PRISM: Streamlined Packet Processing for Containers with Flow Prioritization

(ICDCS '22)

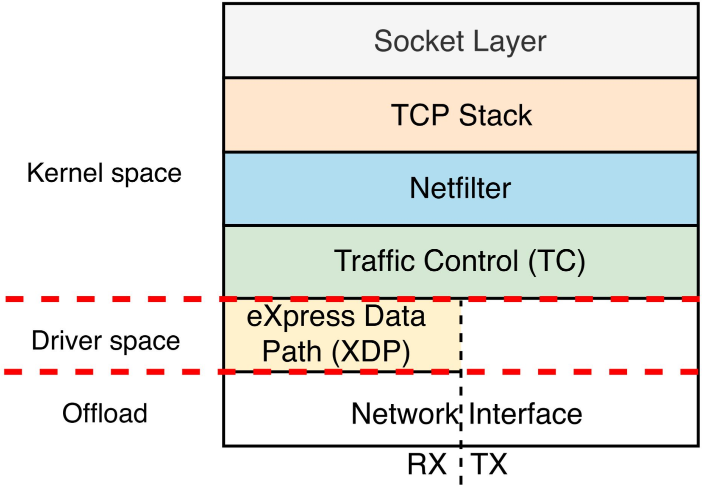
Manish Munikar[1], Jiaxin Lei[2], Hui Lu[2], Jia Rao[1]

[1] *University of Texas at Arlington*

[2] *Binghamton University*

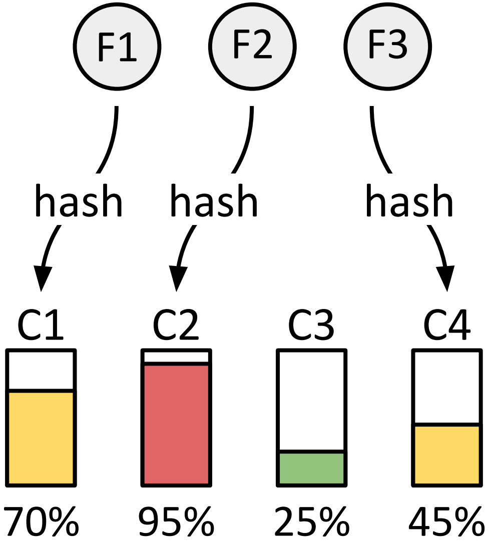
Kernel network stack has been very successful …

- Easy **abstraction** for network I/O
    - Generic sockets interface
- Flexible and **modular**
- Open-source, **stable**, **secure** and **reliable**
- Designed to provide overall efficiency
- Successful for more than **three decades**!



… until recently?

- Network devices are getting faster ***faster***
  - 10 Gbps (*2002*) → 200 Gbps (*2017*)
- CPU speed is stagnating at 2–3 GHz
  - Scaling **horizontally** instead
- Kernel network stack has been trying to keep up...
  - Packet steering to support multi-core
  - NAPI for dynamic interrupt/poll mode
  - Packet coalescing / batching
  - Checksum offloading
- **New bottleneck** for high-performance network applications

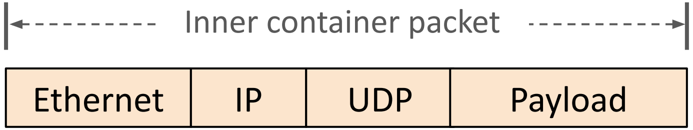


Container networks

- Containers are revolutionizing cloud
  - Lightweight OS-level virtualization
  - Higher consolidation density
  - Faster and easier to manage
- Communicate using overlay network
  - VXLAN encapsulation

Container networks

- Containers are revolutionizing cloud
  - Lightweight OS-level virtualization
  - Higher consolidation density
  - Faster and easier to manage
- Communicate using overlay network
  - VXLAN encapsulation

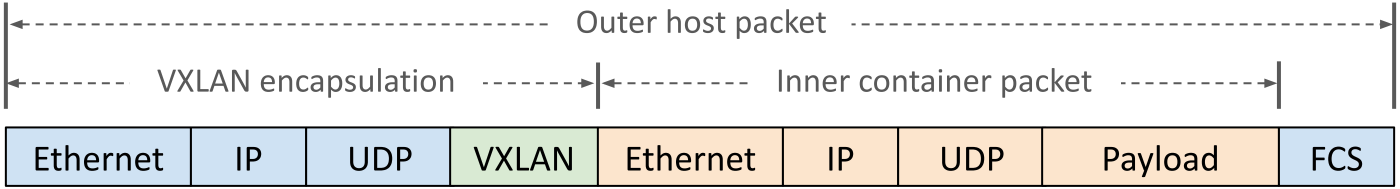


Inner container packet

Ethernet	IP	UDP	Payload

Container networks

- Containers are revolutionizing cloud
  - Lightweight OS-level virtualization
  - Higher consolidation density
  - Faster and easier to manage
- Communicate using overlay network
  - VXLAN encapsulation



Outer host packet

VXLAN encapsulation

Inner container packet

Ethernet	IP	UDP	VXLAN	Ethernet	IP	UDP	Payload	FCS

Cloud application types

Cloud application types

- Bulk data transfer
  - File transfer, backup, sync
  - Video streaming
  - Many/large packets
  - **Throughput-sensitive**

NETFLIX

hadoop

amazon
S3

Cloud application types

- Bulk data transfer
  - File transfer, backup, sync
  - Video streaming
  - Many/large packets
  - Throughput-sensitive

NETFLIX

hadoop

amazon S3

- Request-Response
  - Web / database request, RPC
  - Control plane signals
  - Few/small packets
  - Latency-sensitive

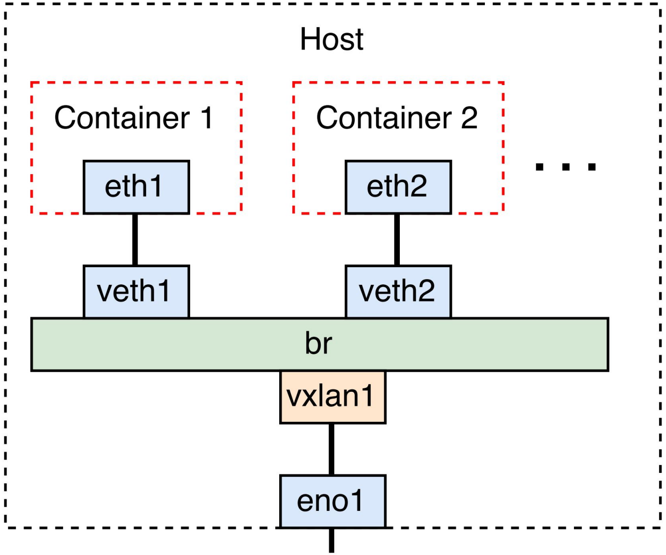
Memcached

NGINX

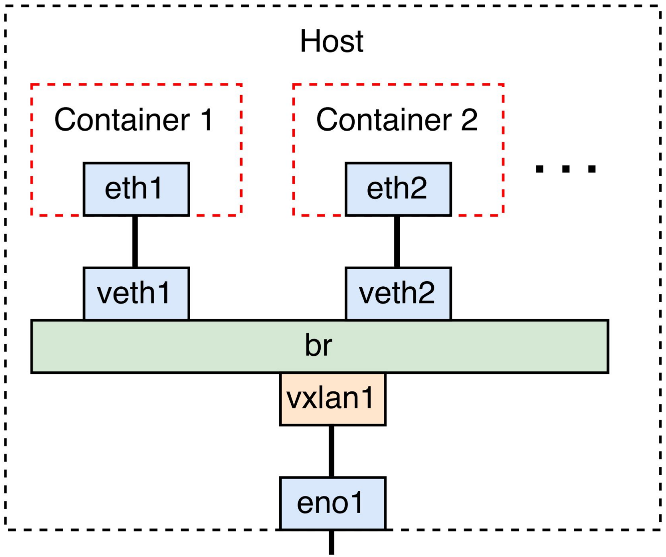
{ REST:API }

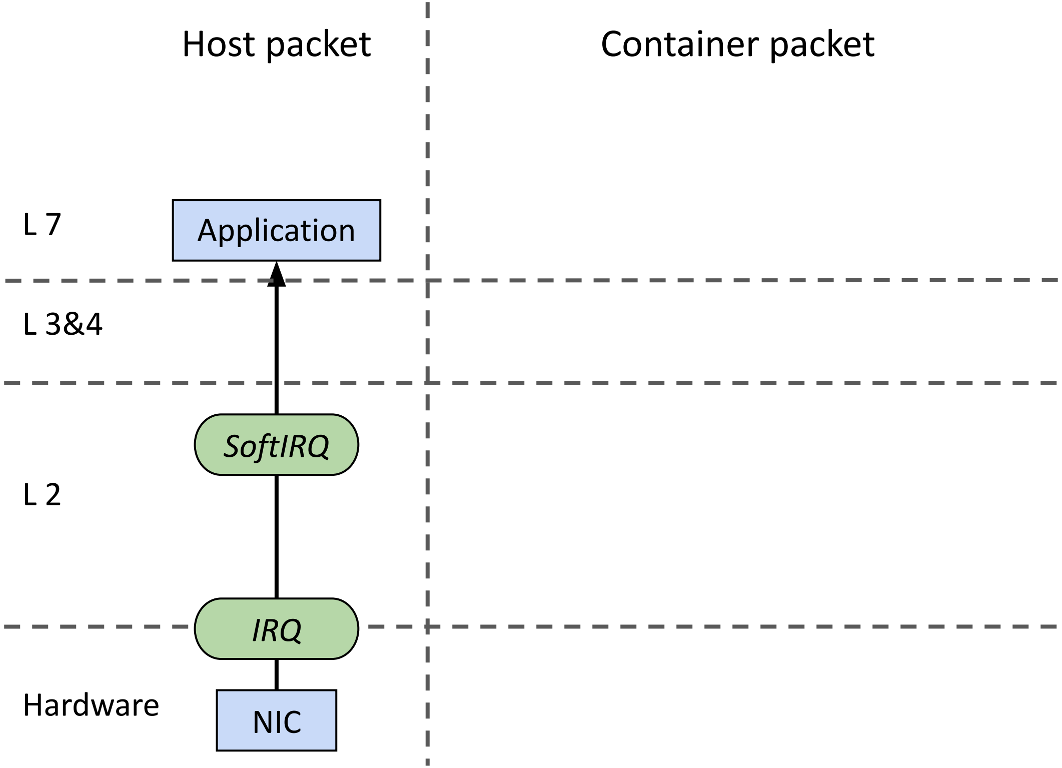
BGP

Container packet processing

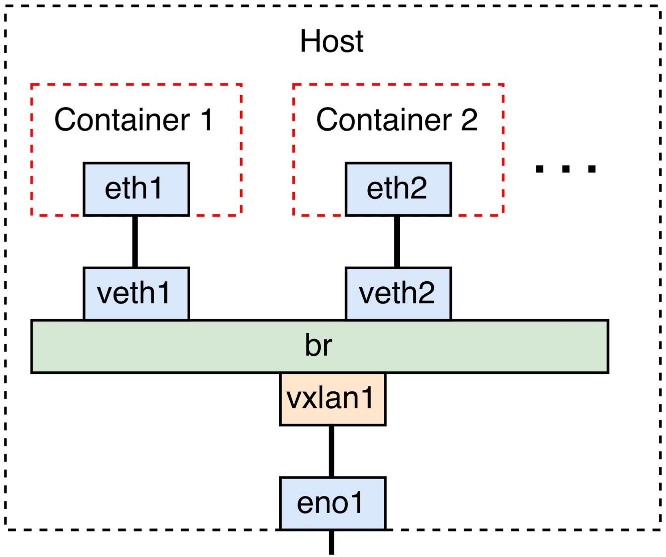


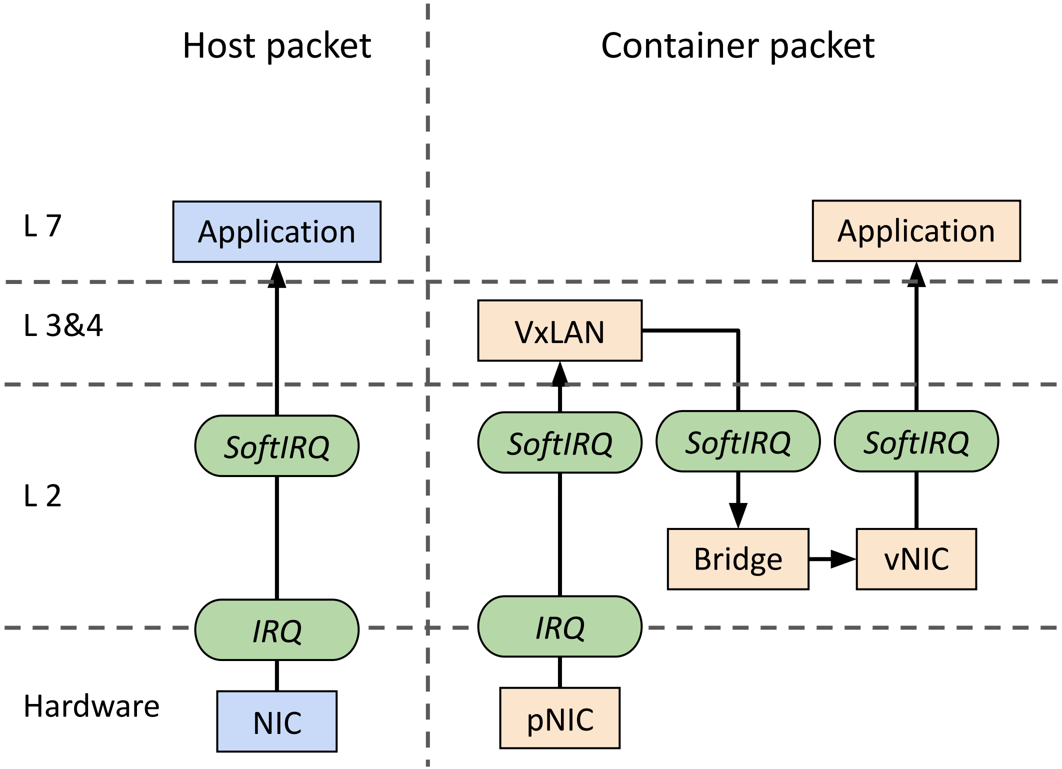
Container packet processing





Container packet processing





Related works

Related works

- Kernel stack optimizations [RPS, RSS, RFS, FlexSC, ...]
    - System call batching
    - Packet steering / load balancing
    - Interrupt coalescing
    - Zero-copy

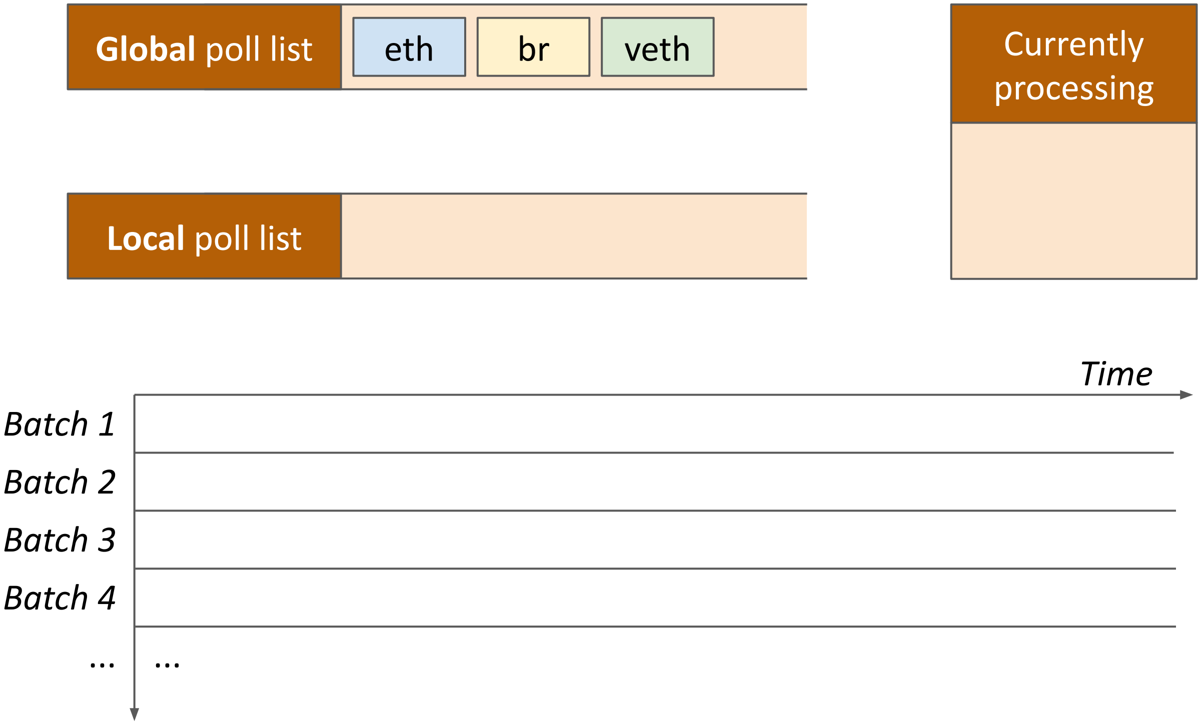
Related works

- Kernel stack optimizations [RPS, RSS, RFS, FlexSC, ...]
  - System call batching
  - Packet steering / load balancing
  - Interrupt coalescing
  - Zero-copy
- Kernel bypass [DPDK, NetMap, FD.io, XDP, Mellanox ASAP^2, ...]
  - Avoids most kernel overheads
  - Dedicates core for packet polling and uses custom network stack
  - Hardware offload

Related works

- Kernel stack optimizations [RPS, RSS, RFS, FlexSC, ...]
    - System call batching
    - Packet steering / load balancing
    - Interrupt coalescing
    - Zero-copy
- Kernel bypass [DPDK, NetMap, FD.io, XDP, Mellanox ASAP^2, ...]
    - Avoids most kernel overheads
    - Dedicates core for packet polling and uses custom network stack
    - Hardware offload
- Container network optimization [Slim, FALCON, FreeFlow, ...]
    - Connection metadata manipulation
    - More fine-grained (flow-device level) packet steering

Interleaved NAPI device processing



Global poll list

eth

br

veth

Local poll list

Currently processing

Time

Batch 1

Batch 2

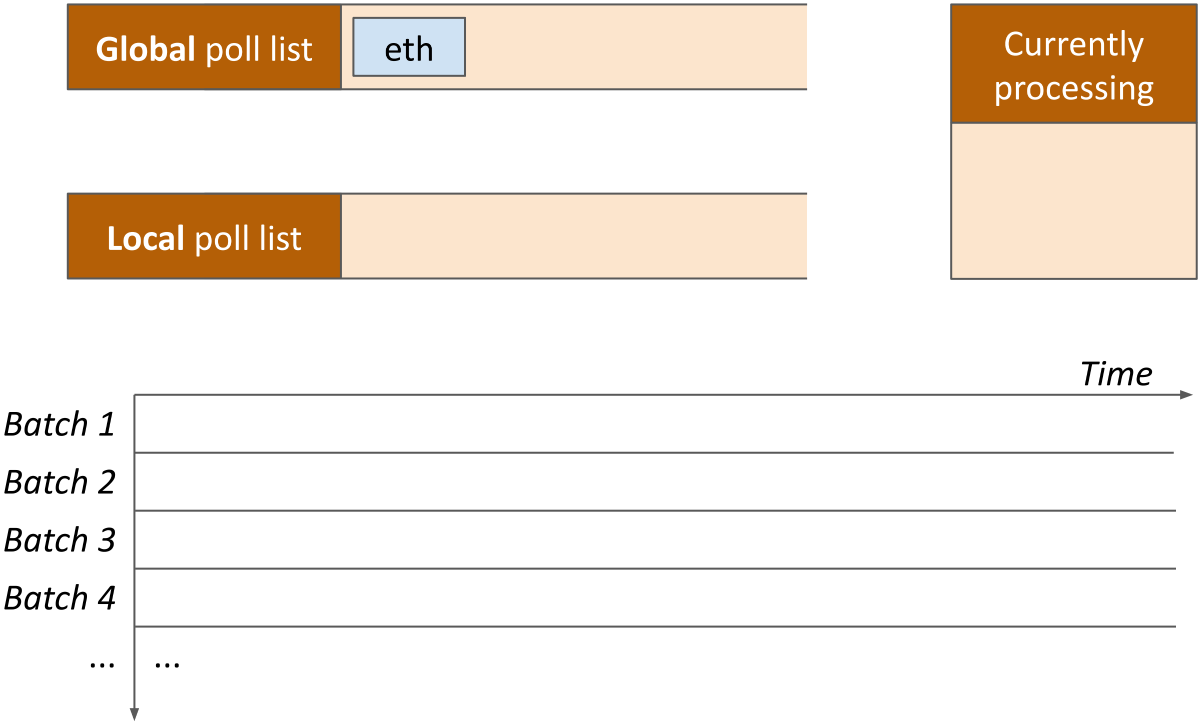
Batch 3

Batch 4

…

…

Interleaved NAPI device processing



Global poll list

eth

Local poll list

Currently processing

Time

Batch 1

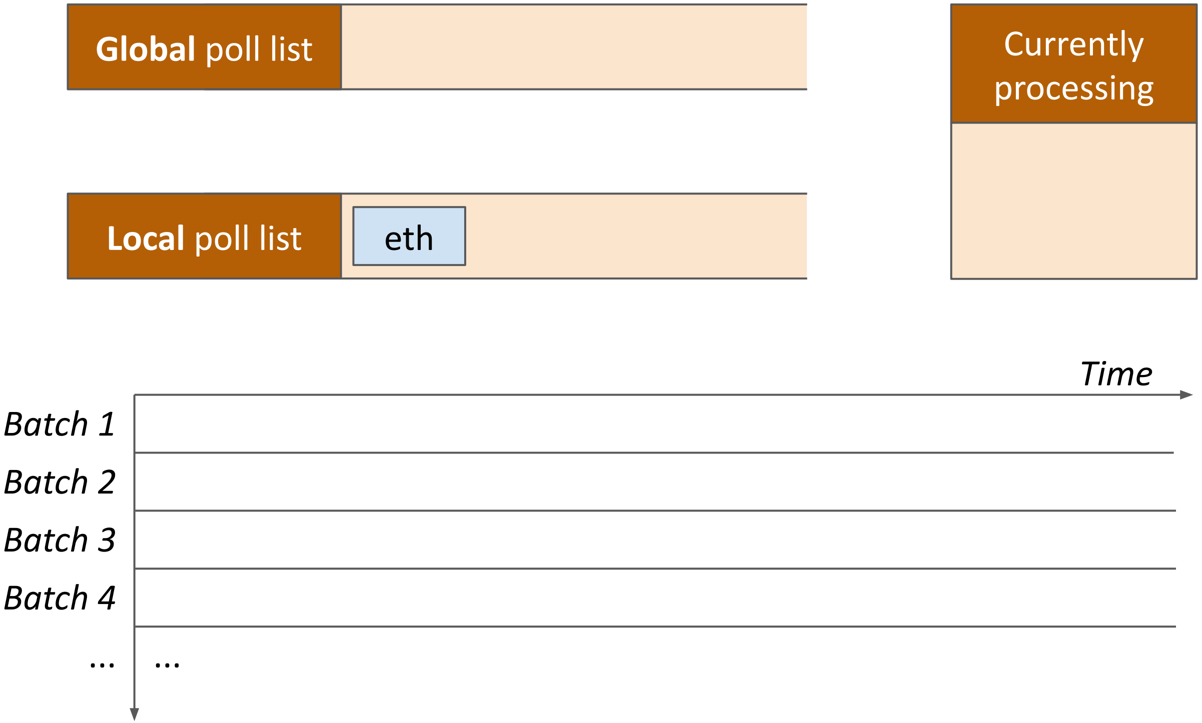
Batch 2

Batch 3

Batch 4

… …

Interleaved NAPI device processing



Global poll list

Local poll list

eth

Currently processing

Time

Batch 1

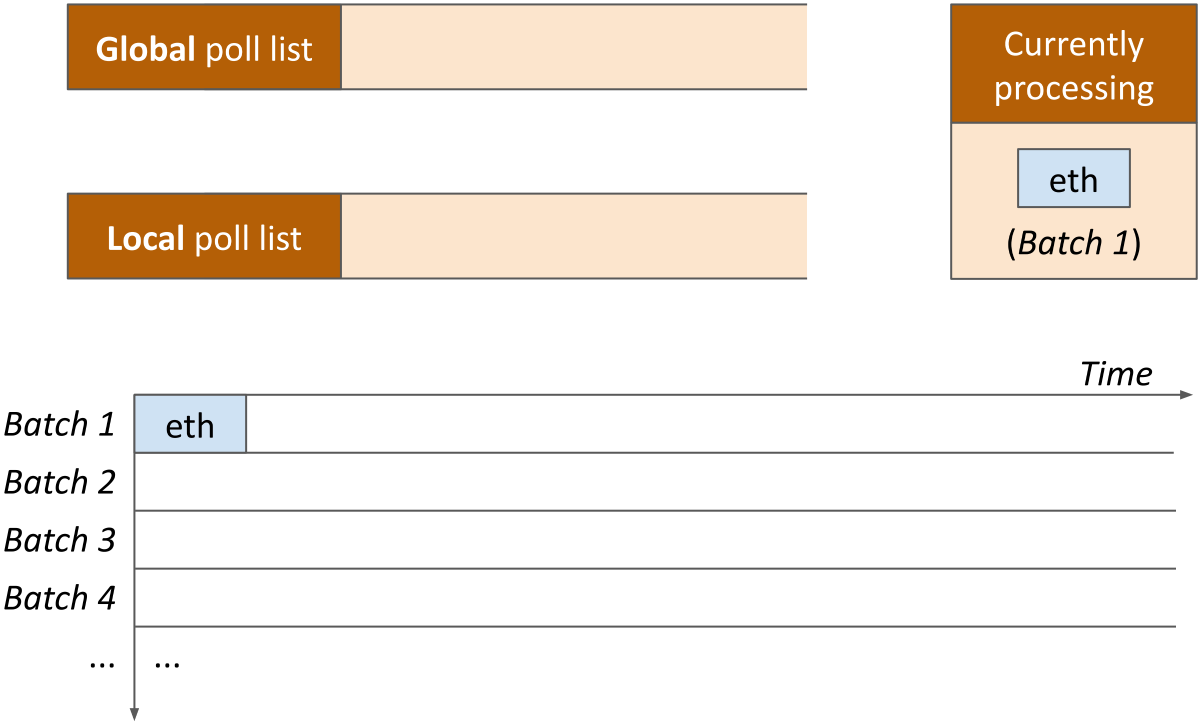
Batch 2

Batch 3

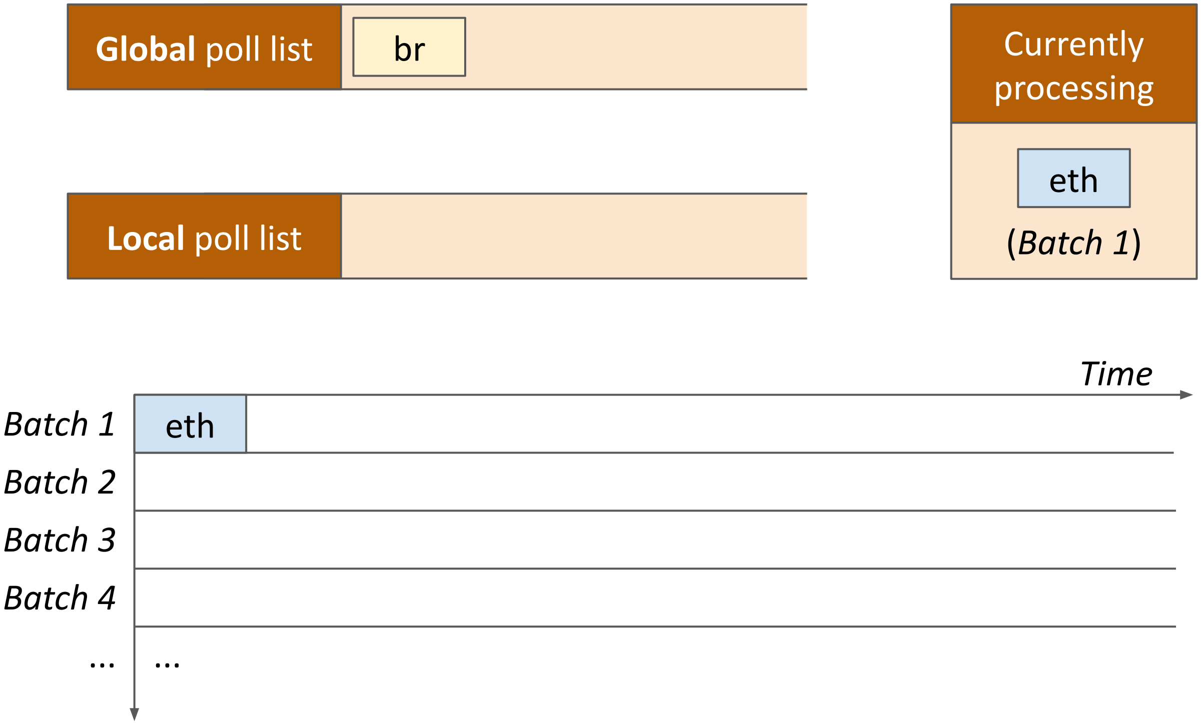
Batch 4

... ...

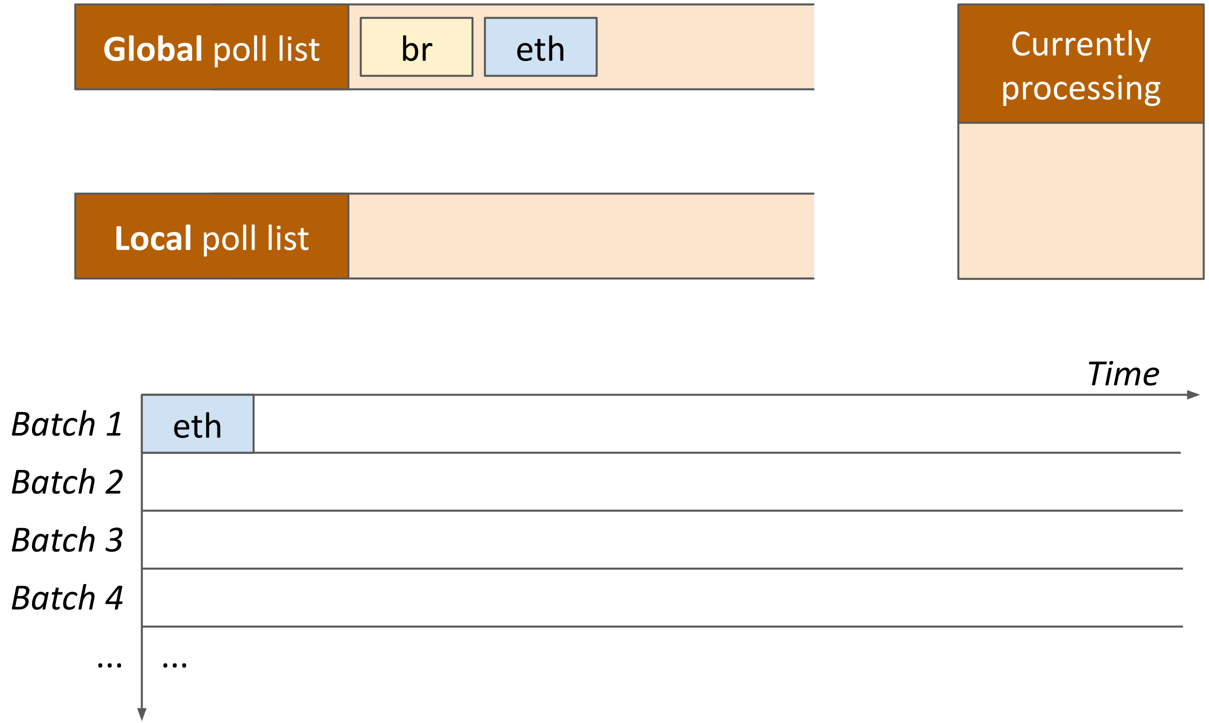
Interleaved NAPI device processing



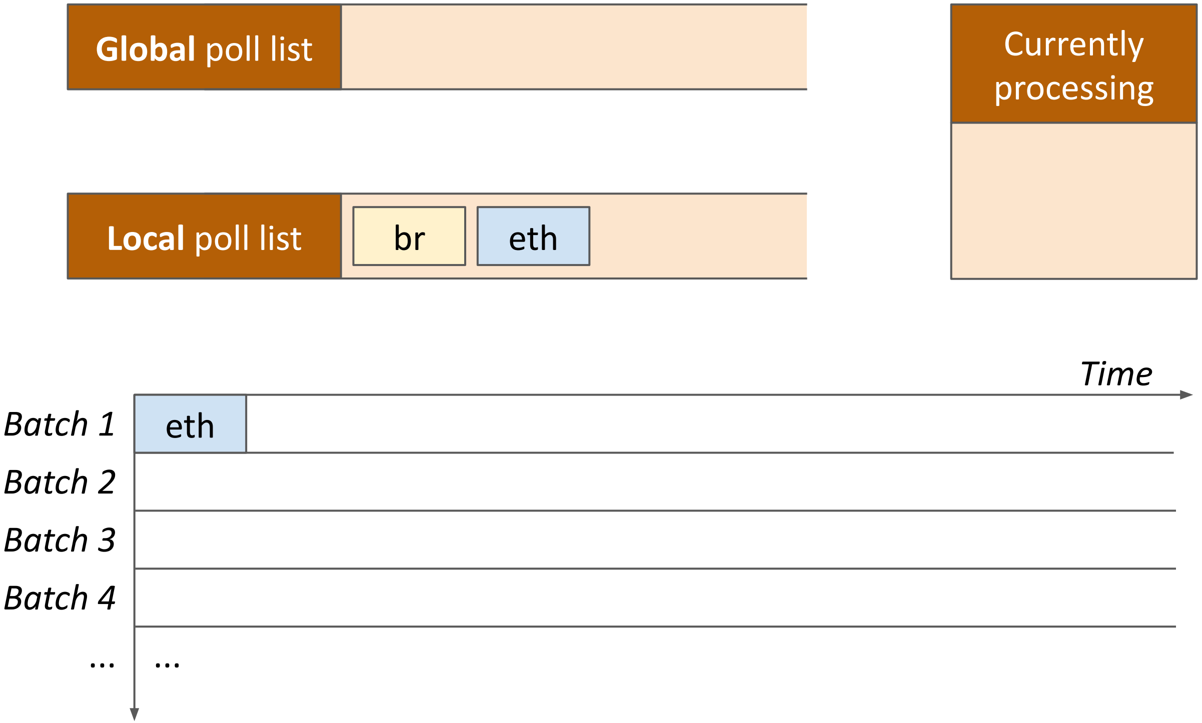
Interleaved NAPI device processing



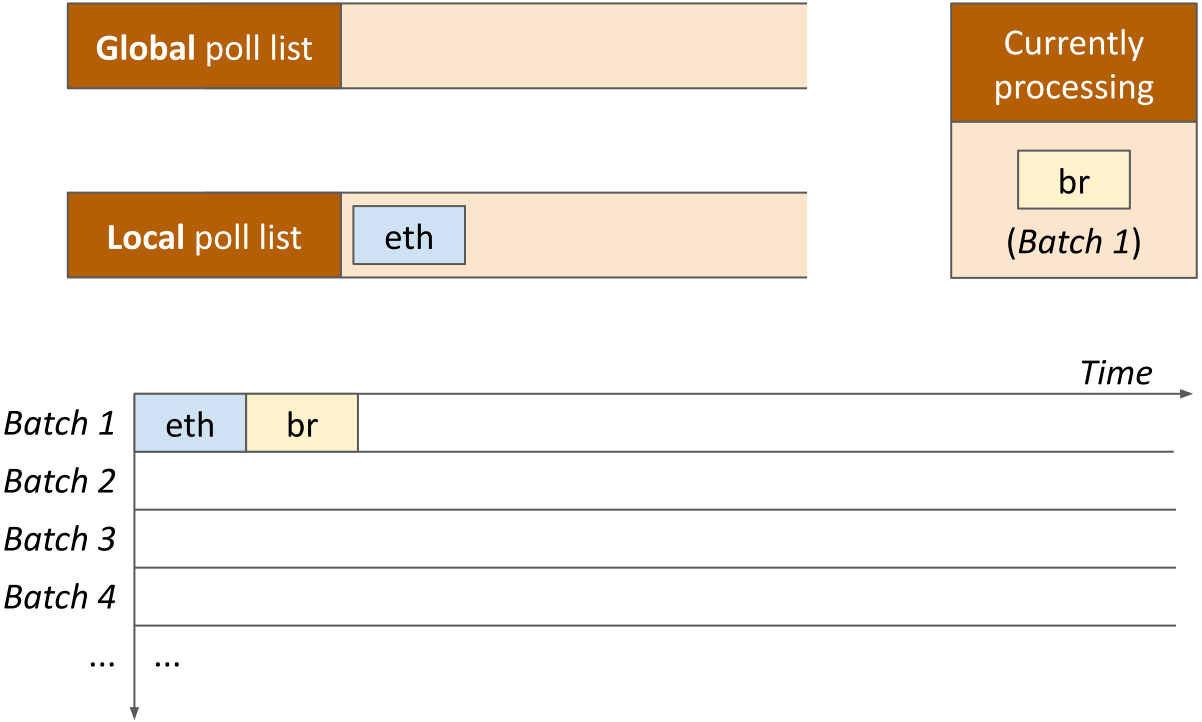
Interleaved NAPI device processing



Interleaved NAPI device processing



Interleaved NAPI device processing



Global poll list

Local poll list

eth

Currently processing

br

(*Batch 1*)

Time

Batch 1 eth br

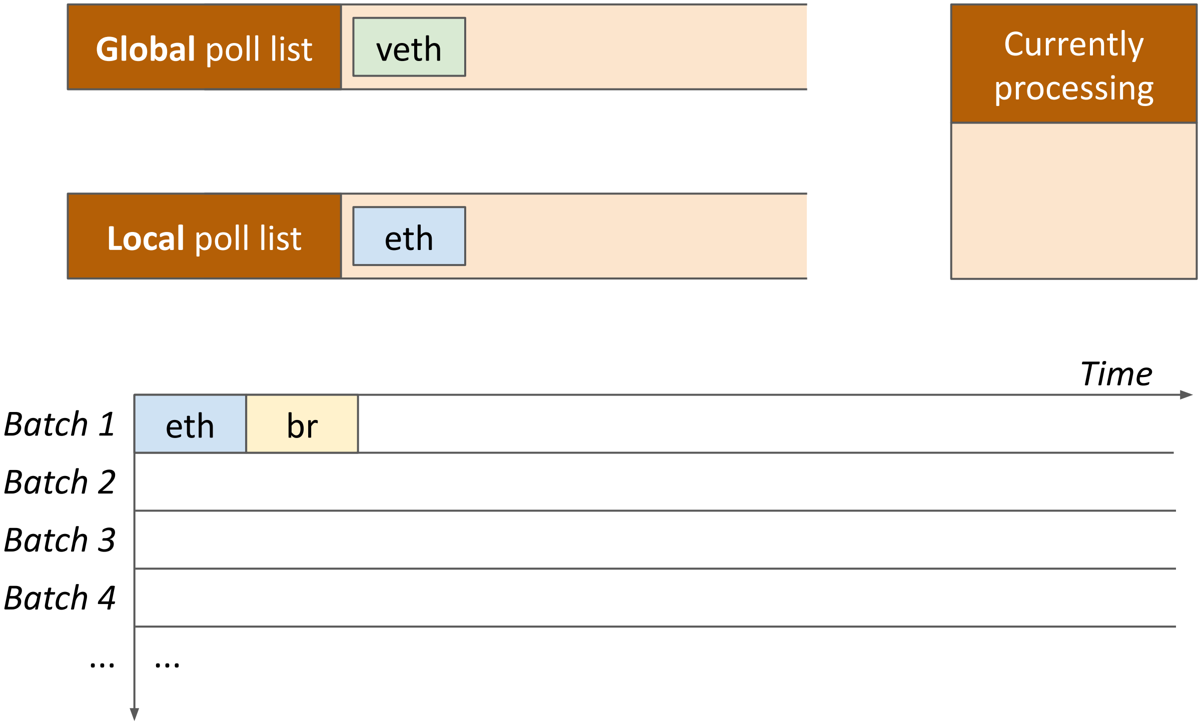
Batch 2

Batch 3

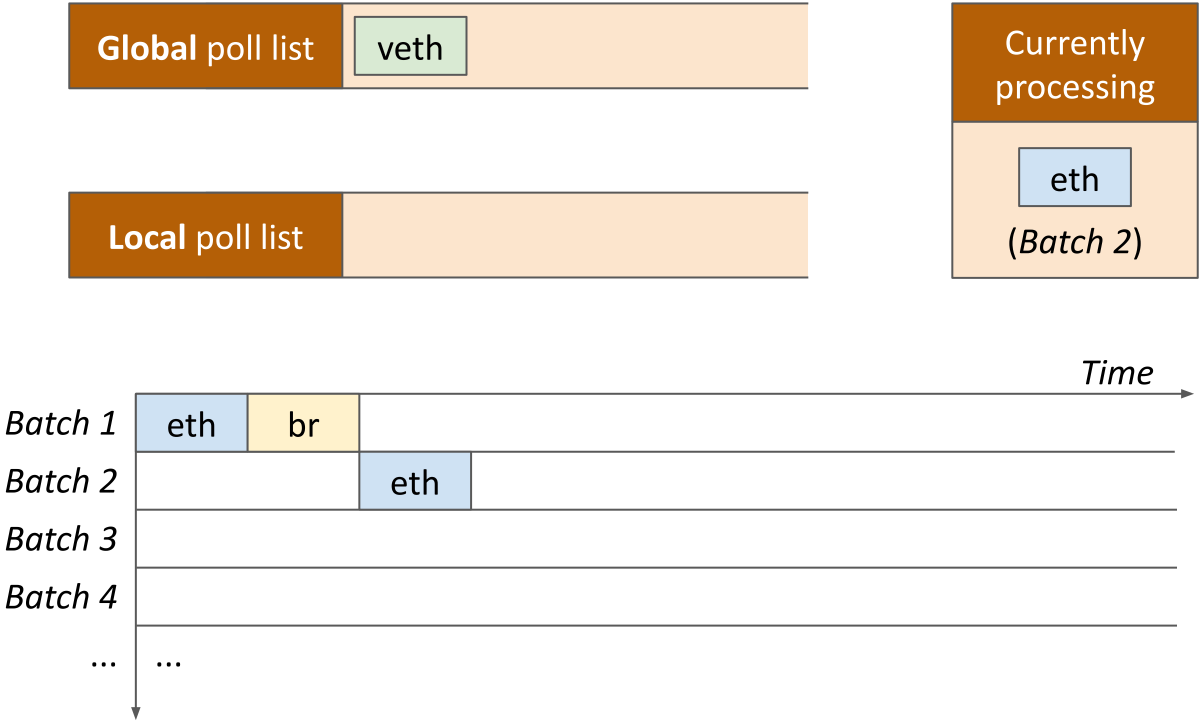
Batch 4

… …

Interleaved NAPI device processing



Interleaved NAPI device processing



Global poll list

veth

Local poll list

Currently processing

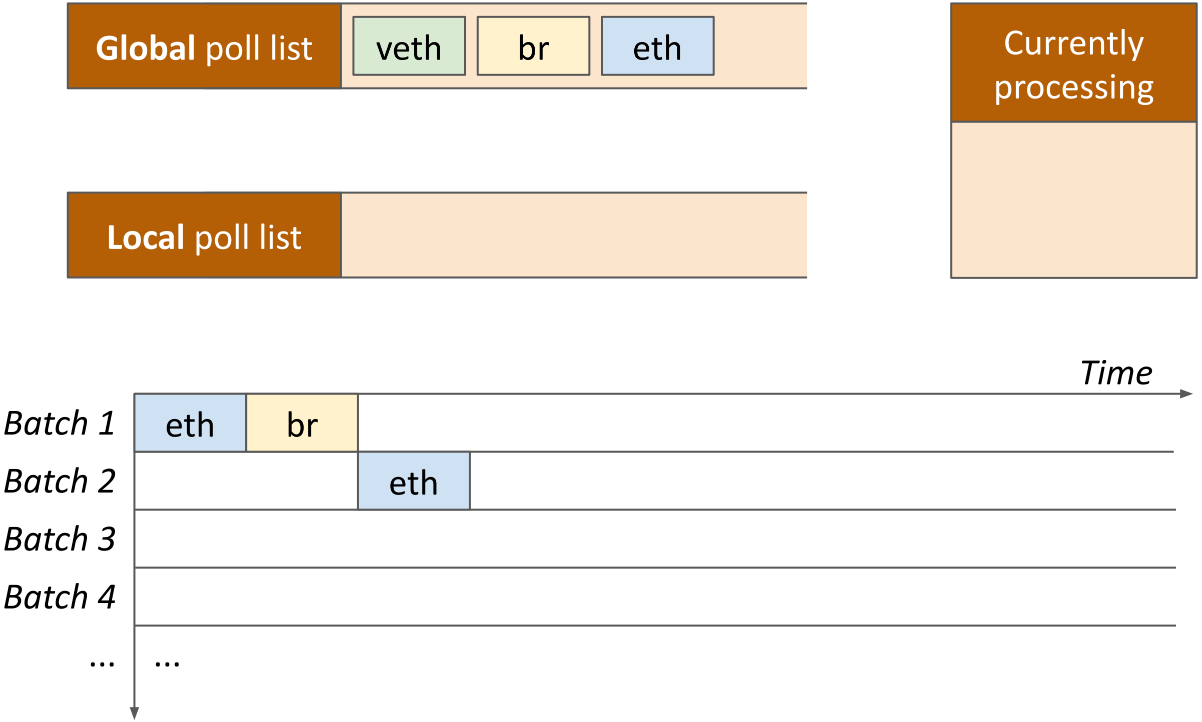
eth

(*Batch 2*)

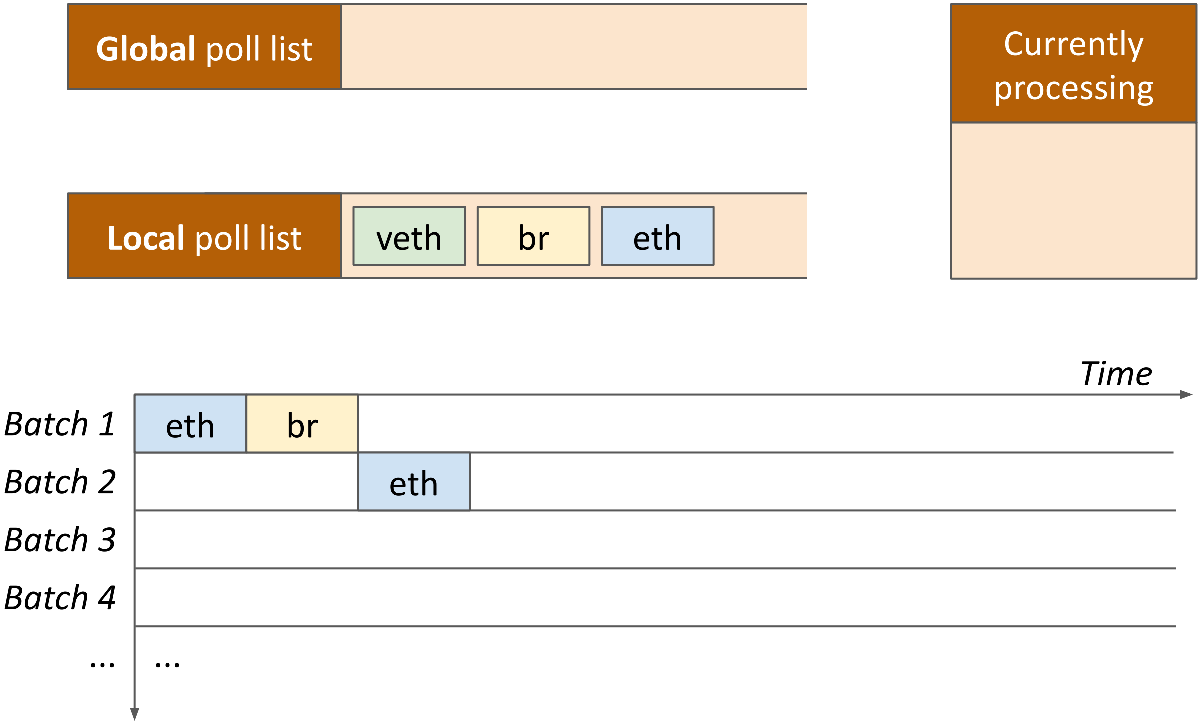
Time

Batch 1	eth	br	
Batch 2			eth
Batch 3			
Batch 4			
...	...		

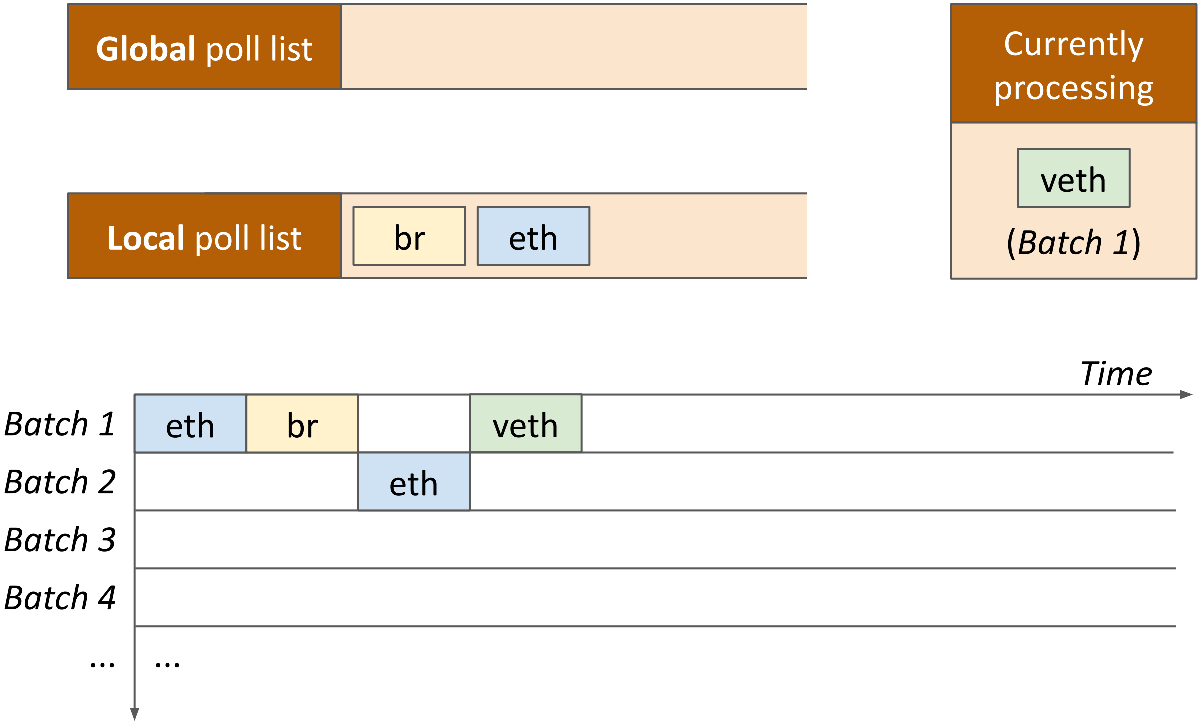
Interleaved NAPI device processing



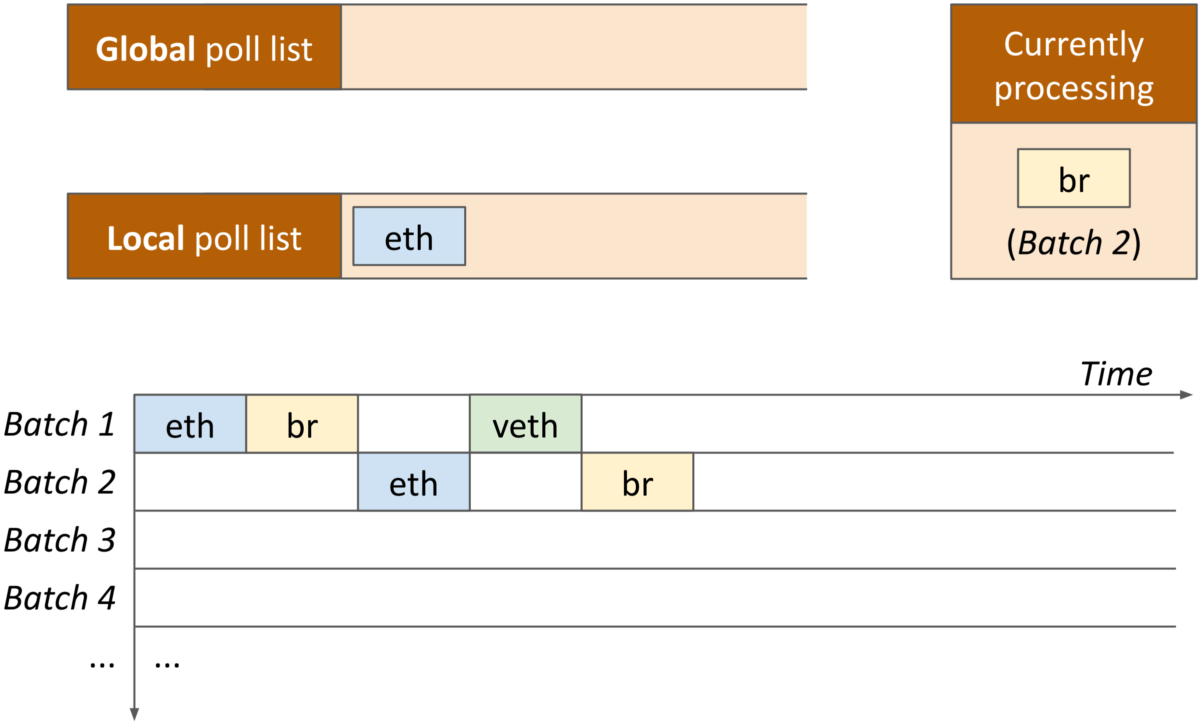
Interleaved NAPI device processing



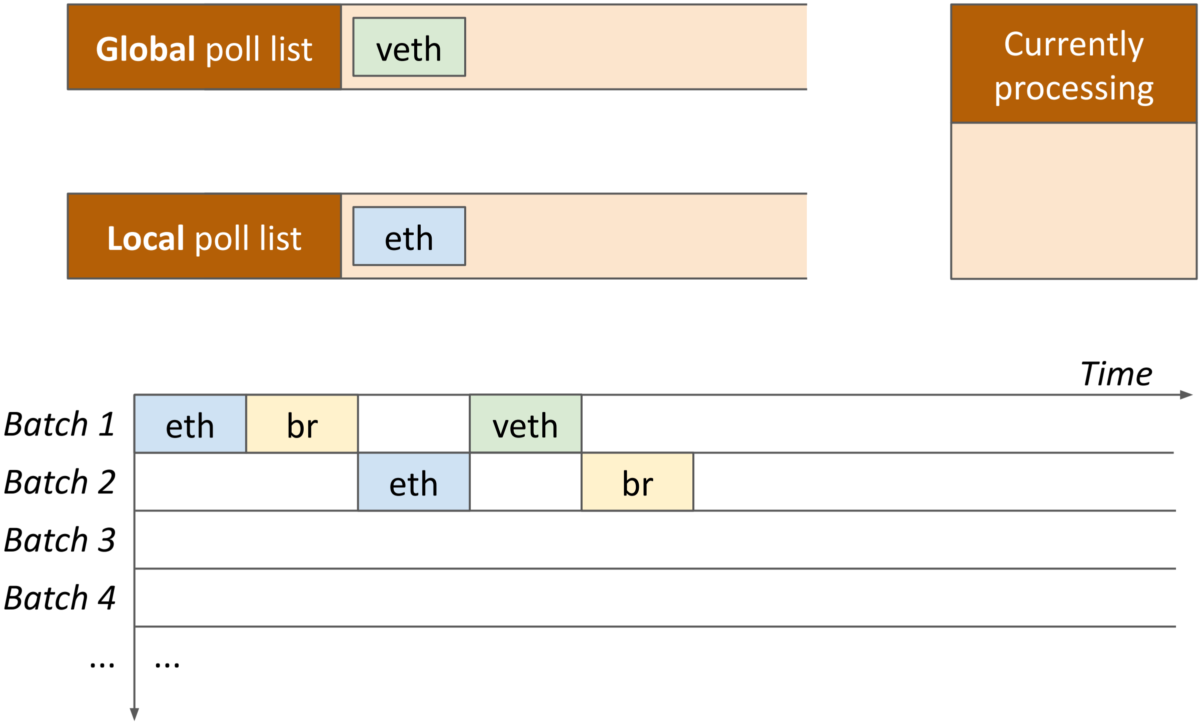
Interleaved NAPI device processing



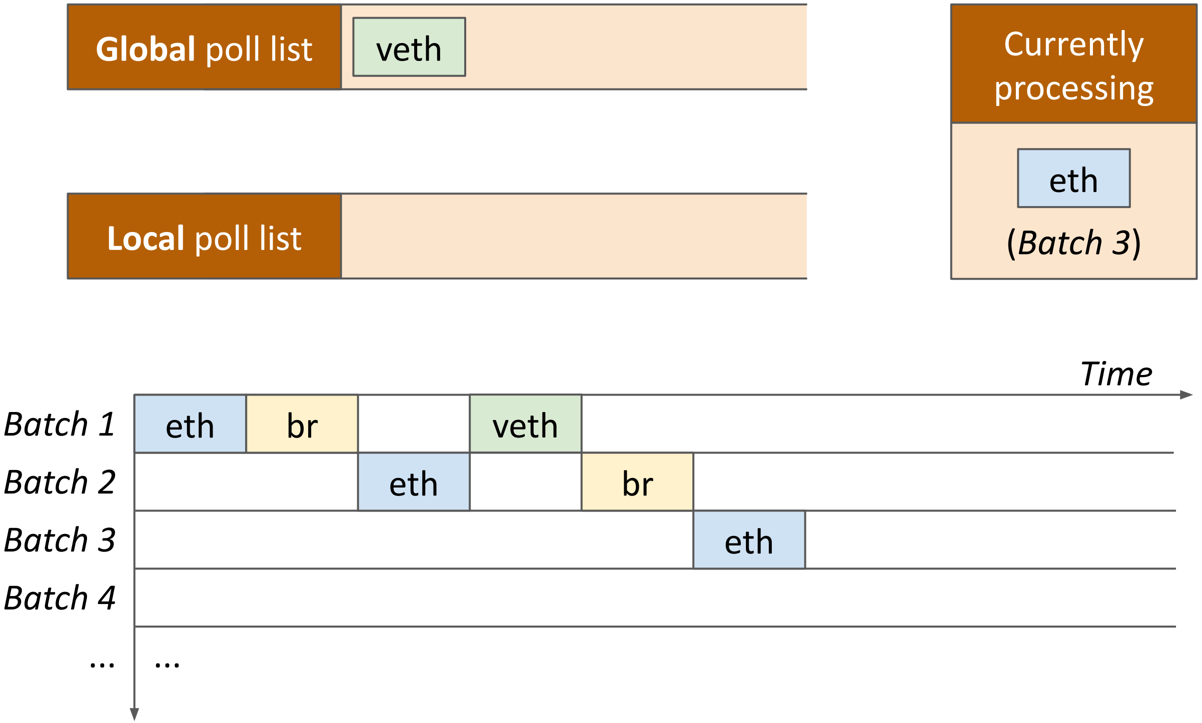
Interleaved NAPI device processing



Interleaved NAPI device processing



Interleaved NAPI device processing



Global poll list: veth

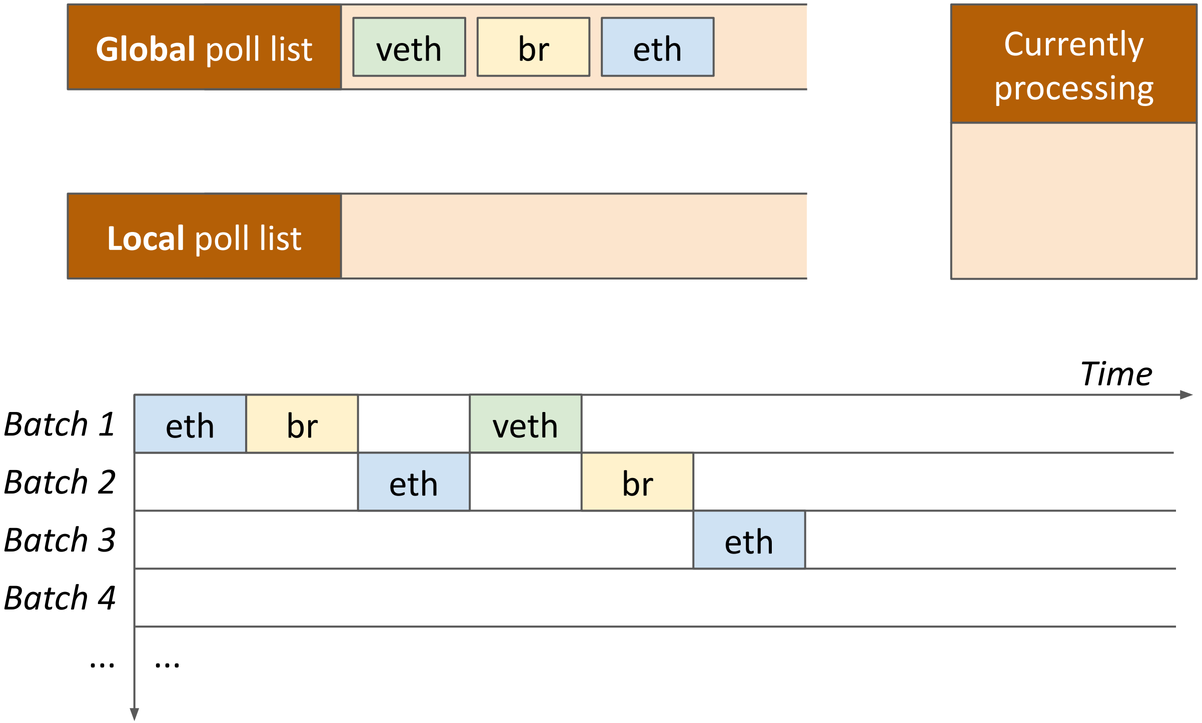
Local poll list:

Currently processing: eth (*Batch 3*)

Time

Batch 1	eth	br		veth		
Batch 2			eth		br	
Batch 3						eth
Batch 4						
…	…					

Interleaved NAPI device processing



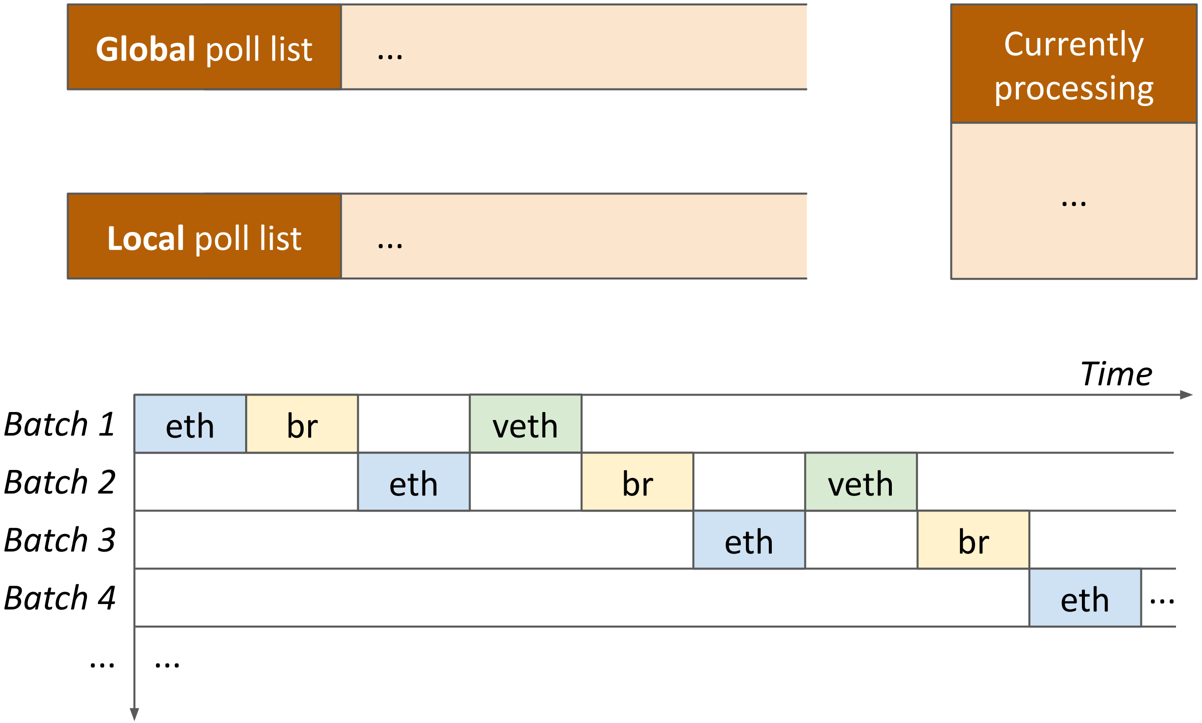
Global poll list: veth | br | eth

Local poll list

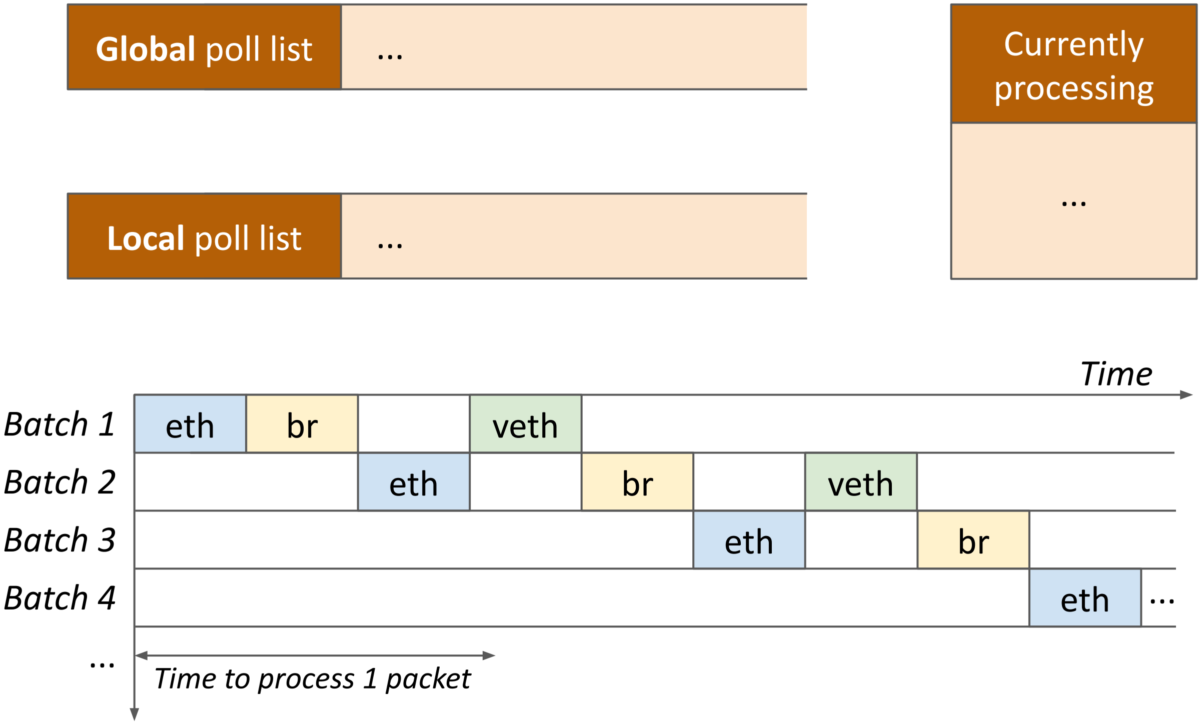
Currently processing

							Time
Batch 1	eth	br		veth			
Batch 2			eth		br		
Batch 3						eth	
Batch 4							
…	…						

Interleaved NAPI device processing



Interleaved NAPI device processing

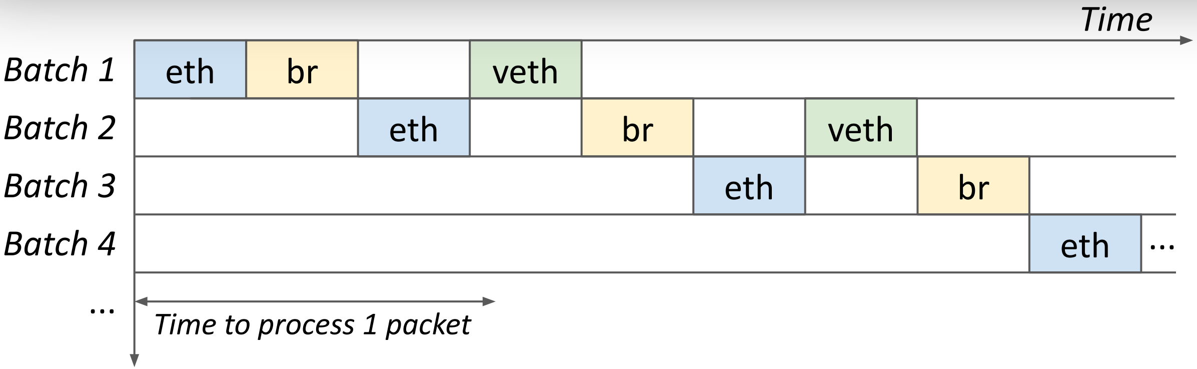


Batch 1	eth	br		veth						
Batch 2			eth		br		veth			
Batch 3						eth		br		
Batch 4									eth	…
…										

Interleaved NAPI device processing

Summary:

Vanilla kernel network stack **hurts latency** due to **interleaved processing** stages of batches.



Batch 1	eth	br		veth						
Batch 2			eth		br		veth			
Batch 3						eth		br		
Batch 4									eth	…
…										

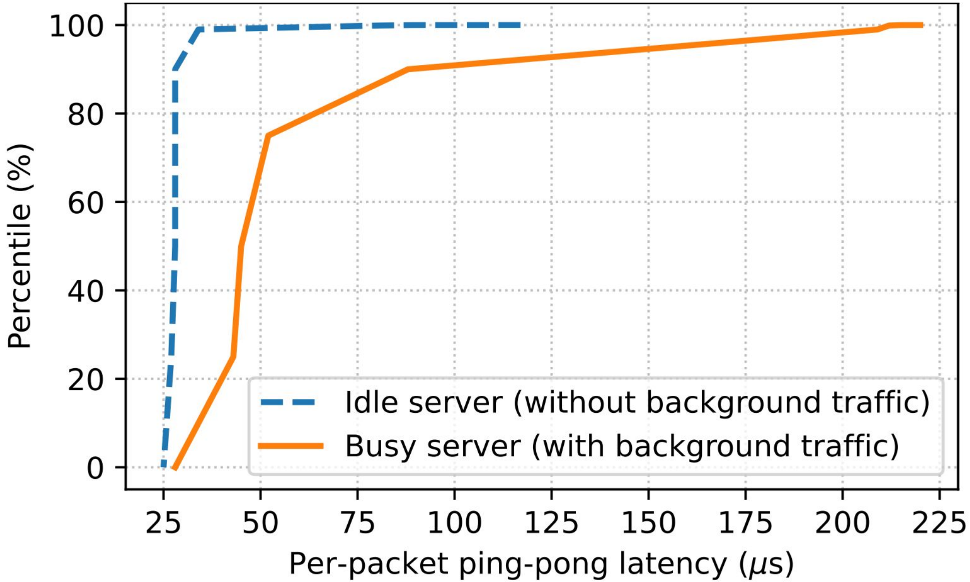
Lack of priority differentiation

- All packets are processed in FIFO order

- *Latency-sensitive* flows get stuck behind long queues filled with *throughput-sensitive* flows

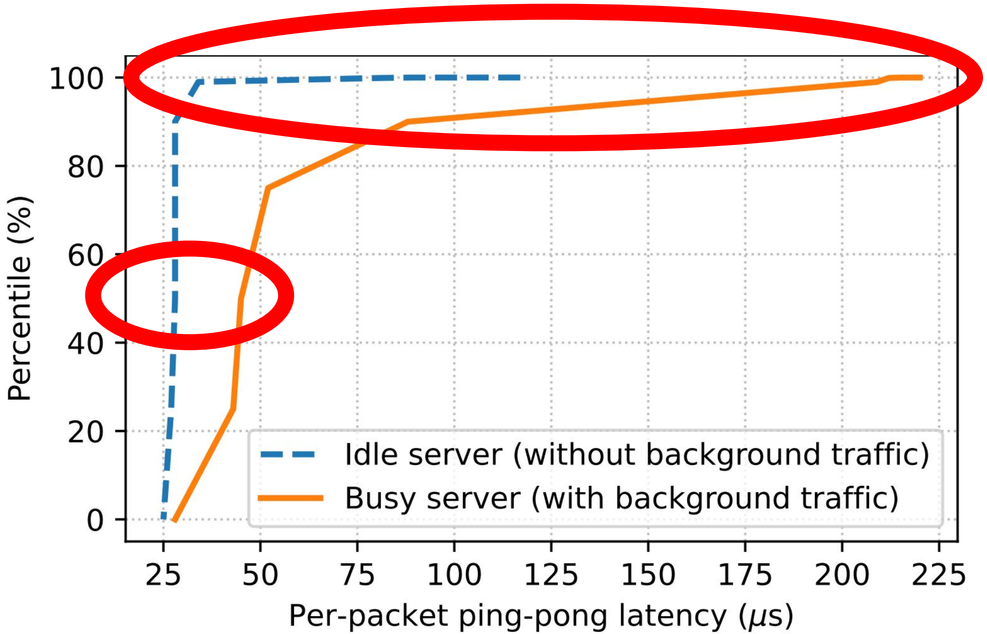
Lack of priority differentiation

- All packets are processed in FIFO order
- *Latency-sensitive* flows get stuck behind long queues filled with *throughput-sensitive* flows



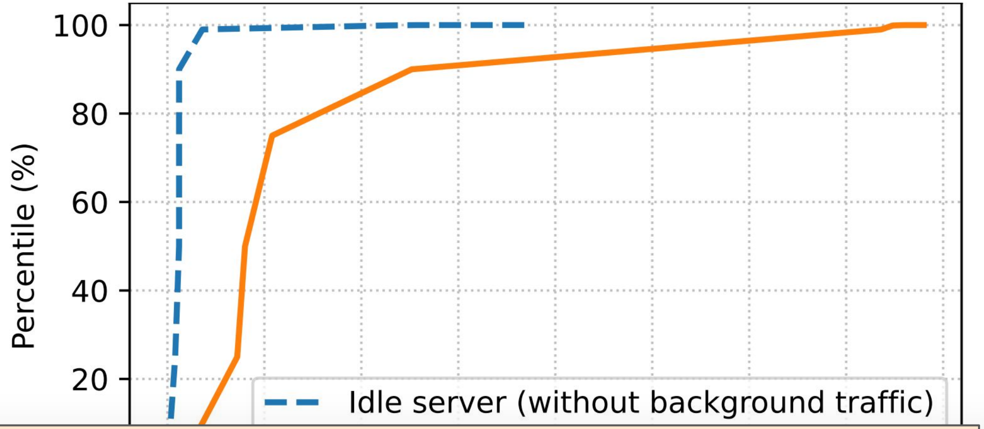
Lack of priority differentiation

- All packets are processed in FIFO order
- *Latency-sensitive* flows get stuck behind long queues filled with *throughput-sensitive* flows



Lack of priority differentiation

- All packets are processed in FIFO order
- *Latency-sensitive* flows get stuck behind long queues filled with



Is it possible to minimize latency of some flows while still offering sufficient throughput for other flows?

PRISM Design

Priority-based **S**trea**m**lined Packet Processing

- Improved NAPI design
- Priority differentiation
- Streamlined NAPI device polling

PRISM Design

Priority-based **S**trea**m**lined Packet Processing

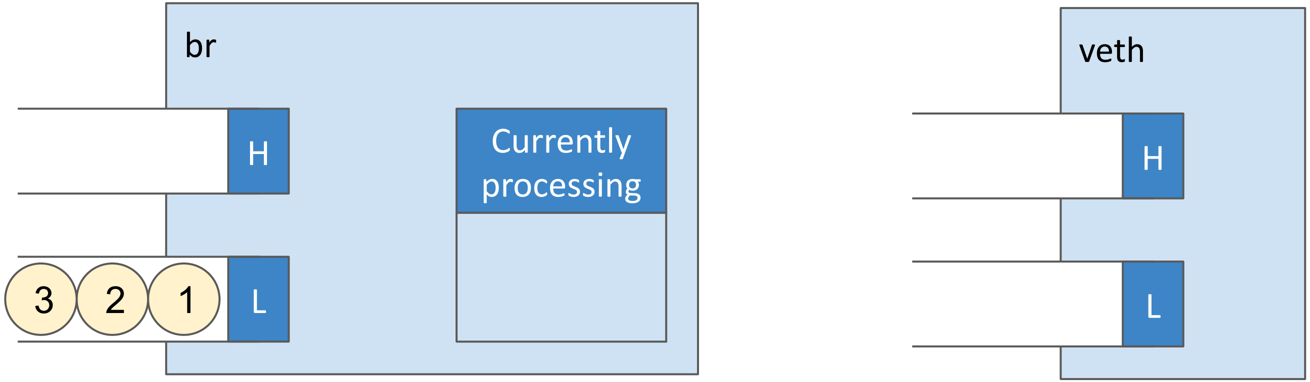
- Improved NAPI design
- Priority differentiation
- Streamlined NAPI device polling

Benefits over kernel-bypass / custom network stack:

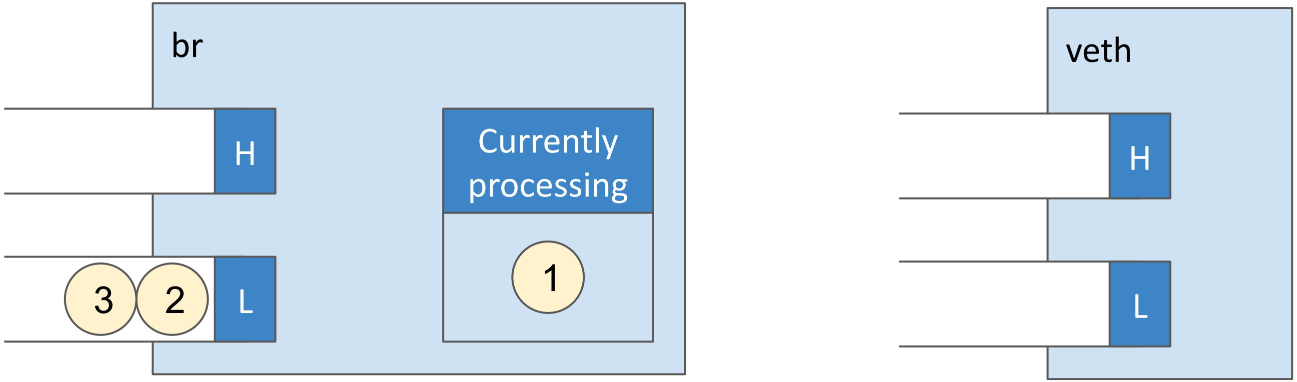
- Completely backward compatible
- No expensive hardware needed
- No need to change application code
- Preserves kernel's security, portability, and reliability

Priority differentiation

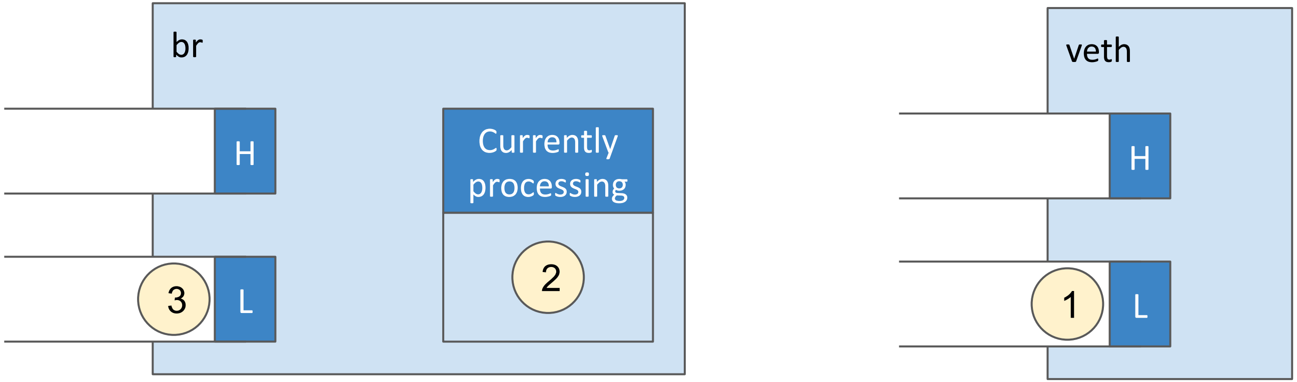
Priority differentiation



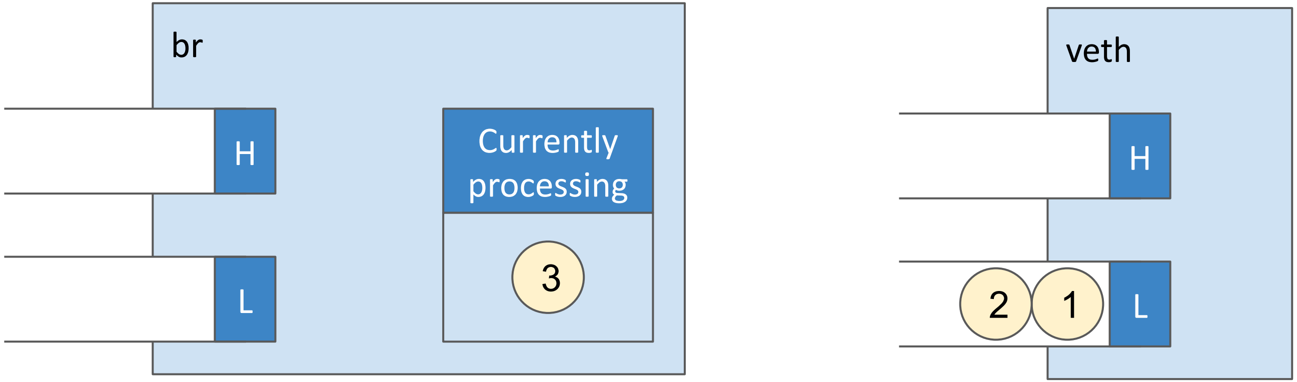
Priority differentiation



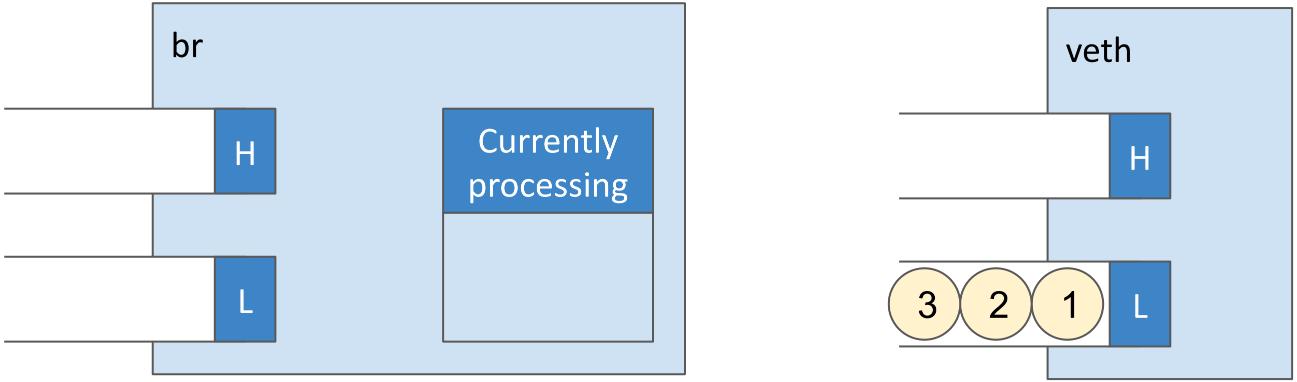
Priority differentiation



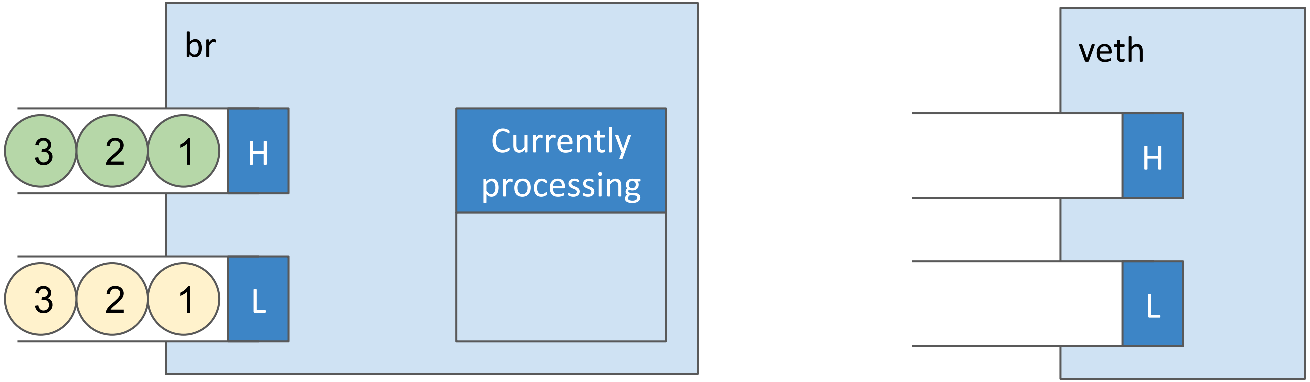
Priority differentiation



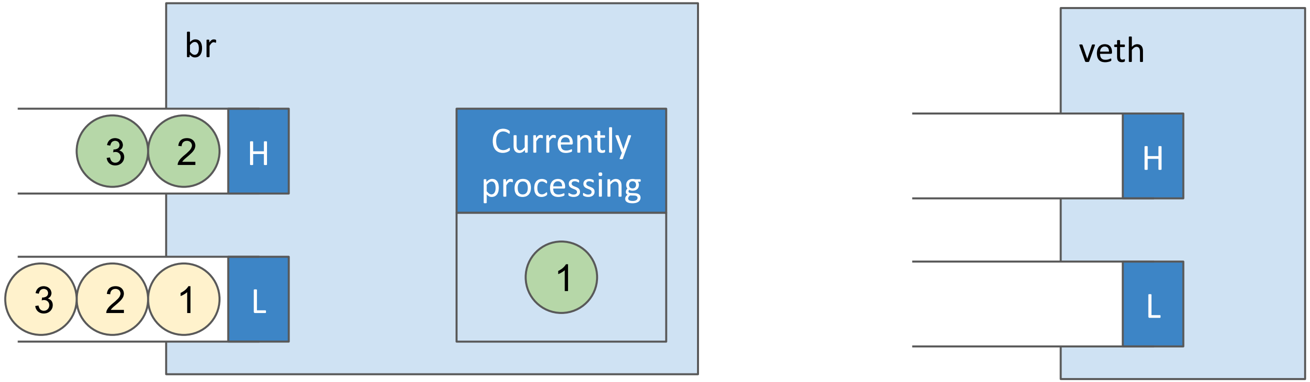
Priority differentiation



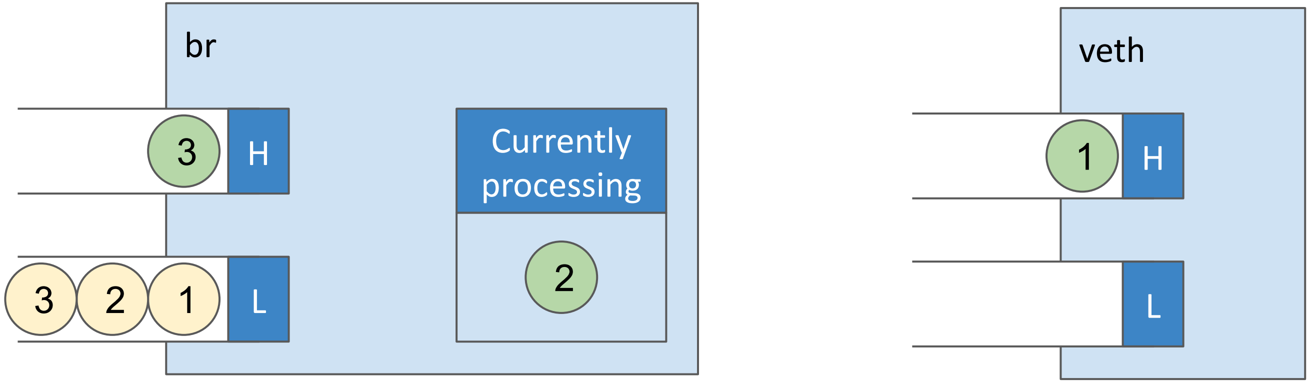
Priority differentiation



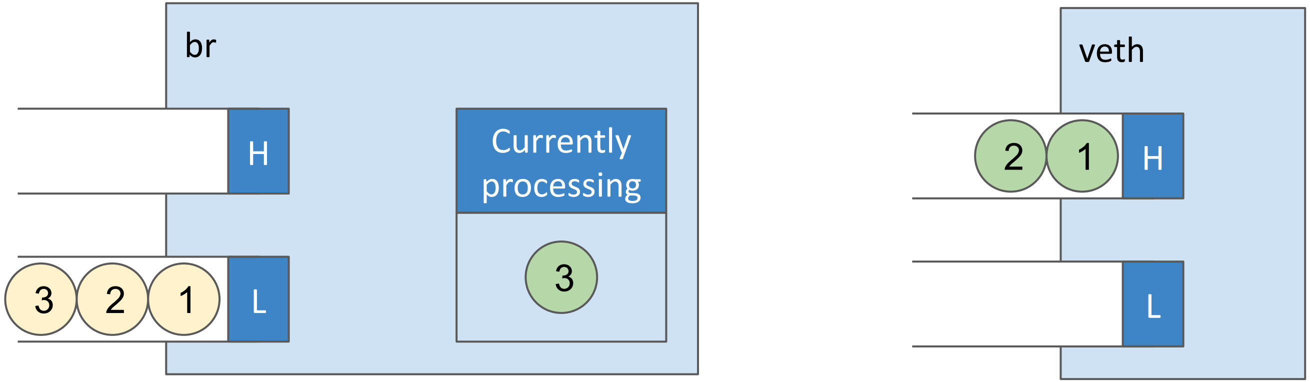
Priority differentiation



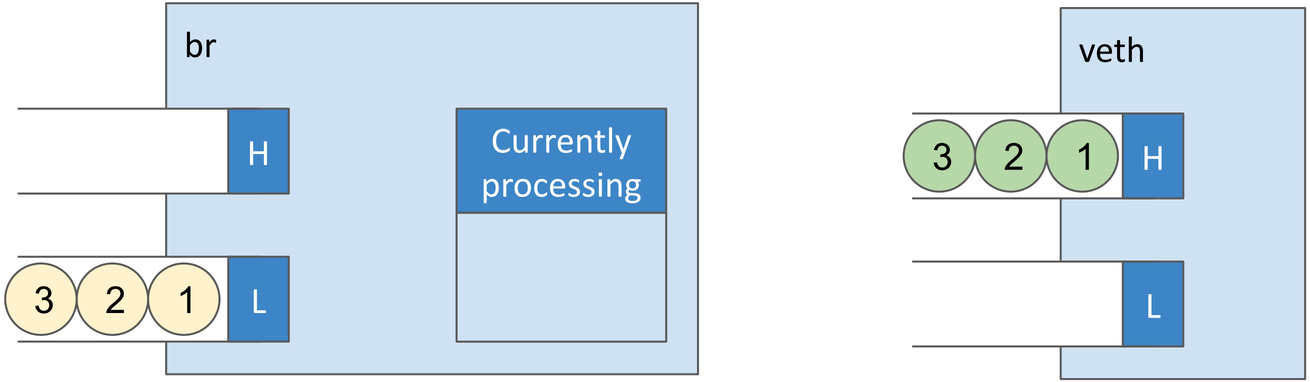
Priority differentiation



Priority differentiation



Priority differentiation



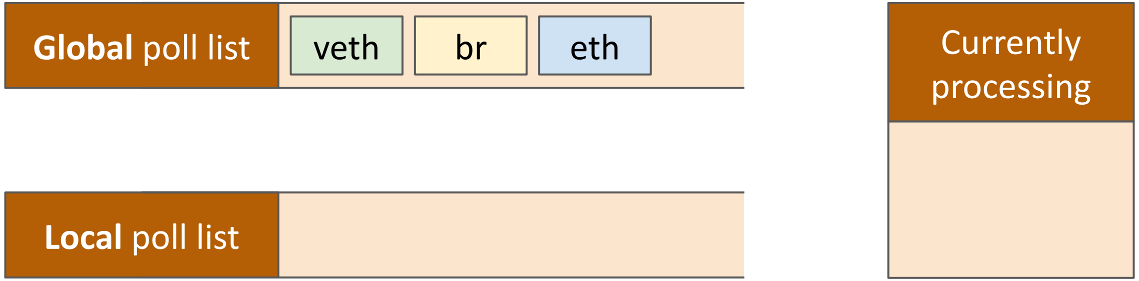
Priority differentiation

PRISM enables priority differentiation by:

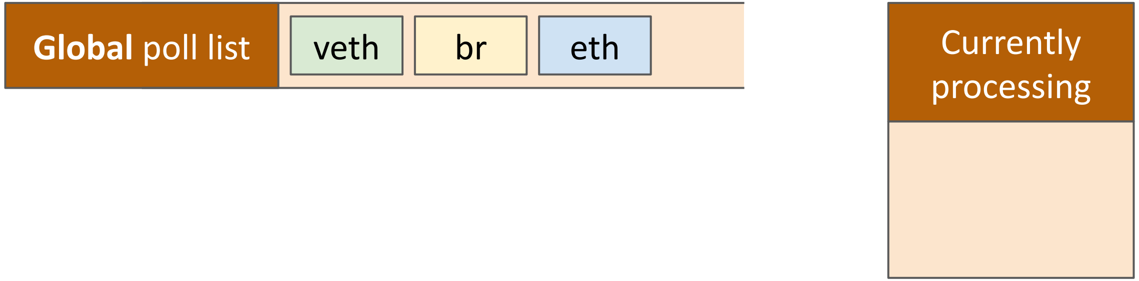
- Using a dedicated high-priority packet queue
- Batch-level preemption

PRISM-batch

PRISM-batch


Global poll list
veth
br
eth
Local poll list
Currently processing

PRISM-batch



Global poll list

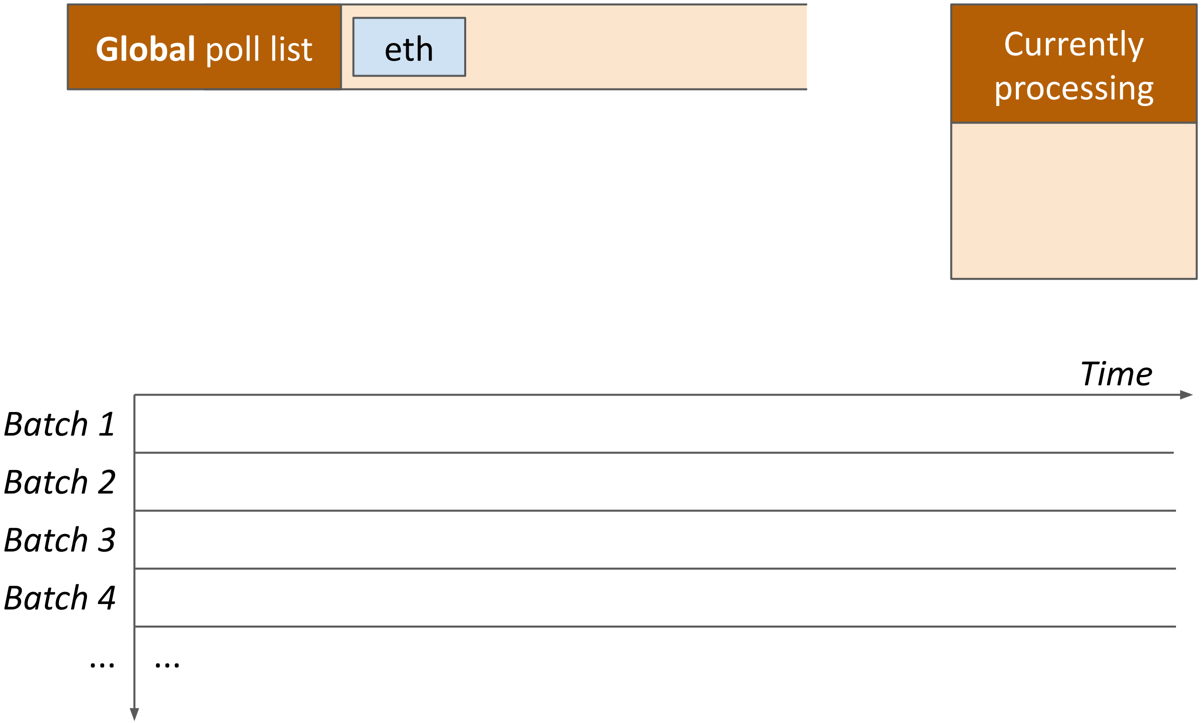
veth

br

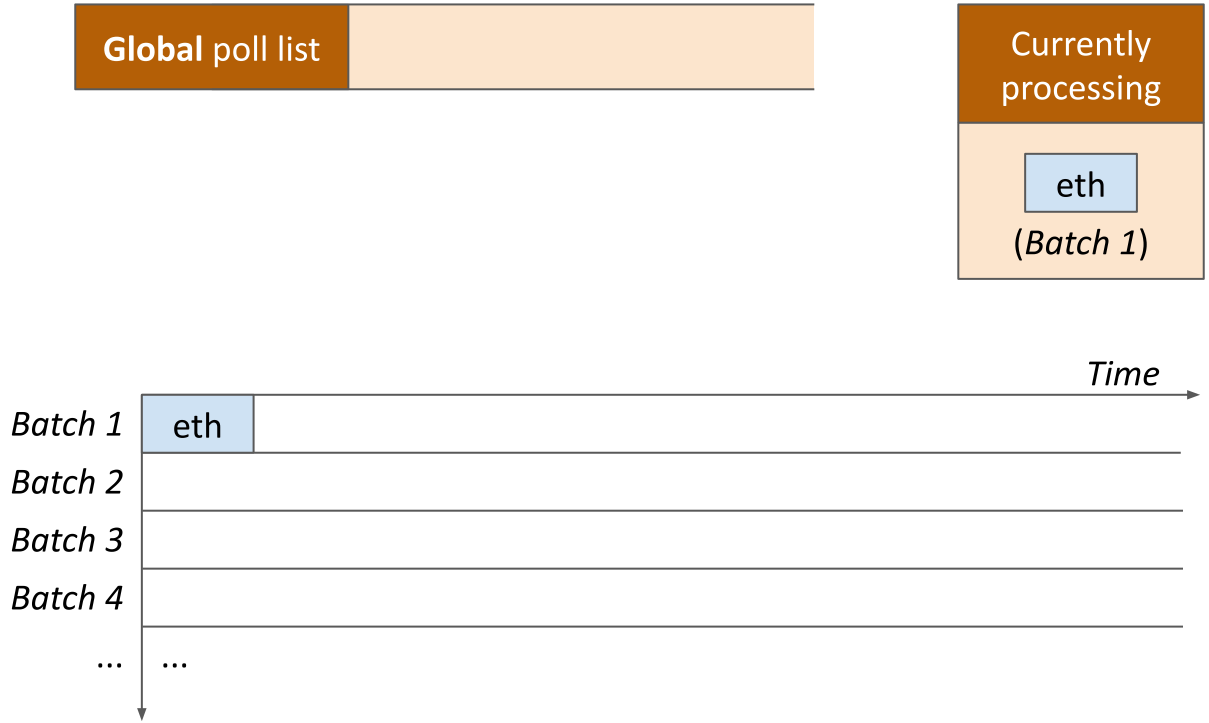
eth

Currently processing

PRISM-batch


Global poll list
eth
Currently processing
Time
Batch 1
Batch 2
Batch 3
Batch 4
…
…

PRISM-batch



Global poll list

Currently processing

eth

(*Batch 1*)

Time

Batch 1 eth

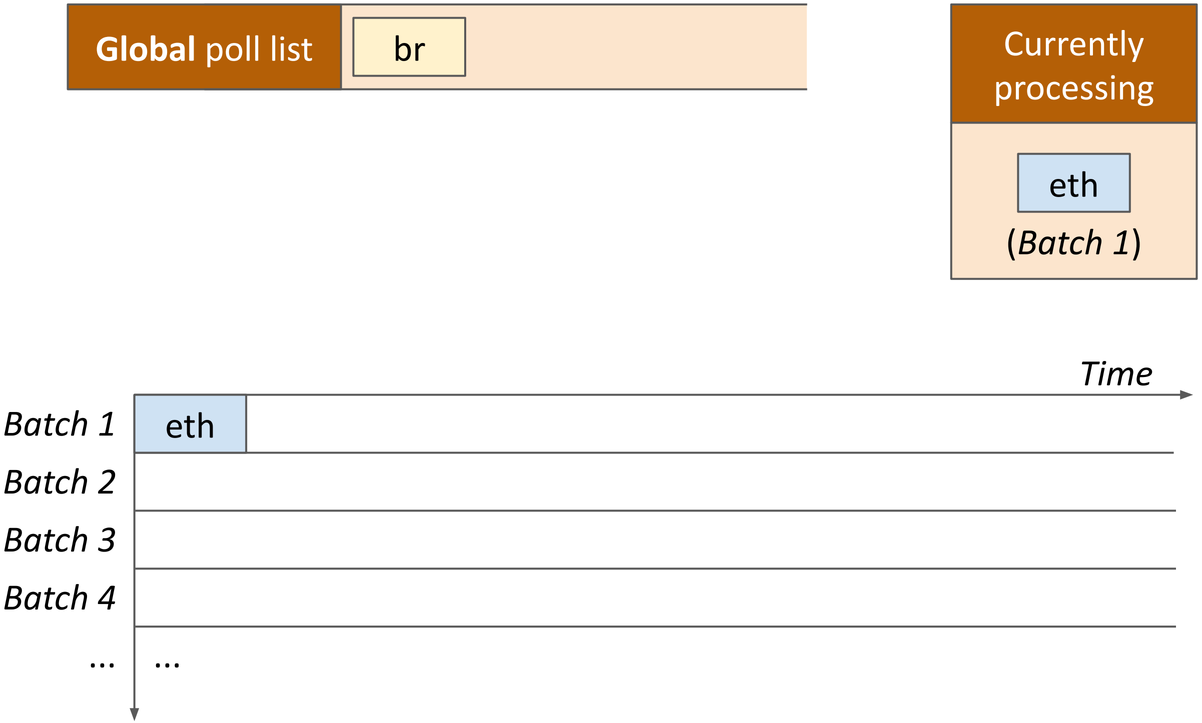
Batch 2

Batch 3

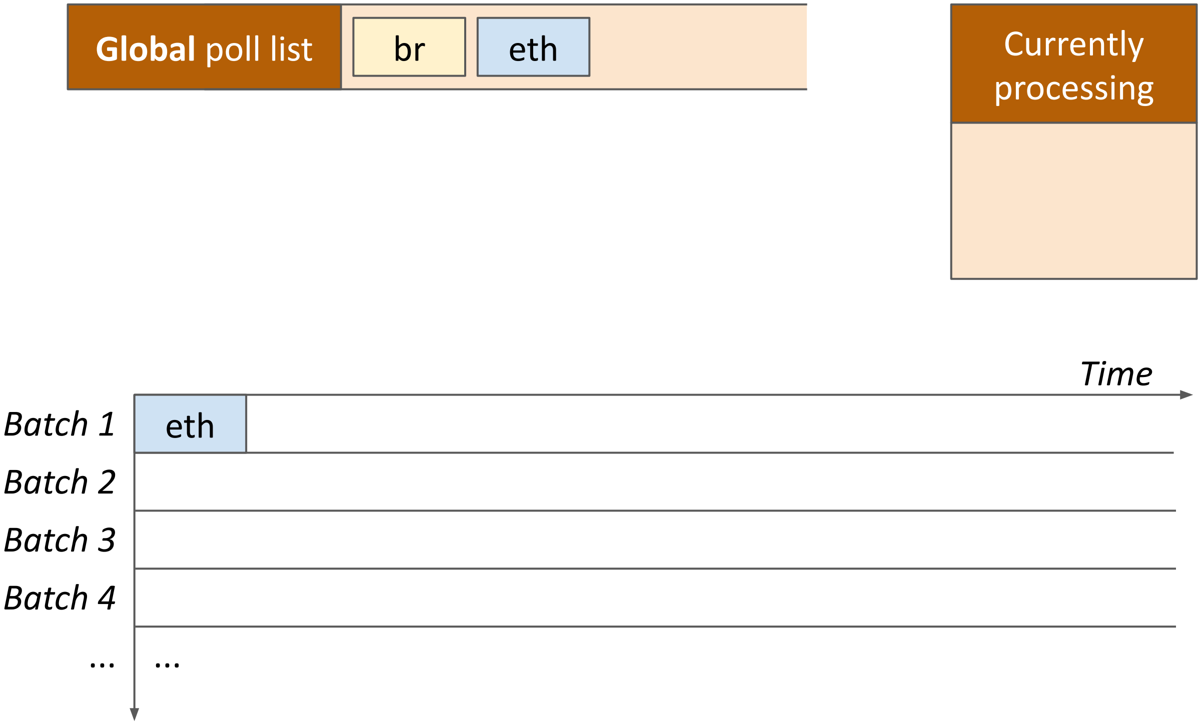
Batch 4

... ...

PRISM-batch


Global poll list
br
Currently processing
eth
(*Batch 1*)
Time
Batch 1
eth
Batch 2
Batch 3
Batch 4
…
…

PRISM-batch



Global poll list

br

eth

Currently processing

Time

Batch 1

eth

Batch 2

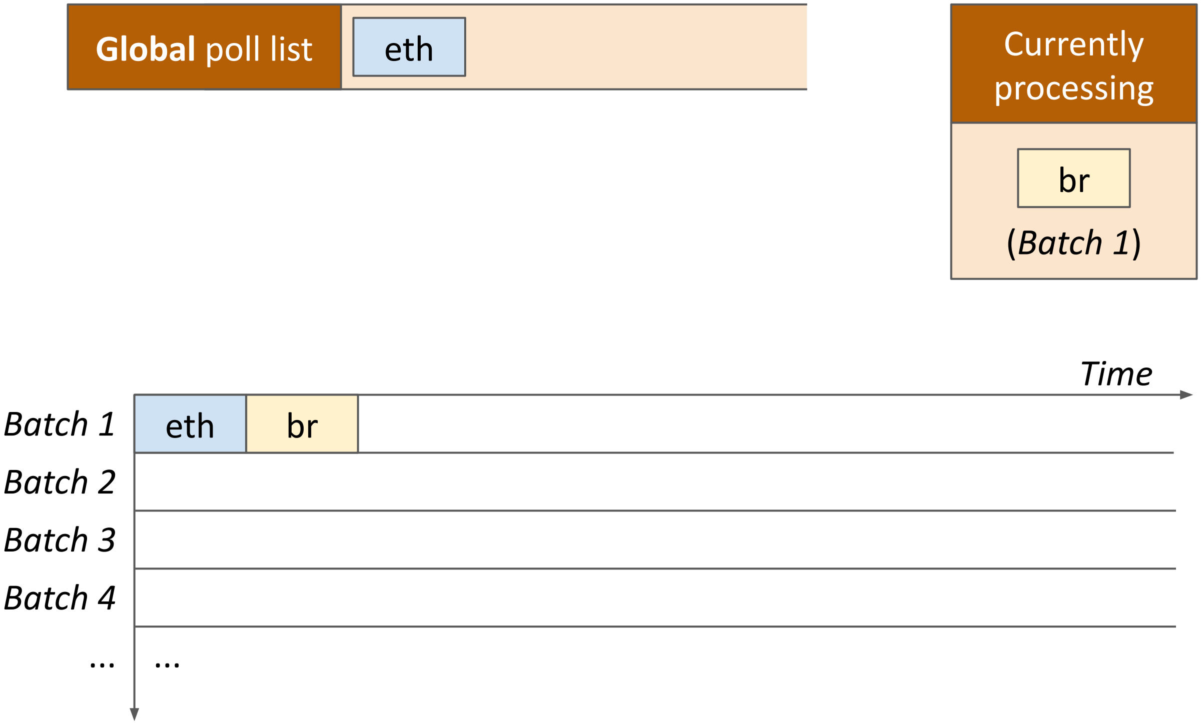
Batch 3

Batch 4

...

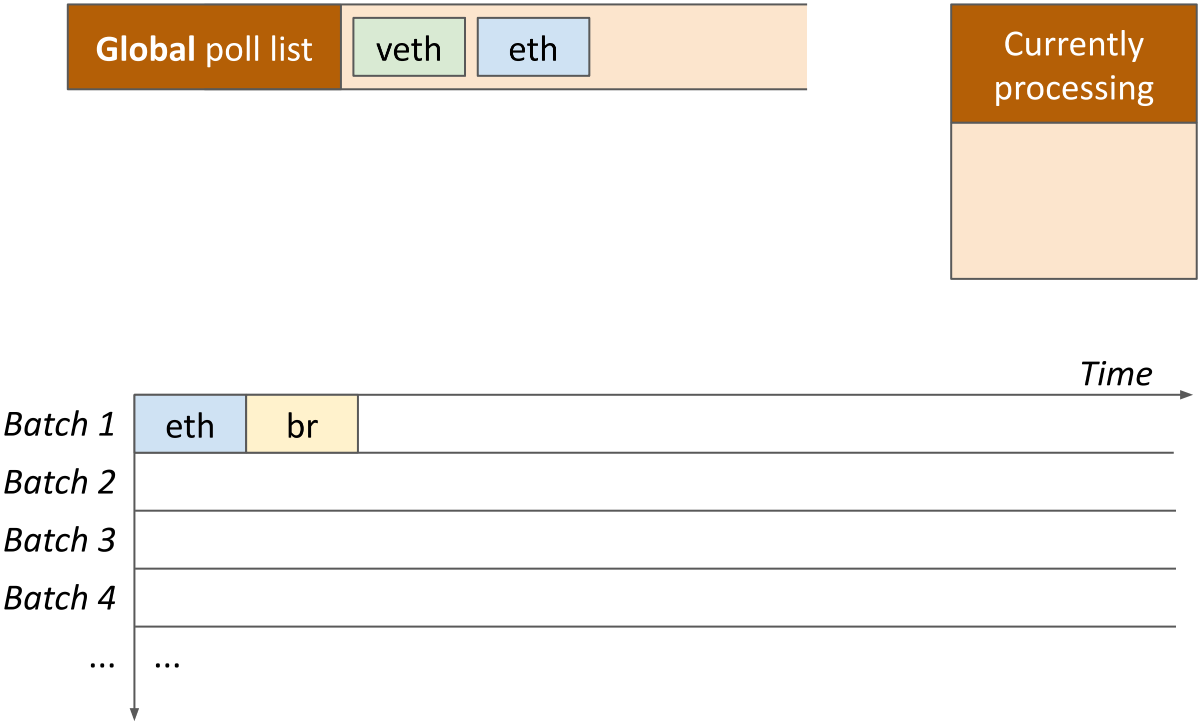
...

PRISM-batch

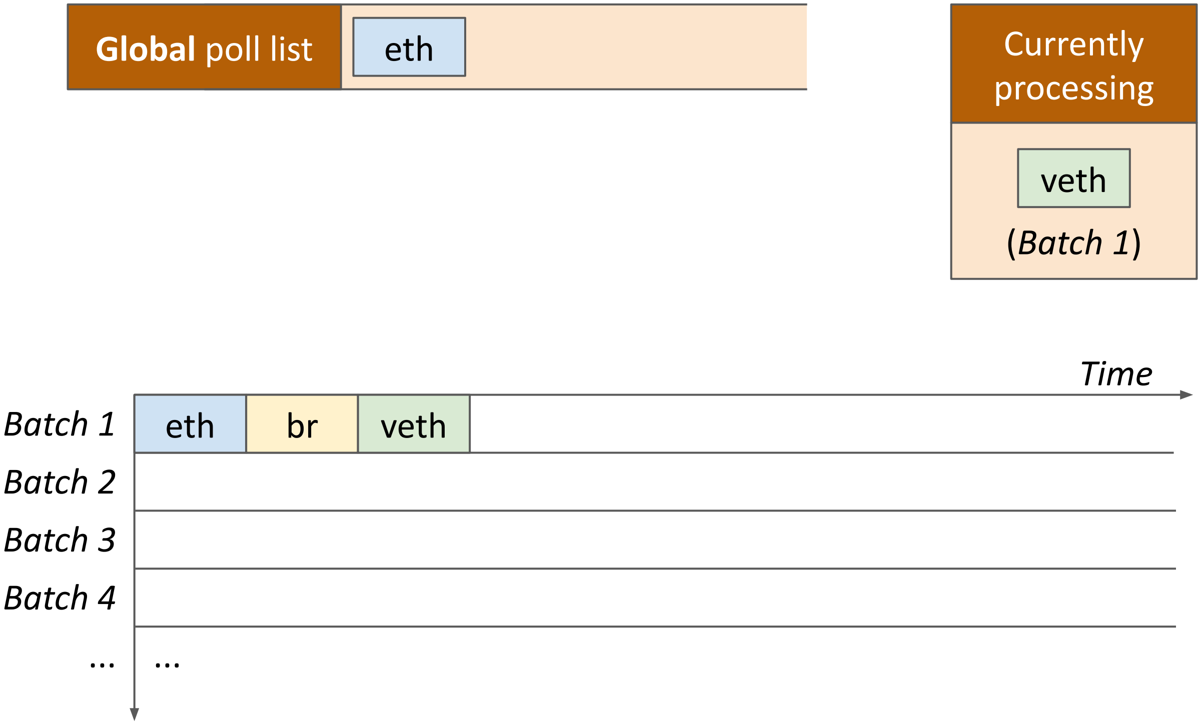


Global poll list
eth
Currently processing
br
(*Batch 1*)
Time
Batch 1
eth
br
Batch 2
Batch 3
Batch 4
...
...

PRISM-batch


Global poll list
veth
eth
Currently processing
Time
Batch 1
eth
br
Batch 2
Batch 3
Batch 4
...
...

PRISM-batch



Global poll list

eth

Currently processing

veth

(*Batch 1*)

Time

Batch 1 eth br veth

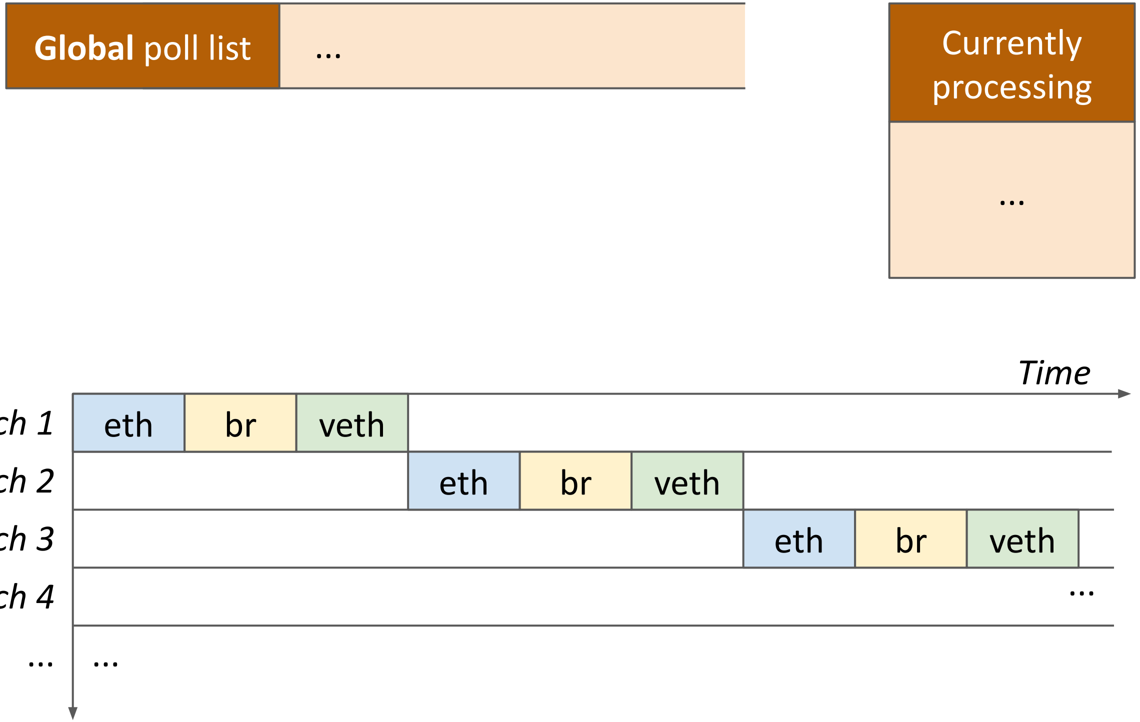
Batch 2

Batch 3

Batch 4

… …

PRISM-batch

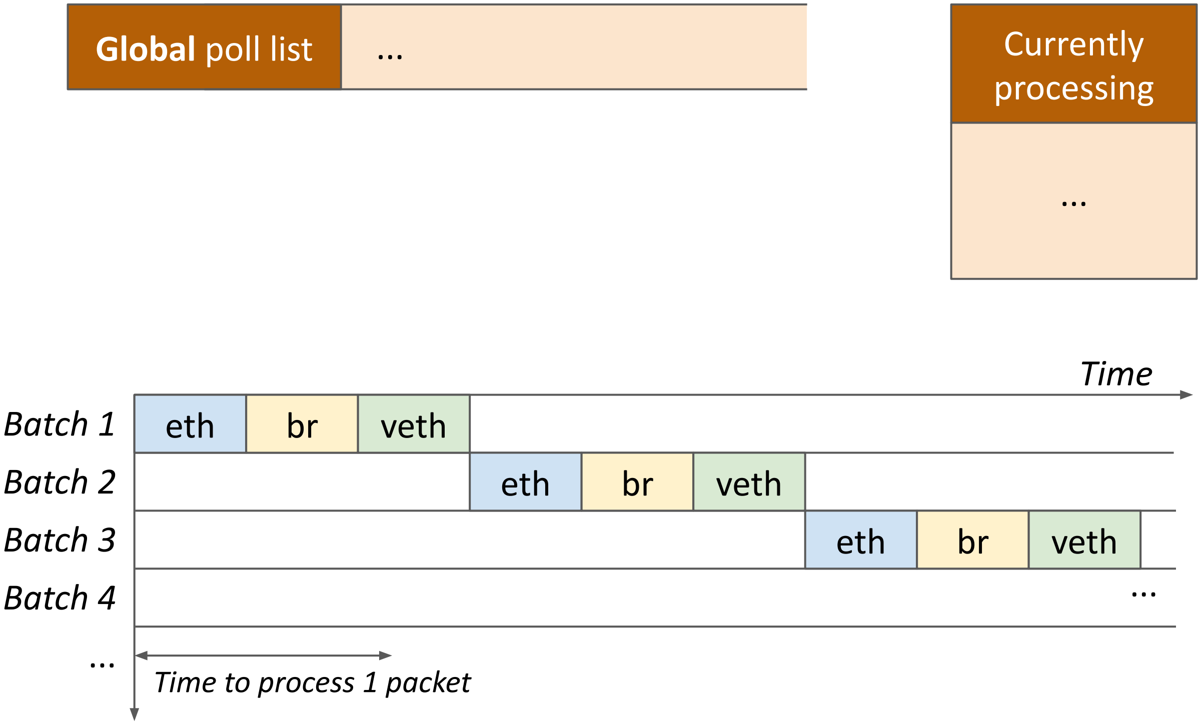


Global poll list	…

Currently processing
…

										Time
Batch 1	eth	br	veth							
Batch 2				eth	br	veth				
Batch 3							eth	br	veth	
Batch 4										…
…	…									

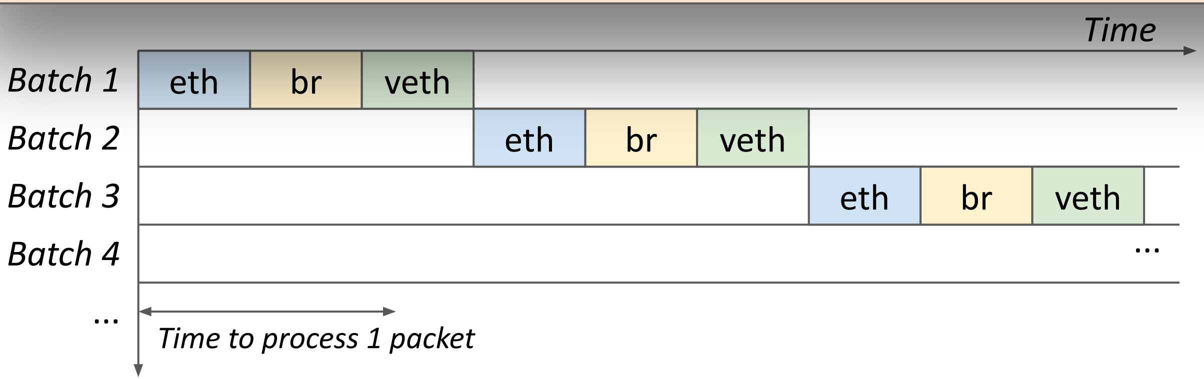
PRISM-batch


Global poll list
...
Currently processing
...
Time
Batch 1
eth
br
veth
Batch 2
eth
br
veth
Batch 3
eth
br
veth
Batch 4
...
...
Time to process 1 packet

PRISM-batch

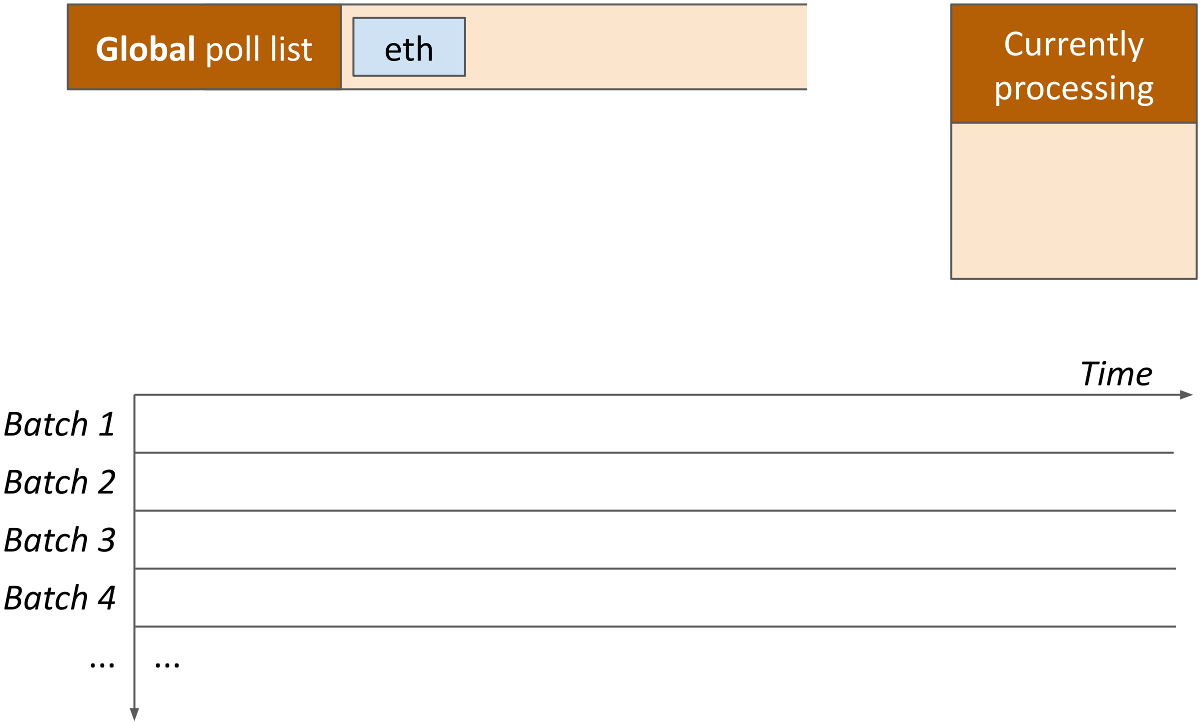
PRISM-batch:

- Improves latency by streamlining processing of batches
- Enables batch-level preemption
- But still maintains batching benefits



PRISM-sync

PRISM-sync



Global poll list

eth

Currently processing

Time

Batch 1

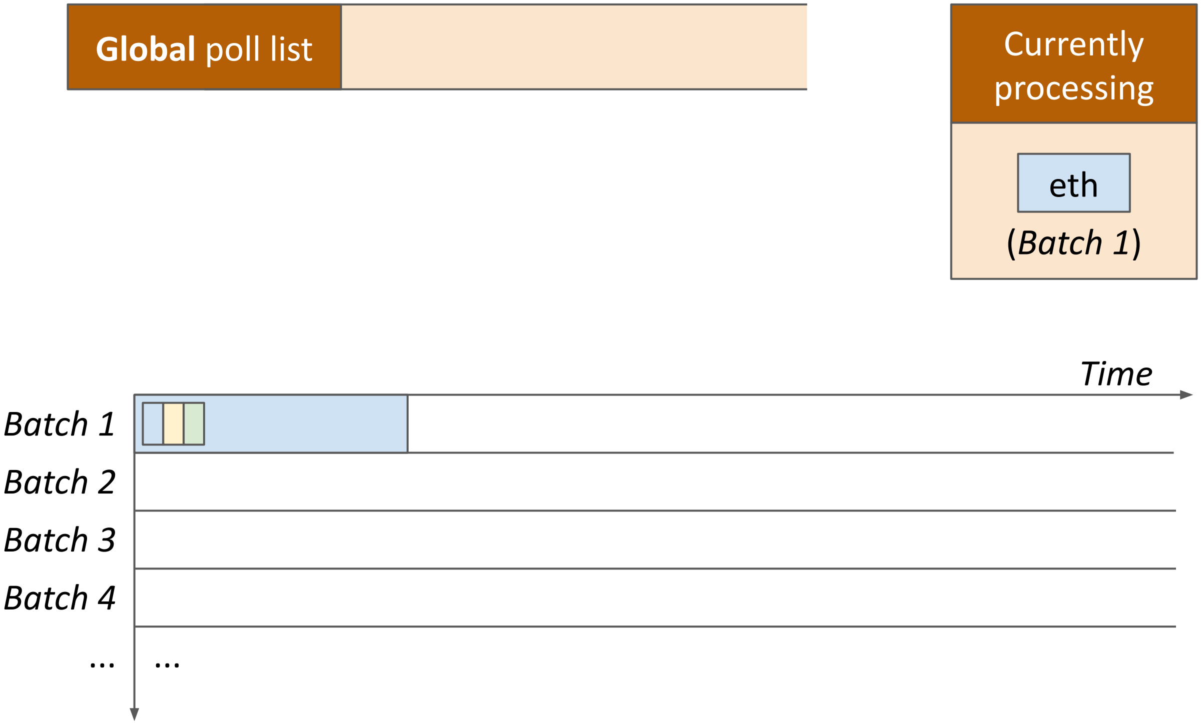
Batch 2

Batch 3

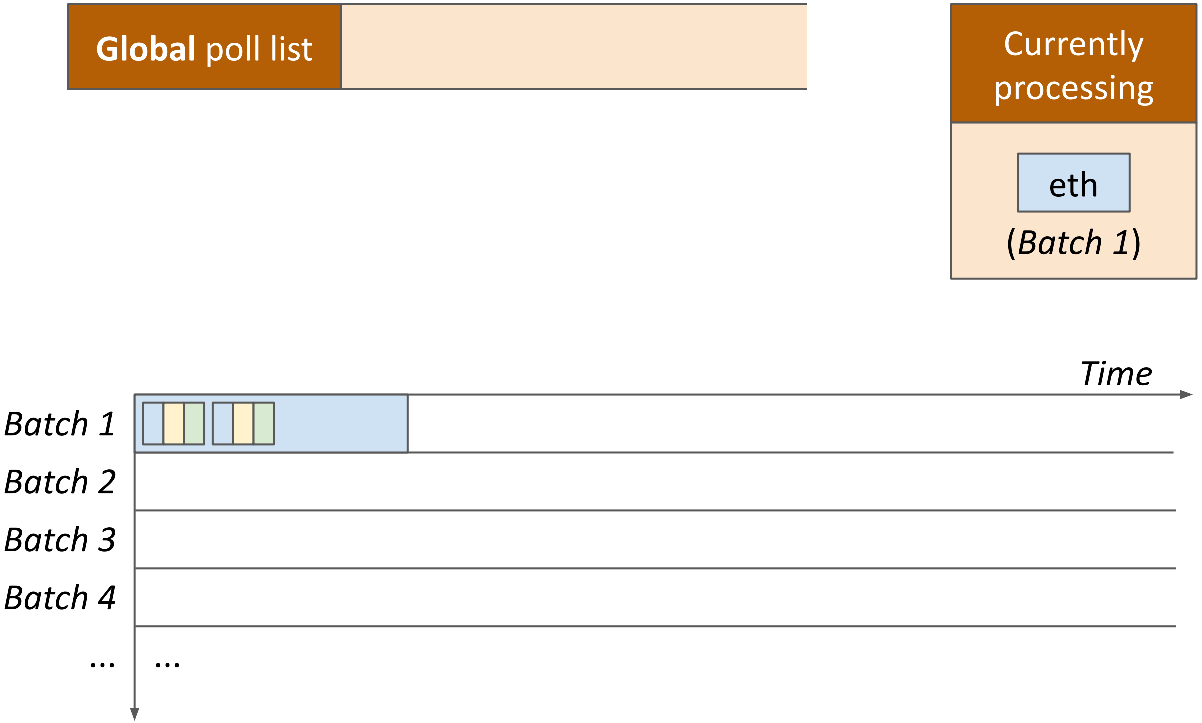
Batch 4

… …

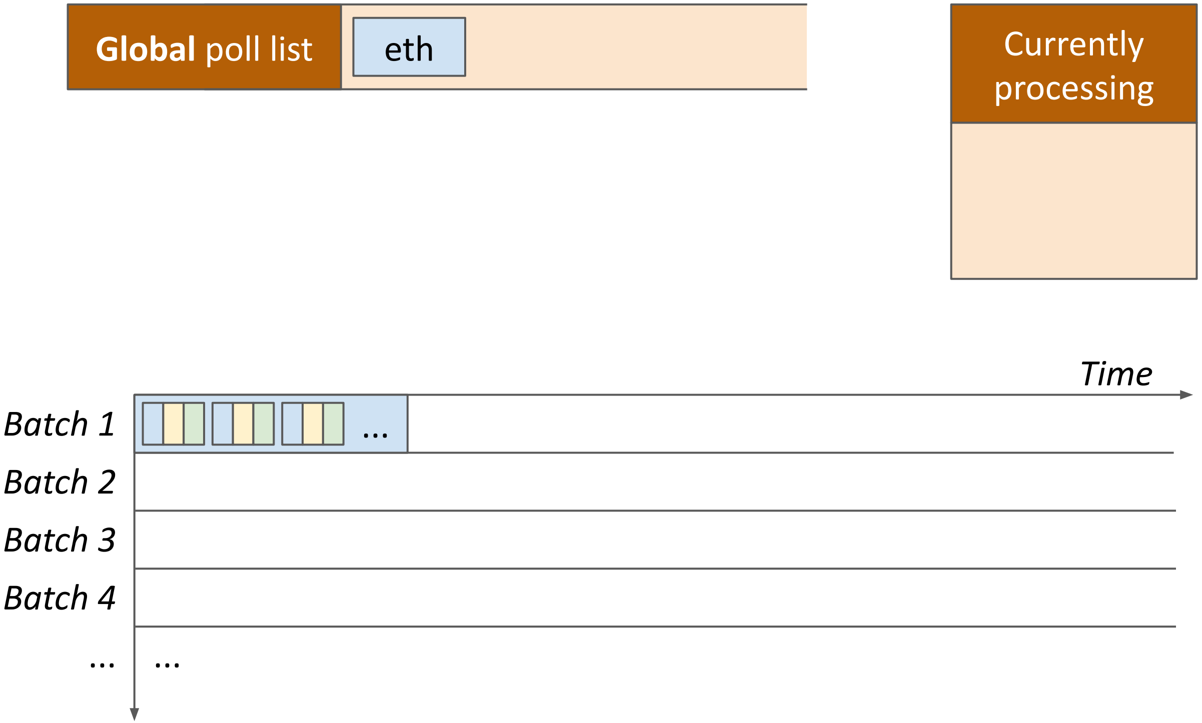
PRISM-sync


Global poll list
Currently processing
eth
(*Batch 1*)
Time
Batch 1
Batch 2
Batch 3
Batch 4
...
...

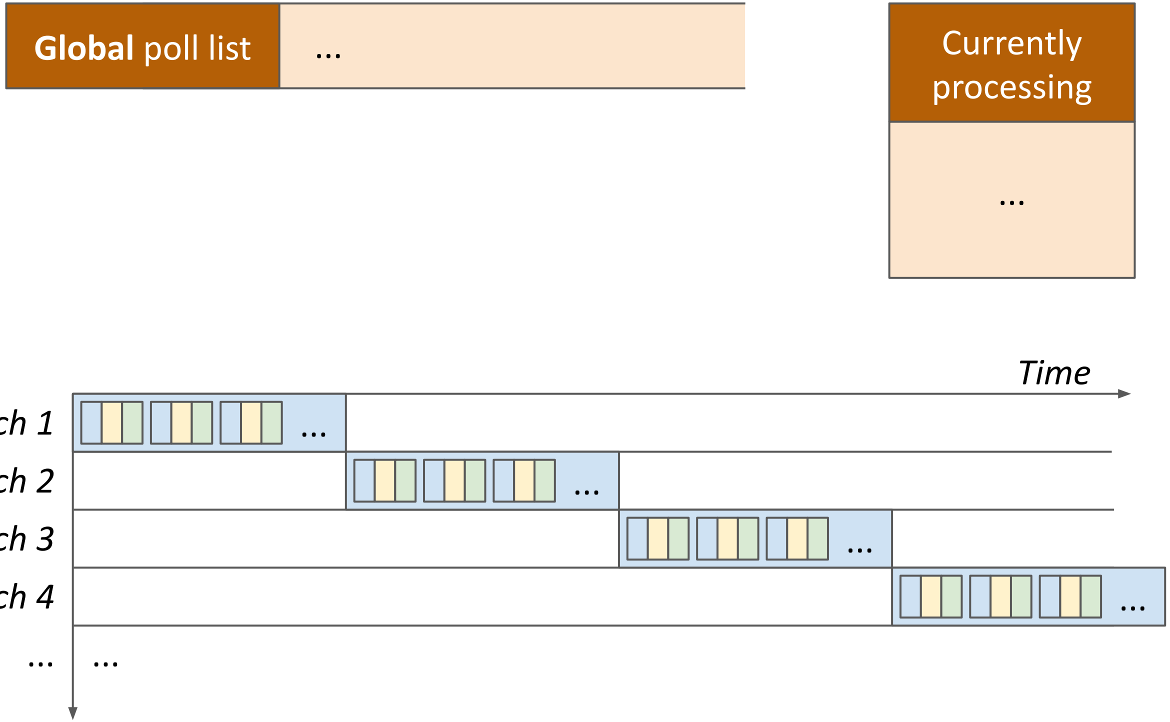
PRISM-sync


Global poll list
Currently processing
eth
(*Batch 1*)
Time
Batch 1
Batch 2
Batch 3
Batch 4
...
...

PRISM-sync


Global poll list
eth
Currently processing
Time
Batch 1
…
Batch 2
Batch 3
Batch 4
…
…

PRISM-sync



Global poll list

...

Currently processing

...

Time

Batch 1

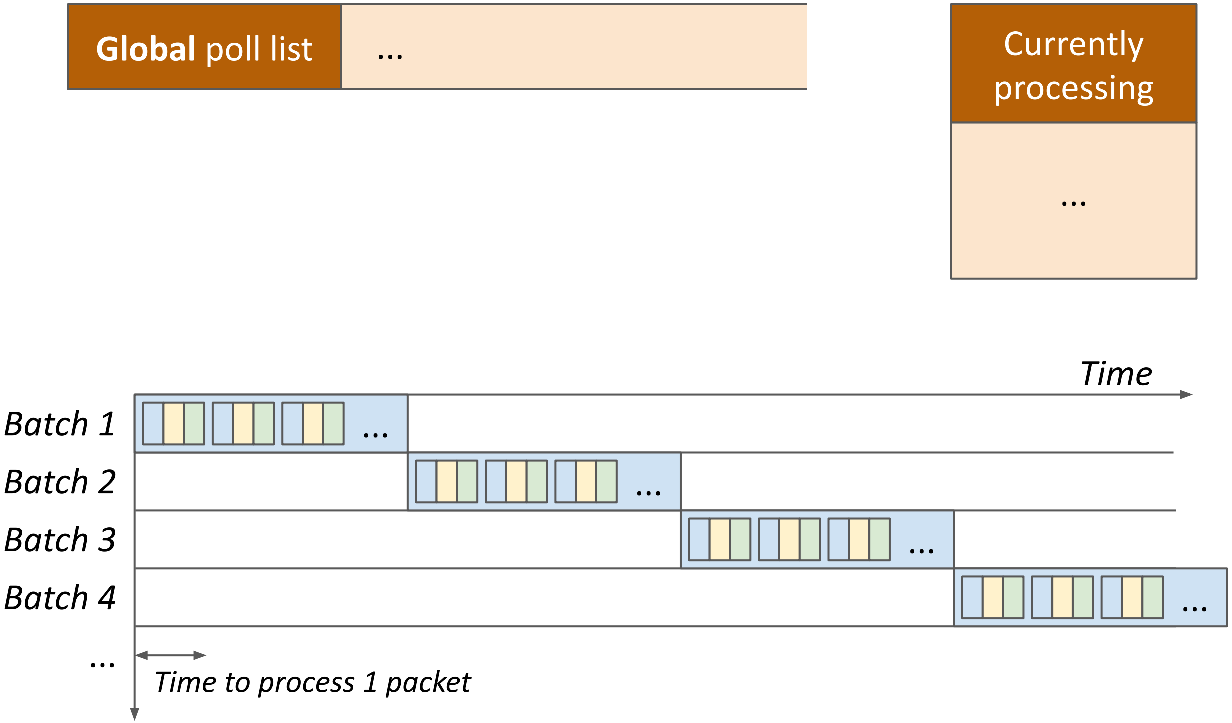
Batch 2

Batch 3

Batch 4

...

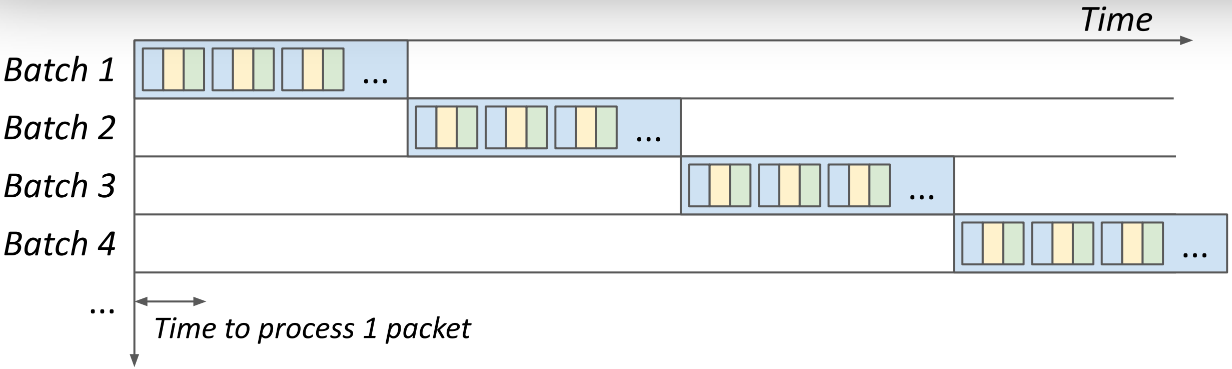
PRISM-sync


Global poll list
…
Currently processing
…
Time
Batch 1
…
Batch 2
…
Batch 3
…
Batch 4
…
…
Time to process 1 packet

PRISM-sync

PRISM-sync:

- minimizes latency by processing each packet to completion
- loses batching benefits



Implementation

- Prototype implemented on top of Linux 5.4
- ~550 lines of code in the kernel network stack
- Publicly available at: github.com/munikarmanish/prism

Implementation

- Prototype implemented on top of Linux 5.4
- ~550 lines of code in the kernel network stack
- Publicly available at: github.com/munikarmanish/prism

Some things to note:

- Priority identification is left to user
- Prioritization in the first stage (NIC)

Evaluation — Setup

Hardware:	Intel Xeon, 40 logical cores @ 2.2GHz, 128GB memory
NIC:	Mellanox ConnectX-5 EN (100GbE)
Software:	Ubuntu 18.04, with Linux kernel 5.4

Evaluation — Setup

Hardware: Intel Xeon, 40 logical cores @ 2.2GHz, 128GB memory

NIC: Mellanox ConnectX-5 EN (100GbE)

Software: Ubuntu 18.04, with Linux kernel 5.4

Comparison:

- Vanilla vs. PRISM-batch vs. PRISM-sync
- Idle vs. Busy server

Evaluation — Setup

Hardware: Intel Xeon, 40 logical cores @ 2.2GHz, 128GB memory

NIC: Mellanox ConnectX-5 EN (100GbE)

Software: Ubuntu 18.04, with Linux kernel 5.4

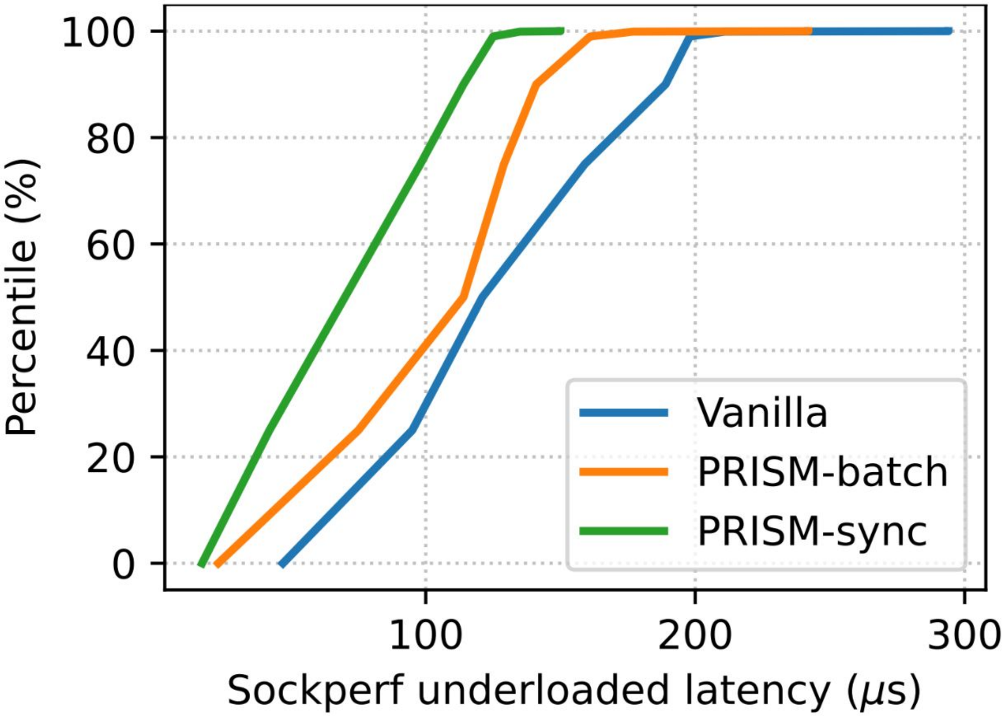
Comparison:

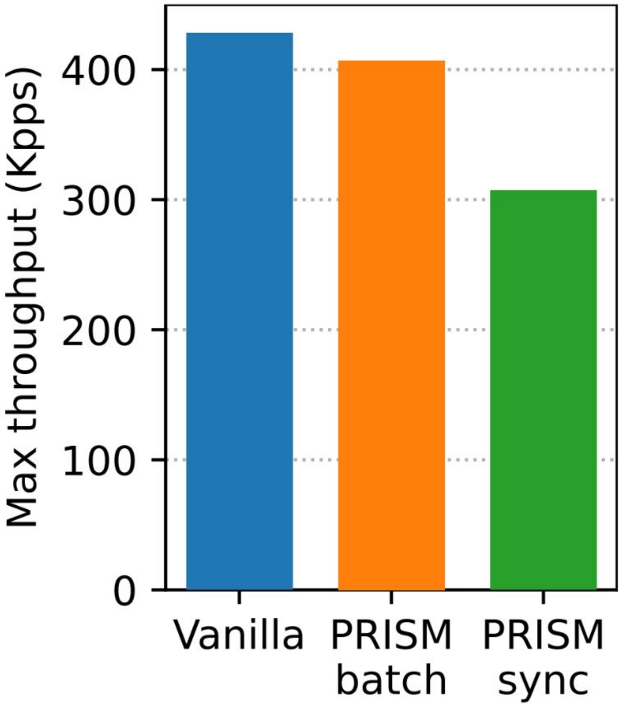
- Vanilla vs. PRISM-batch vs. PRISM-sync
- Idle vs. Busy server

Experiments:

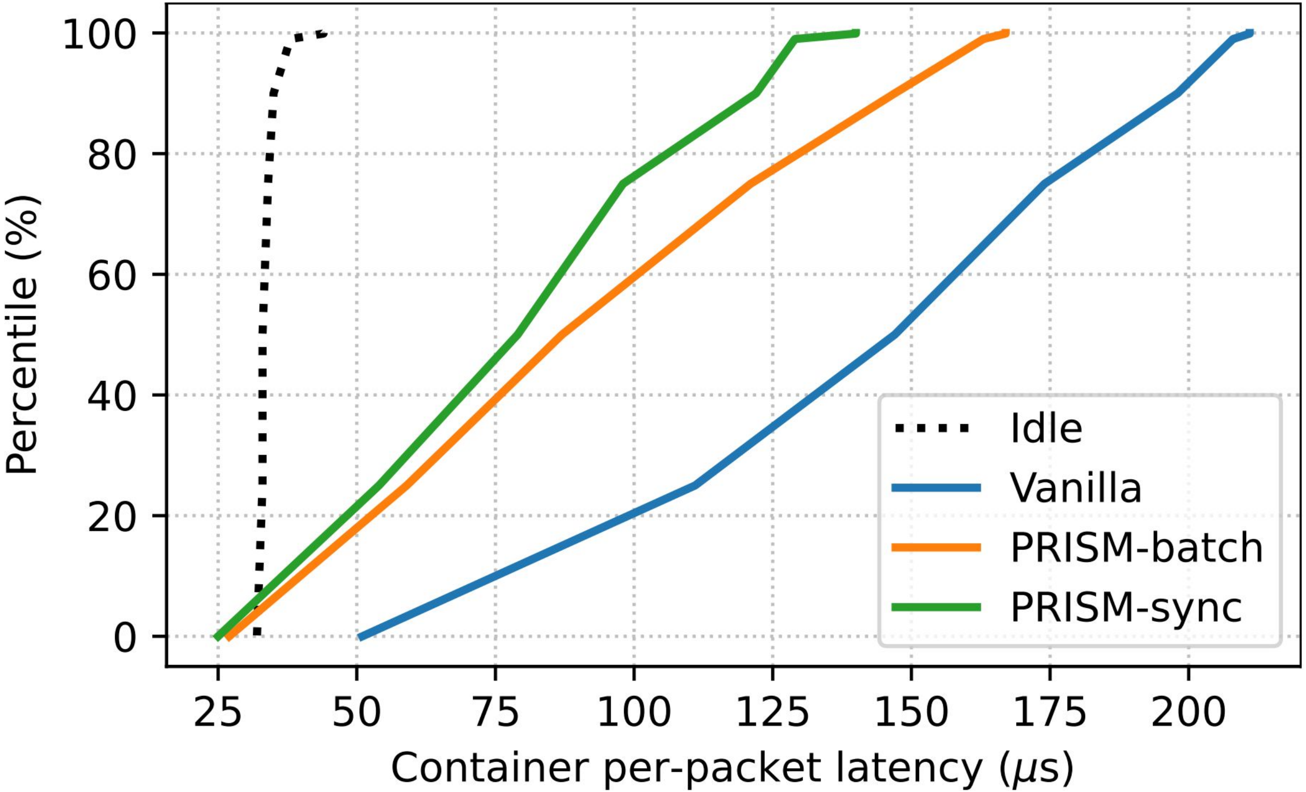
- Microbenchmarks
- Application benchmarks (Memcached, Web serving)
- *[More in the paper]*

Microbenchmark — Streamlined NAPI processing

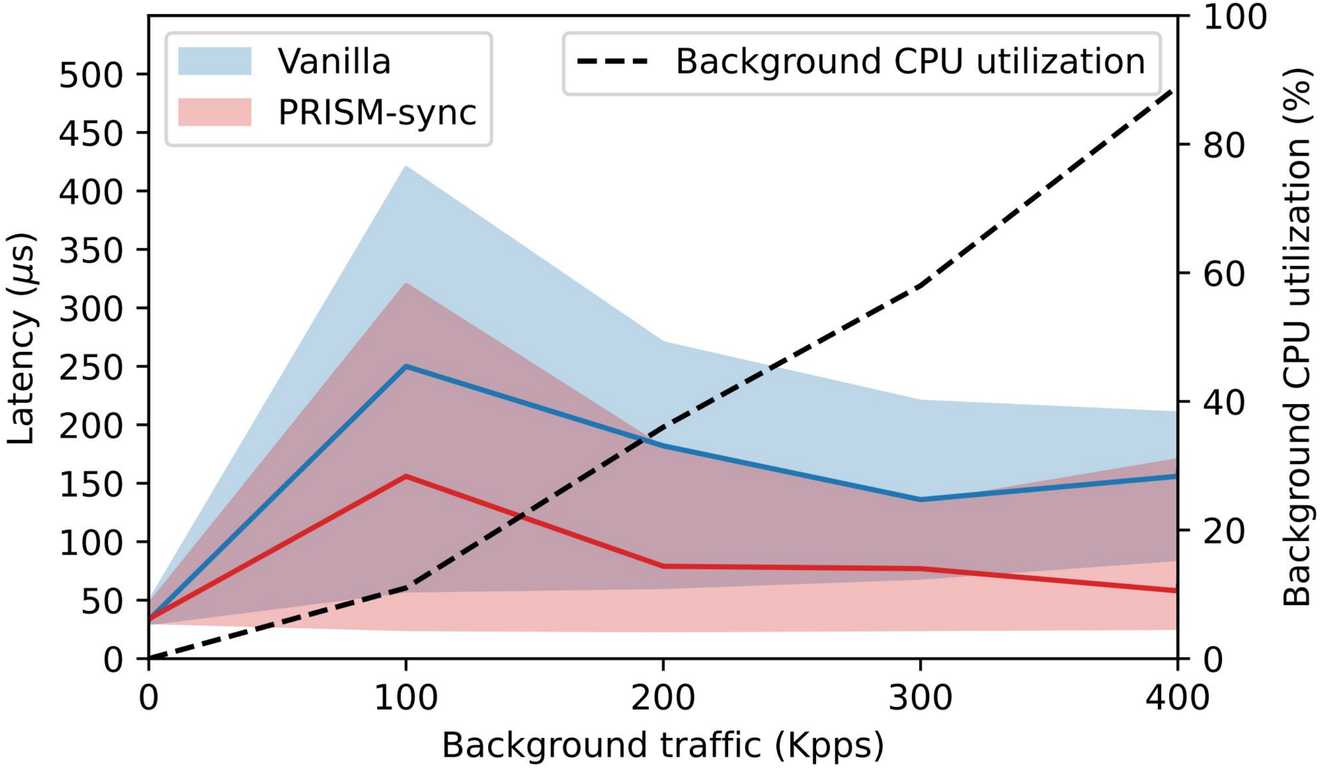




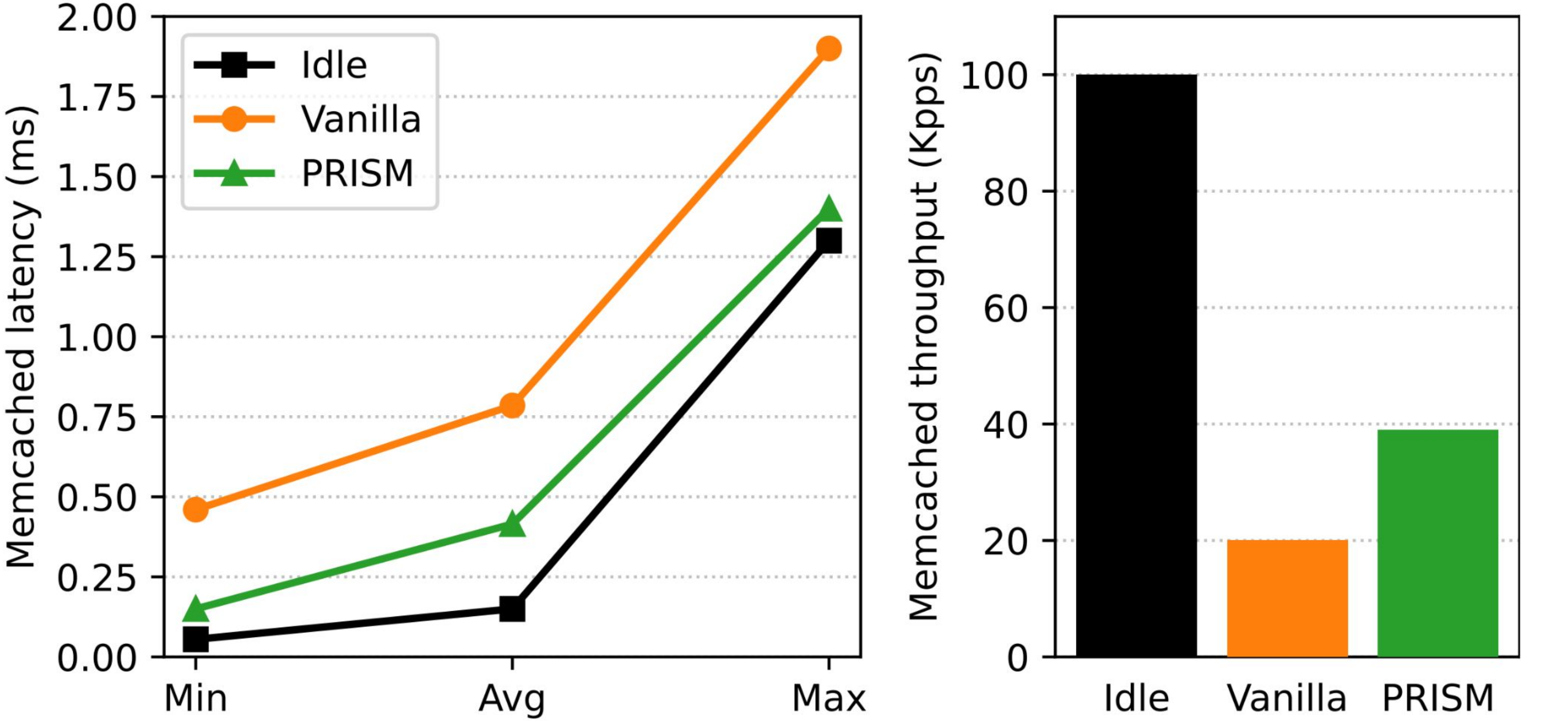
Microbenchmark — Priority differentiation



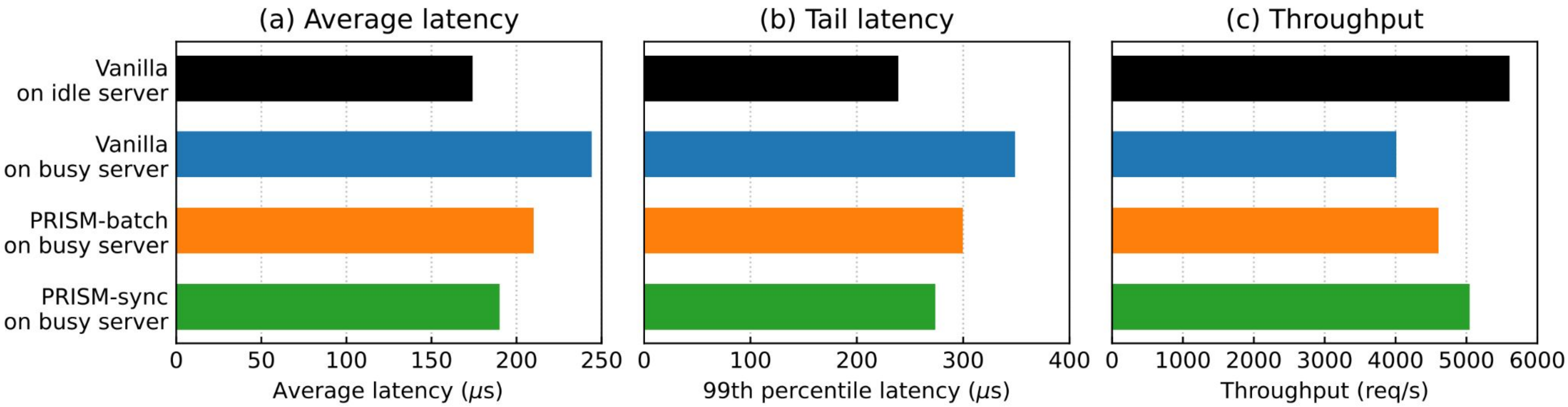
Microbenchmark — Priority differentiation



Memcached benchmark



Web serving benchmark



Conclusion

- Linux kernel has many sources of inefficiencies
  - Especially for overlay packet processing
  - Interleaved packet processing logic
  - No support for flow prioritization
- PRISM improves the performance of container flows
  - **Priority-based** packet queue management
  - **Streamlined** multi-stage processing pipeline
  - **Batch-level preemption** or synchronous processing
- Implementation: github.com/munikarmanish/prism

Conclusion

- Linux kernel has many sources of inefficiencies
  - Especially for overlay packet processing
  - Interleaved packet processing logic
  - No support for flow prioritization
- PRISM improves the performance of container flows
  - **Priority-based** packet queue management
  - **Streamlined** multi-stage processing pipeline
  - **Batch-level preemption** or synchronous processing
- Implementation: github.com/munikarmanish/prism

For more info, please contact me at manish.munikar@uta.edu.

Backup slides

Background: NAPI

- Interrupt-based packet processing
  - IRQ raised for each packet received by the NIC
  - Good for light load, bad for heavy load
- NAPI
  - Many packets are processed (in batches) after a single interrupt
  - Interrupt-mode for light load
  - Poll-mode for heavy load
- RSS / RPS / RFS
  - Distributes independent flows to separate cores for processing
- Still naive FIFO queueing!
  - High-priority packets can get stuck behind long queues of low-priority packets

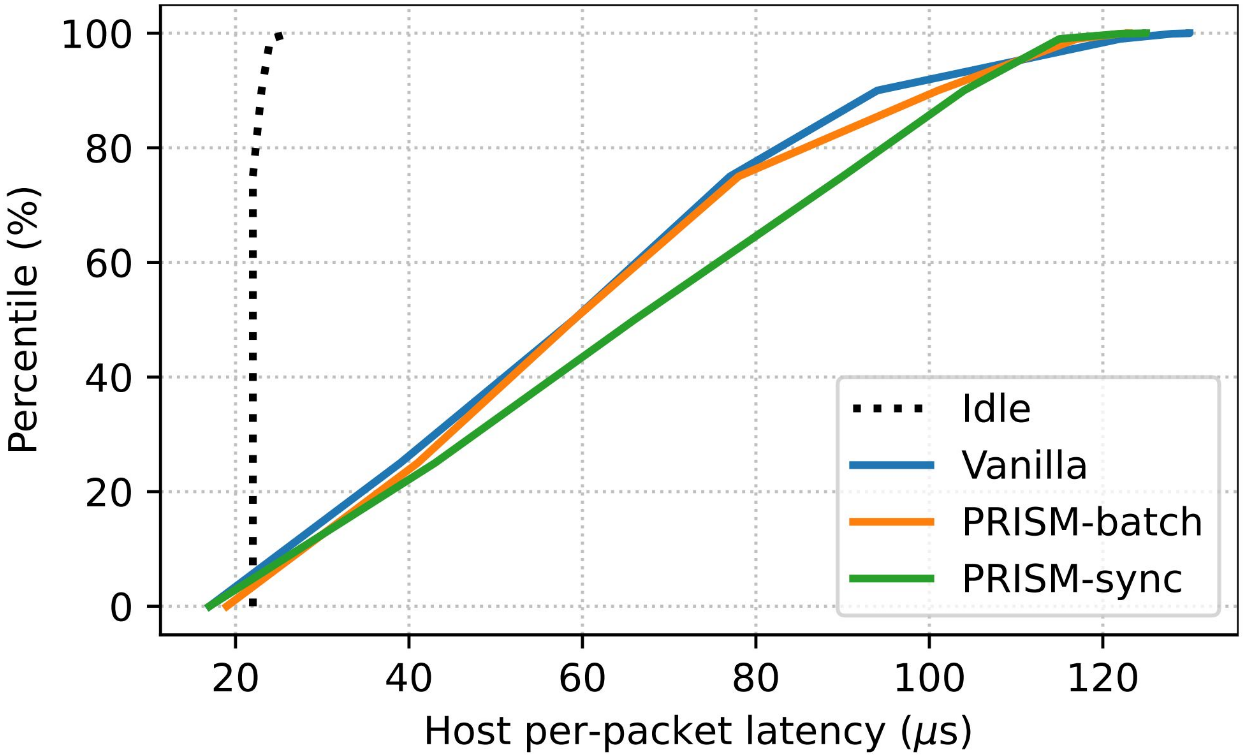
Remaining issues

- Flow prioritization in the NIC queue
- Priority synchronization between user application and network stack
- Multiple priority levels

Why this problem?

- NICs are becoming faster quicker
- CPUs cannot keep up
- OS packet processing logic is becoming performance bottleneck
- Moreover, containers are everywhere
- Containers use container overlay networks
- Container network incurs high overheads
- In a highly utilized system, the high priority messages experience large queueing delays

Microbenchmark — Priority differentiation (host network)


Percentile (%)
Host per-packet latency (μs)
Idle
Vanilla
PRISM-batch
PRISM-sync