6.5810: PicNIC

Adam Belay <abelay@mit.edu>

Logistics

- Next week: Draft project report is due
- Describes the motivation and design of your system
- Can be short, target 2-4 pages
- Implementation and evaluation will be in the final report

Motivation for PicNIC

- So far, the focus has been on raw net performance and efficiency
- But clouds have multiple tenants that share the network
- PicNIC focuses on providing performance isolation
- Unusual challenge: CPU is processing network packets for VMs
  - Authors observe contention inside the "host stack"
  - Usually, the host stack runs on a limited set of cores

Example

- One VM is experiencing a denial-of-service attack
- Host stack becomes congested, spends all cycles processing DOS attack packets
- All other VMs on the same machine experience high latency
- This is a breakage of performance isolation

Network virtualization?

- Key idea: Give cloud users the abstraction of their own private network on top of a physical network
- Host stack translates virtual IP addresses to real IP addresses via encapsulation

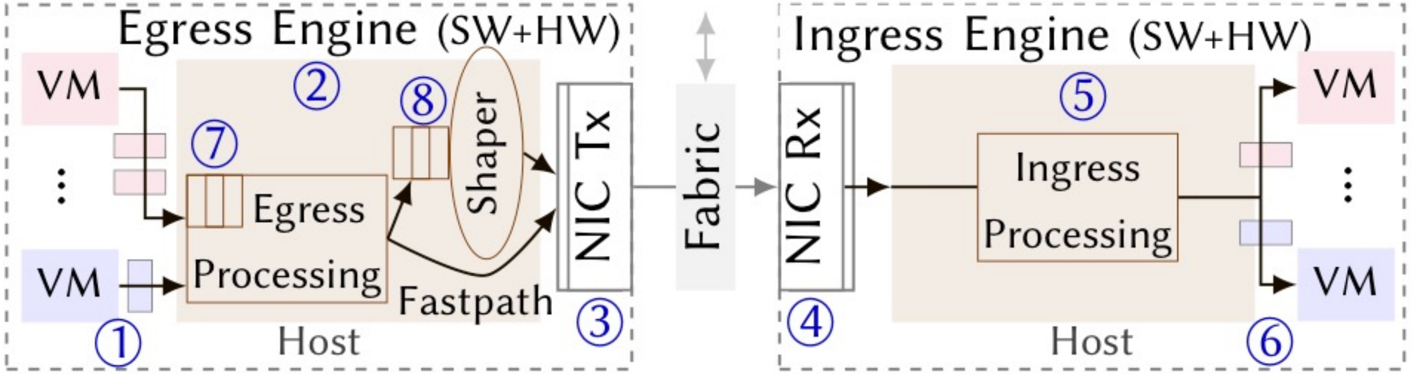


Figure 1: Overview of an on-host network virtualization stack.

Example incident (egress)

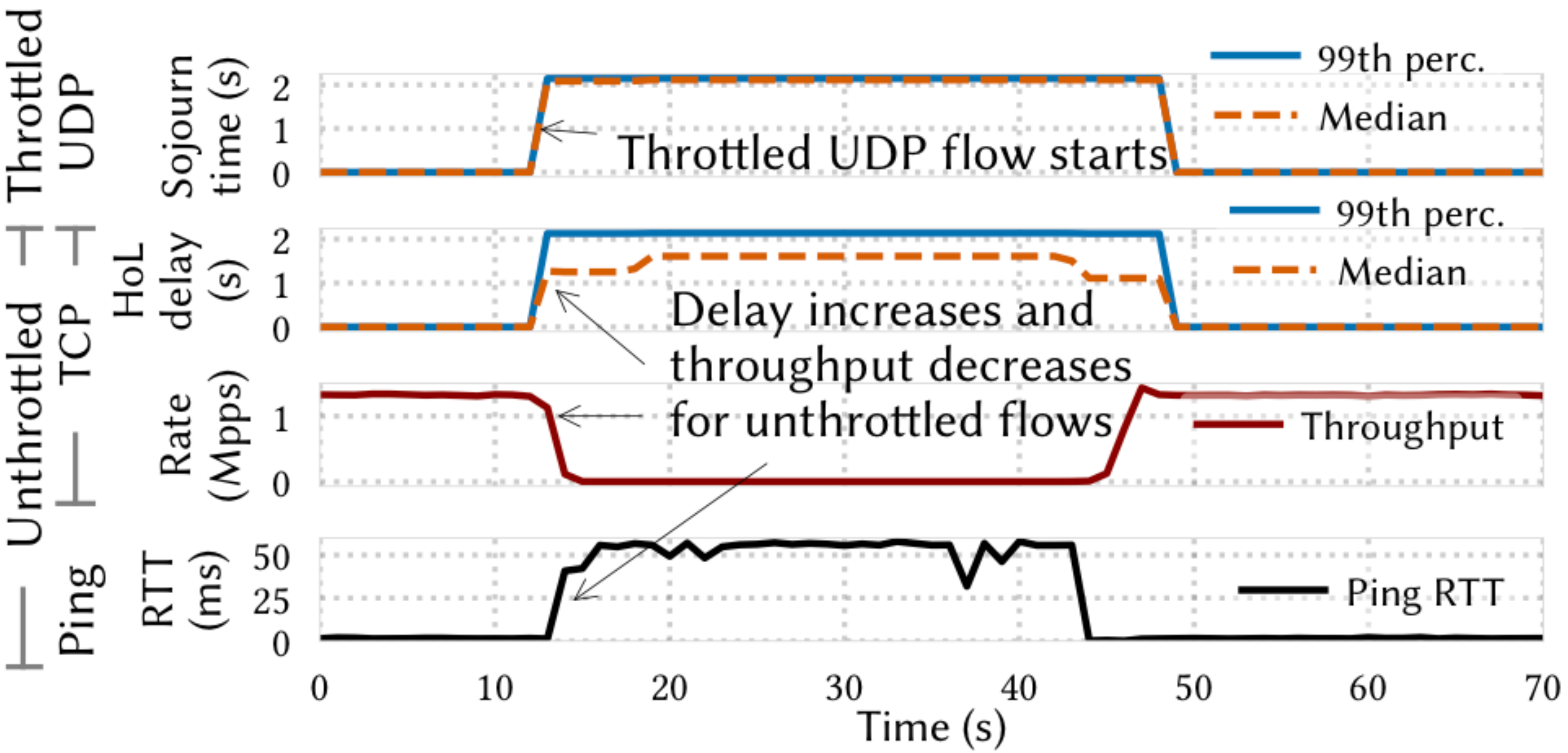


Figure 2: HoL blocking and isolation breakage at egress.

PicNIC’s design principles

1. Host stack resources should be proportional to each VM’s SLO

2. Under overload, apply backpressure, otherwise drop early

Defining predictability

Metric	Predictable vNIC SLO
Bandwidth	*min.* and *max.* envelope (hose model)
Delay	Low; predictable distribution*
Loss rate	No drops for cooperating traffic*

*for well-behaved traffic in bandwidth envelope

Table 2: Abstraction: Predictable vNIC SLO metrics

Design overview

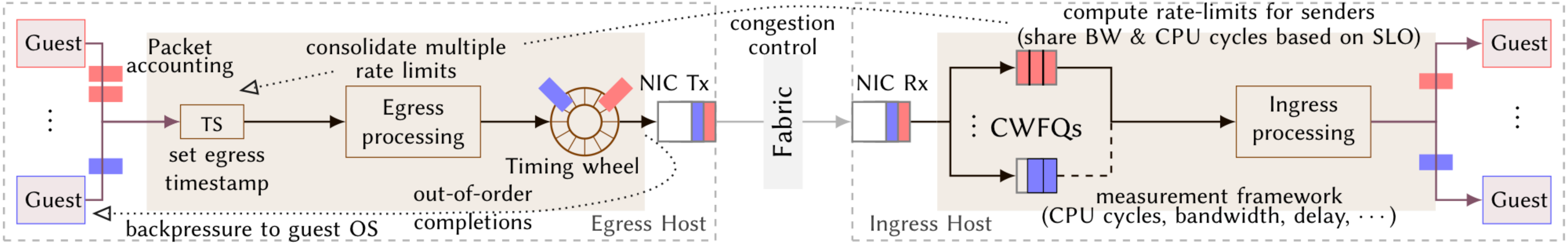


Figure 5: PicNIC architecture. Local constructs (ingress CWFQs and egress sender-side admission control) working in coherence with end-to-end receiver-driven congestion control to achieve the predictable vNIC abstraction.

Idea: Ingress CPU-Fair WFQs (CWFQs)

- Pull ingress (RX) packets from NIC as fast as possible
- Place packets in per-VM queues
- Calculate per-packet processing overhead in each queue
- Allocate CPU resources in a weighted but fair way across VMs

Idea: Congestion control

- Combination of rate limiters at sender
  - Set bandwidth per second limit to control bandwidth use
  - Set packets per second limit to minimize CPU use and delay
- Congestion control is left off for low load VMs
- High load VMs experience delays in their CFWFQs
- Use this delay as a signal to control transmit rate at sender

Idea: Egress admission control

- Traffic shapping: Controls the rate of packet transmissions
- PicNIC uses a timing wheel; a data structure that stores packets by their transmission timestamp
- Problem: VM guests can spam egress packets
- Solution #1: PicNIC places a limit on buffer use to avoid too many buffers waiting in the timing wheel
- Solution #2: PicNIC applies backpressure to VMs so they can voluntarily stop sending

Backpressure inside the guest OS

- Note: Guest OS is willing participant in this scheme; many modifications were made to Linux
- #1: PicNIC sends out-of-order completions when each packet leaves the wheel
- #2: TCP small queues (TSQs) limits the number of packets each flow can have in flight

Q: Why can PicNIC not use in-order completions?

- Note: In order completions are the normal approach used by NICs

Q: What happens if the buffer limit is hit?

Egress admission control design

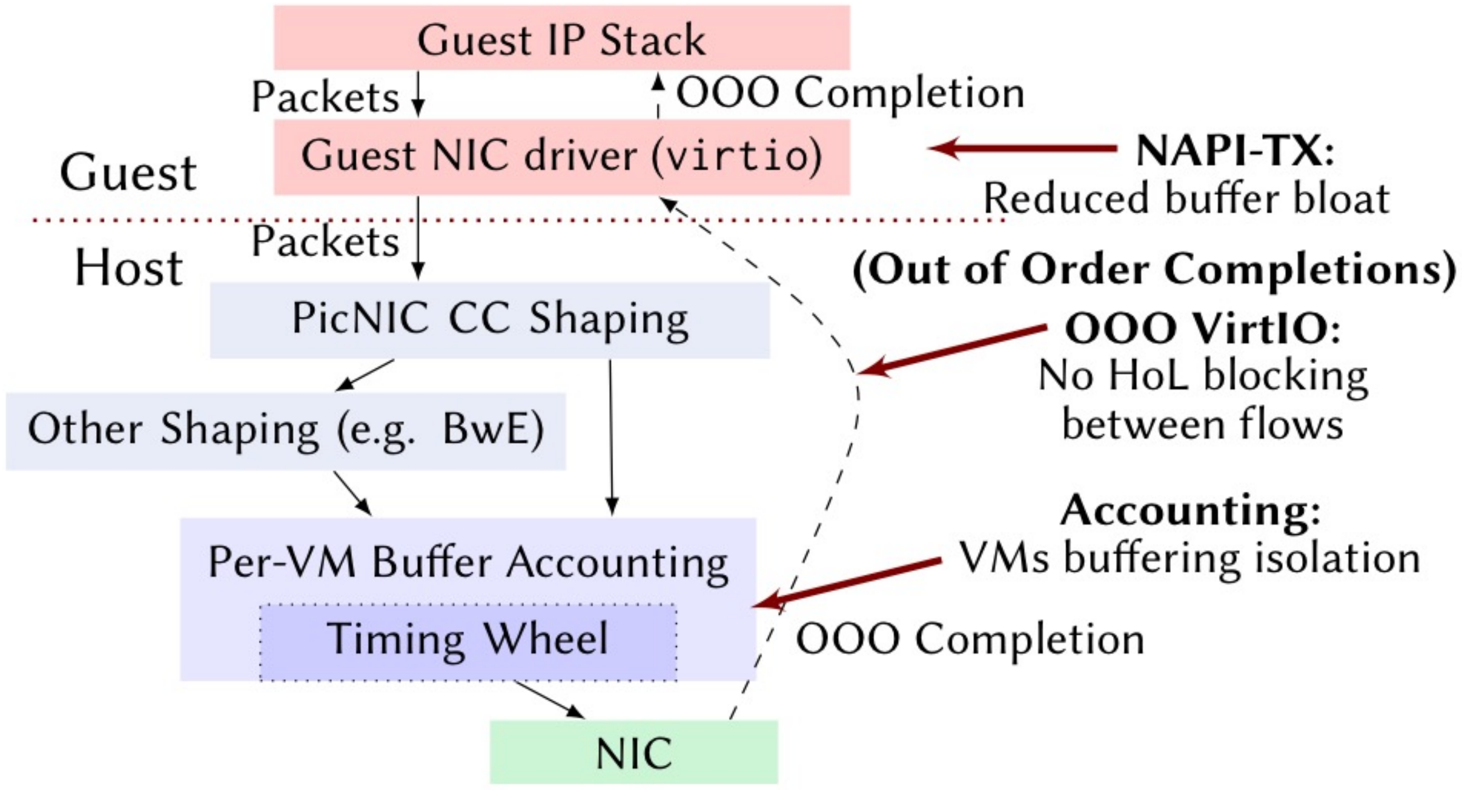


Figure 6: PicNIC's sender-side admission control.

Back to the UDP flow overload problem

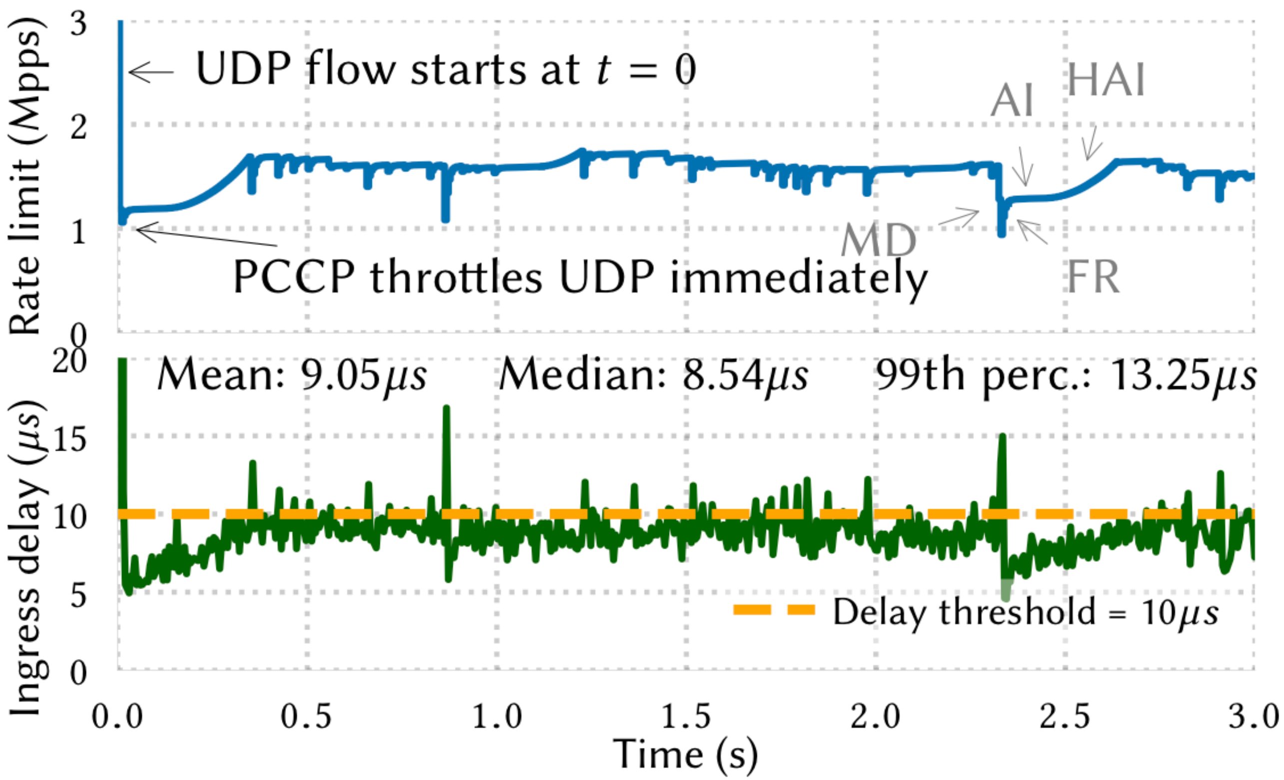
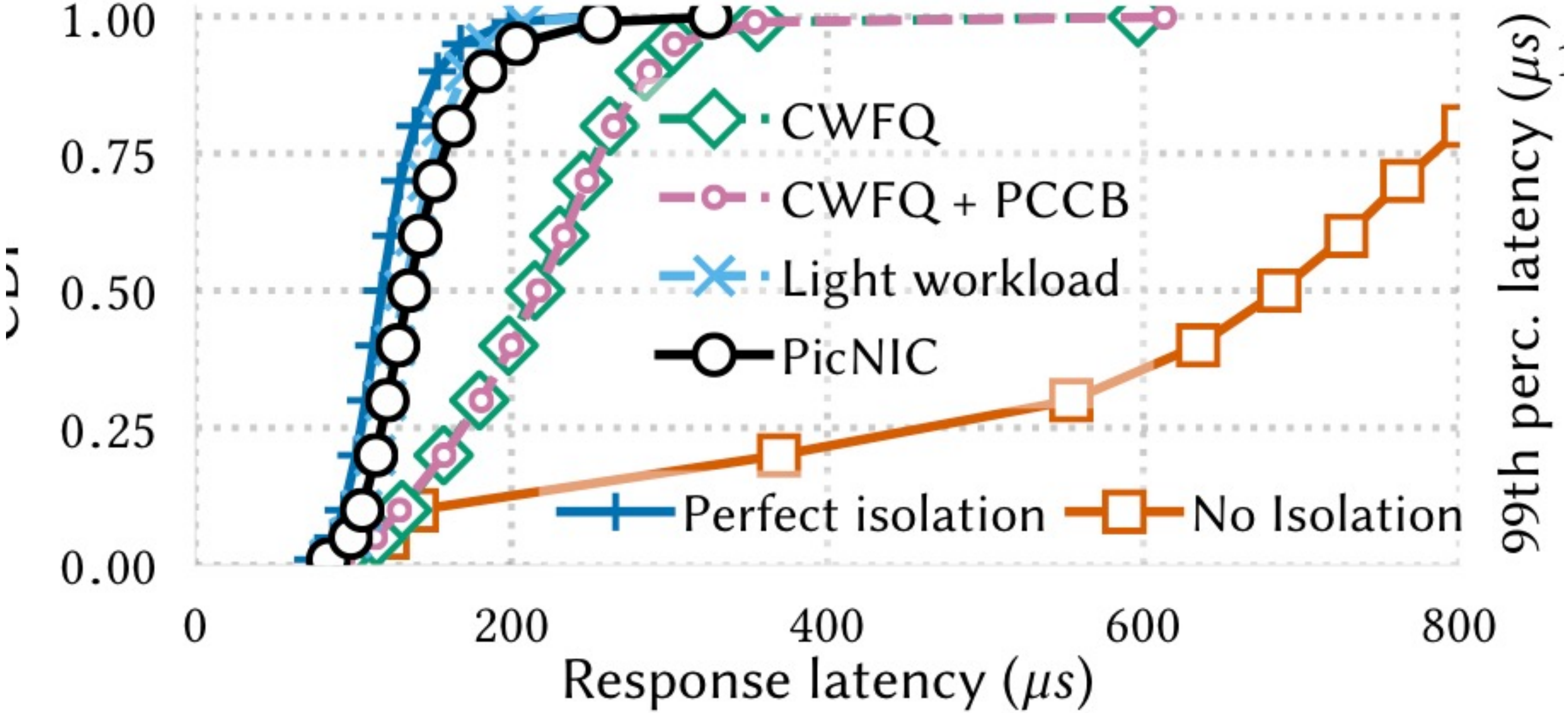


Figure 10: Predictable low latency with PicNIC.

Contribution of PicNIC's mechanisms



Discussion

- Could we put PicNIC in hardware? What parts?
- How does PicNIC compare to Ethernet PFC?
- Do RDMA NICs have the same problem of unpredictable packet processing work? Do packets consume resources? Does 1RMA?
- Are there any downsides to out-of-order completions or TCP small queues?

Deeper discussion

- Should VMs participate in congestion control at all?
- Why or why not?