Advanced Computer Networks

Network Virtualization in Data Center Networks (II)

https://pages.cs.wisc.edu/~mgliu/CS740/F25/index.html

Ming Liu
mgliu@cs.wisc.edu

Outline

- Last lecture
  - Network virtualization in data center networks (I)
- Today
  - Network virtualization in data center networks (II)
- Announcements
  - Lab2 due 11/05/2025 11:59 PM
  - Midterm report due 11/04/2025 11:59 PM

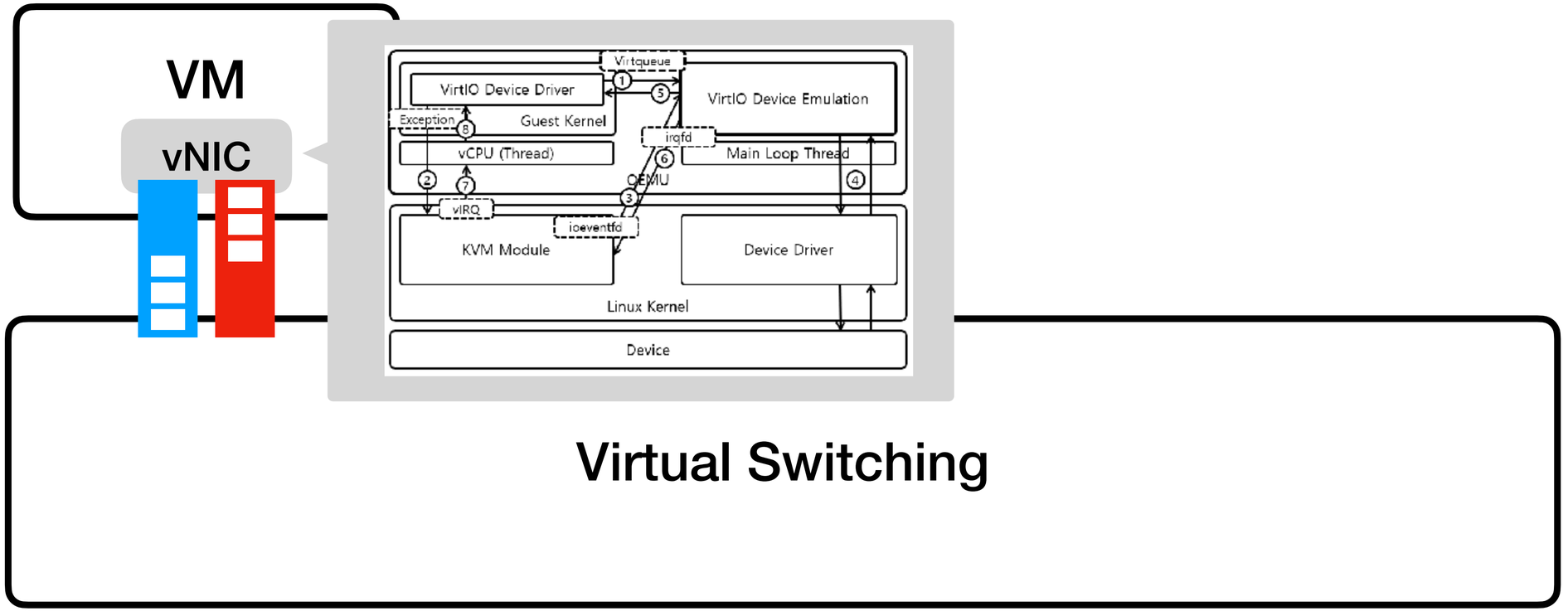
Network Virtualization @Endhost

VM

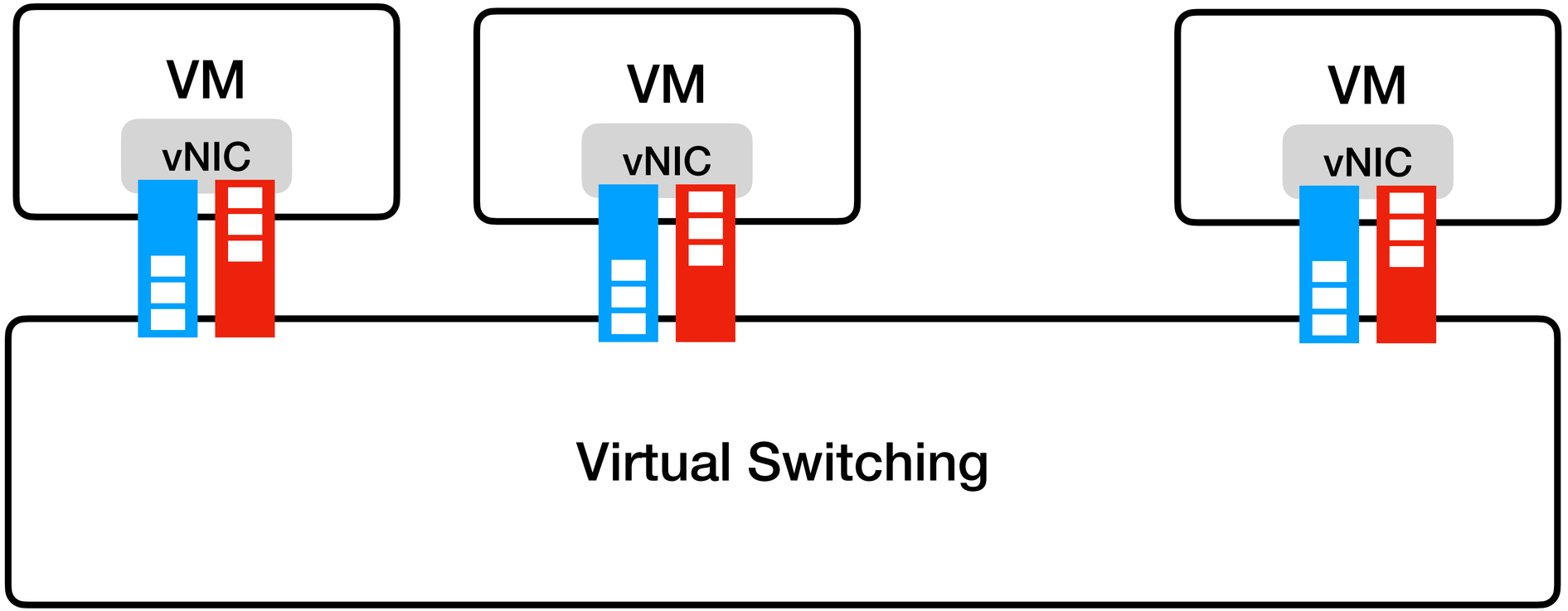
vNIC

Virtual Switching

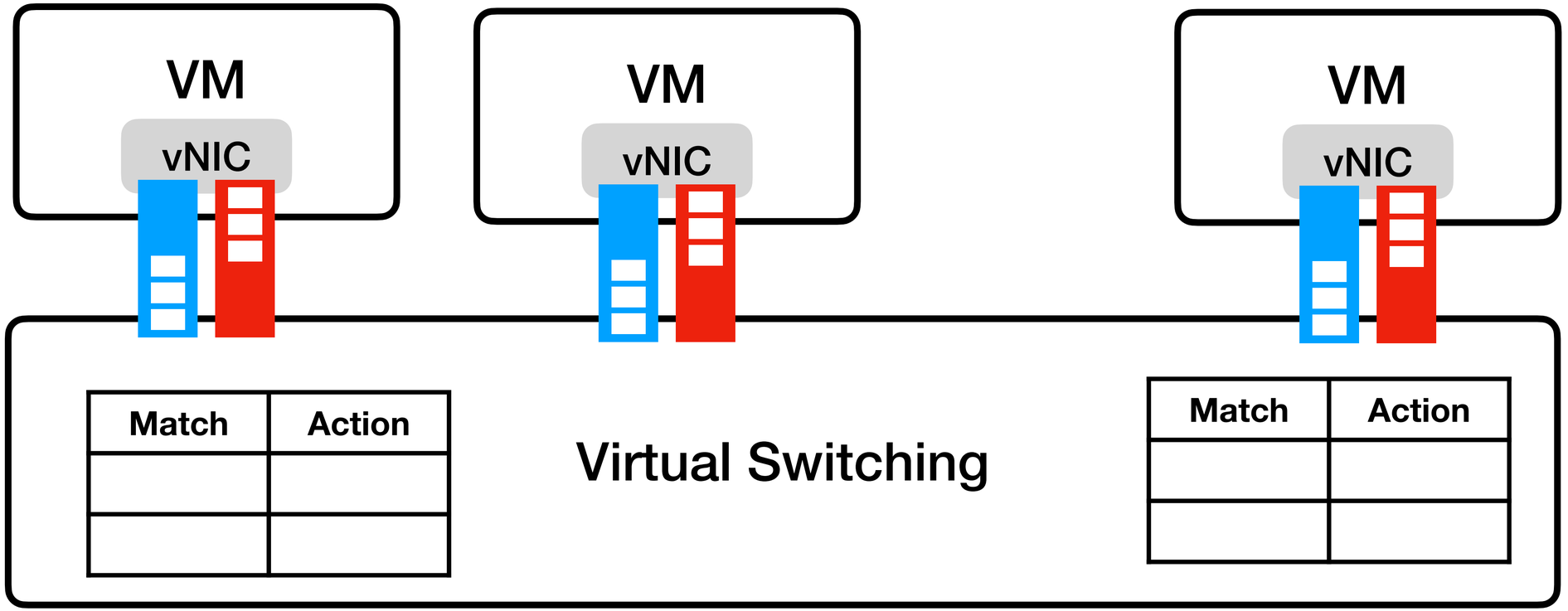
Network Virtualization @Endhost



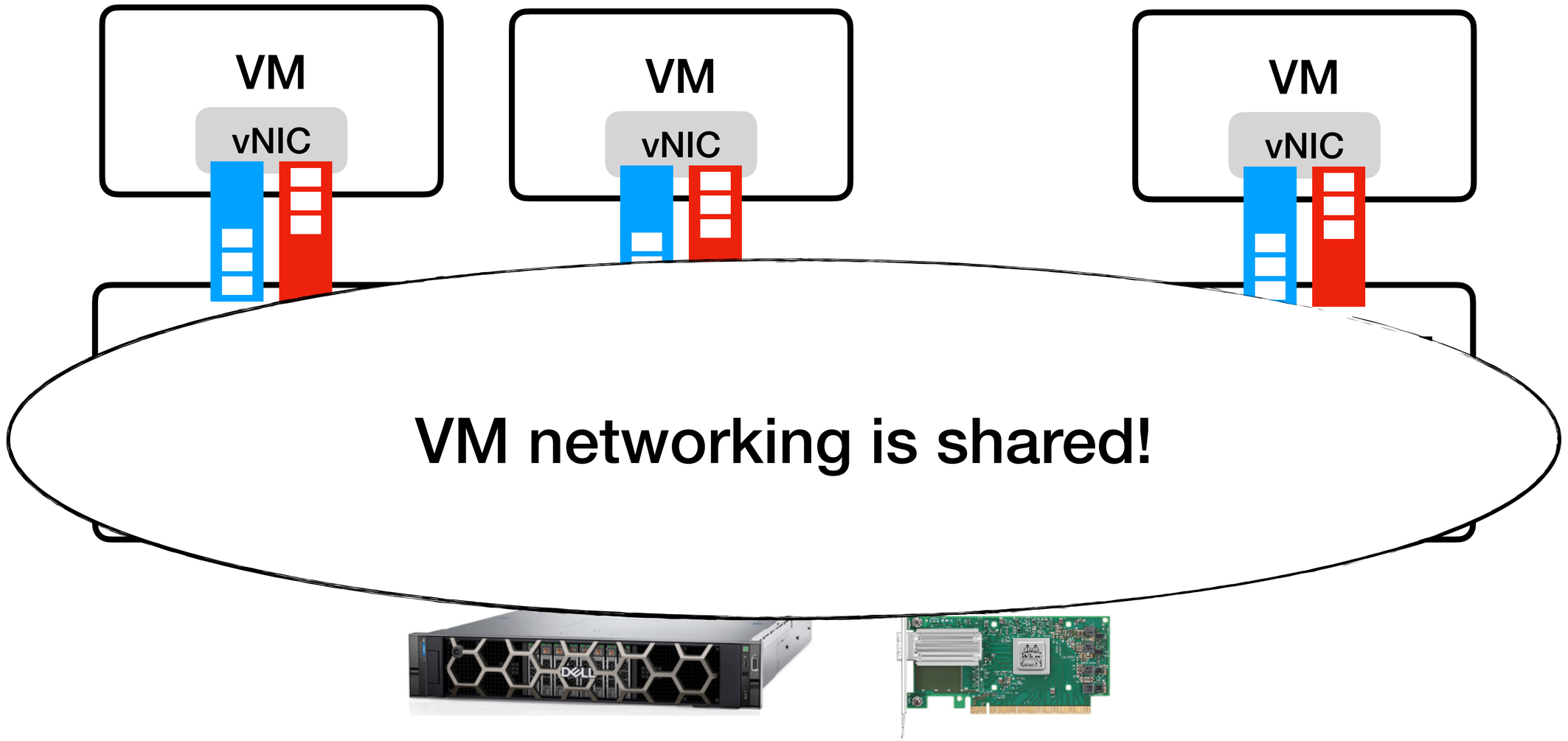
Network Virtualization @Endhost



Network Virtualization @Endhost



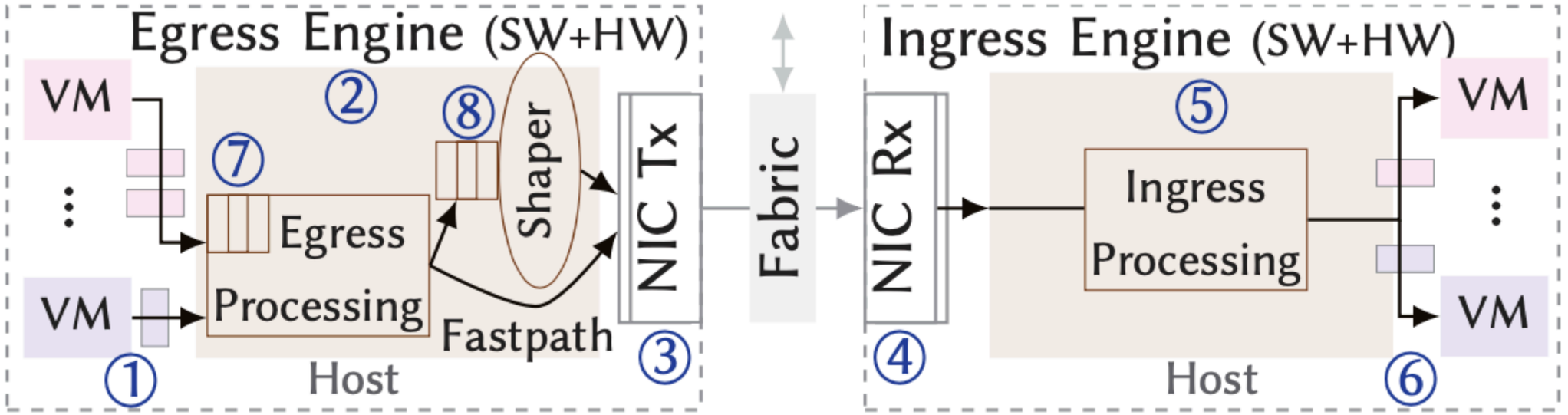
Network Virtualization @Endhost



How can we achieve network performance isolation at the endhost?

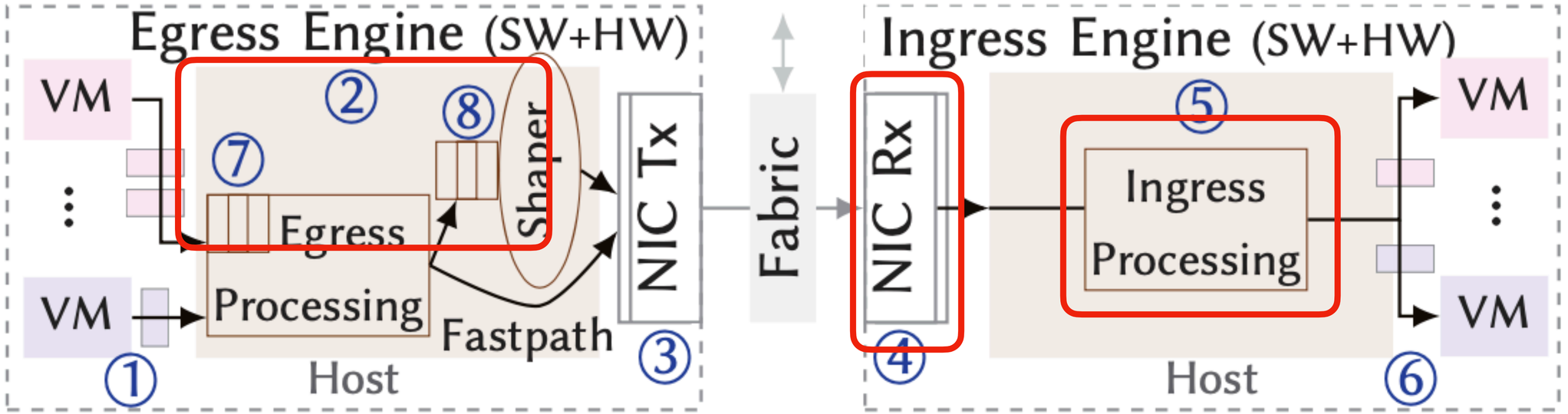
Packet Path Between Two VMs

- Per-packet processing costs are not const.



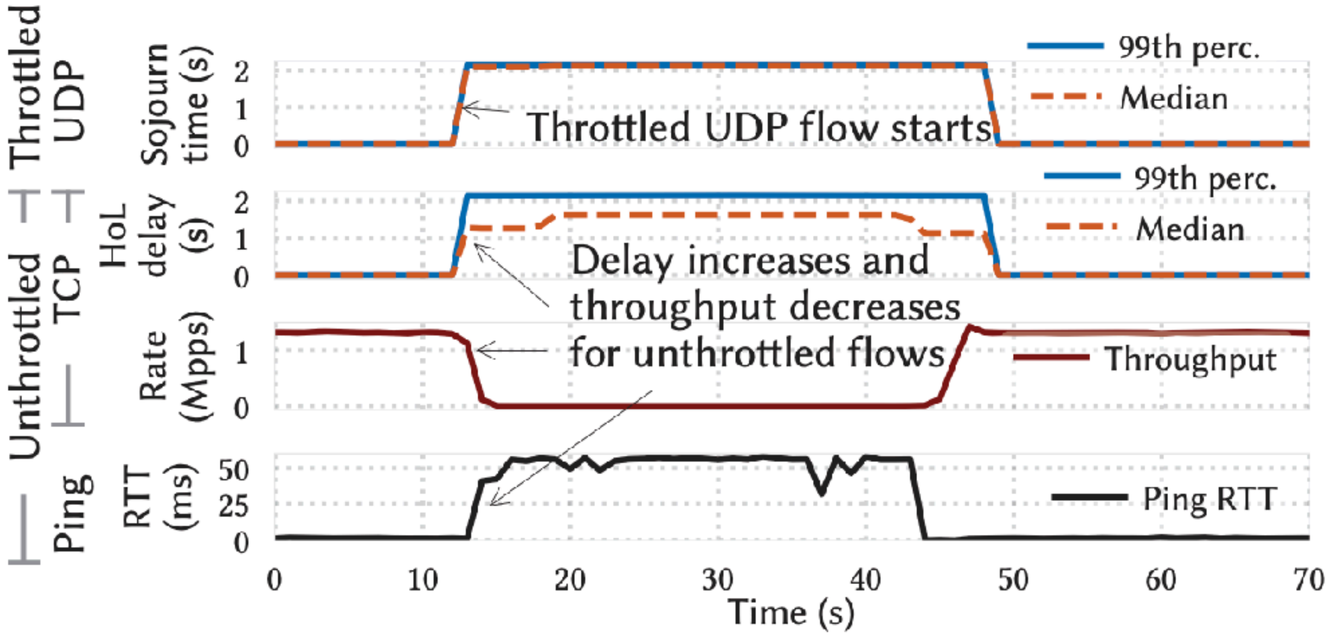
Packet Path Between Two VMs

- Per-packet processing costs are not const.



Issue #1: Egress Buffer Contention

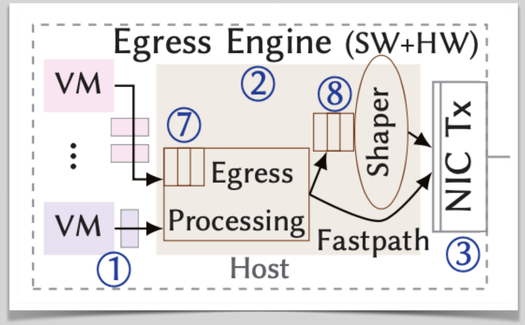
- Experiment setup:
  - VM1 —> VM2: unthrottled TCP flow
  - VM1 —> VM3: throttled UDP flow @ 10Mbps

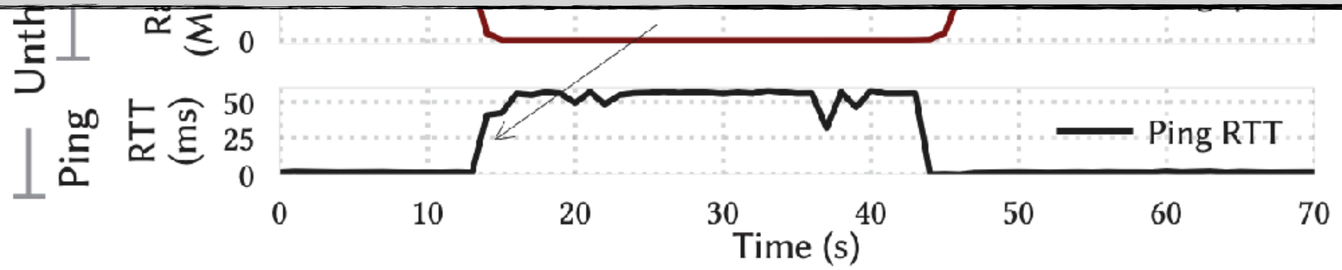


Issue #1: Egress Buffer Contention

- Experiment setup:
  - VM1 —> VM2: unthrottled TCP flow
  - VM1 —> VM3: throttled UDP flow @ 10Mbps

HoL blocking happens when egress buffers are full.

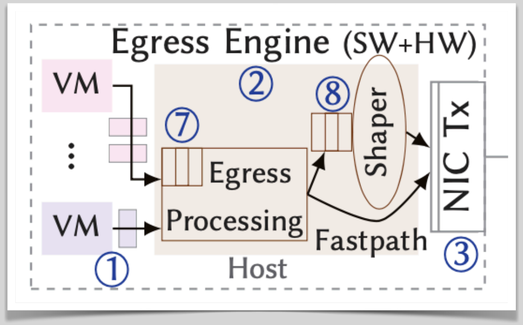


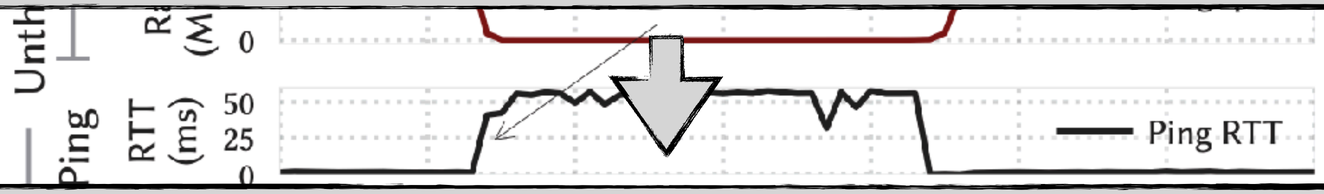


Issue #1: Egress Buffer Contention

- Experiment setup:
  - VM1 —> VM2: unthrottled TCP flow
  - VM1 —> VM3: throttled UDP flow @ 10Mbps

HoL blocking happens when egress buffers are full.

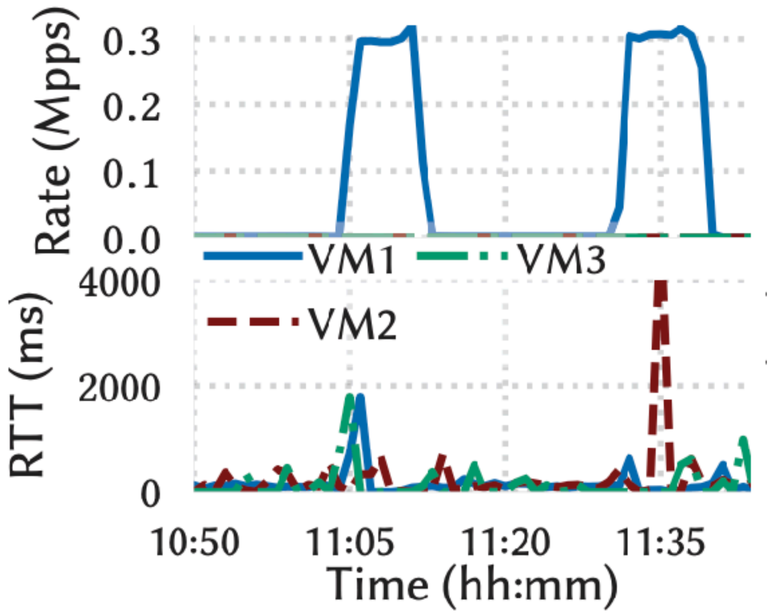




Implication: rethinking egress buffer sharing and admission control @TX-Host

Issue #2: Ingress NIC Contention

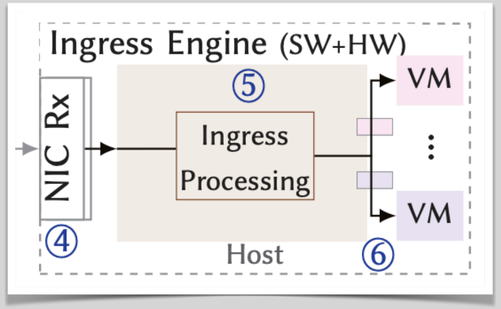
- Experiment setup:
  - Three co-located: VM1, VM2, VM3
  - VM1 receives a packet burst from small packets

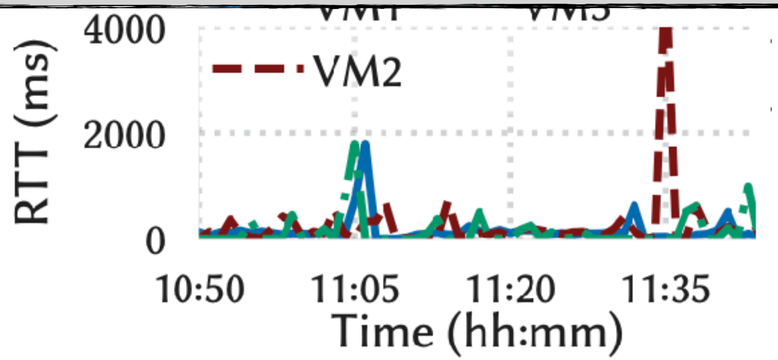


Issue #2: Ingress NIC Contention

- Experiment setup:
  - Three co-located: VM1, VM2, VM3
  - VM1 receives a packet burst from small packets

A storm of small packets can exceed the NIC packet processing capacity.

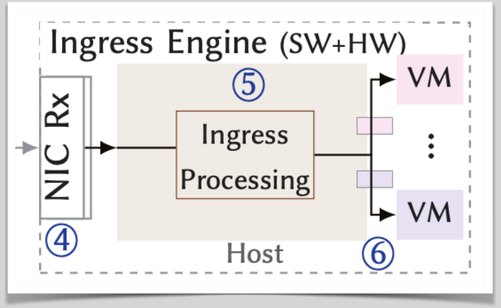


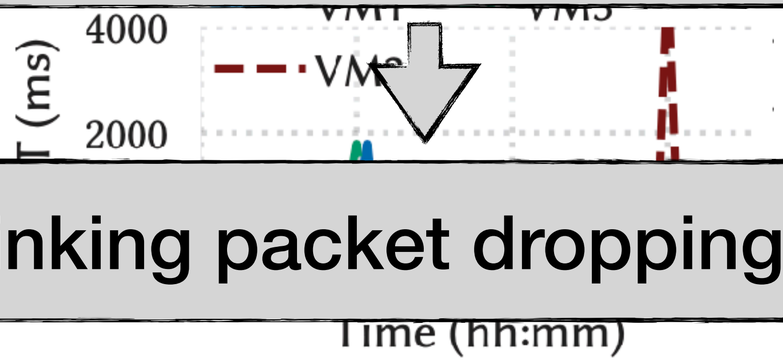


Issue #2: Ingress NIC Contention

- Experiment setup:
  - Three co-located: VM1, VM2, VM3
  - VM1 receives a packet burst from small packets

A storm of small packets can exceed the NIC packet processing capacity.

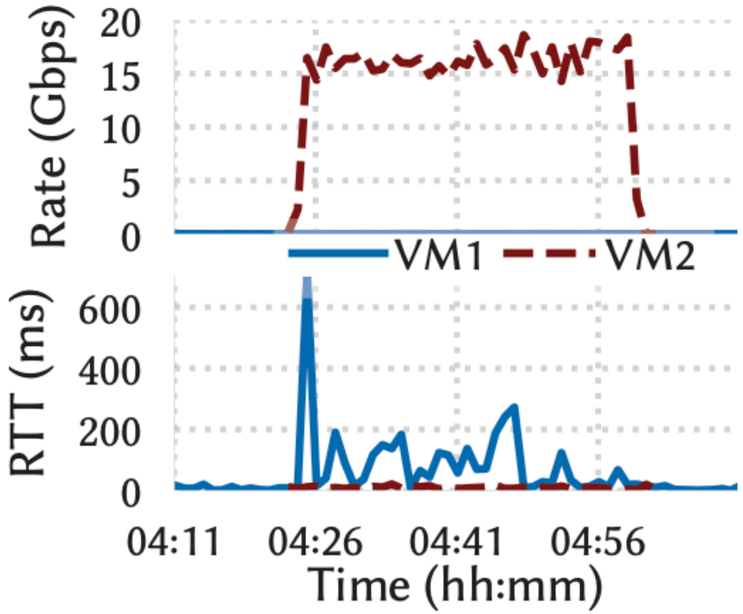




Implication: rethinking packet dropping policy @ RX-NIC

Issue #3: Ingress Engine Contention

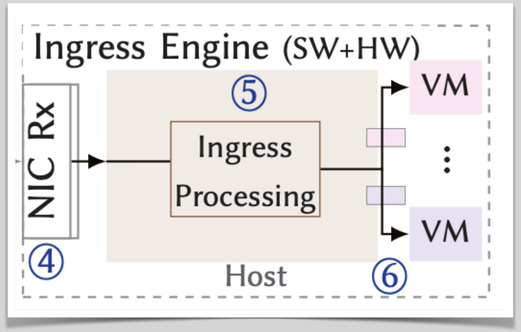
- Experiment setup:
  - Two co-located: VM1, VM2
  - VM2 receives a burst of ~16Gbps traffic

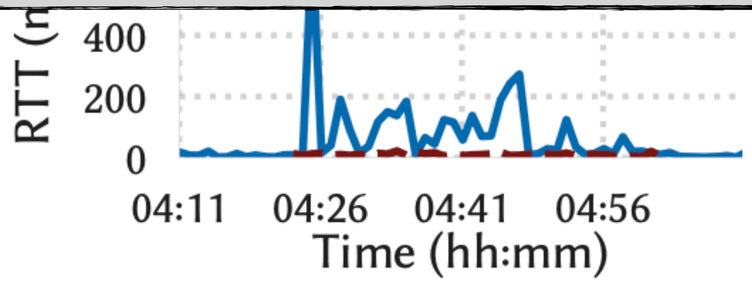


Issue #3: Ingress Engine Contention

- Experiment setup:
  - Two co-located: VM1, VM2
  - VM2 receives a burst of ~16Gbps traffic

A packet burst can exceed the ingress engine's packet processing capacity.

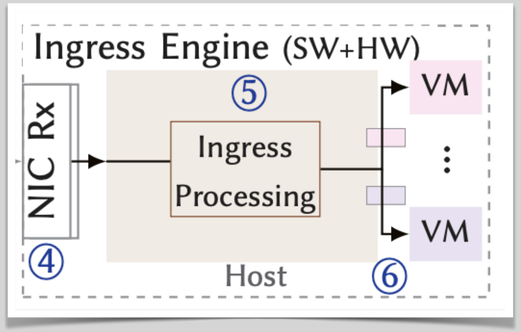


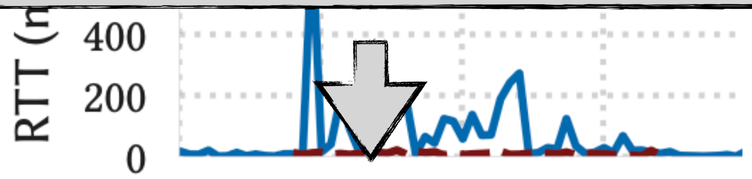


Issue #3: Ingress Engine Contention

- Experiment setup:
  - Two co-located: VM1, VM2
  - VM2 receives a burst of ~16Gbps traffic

A packet burst can exceed the ingress engine's packet processing capacity.





Implication: rethinking computation allocation@RX-Host

Design principles behind PicNIC:

- P1: SLO-based resource sharing => Sacrifice utilization
- P2: Backpressure and early drops => Traffic regulation

Technique #1: Redefine SLO

Technique #1: Redefine SLO

- #1: Bandwidth
  - [MIN_BPS, MAX_BPS]
  - Consider PPS (packets per second)

Technique #1: Redefine SLO

- #1: Bandwidth
  - [MIN_BPS, MAX_BPS]
  - Consider PPS (packets per second)
- #2: Delay
  - Ingress delay: delay in NIC + delay in engine
  - Egress delay: delay from the guest OS Tx queue to the wire

Technique #1: Redefine SLO

- #1: Bandwidth
  - [MIN_BPS, MAX_BPS]
  - Consider PPS (packets per second)
- #2: Delay
  - Ingress delay: delay in NIC + delay in engine
  - Egress delay: delay from the guest OS Tx queue to the wire
- #3: Loss rate
  - Well-behaved VMs: no drop
  - Uncooperative VMs: drop

Technique #2: Ingress CPU-Fair WFQs

- CPU-fair weighted fair queues
  - Per-VM packet queues

Technique #2: Ingress CPU-Fair WFQs

- CPU-fair weighted fair queues
  - Per-VM packet queues
- Enqueue
  - Packet classification

Technique #2: Ingress CPU-Fair WFQs

- CPU-fair weighted fair queues
  - Per-VM packet queues
- Enqueue
  - Packet classification
- Dequeue
  - Monitor the packet processing time for each VM and apply EMWA
  - Allocate dequeueing CPU works based on SLO

Technique #2: Ingress CPU-Fair WFQs

- CPU-fair weighted fair queues
  - Per-VM packet queues
- Enqueue
  - Packet classification
- Dequeue
  - Monitor the packet processing time for each VM and apply EMWA
  - Allocate dequeueing CPU works based on SLO

Target: issue #3
Benefits: delay and drops

Technique #3: PicNIC Congestion Control

- Receiver-driven
  - PCCB: throughput mode
  - PCCP: latency mode

Technique #3: PicNIC Congestion Control

- Receiver-driven
  - PCCB: throughput mode
  - PCCP: latency mode

PCCB

- Receivers use the Max-min fairness to determine the rate
- Senders apply an RCP-like approach to control the rate

Technique #3: PicNIC Congestion Control

- Receiver-driven
  - PCCB: throughput mode
  - PCCP: latency mode

PCCB

- Receivers use the Max-min fairness to determine the rate
- Senders apply an RCP-like approach to control the rate

PCCP

Input: $delay\_in$
$delay \leftarrow EWMA(delay, delay\_in)$
if $delay > threshold$ **then** ▷ multiplicative decrease (MD)
  $rate \leftarrow (1 - \beta \cdot (1 - \frac{threshold}{delay})) \cdot rate$
  $counter \leftarrow 0$ ▷ enter fast recovery (FR)
  $target\_rate \leftarrow rate$
else ▷ default: fast recovery (FR)
  if $N_{AI} < counter \leq N_{HAI}$ **then** ▷ additive increase (AI)
    $target\_rate \leftarrow target\_rate + \delta$
  if $counter > N_{HAI}$ **then** ▷ hyper-active increase (HAI)
    $target\_rate \leftarrow target\_rate + (counter - N_{HAI}) \cdot \delta$
  $rate \leftarrow \frac{rate + target\_rate}{2}$
  $counter \leftarrow counter + 1$
Output: $rate$ ▷ initial value = goodput / approx. #senders

Technique #3: PicNIC Congestion Control

- Receiver-driven
  - PCCB: throughput mode
  - PCCP: latency mode

PCCB

- Receivers use the Max-min fairness to determine the rate
- Senders apply an RCP-like approach to

PCCP

Input: $delay\_in$
$delay \leftarrow EWMA(delay, delay\_in)$
if $delay > threshold$ **then** ▷ multiplicative decrease (MD)
  $rate \leftarrow (1 - \beta \cdot (1 - \frac{threshold}{delay})) \cdot rate$
  $counter \leftarrow 0$ ▷ enter fast recovery (FR)
  $target\_rate \leftarrow rate$

Target: cause #2

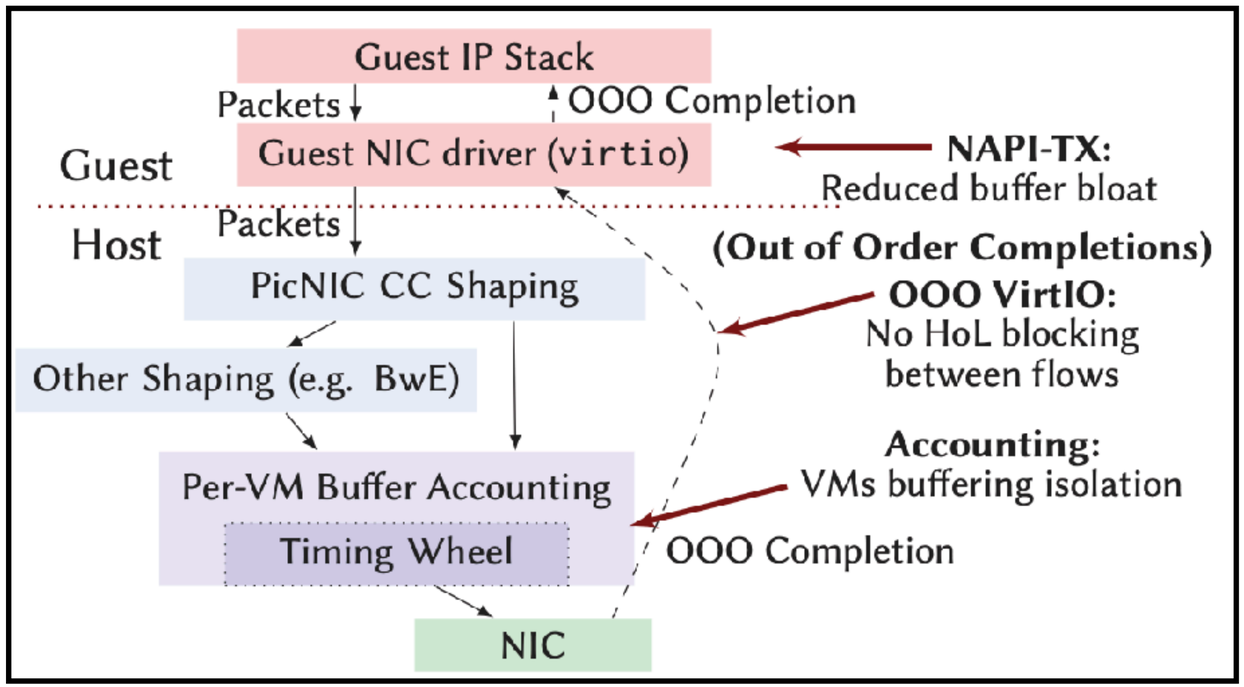
Benefits: bandwidth, delay, and drops

Technique #4: Sender-side Admission Control

- Smart traffic shaper
  - Per-VM packet counting
  - Backpressure to guest OS

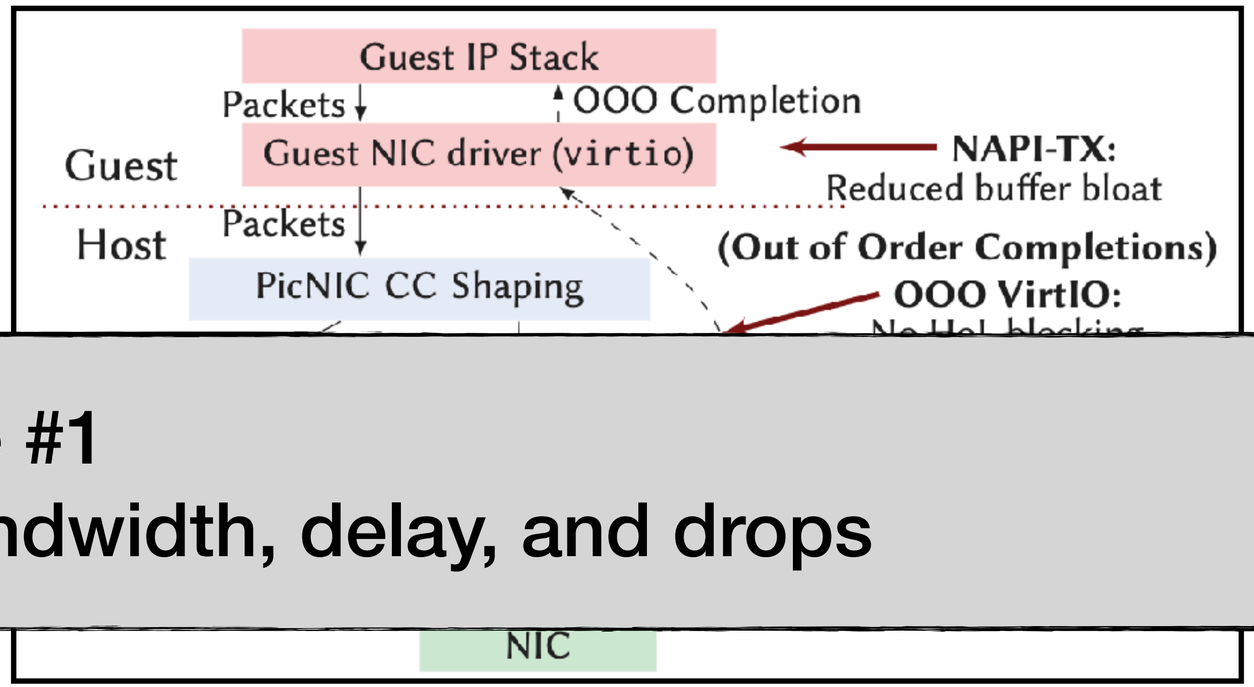
Technique #4: Sender-side Admission Control

- Smart traffic shaper
  - Per-VM packet counting
  - Backpressure to guest OS



Technique #4: Sender-side Admission Control

- Smart traffic shaper
  - Per-VM packet counting
  - Backpressure to guest OS



Target: issue #1
Benefits: bandwidth, delay, and drops

Network performance isolation requires considering (1) the entire communication path; (2) the interaction between network with CPU/memory.

Summary

- Today
  - Network virtualization in data center networks (II)

- Next topic: SDN and programmable networks
  - Ethane (Sigcomm'07) and OpenFlow (Sigcomm CCR'08)
  - RMT (Sigcomm'13) and AFQ (NSDI'18)
  - AccelNet (NSDI'18) and iPipe (Sigcomm'19)