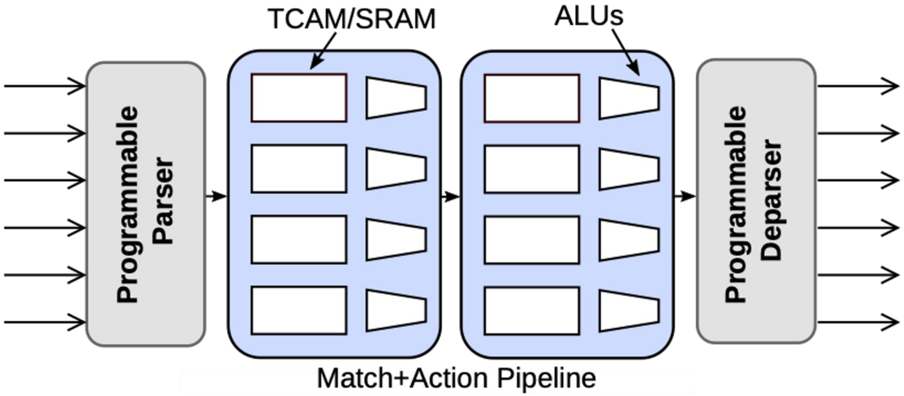
Our proposed solution: P4Mite

- **A Load Balancer which:**
  - *Works at a per-accelerator granularity*
    - **Challenge #1**: It must run a heterogeneous environment
    - **How to address?** The system collects resource statistics from hosts AND their accelerators and then distributes the load
  - *Runs on a programmable switch*
    - **Challenge #2**: It must handle many connections using negligible memory footprint
    - **How to address?** It stores compressed information inside the switch

Background and State-of-the-art

Data Plane Programming

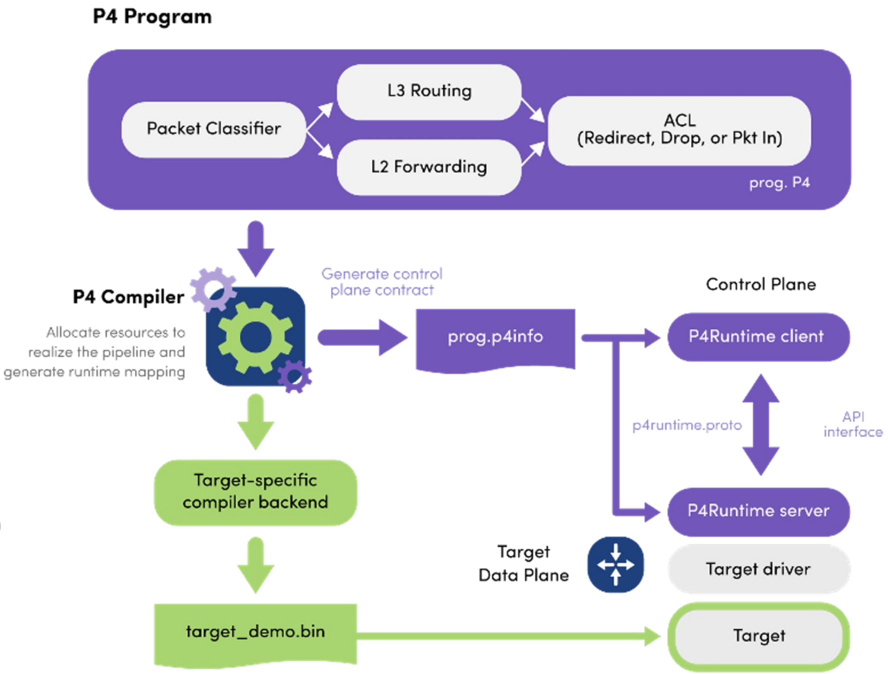
- Protocol Independent Switch Architecture (PISA)
    1. Parses packets
    2. Performs a set of match+action rules on packets
    3. Deparses packets
- Works at line-rates
- Can work on every packet if needed



Programmable Switches

- Programming Protocol-independent Packet Processors (P4) is a domain-specific language for data plane programming

- We use the Tofino switch in this research (Tofino Native Architecture)



Source: p4.org

Existing Load Balancers

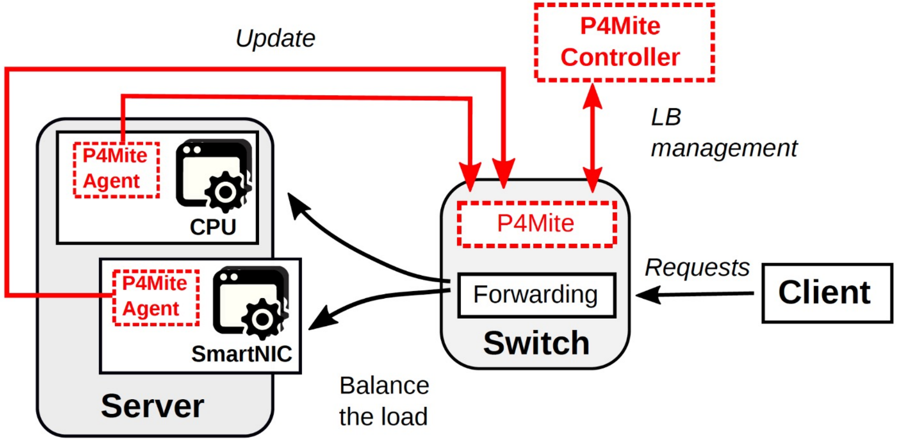
Load Balancer	Resource Awareness	Accelerator Visibility	Deployment
SilkRoad (SIGCOMM'17)	✗	✗	In a programmable switch
Loom (ICNP'21)	✗	✗	In a programmable switch
Cheetah (USENIX'20)	✗	✗	In a programmable switch
Tiara (USENIX'22)	✗	✗	In a switch + SmartNIC + server
CrossRSS (CoNEXT'20)	✓	✗	In a SmartNIC
Charon (CNSM'21)	✓	✗	In a SmartNIC
P4Mite	✓	✓	In a programmable switch

Our contributions

- P4Mite: an **accelerator-aware in-network** load balancer
- Implemented a prototype of P4Mite
- Evaluated P4Mite over different applications

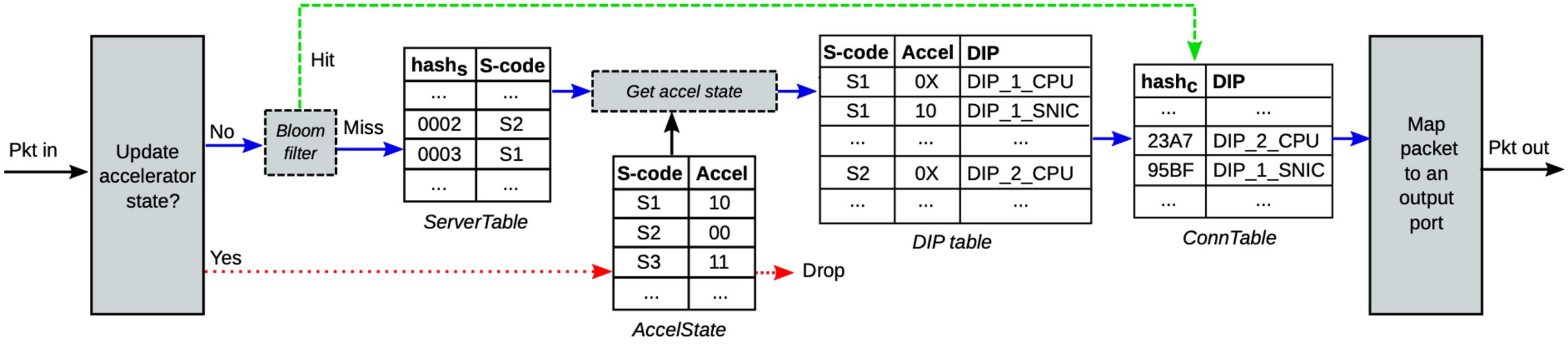
P4Mite Design

P4Mite system overview



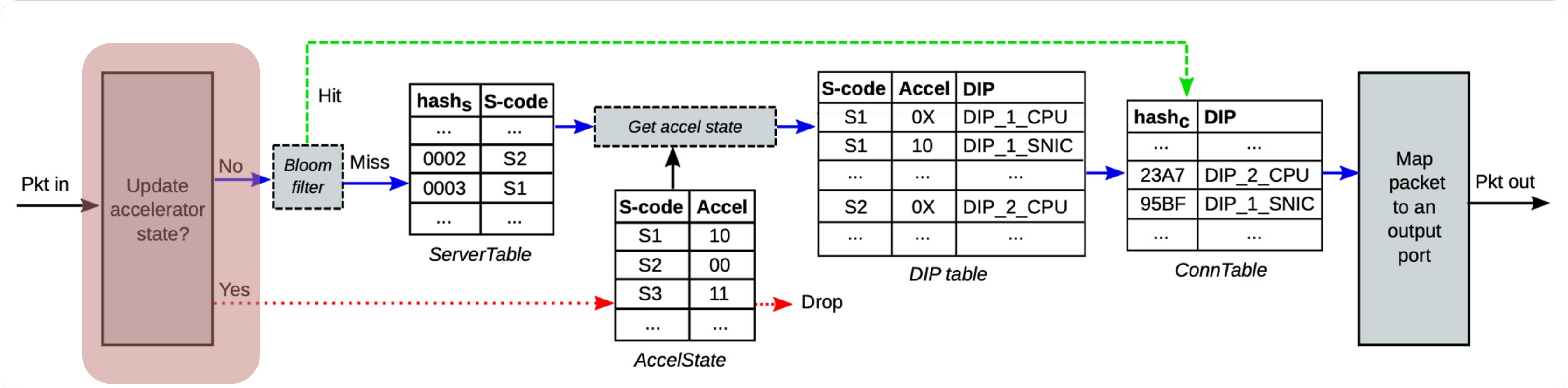
- The controller implements a policy and configures the associated switch
- Its data plane processes requests accordingly
- Agents actively monitor servers and accelerators and update the switch based on a target metric (CPU utilization, processing time, etc.)

Data Plane Design



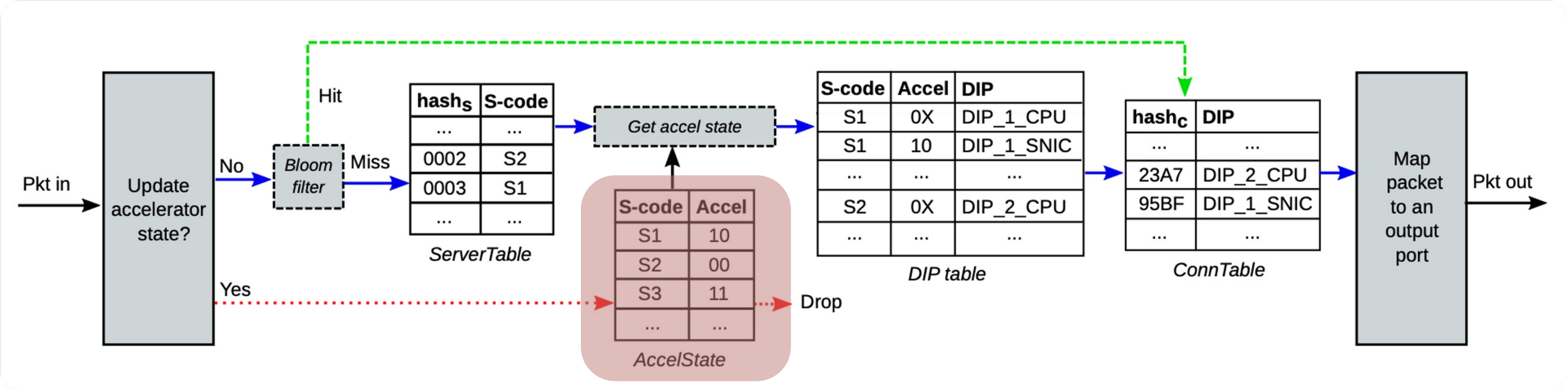
Data Plane Design

- First, the switch checks the type of packets: state vs. request



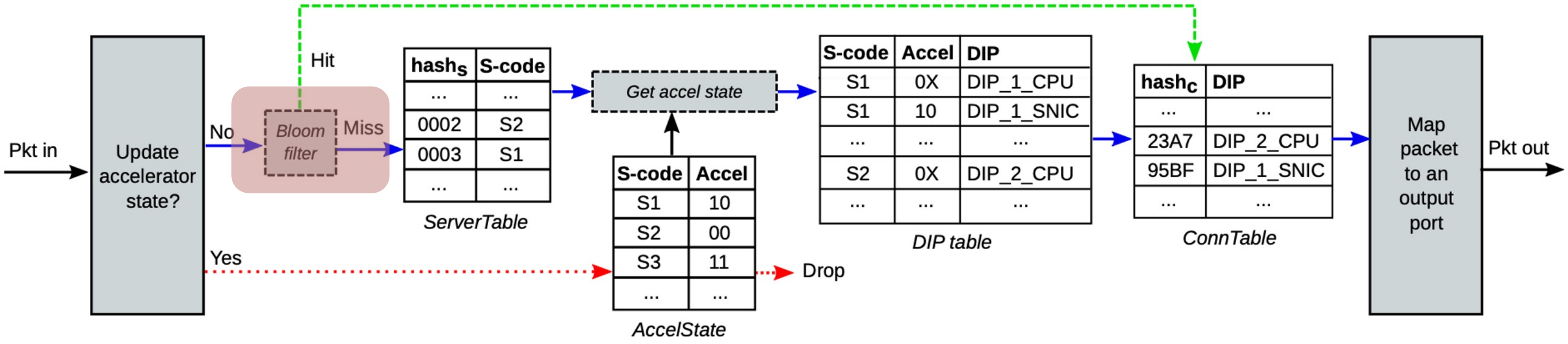
Data Plane Design

- First, the switch checks the type of packets: **state** vs. request
  - Updates bitmaps from the ***AccelState*** table



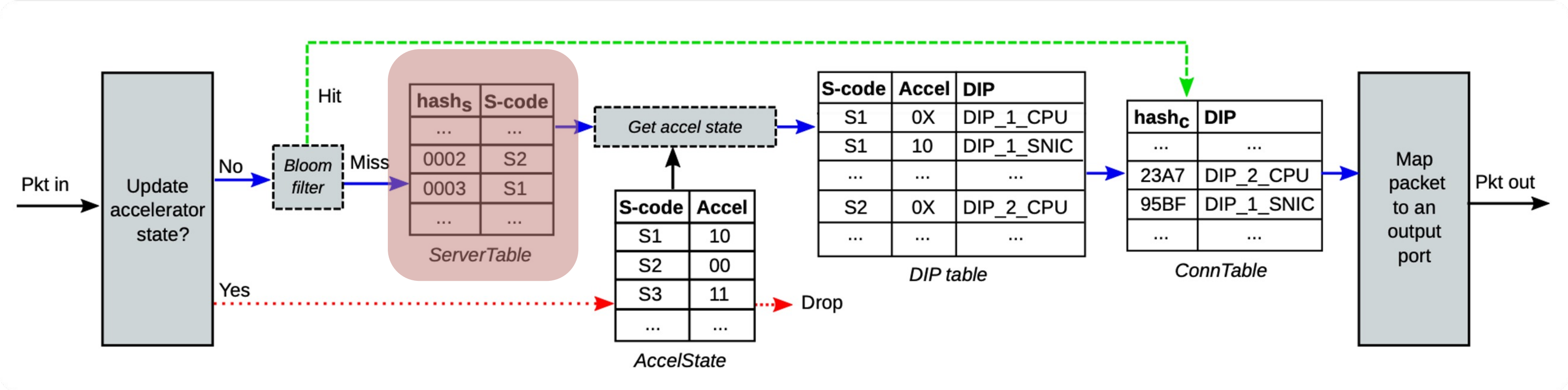
Data Plane Design

- First, the switch checks the type of packets: state vs. **request**
  - Computes hash for the bloom filter
    - Hits are from existing connections
    - Misses are from new connections



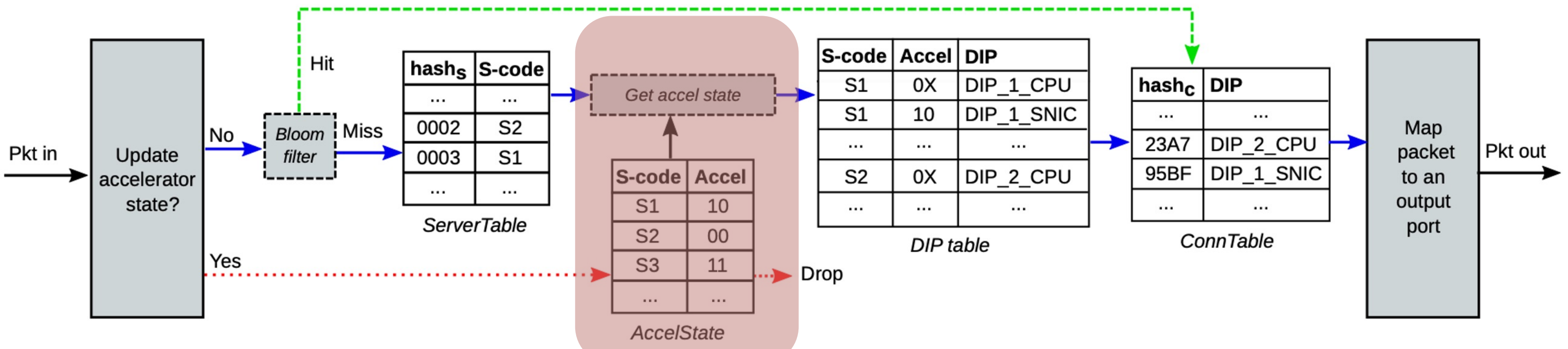
Data Plane Design

- In the case of a new connection, the request is dispatched to a server using the *ServerTable*



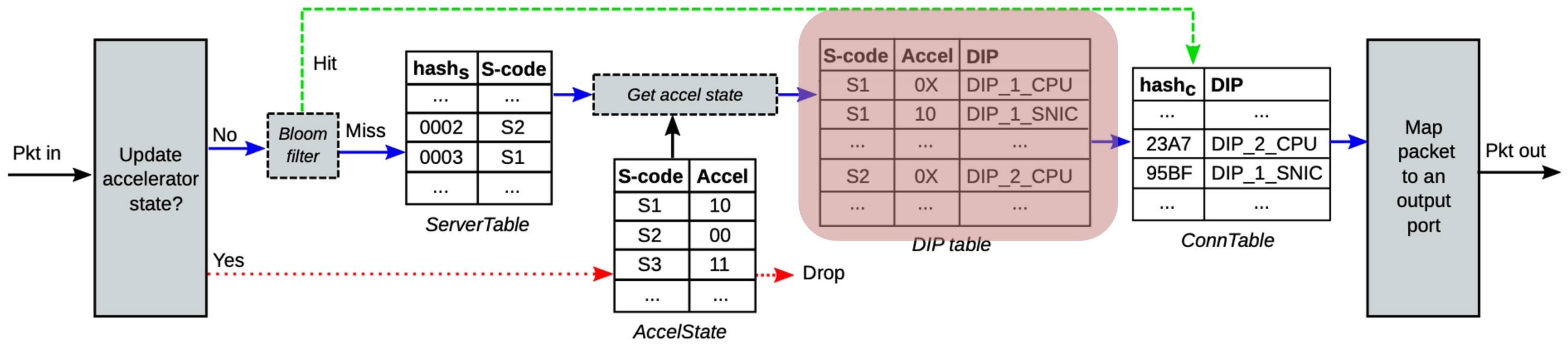
Data Plane Design

- In the case of a new connection, the request is dispatched to a server using the ***ServerTable***
- Next, it keeps checking the state of that server from the ***AccelState***



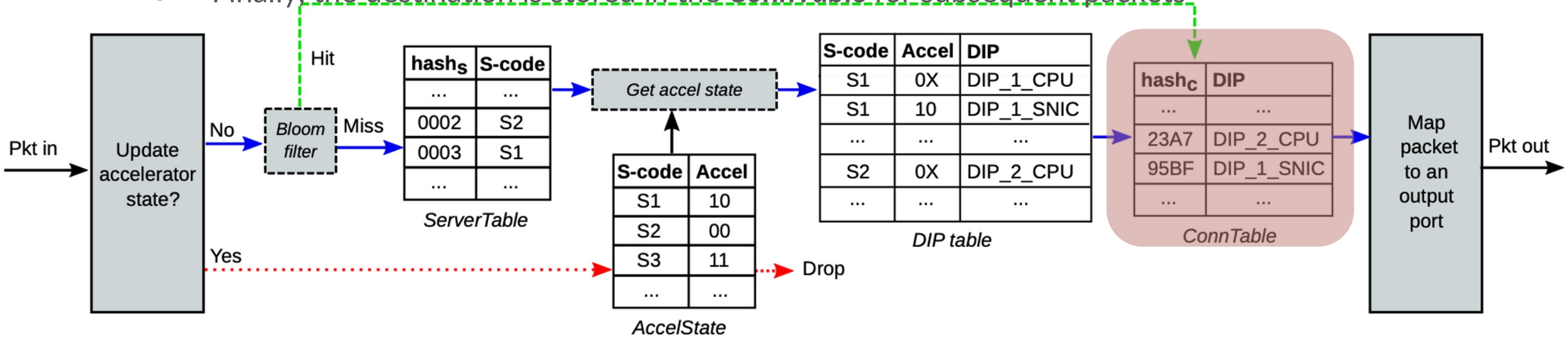
Data Plane Design

- In the case of a new connection, the request is dispatched to a server using the ***ServerTable***
- Next, it keeps checking the state of that server from the ***AccelState***
- Thirdly, it maps to the server's CPU or an accelerator based on the ***DIP table***



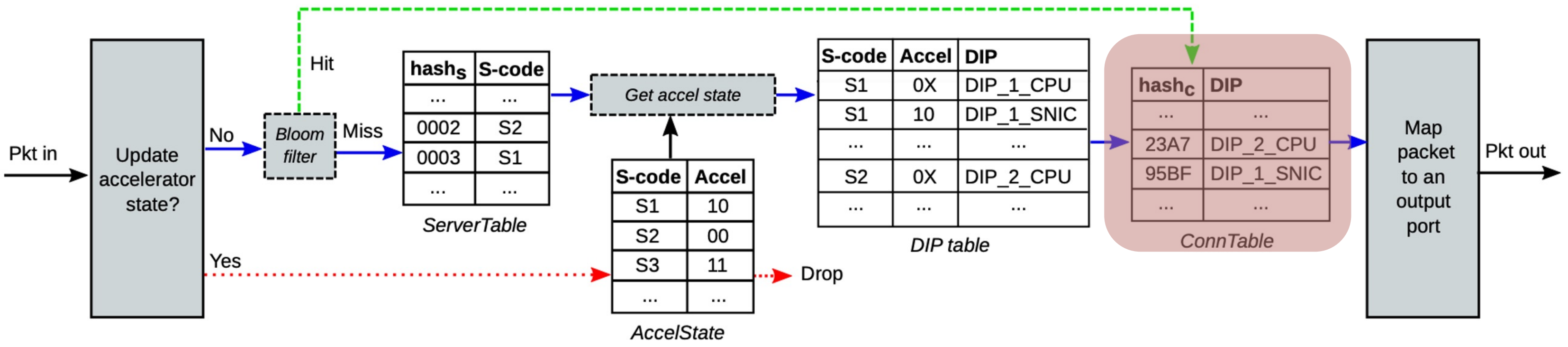
Data Plane Design

- In the case of a new connection, the request is dispatched to a server using the ***ServerTable***
- Next, it keeps checking the state of that server from the ***AccelState***
- Thirdly, it maps to the server's CPU or an accelerator based on the ***DIP table***
- Finally, the destination is stored in the ***ConnTable*** for subsequent packets



Data Plane Design

- In the case of an existing connection, the destination is available in the ***ConnTable***



P4Mite Implementation and Evaluation

Implementation

P4Mite Switch and Controller: Python3 and P4-16

Programmable switch: Intel Tofino

Hash Function: CRC16

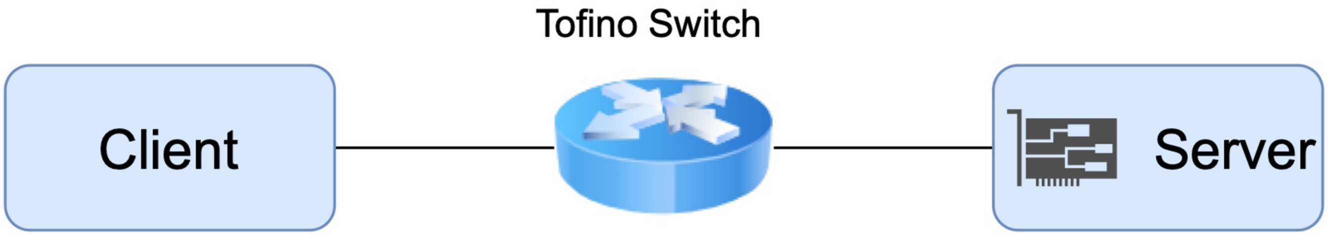
Agents: Python3

Monitoring metric: Processing Time

https://github.com/PINetDalhousie/p4mite

Evaluation Setup

- Switch: Wedge 100BF-32X 32-port programmable switch with a 3.2Tbps Tofino ASIC
- Server and client: Intel(R) Xeon(R) Silver 4210R CPU @ 2.4GHz with 10 cores and 32GB memory
- SmartNIC: Dual-port SFP28, PCIe Gen3.0/4.0 x8



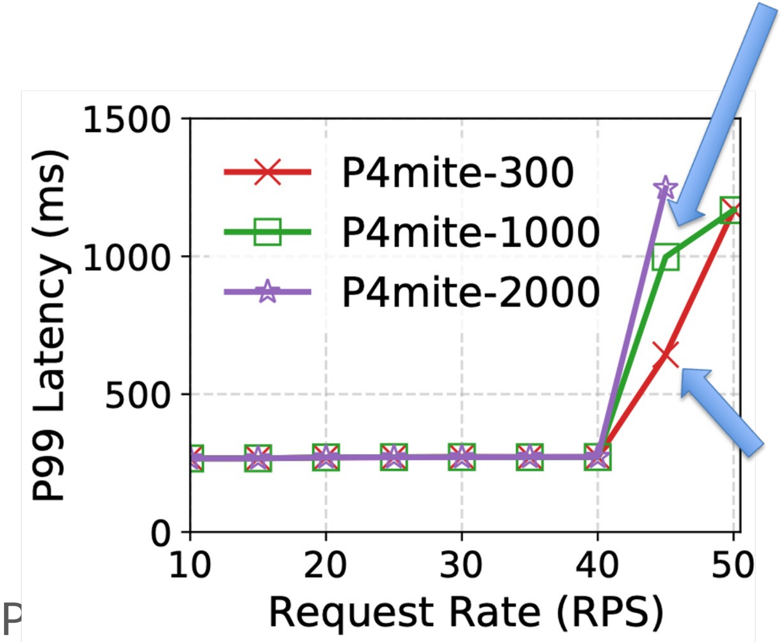
Experiment 1: A synthetic workload

- A synthetic workload is implemented
  - **The server application**: performs different amounts of floating-point operations upon receiving a request
  - **The client application**: sends different numbers of requests per second (RPS)

- Two parameter are evaluated in this experiment:
  - Request rate
  - Request size

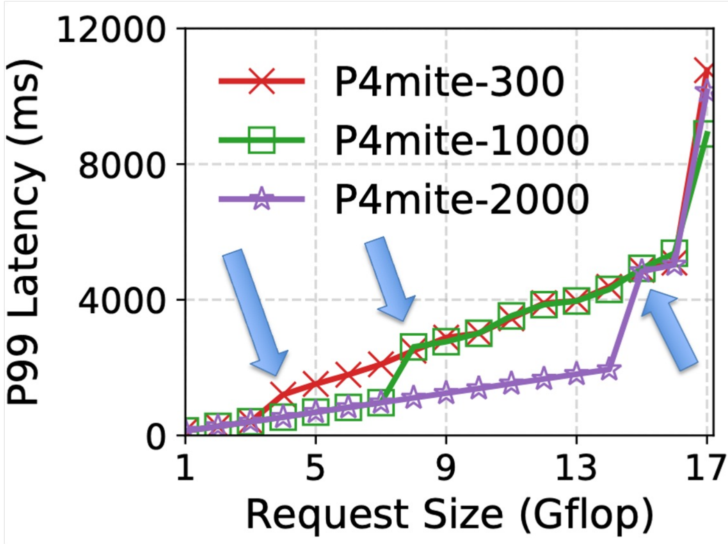
- Three processing time thresholds are tested (300ms, 1000ms, and 2000ms)

Exp1- evaluating request rate



- The request size is set to 2 Gflop
- All options work similarly up to 40 RPS
- P4mite-300 and P4mite-1000 react at 45 RP
- P4mite-300 is the fastest one as its agent gets triggered quicker

Exp1- evaluating request size



- The request rate is set to 5 RPS
- All options work similarly up to 4 Gflop
- The agent of P4mite-300, P4mite-1000, and P4mite-2000 get triggered at size 4, 8, and 15 Gflop, respectively.

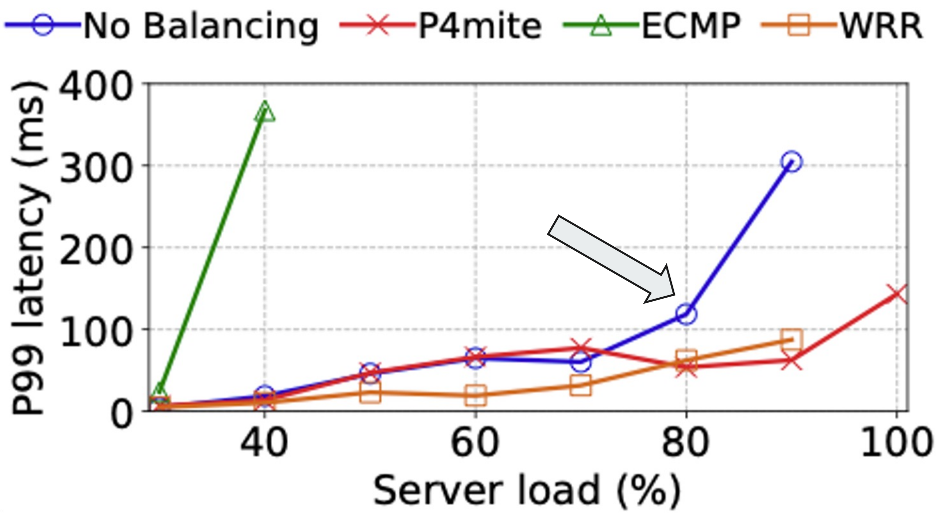
Experiment 2: Three Real-world applications

- Three applications are implemented and tested separately
  - DNS
  - VGG16
  - KNN*
- We compared P4Mite with two existing load balancing algorithms:
  - Equal-cost multi-path routing (EMCP): splits load evenly
  - Weighted Round Robin (WRR): The server and SmartNIC handle 5/6 and 1/6 of the load, respectively. These ratios are used according to the *roofline*'s results.

- We configured a suitable threshold on the agents for each application

*KNN results are available in the full text

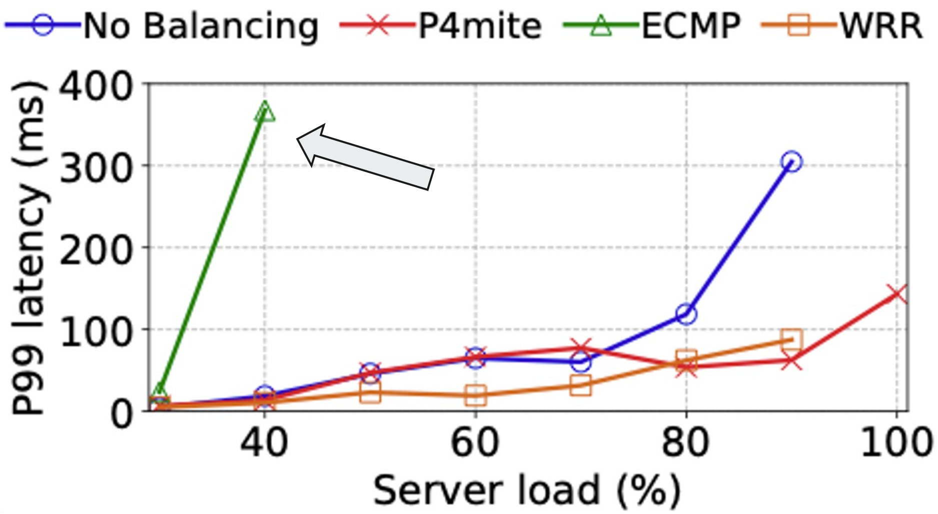
Exp2- DNS



It is a lightweight application

- **No balancing (baseline): latency increases after 80% load**
- **ECMP**: fully utilizes its SmartNIC at 40% load
- **WRR:** proactively splits the load and can reach 90% load
- **P4Mite**: handles **100%** load and reduces the latency up to **50%**

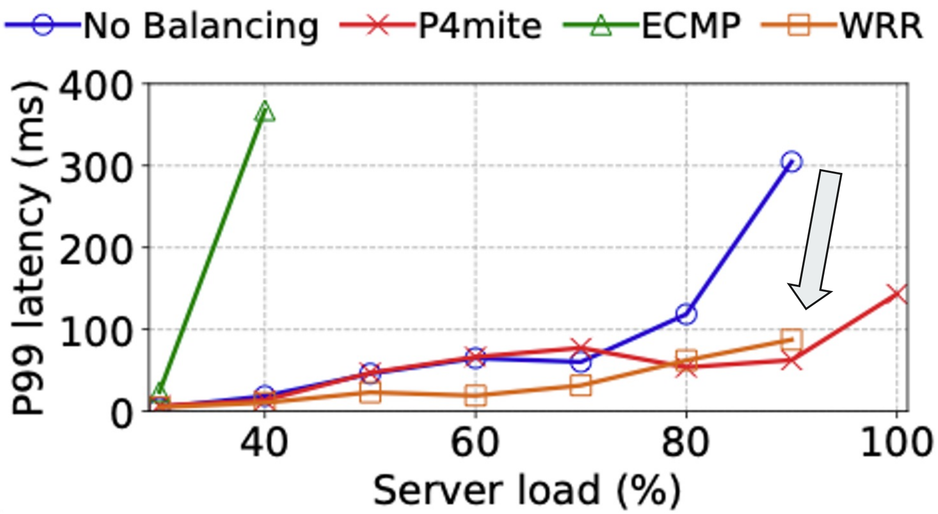
Exp2- DNS



It is a lightweight application

- **No balancing (baseline):** latency increases after 80% load
- **ECMP: fully utilizes its SmartNIC at 40% load**
- **WRR:** proactively splits the load and can reach 90% load
- **P4Mite**: handles **100%** load and reduces the latency up to **50%**

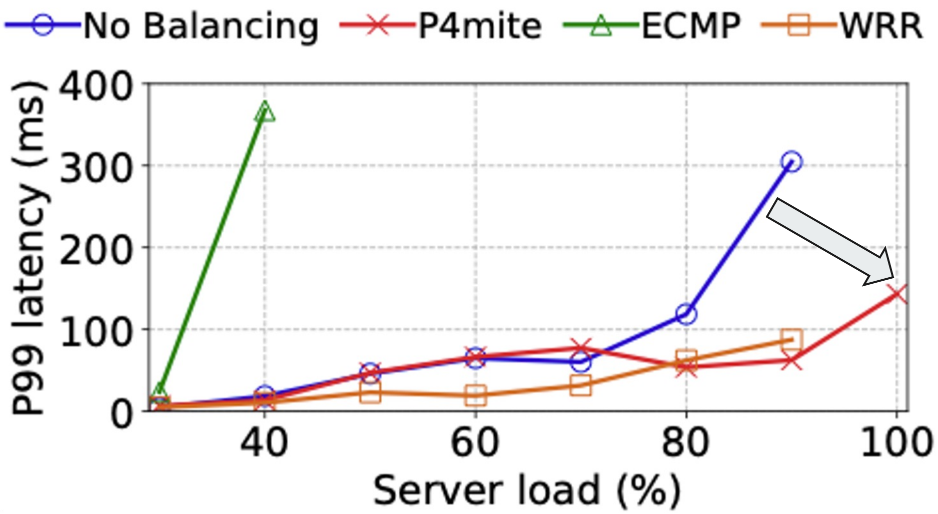
Exp2- DNS



It is a lightweight application

- **No balancing (baseline):** latency increases after 80% load
- **ECMP**: fully utilizes its SmartNIC at 40% load
- **WRR: proactively splits the load and can reach 90% load**
- **P4Mite**: handles **100%** load and reduces the latency up to **50%**

Exp2- DNS

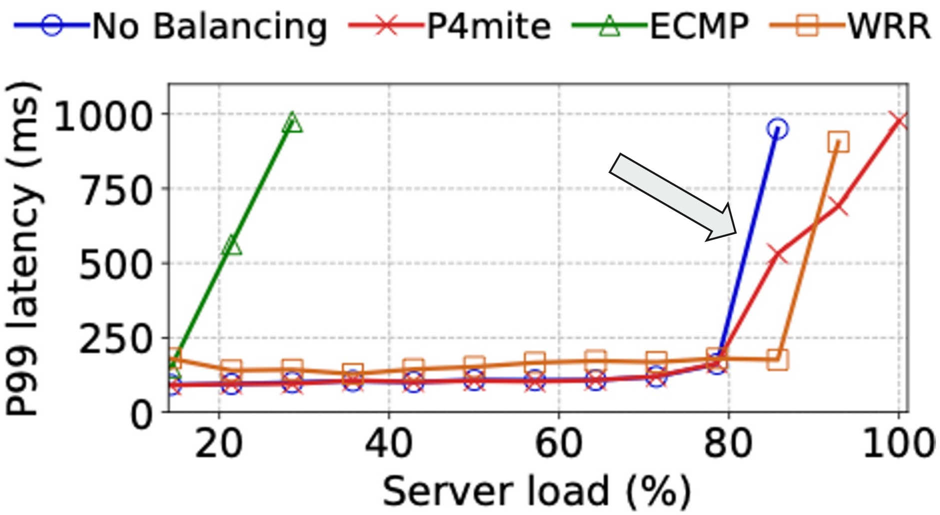


It is a lightweight application

- **No balancing (baseline):** latency increases after 80% load
- **ECMP**: fully utilizes its SmartNIC at 40% load
- **WRR:** proactively splits the load and can reach 90% load
- **P4Mite: handles 100% load and reduces the latency up to 50%**

Exp2- VGG16

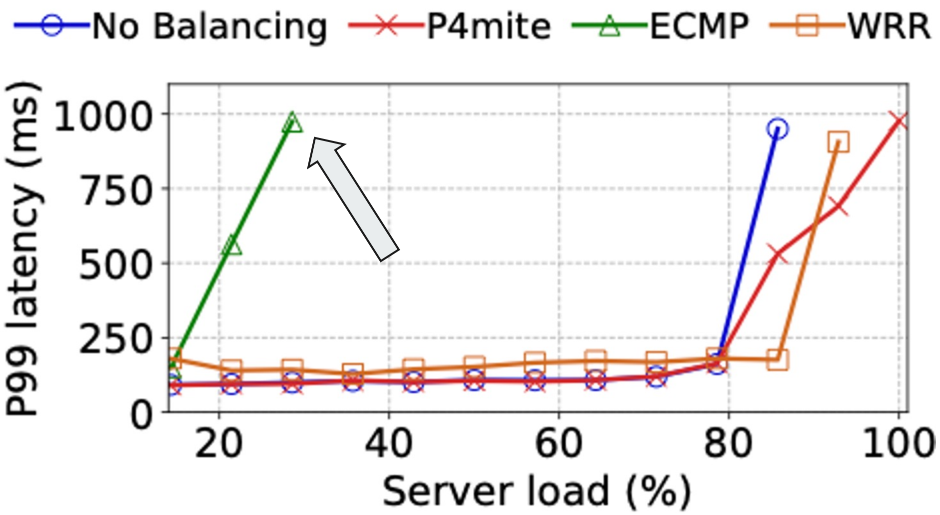
It is computationally heavier than DNS
Server's latency: 80ms
SmartNIC's latency: 120ms



- **Baseline: latency increases after 80% load**
- **ECMP**: fully utilizes its SmartNIC at 30% load
- **WRR**: performs worse than the baseline and P4mite until 80% load
- **P4mite**: handles **100%** load

Exp2- VGG16

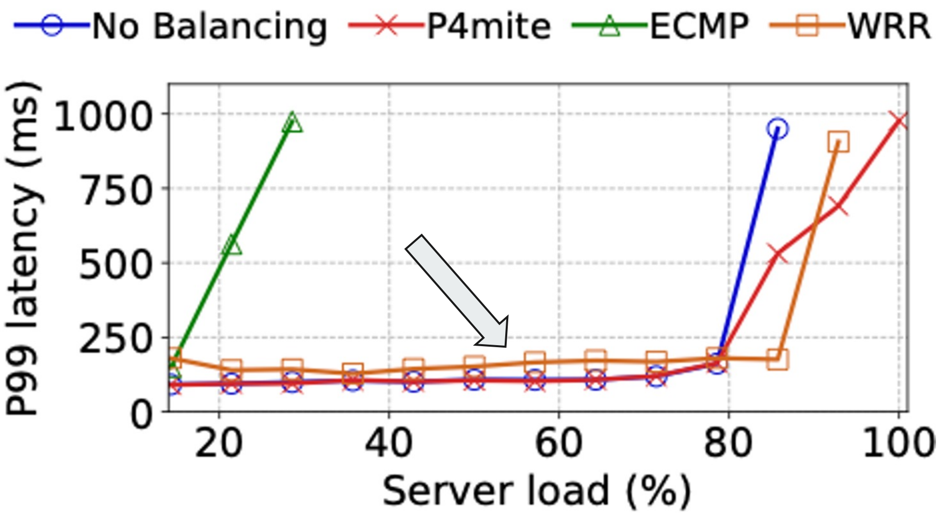
It is computationally heavier than DNS
Server's latency: 80ms
SmartNIC's latency: 120ms



- **Baseline**: latency increases after 80% load
- **ECMP: fully utilizes its SmartNIC at 30% load**
- **WRR**: performs worse than the baseline and P4mite until 80% load
- **P4mite**: handles **100%** load

Exp2- VGG16

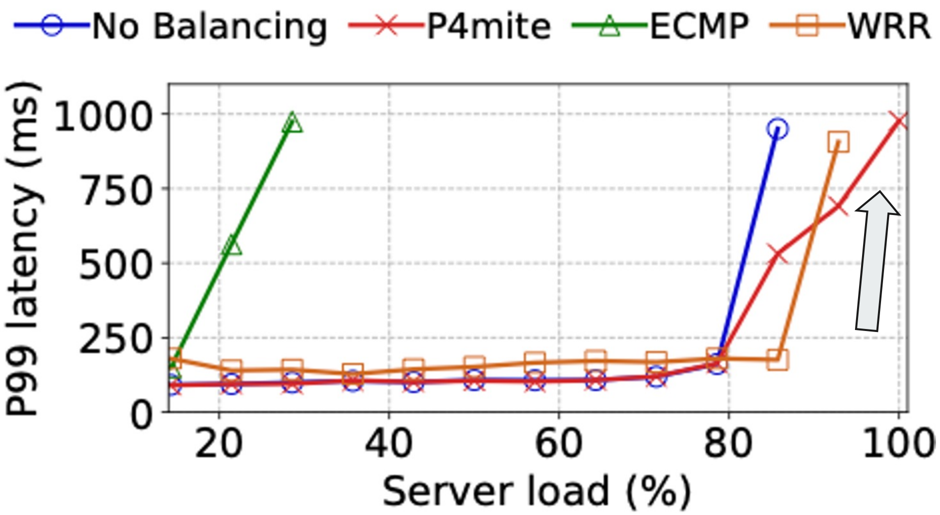
It is computationally heavier than DNS
Server's latency: 80ms
SmartNIC's latency: 120ms



- **Baseline**: latency increases after 80% load
- **ECMP**: fully utilizes its SmartNIC at 30% load
- **WRR: performs worse than the baseline and P4mite until 80% load**
- **P4mite**: handles **100%** load

Exp2- VGG16

It is computationally heavier than DNS
Server's latency: 80ms
SmartNIC's latency: 120ms

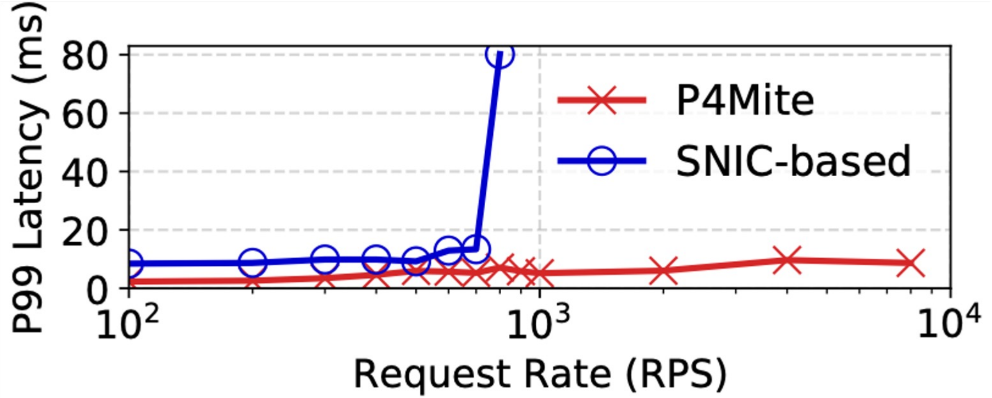


- **Baseline**: latency increases after 80% load
- **ECMP**: fully utilizes its SmartNIC at 30% load
- **WRR**: performs worse than the baseline and P4mite until 80% load
- P4mite: handles 100% load

P4Mite overhead and Scalability

Agents: use less than 5% of CPU

P4Mite data plane: uses at most 6% resources (for 256 Servers, 2 accelerators on each, 50K connections)



Future Work

- Implementing more advanced policies
  - Monitoring multiple metrics on servers and accelerator

- Employing more powerful accelerators such as GPUs
- Evaluating stateful applications

Conclusion

- We introduced P4Mite, an accelerator-aware in-network load balancer
- It offers per-accelerator granularity
- We implemented P4Mite on top of a Tofino switch and evaluated its performance over three real-world applications
- It can reduce end-to-end communication latency by up to 50%
- We shared the prototype implementation

Questions?

Hesam Tajbakhsh (hs762129@dal.ca)

Ricardo Parizotto (rparizotto@inf.ufrgs.br)
Miguel Neves (mg478789@dal.ca)
Alberto Schaeffer-Filho (alberto@inf.ufrgs.br)
Israat Haque (israat@dal.ca)