White-Boxing RDMA with Packet-Granular Software Control

Frank Chenxingyu Zhao[1], Jaehong Min[1], Ming Liu[2], Arvind Krishnamurthy[1]

University of Washington[1]
University of Wisconsin-Madison[2]

NSDI 2025

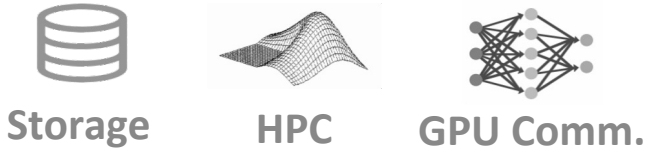
UNIVERSITY of WASHINGTON

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

RDMA Transport Innovations are in Full Swing

RDMA Transport Innovations are in Full Swing

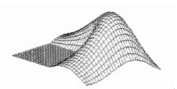
Driven by Workloads



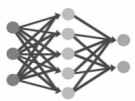
RDMA Transport Innovations are in Full Swing

Driven by Workloads

Storage



HPC



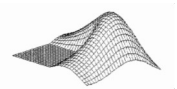
GPU Comm.

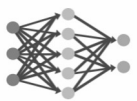
Driven by Deployments

RDMA Transport Innovations are in Full Swing

Driven by Workloads

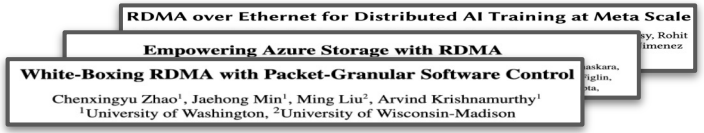

Storage


HPC


GPU Comm.

Driven by Deployments

Numerous Ideas


RDMA over Ethernet for Distributed AI Training at Meta Scale

Empowering Azure Storage with RDMA

White-Boxing RDMA with Packet-Granular Software Control

Chenxingyu Zhao[1], Jaehong Min[1], Ming Liu[2], Arvind Krishnamurthy[1]
[1]University of Washington, [2]University of Wisconsin-Madison

Past 5 years, 30+ NSDI/SIGCOMM papers explore RDMA innovations.

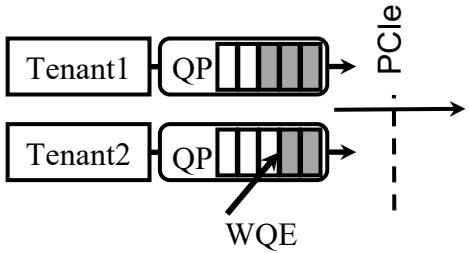
How to Land New Ideas into RDMA NIC?

How to Land New Ideas into RDMA NIC?

- Inflexibility: fixed hardware for both **Ctrl** and **Data** logic

How to Land New Ideas into RDMA NIC?

- Inflexibility: fixed hardware for both **Ctrl** and **Data** logic

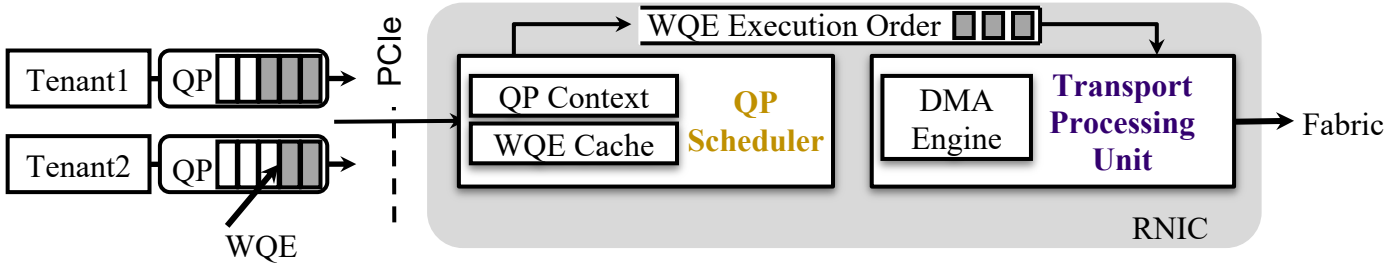


How to Land New Ideas into RDMA NIC?

- Inflexibility: fixed hardware for both **Ctrl** and **Data** logic

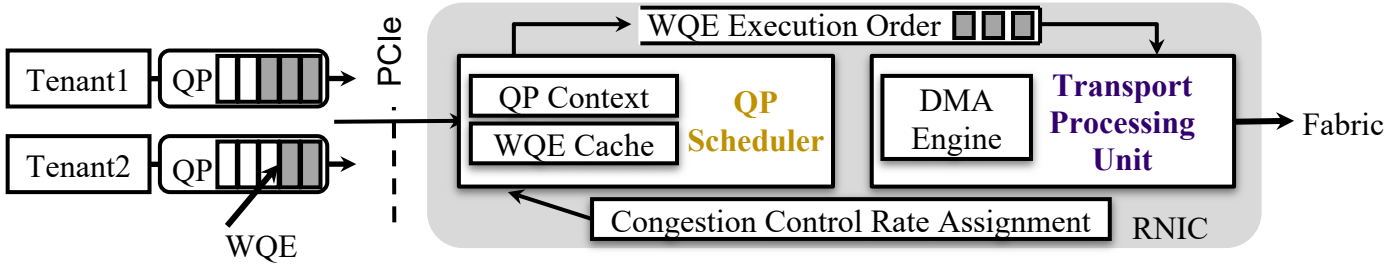
How to Land New Ideas into RDMA NIC?

- Inflexibility: fixed hardware for both **Ctrl** and **Data** logic



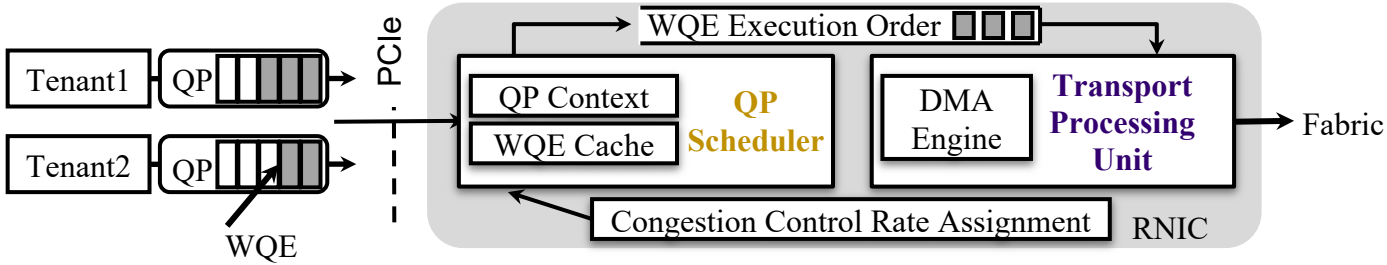
How to Land New Ideas into RDMA NIC?

- Inflexibility: fixed hardware for both **Ctrl** and **Data** logic



How to Land New Ideas into RDMA NIC?

- Inflexibility: fixed hardware for both **Ctrl** and **Data** logic

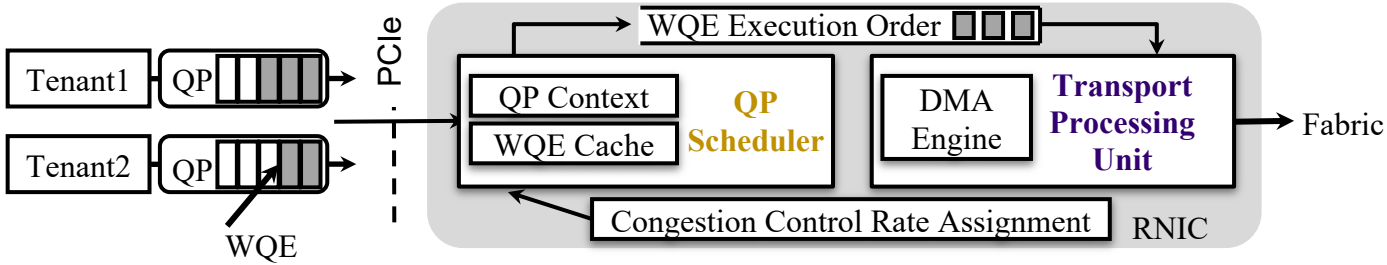


FPGA

High Dev. Cost

How to Land New Ideas into RDMA NIC?

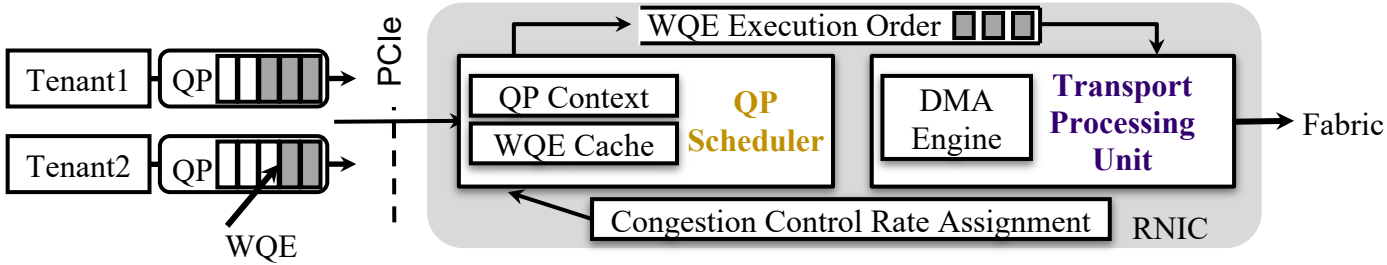
- Inflexibility: fixed hardware for both **Ctrl** and **Data** logic



FPGA	**ASIC**
High Dev. Cost	Wait for Years

How to Land New Ideas into RDMA NIC?

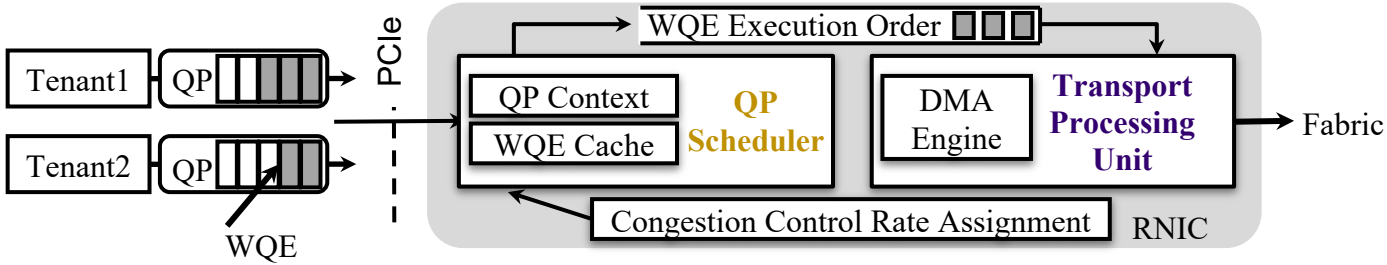
- Inflexibility: fixed hardware for both **Ctrl** and **Data** logic



FPGA	**ASIC**	**SW Emulate**
High Dev. Cost	Wait for Years	Perf. Loss

How to Land New Ideas into RDMA NIC?

- Inflexibility: fixed hardware for both **Ctrl** and **Data** logic



FPGA	**ASIC**	**SW Emulate**	**Whiteboxing**
High Dev. Cost	Wait for Years	Perf. Loss	SW Ctrl HW Data

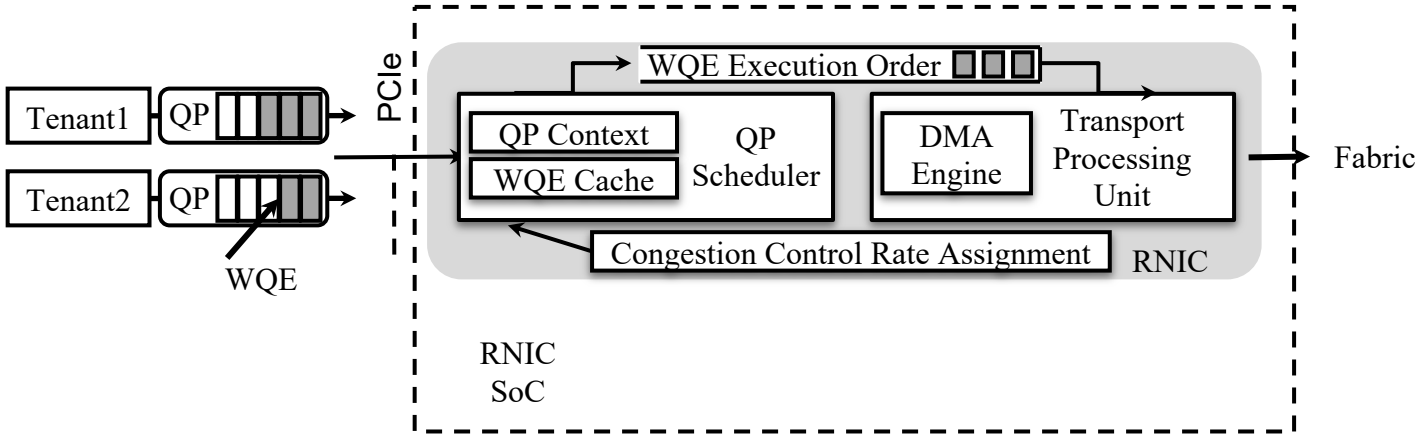
- **Whiteboxing**: Inspired by SDN, control in software, data path in hardware

New RNICs with Lightweight Programmability

- NV BF3/CX-8 embed low-profile datapath processor.

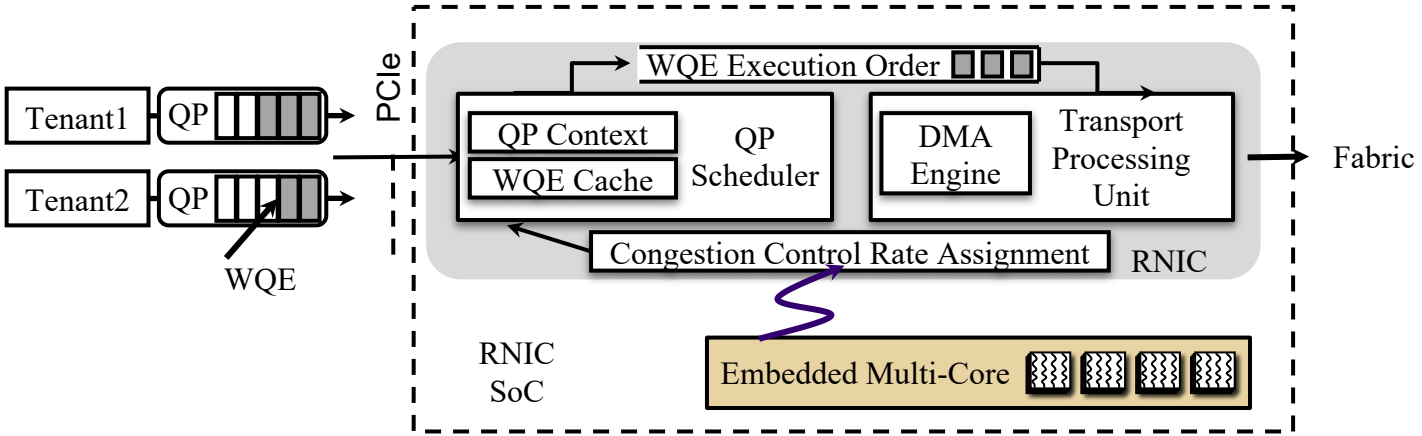
New RNICs with Lightweight Programmability

- NV BF3/CX-8 embed low-profile datapath processor.



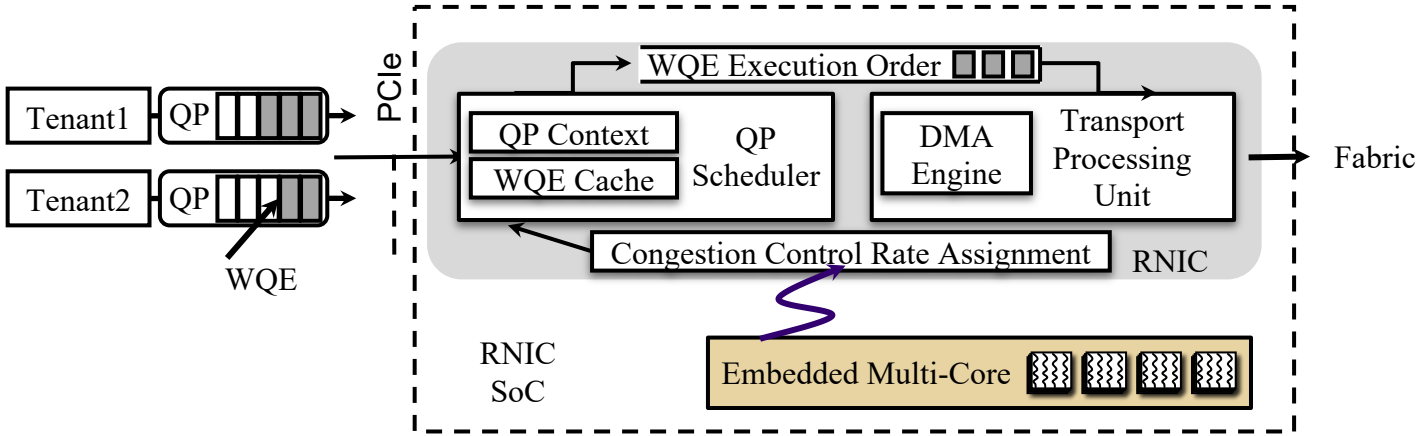
New RNICs with Lightweight Programmability

- NV BF3/CX-8 embed low-profile datapath processor.



New RNICs with Lightweight Programmability

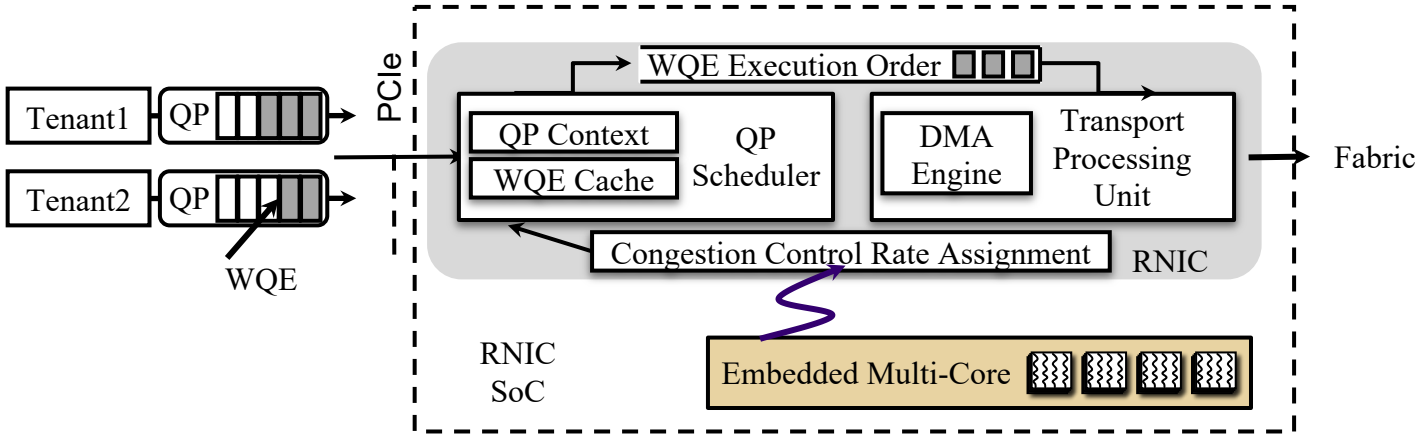
- NV BF3/CX-8 embed low-profile datapath processor.



- Building blocks from embedded cores:

New RNICs with Lightweight Programmability

- NV BF3/CX-8 embed low-profile datapath processor.

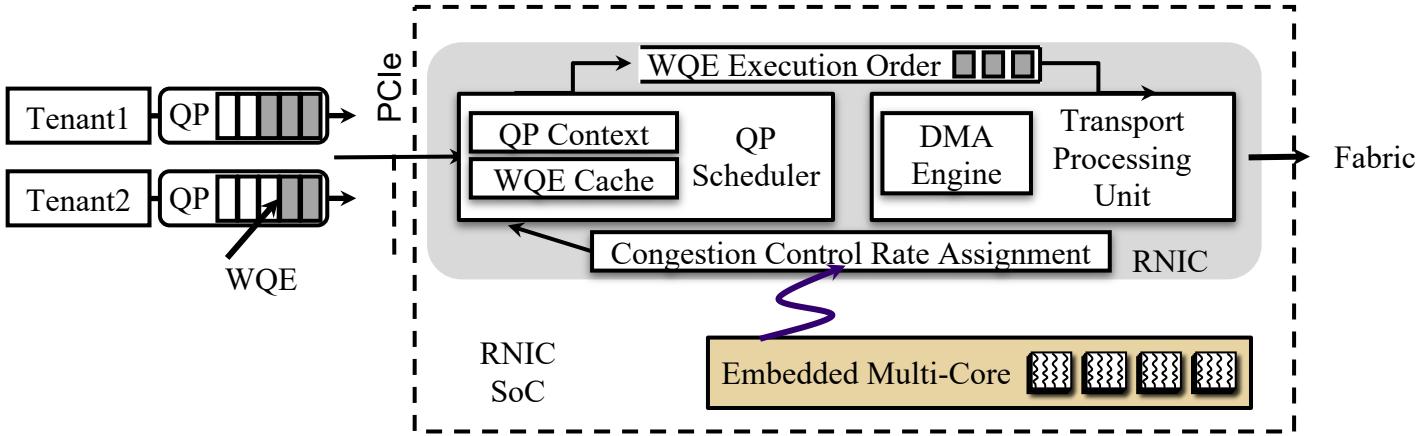


- Building blocks from embedded cores:

General Computation

New RNICs with Lightweight Programmability

- NV BF3/CX-8 embed low-profile datapath processor.

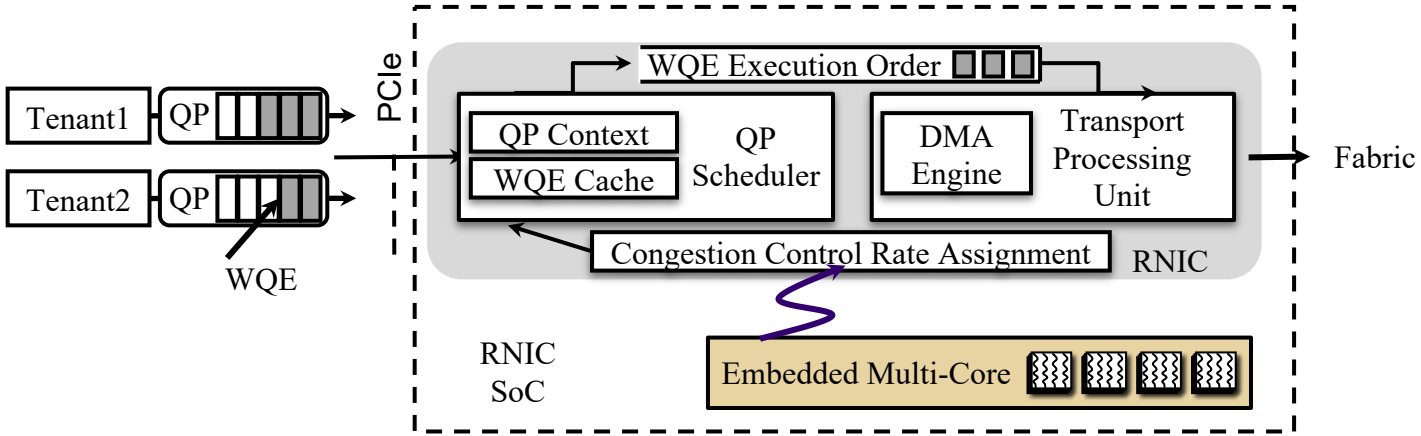


- Building blocks from embedded cores:

General Computation	Rate Control Interface

New RNICs with Lightweight Programmability

- NV BF3/CX-8 embed low-profile datapath processor.



- Building blocks from embedded cores:

General Computation	Rate Control Interface	Congestion Events (e.g, ECN)

Challenges to Realize Whiteboxing RDMA

- With SmartNICs building blocks, we need to further address challenges:

Challenges to Realize Whiteboxing RDMA

- With SmartNICs building blocks, we need to further address challenges:

Flexibility

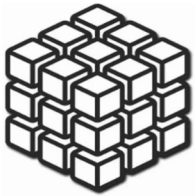
Express Diverse Customizations

Challenges to Realize Whiteboxing RDMA

- With SmartNICs building blocks, we need to further address challenges:

Flexibility

Express Diverse Customizations



Granularity

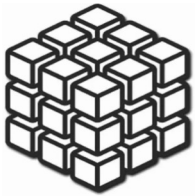
Packet-Granular Events from High Line Rates

Challenges to Realize Whiteboxing RDMA

- With SmartNICs building blocks, we need to further address challenges:

Flexibility

Express Diverse Customizations



Granularity

Packet-Granular Events from High Line Rates

Scalability

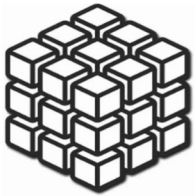
Handle Many Concurrent Flows

Challenges to Realize Whiteboxing RDMA

- With SmartNICs building blocks, we need to further address challenges:

Flexibility

Express Diverse Customizations



Granularity

Packet-Granular Events from High Line Rates

Scalability

Handle Many Concurrent Flows

- Next, we introduce **Software Controlled RDMA (SCR)** to achieve all.

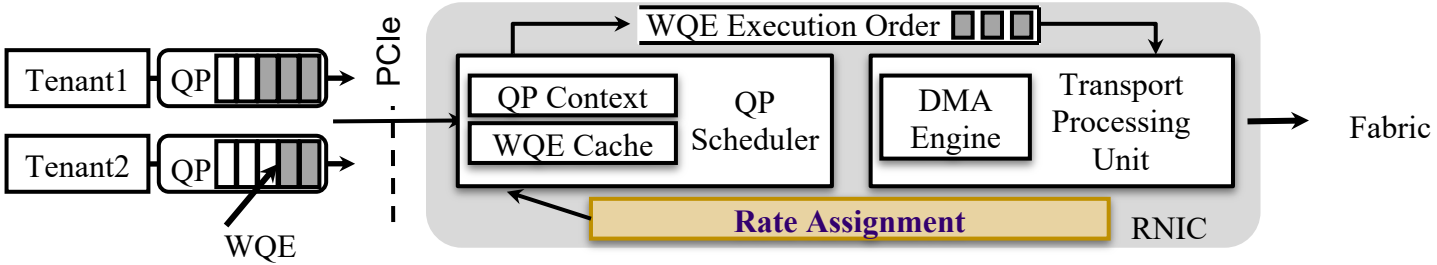
#1 Flexibility: How to Support Diverse Policies?

#1 Flexibility: How to Support Diverse Policies?

- Repurpose the rate assignment knob for ***Dequeue Rate Control***

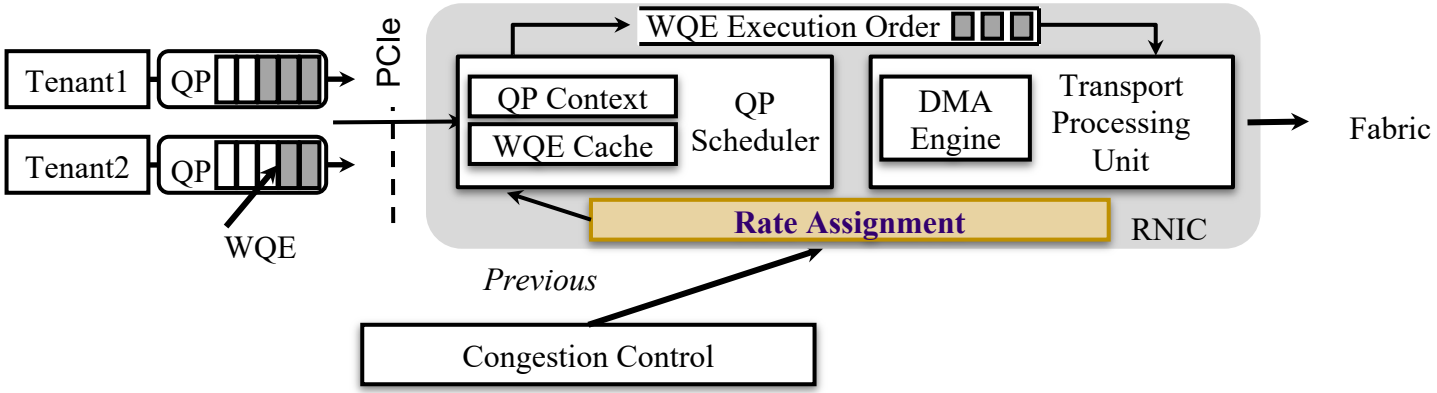
#1 Flexibility: How to Support Diverse Policies?

- Repurpose the rate assignment knob for ***Dequeue Rate Control***



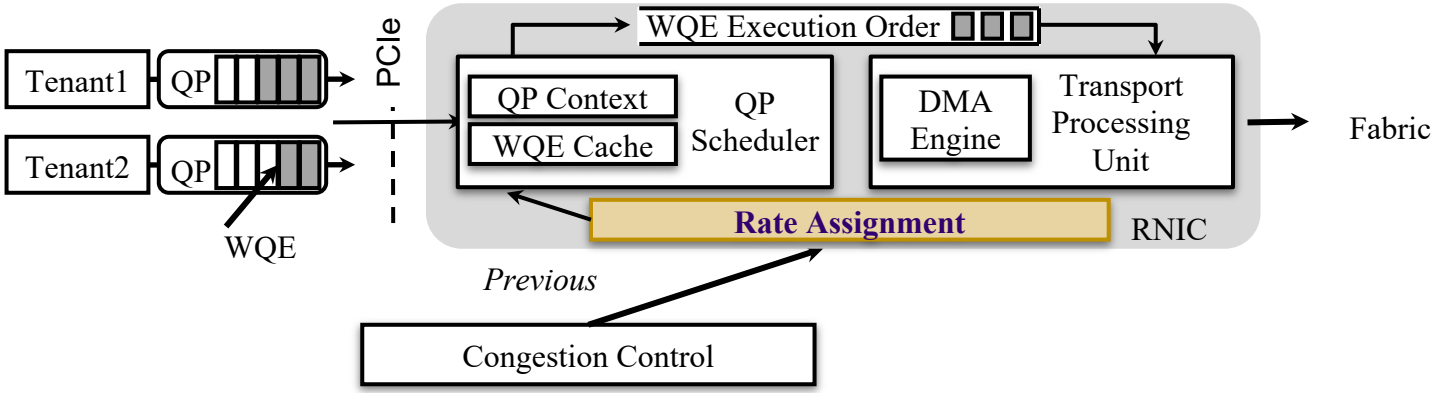
#1 Flexibility: How to Support Diverse Policies?

- Repurpose the rate assignment knob for ***Dequeue Rate Control***



#1 Flexibility: How to Support Diverse Policies?

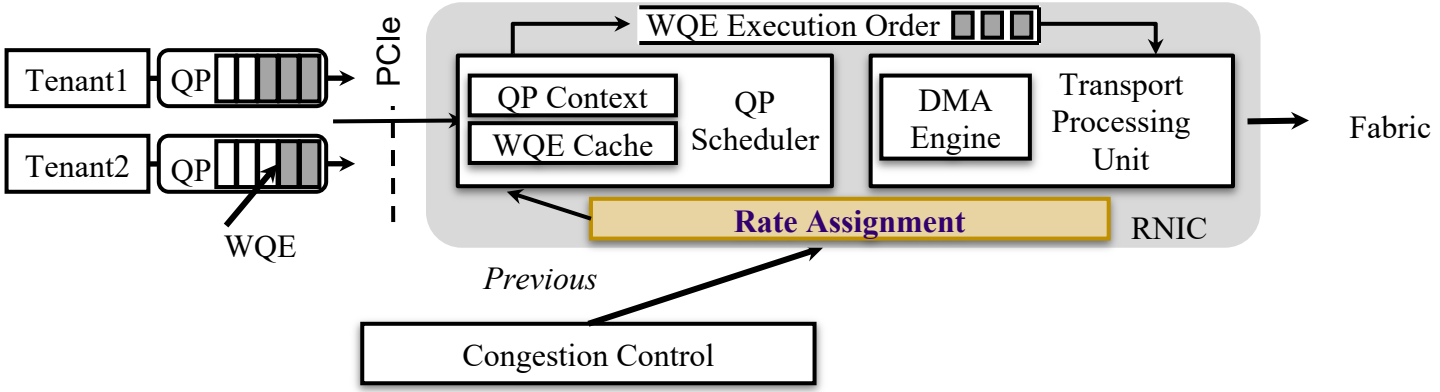
- Repurpose the rate assignment knob for ***Dequeue Rate Control***



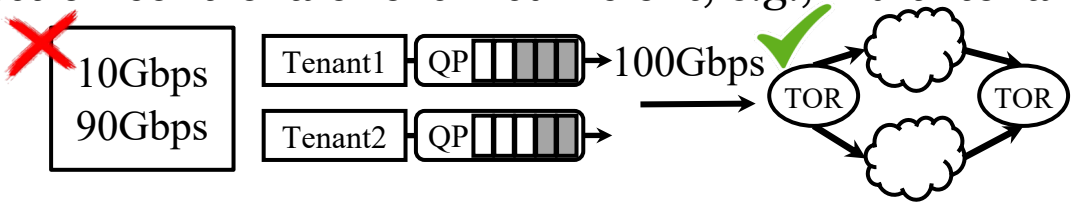
- Congestion control alone is insufficient, e.g., multi-tenant isolation

#1 Flexibility: How to Support Diverse Policies?

- Repurpose the rate assignment knob for ***Dequeue Rate Control***

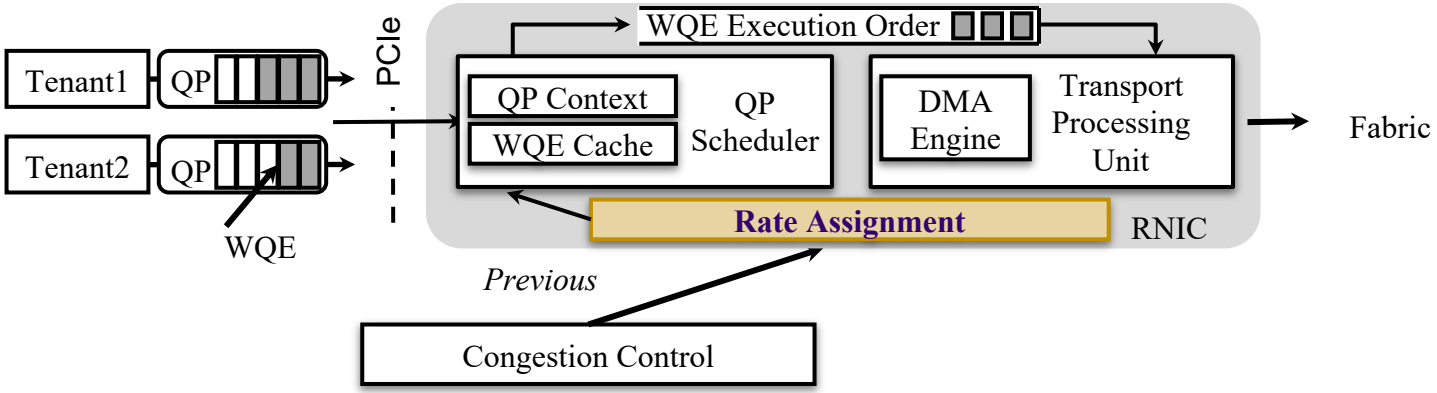


- Congestion control alone is insufficient, e.g., multi-tenant isolation



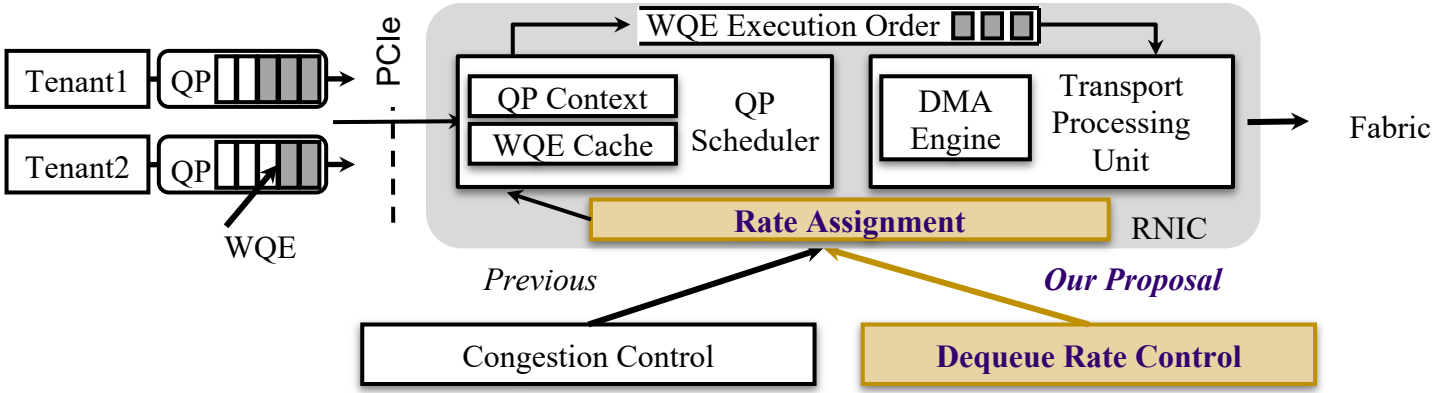
#1 Flexibility: How to Support Diverse Policies?

- Repurpose the rate assignment knob for ***Dequeue Rate Control***



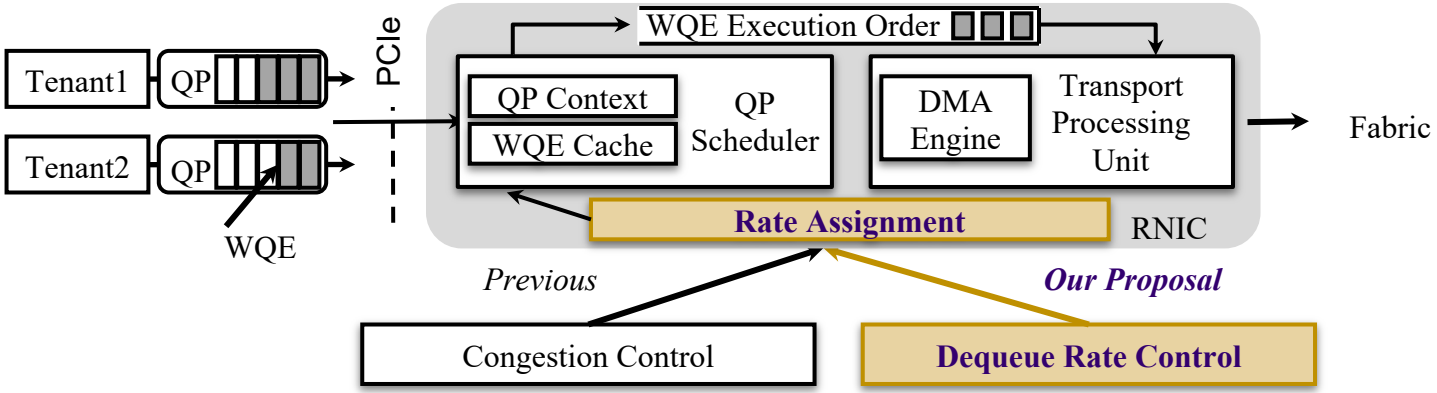
#1 Flexibility: How to Support Diverse Policies?

- Repurpose the rate assignment knob for ***Dequeue Rate Control***



#1 Flexibility: How to Support Diverse Policies?

- Repurpose the rate assignment knob for ***Dequeue Rate Control***



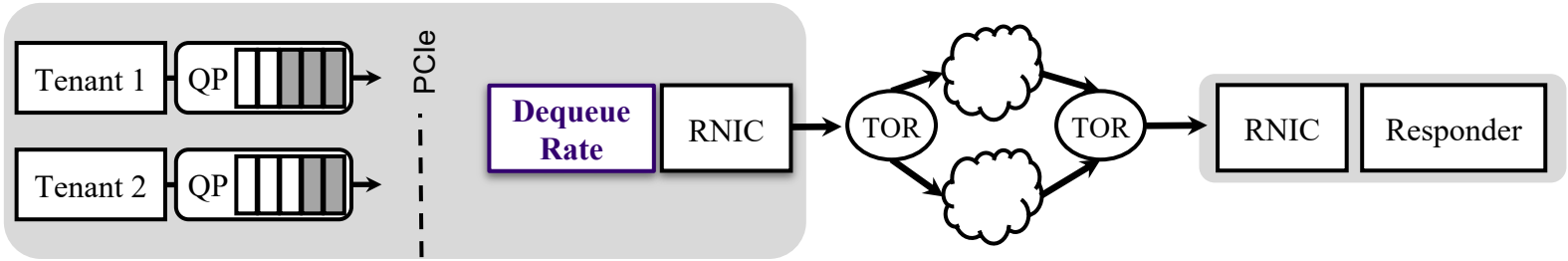
- Rate at which the RNIC retrieves messages from memory regions

#1 Flexibility: Why Dequeue Rate?

- Dequeue Rate considers events from multiple domains:

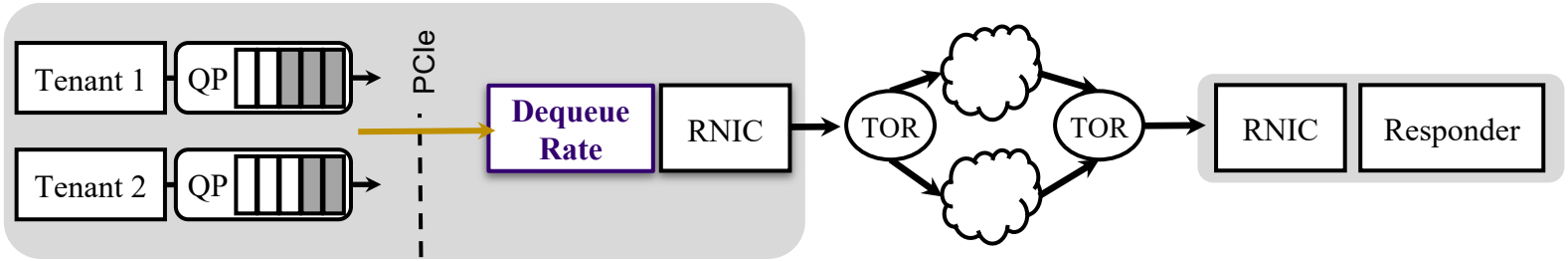
#1 Flexibility: Why Dequeue Rate?

- Dequeue Rate considers events from multiple domains:



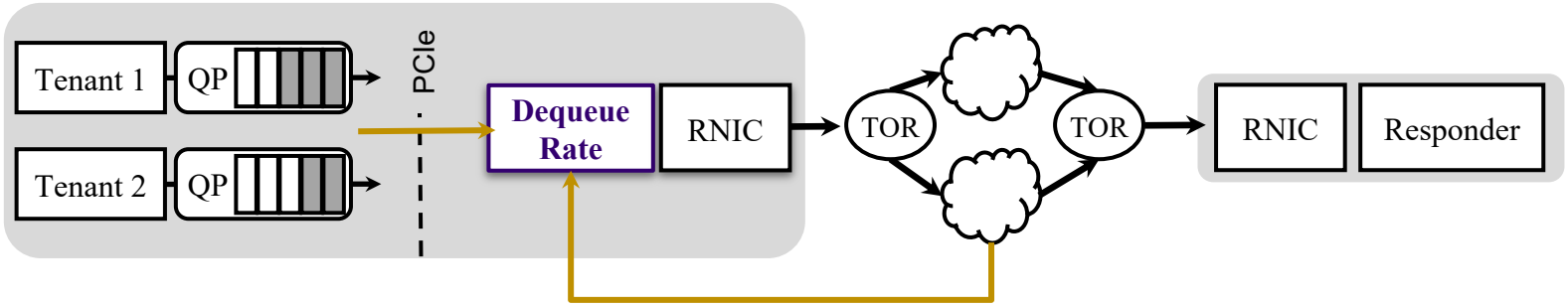
#1 Flexibility: Why Dequeue Rate?

- Dequeue Rate considers events from multiple domains:
  - Host Domain: e.g., RNIC Utilization



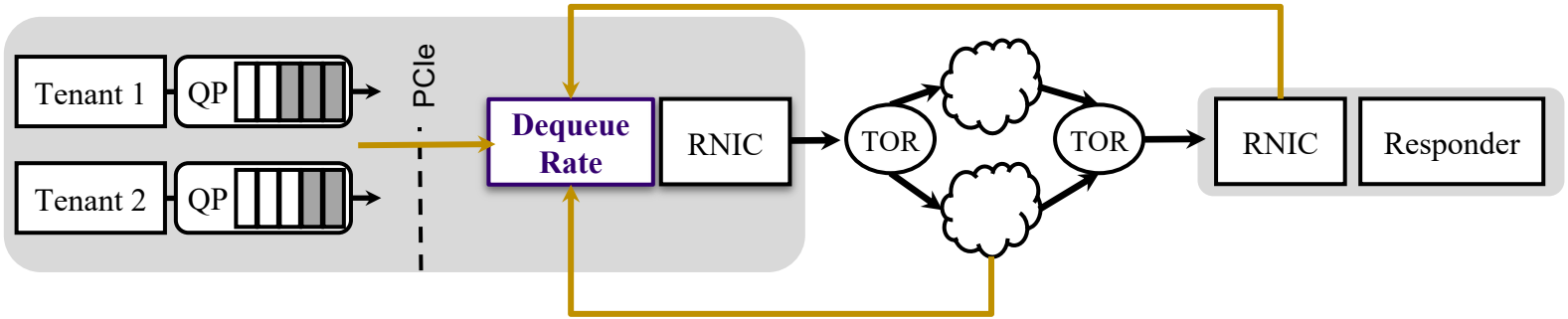
#1 Flexibility: Why Dequeue Rate?

- Dequeue Rate considers events from multiple domains:
  - Host Domain: e.g., RNIC Utilization
  - Fabric Domain: e.g., In-Network Congestion



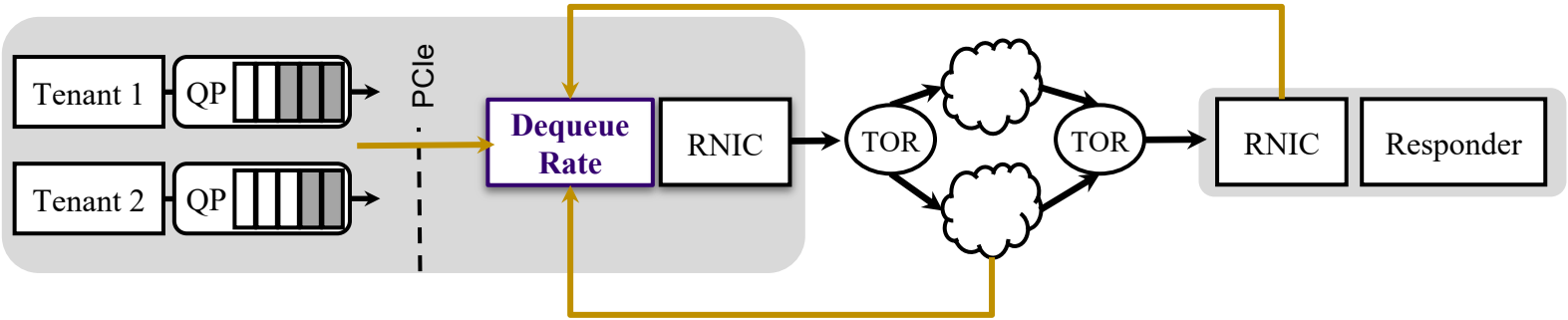
#1 Flexibility: Why Dequeue Rate?

- Dequeue Rate considers events from multiple domains:
  - Host Domain: e.g., RNIC Utilization
  - Fabric Domain: e.g., In-Network Congestion
  - Peer Domain: e.g., Receiver-Driven Feedbacks



#1 Flexibility: Why Dequeue Rate?

- Dequeue Rate considers events from multiple domains:
  - Host Domain: e.g., RNIC Utilization
  - Fabric Domain: e.g., In-Network Congestion
  - Peer Domain: e.g., Receiver-Driven Feedbacks



- See paper for multi-domain events and the collection process.

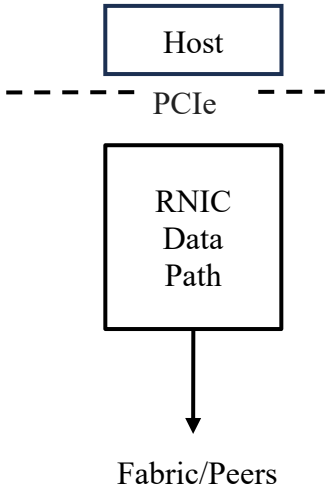
#2 Granularity: How to Control at Packet Level?

#2 Granularity: How to Control at Packet Level?

- Challenge: 100Gbps, 1500B MTU, sub-μs packet-granular events (120 ns!)

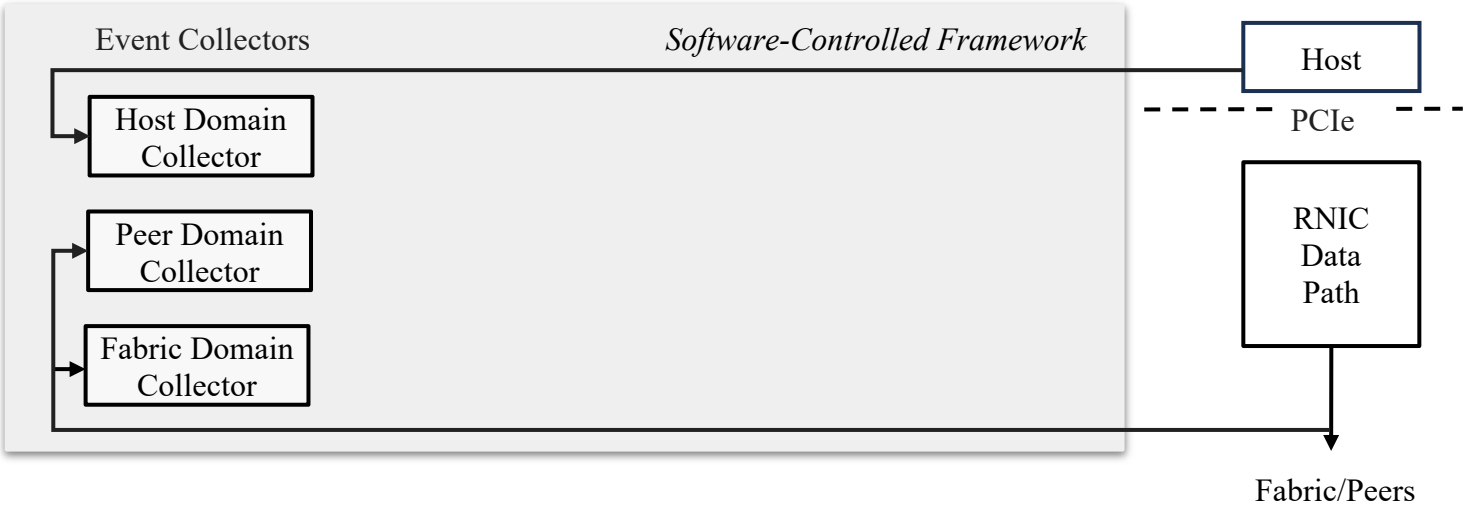
#2 Granularity: How to Control at Packet Level?

- Challenge: 100Gbps, 1500B MTU, sub-µs packet-granular events (120 ns!)



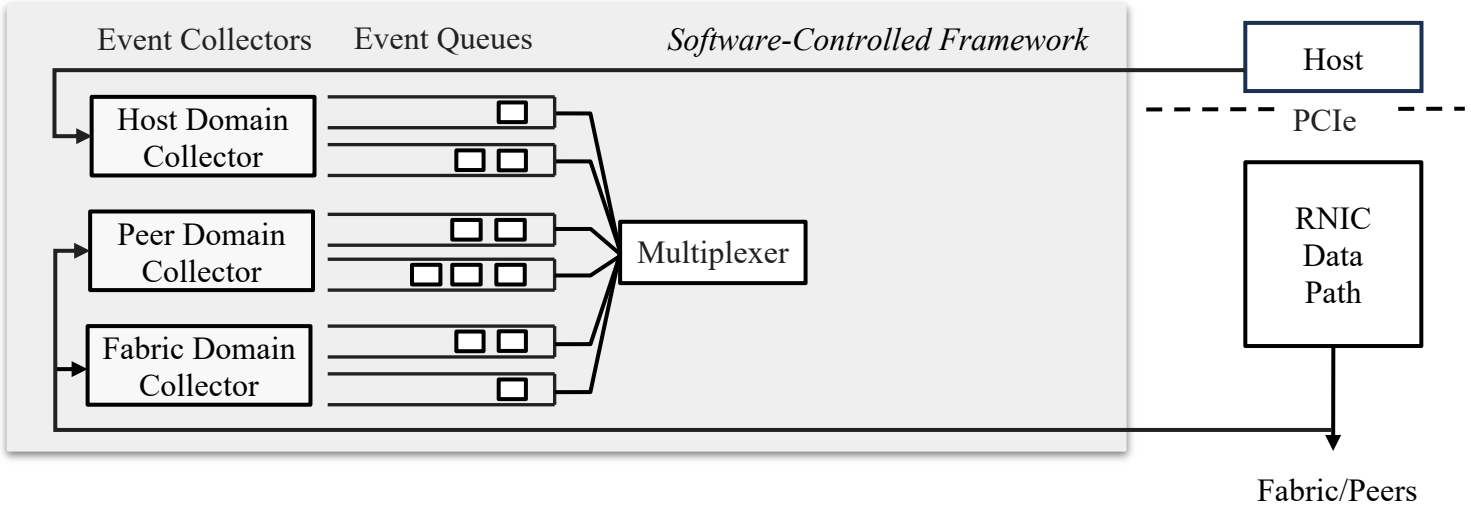
#2 Granularity: How to Control at Packet Level?

- Challenge: 100Gbps, 1500B MTU, sub-µs packet-granular events (120 ns!)



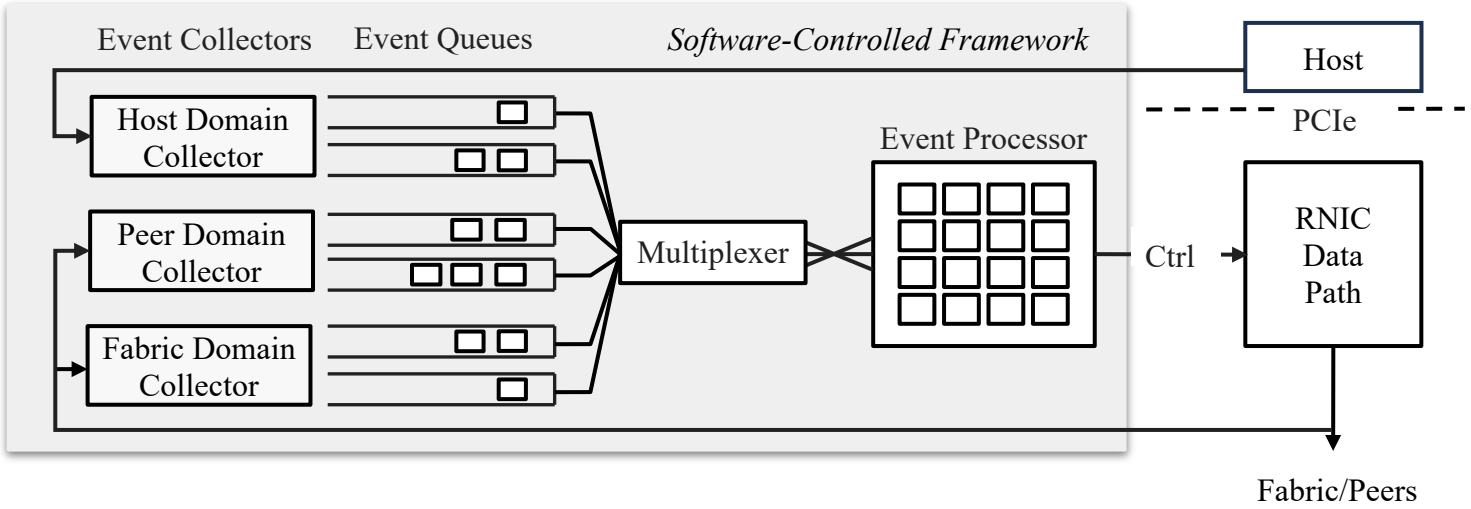
#2 Granularity: How to Control at Packet Level?

- Challenge: 100Gbps, 1500B MTU, sub-µs packet-granular events (120 ns!)



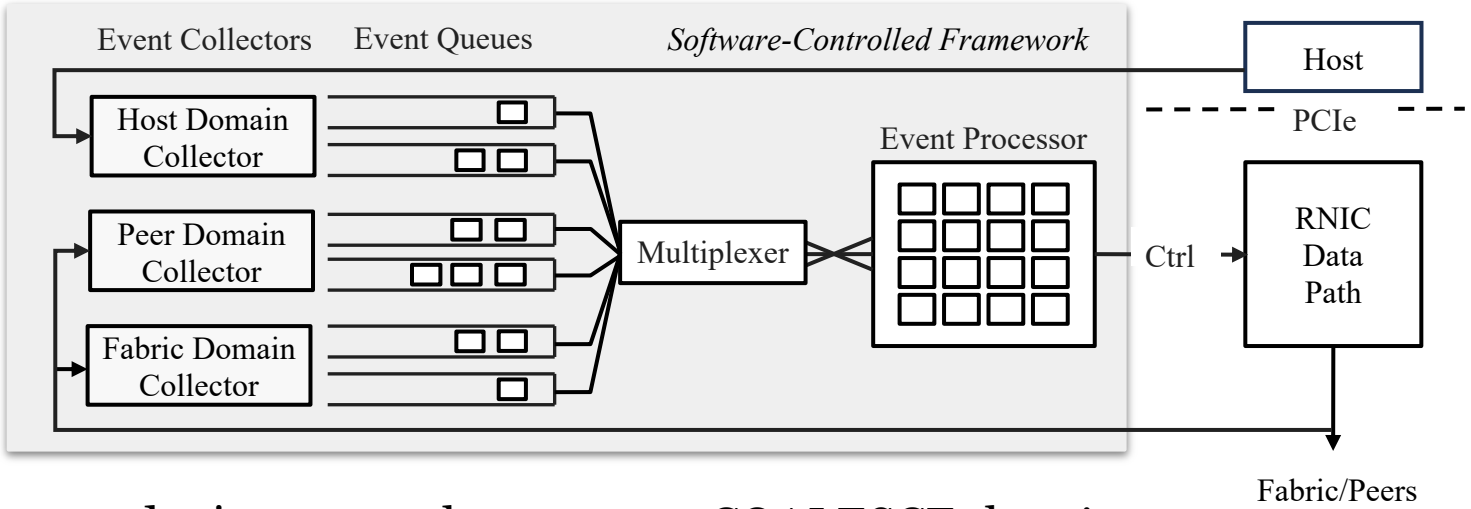
#2 Granularity: How to Control at Packet Level?

- Challenge: 100Gbps, 1500B MTU, sub-µs packet-granular events (120 ns!)



#2 Granularity: How to Control at Packet Level?

- Challenge: 100Gbps, 1500B MTU, sub-µs packet-granular events (120 ns!)



- Queues don't matter when you can COALESCE them!

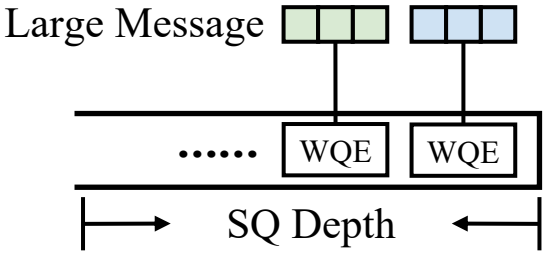
#2 Granularity: Event Queuing and Coalescing

#2 Granularity: Event Queuing and Coalescing

- **Coalescing**: Managing packet-granular events

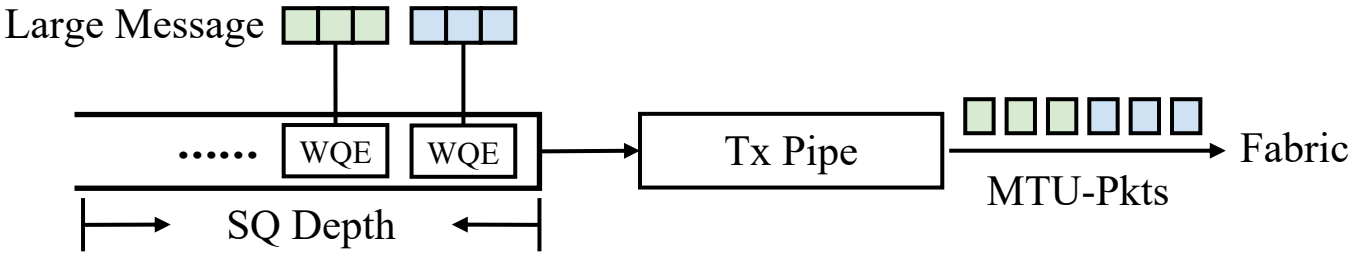
#2 Granularity: Event Queuing and Coalescing

- **Coalescing**: Managing packet-granular events



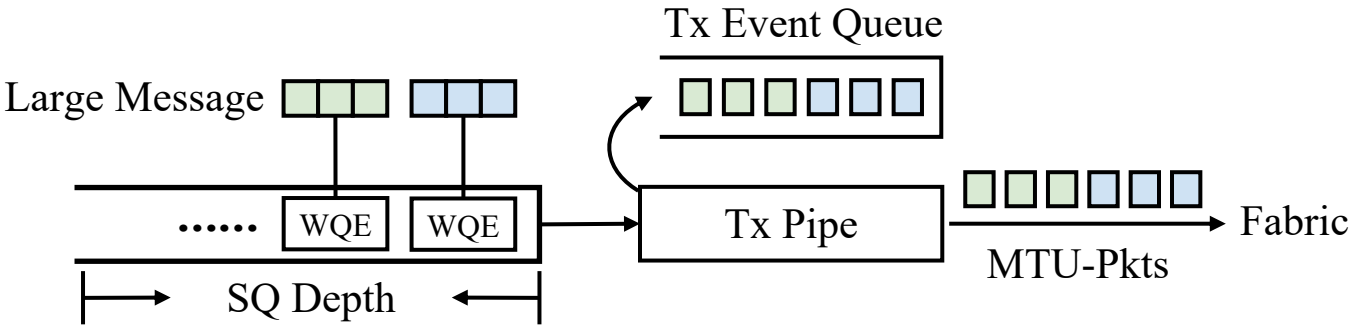
#2 Granularity: Event Queuing and Coalescing

- **Coalescing**: Managing packet-granular events



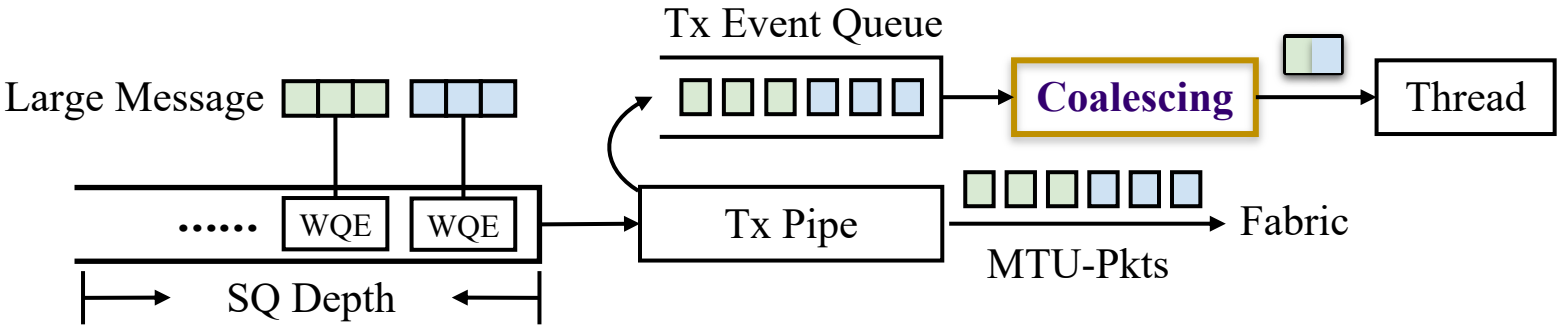
#2 Granularity: Event Queuing and Coalescing

- **Coalescing**: Managing packet-granular events



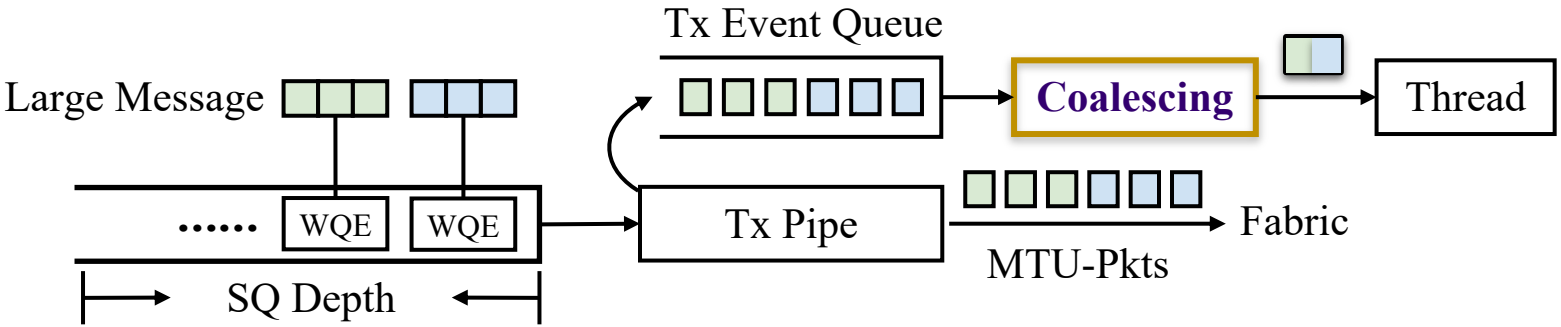
#2 Granularity: Event Queuing and Coalescing

- **Coalescing**: Managing packet-granular events



#2 Granularity: Event Queuing and Coalescing

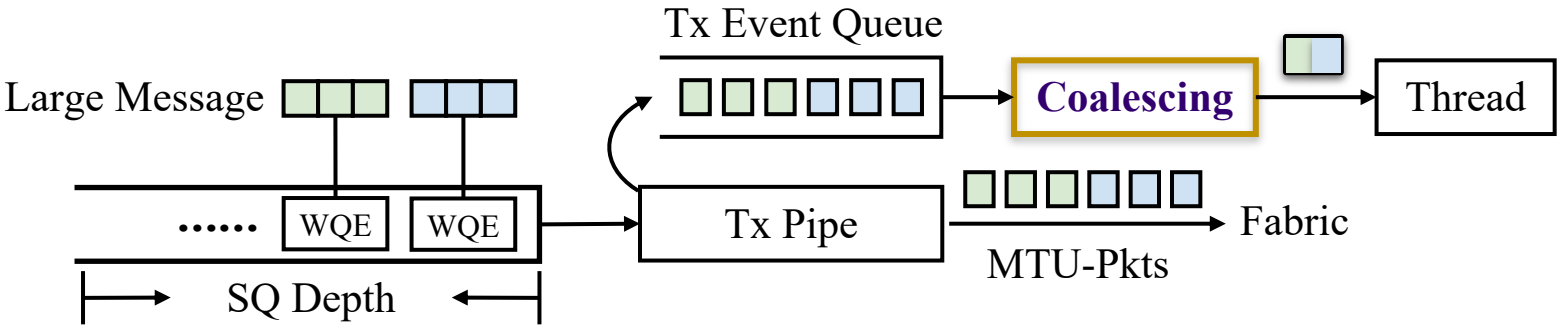
- **Coalescing**: Managing packet-granular events



- HW/SW Hybrid Coalescing:

#2 Granularity: Event Queuing and Coalescing

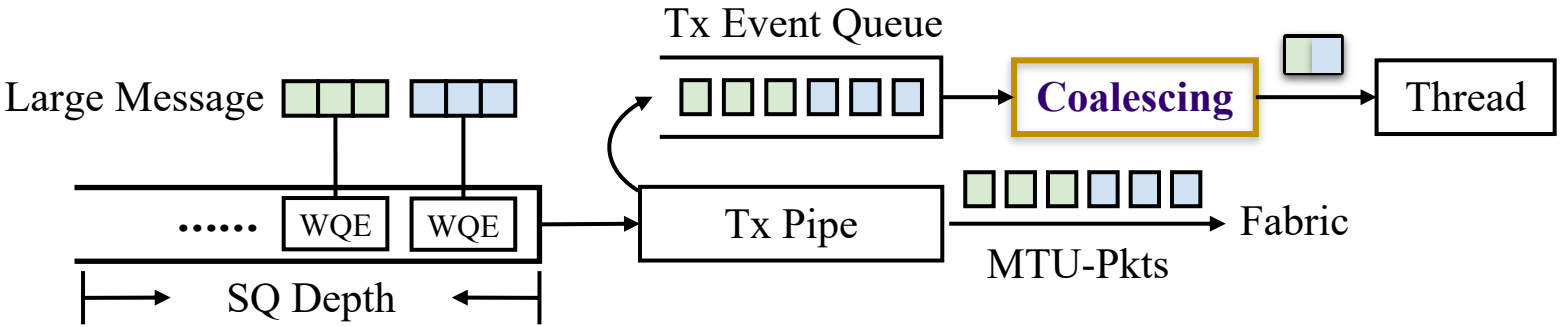
- **Coalescing**: Managing packet-granular events



- HW/SW Hybrid Coalescing:
  - HW coalescing is efficient, but not all events are supported:
    - E.g., fabric domain has HW modules; host domain does not.

#2 Granularity: Event Queuing and Coalescing

- **Coalescing**: Managing packet-granular events



- HW/SW Hybrid Coalescing:
  - HW coalescing is efficient, but not all events are supported:
    - E.g., fabric domain has HW modules; host domain does not.
  - SW coalescing is programmable:
    - E.g., keep latest RTT, drop others.

#3 Scalability: How to Control Many Flows?

#3 Scalability: How to Control Many Flows?

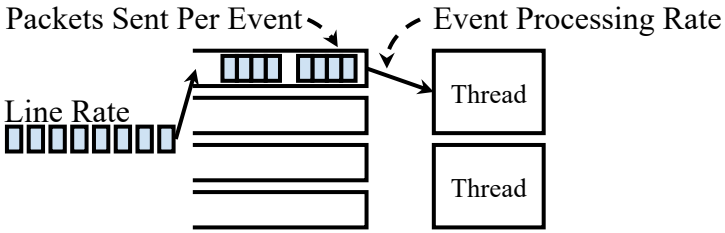
- Increasing Flows, Compromised Granularity? **No!**

#3 Scalability: How to Control Many Flows?

- Increasing Flows, Compromised Granularity? **No!**
  - **Observation:** Adding flows doesn't hurt per-flow coalescing granularity!

#3 Scalability: How to Control Many Flows?

- Increasing Flows, Compromised Granularity? **No!**
  - **Observation:** Adding flows doesn't hurt per-flow coalescing granularity!

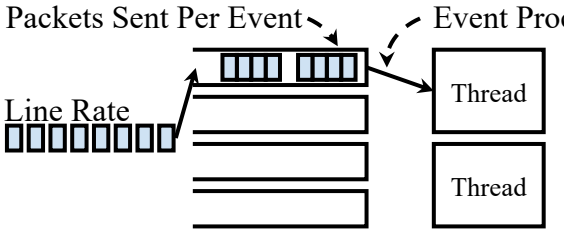


1 QP : *2* Threads

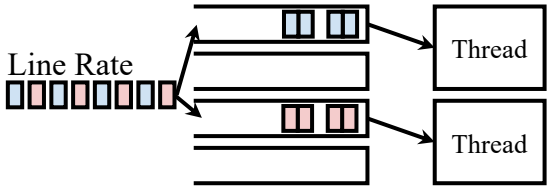
Granularity = 4

#3 Scalability: How to Control Many Flows?

- Increasing Flows, Compromised Granularity? **No!**
  - **Observation:** Adding flows doesn't hurt per-flow coalescing granularity!



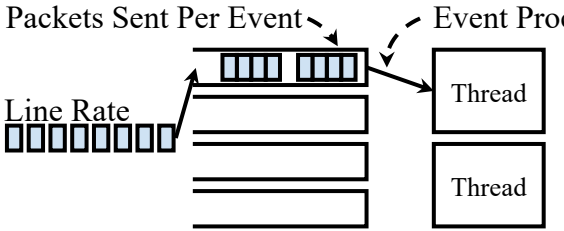
1 QP : *2* Threads
Granularity = 4



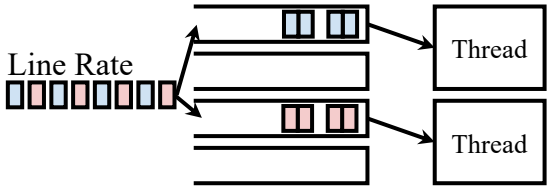
2 QPs : *2* Threads
Granularity = 2

#3 Scalability: How to Control Many Flows?

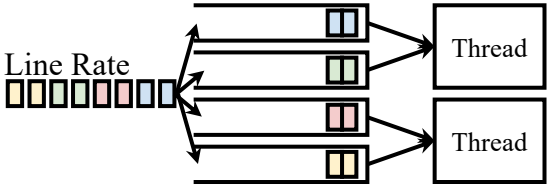
- Increasing Flows, Compromised Granularity? **No!**
  - **Observation:** Adding flows doesn't hurt per-flow coalescing granularity!



1 QP : *2* Threads
Granularity = 4



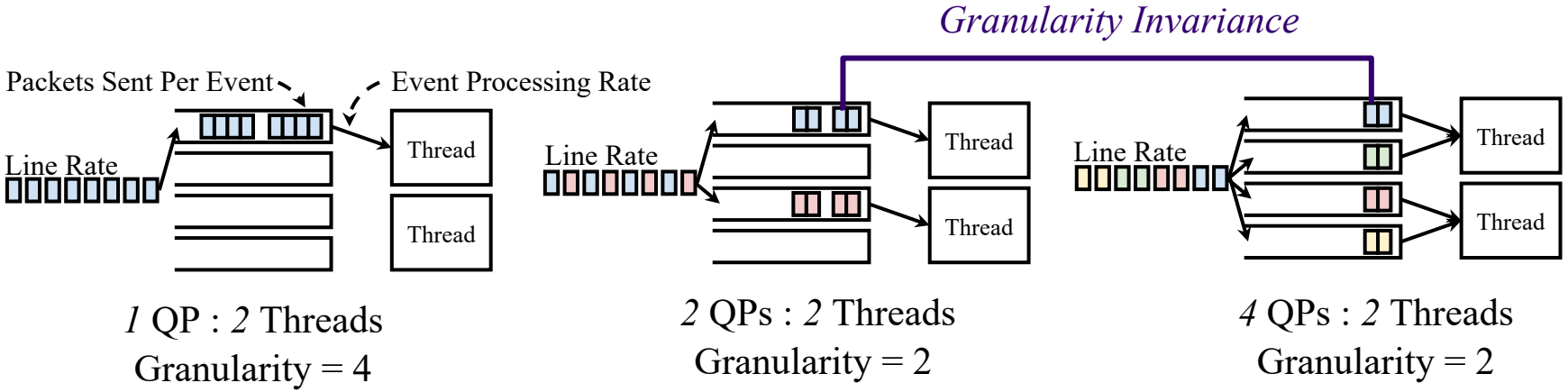
2 QPs : *2* Threads
Granularity = 2



4 QPs : *2* Threads
Granularity = 2

#3 Scalability: How to Control Many Flows?

- Increasing Flows, Compromised Granularity? **No!**
  - **Observation:** Adding flows doesn't hurt per-flow coalescing granularity!



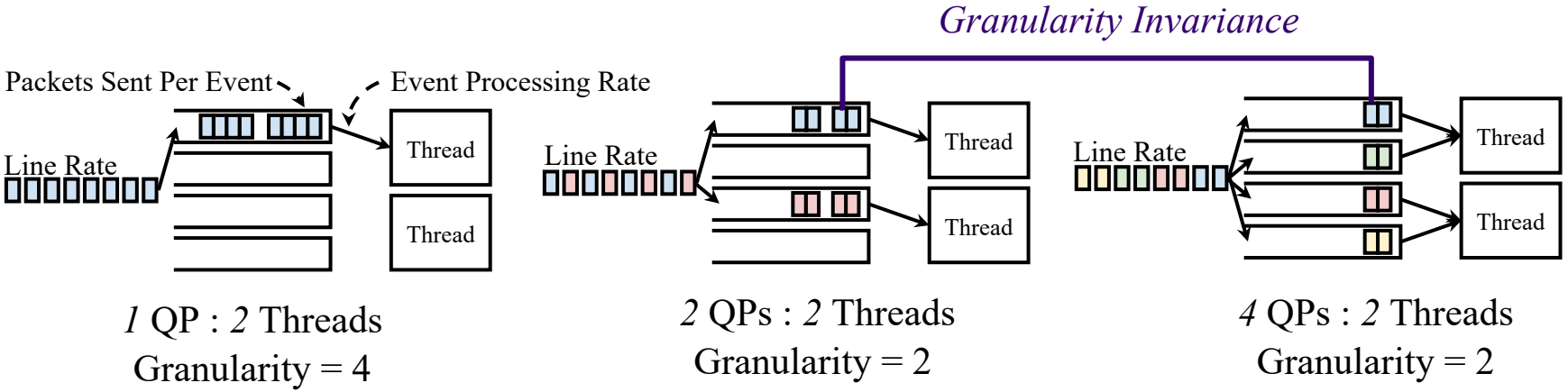
1 QP : *2* Threads
Granularity = 4

2 QPs : *2* Threads
Granularity = 2

4 QPs : *2* Threads
Granularity = 2

#3 Scalability: How to Control Many Flows?

- Increasing Flows, Compromised Granularity? **No!**
  - **Observation:** Adding flows doesn't hurt per-flow coalescing granularity!



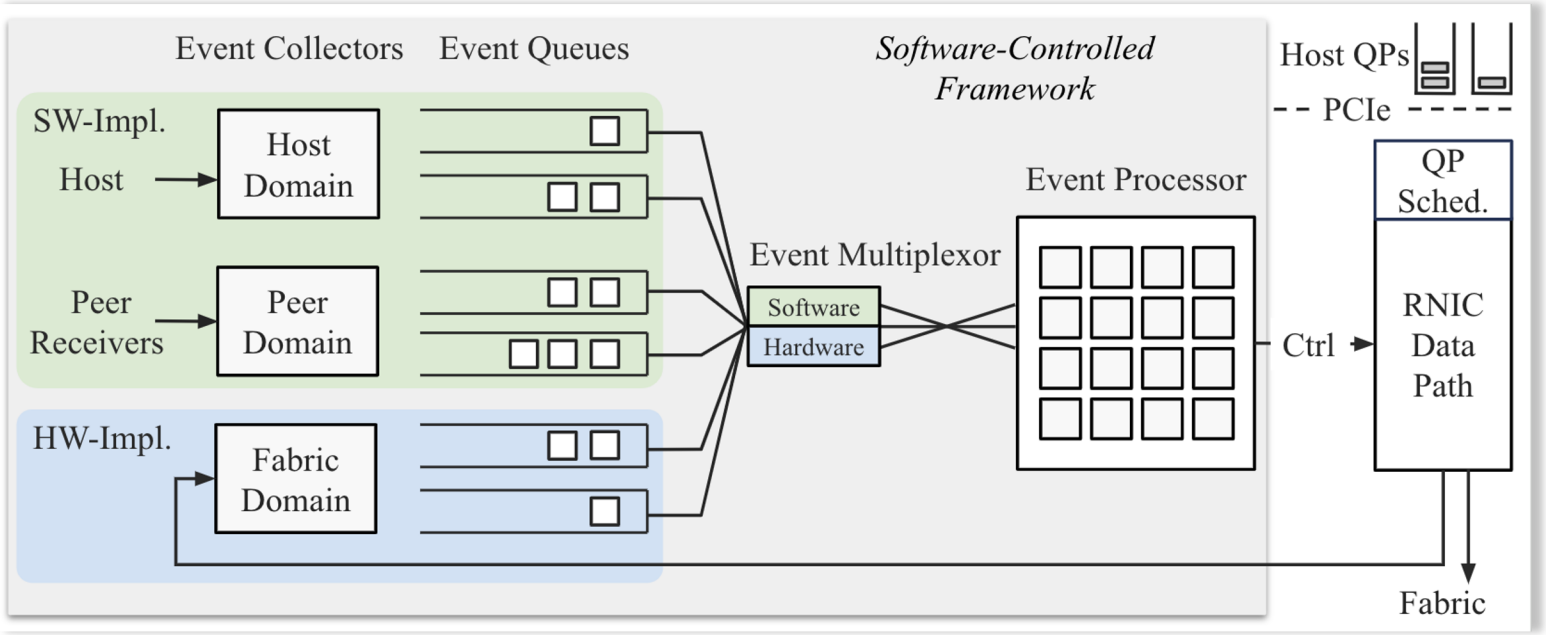
1 QP : *2* Threads
Granularity = 4

2 QPs : *2* Threads
Granularity = 2

4 QPs : *2* Threads
Granularity = 2

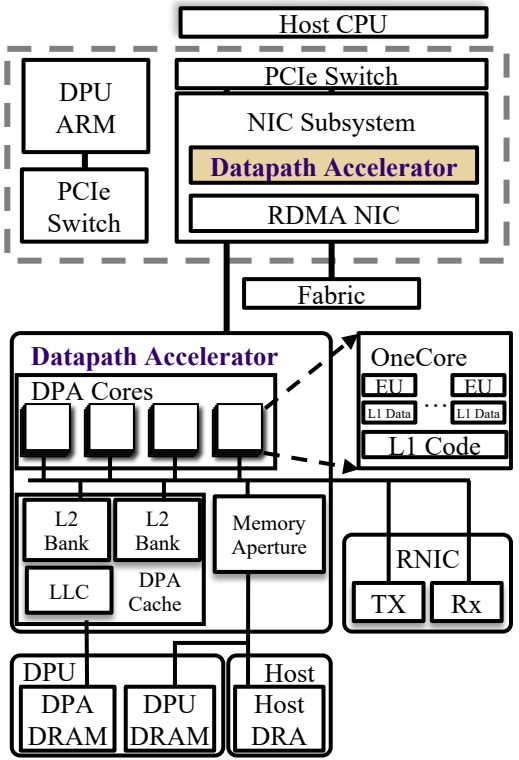
  - Principle: Line rate is invariant—catch up, done!

Addressing Challenges, SCR Framework



Implementation: SCR on BlueField-3 DPA

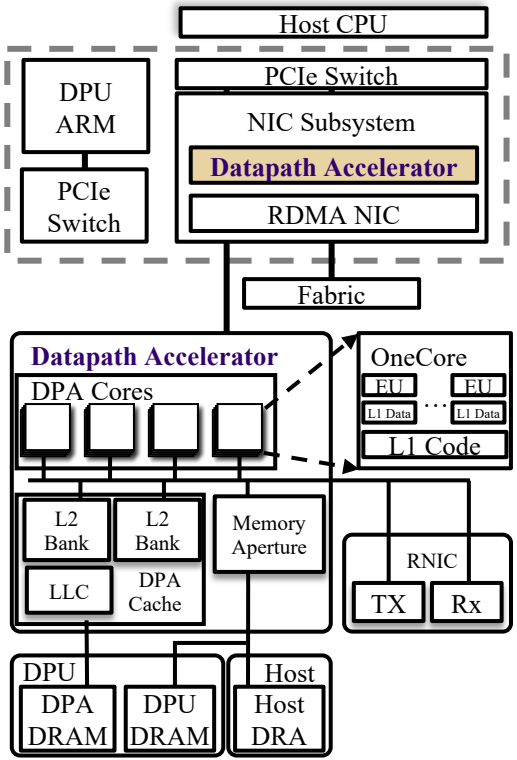
Implementation: SCR on BlueField-3 DPA



[Figures are from NVIDIA Public Information]

Implementation: SCR on BlueField-3 DPA

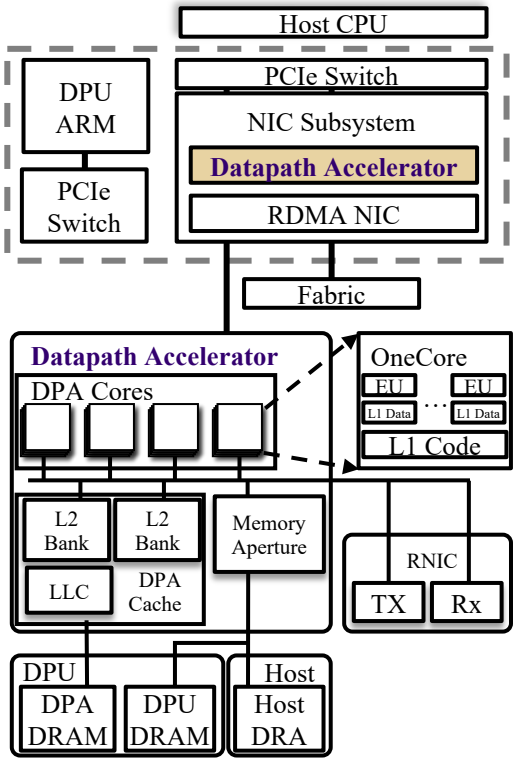
- Data Path Accelerator (DPA)
  - Embedded Processor
  - RISC-V 256 Hardware Threads
  - Real-Time OS (RTOS)
  - Low-Cost cache and memory subsystem



[Figures are from NVIDIA Public Information]

Implementation: SCR on BlueField-3 DPA

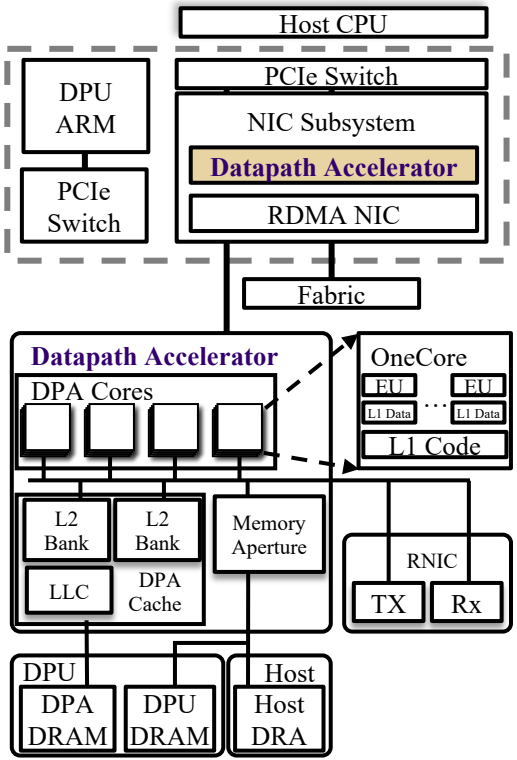
- Data Path Accelerator (DPA)
  - Embedded Processor
  - RISC-V 256 Hardware Threads
  - Real-Time OS (RTOS)
  - Low-Cost cache and memory subsystem
- Enhance Efficiency
  - Run-to-completion
  - Improve cache locality
  - Overcome RTOS constraints



[Figures are from NVIDIA Public Information]

Implementation: SCR on BlueField-3 DPA

- Data Path Accelerator (DPA)
  - Embedded Processor
  - RISC-V 256 Hardware Threads
  - Real-Time OS (RTOS)
  - Low-Cost cache and memory subsystem
- Enhance Efficiency
  - Run-to-completion
  - Improve cache locality
  - Overcome RTOS constraints
- See paper for detailed characterization on new hardware!



[Figures are from NVIDIA Public Information]

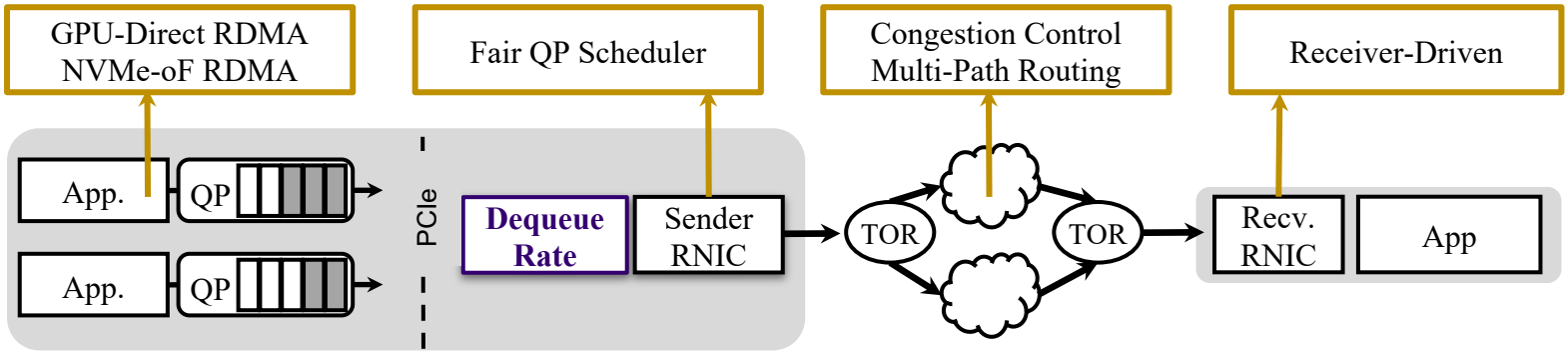
Use Cases: Spectrum of New Customizations

Use Cases: Spectrum of New Customizations

- Different Domains: Host/Fabric/Peer
- Different Workloads: NVMe-oF and GPU-Direct

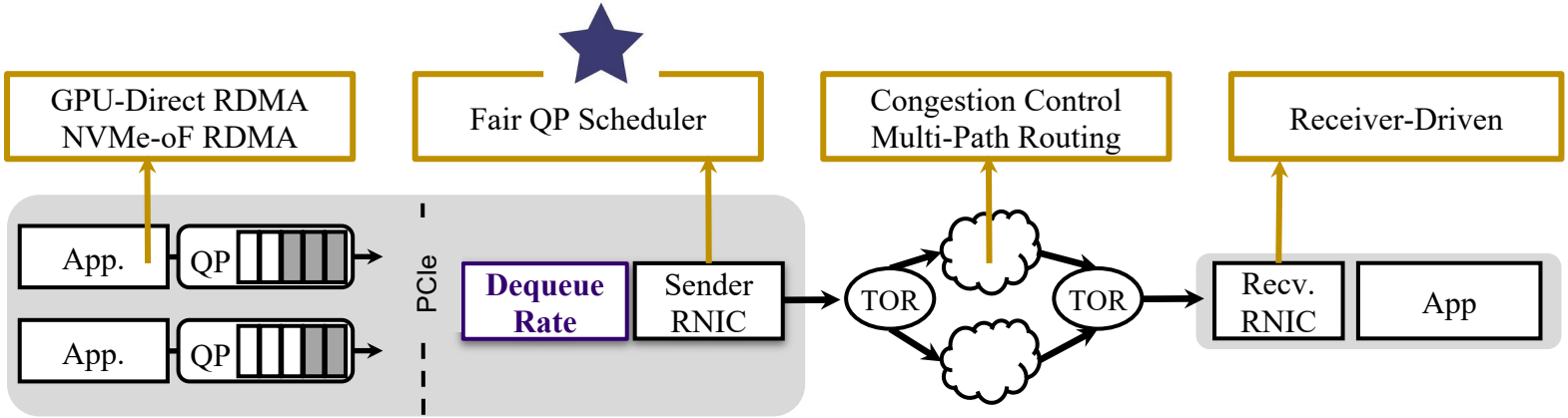
Use Cases: Spectrum of New Customizations

- Different Domains: Host/Fabric/Peer
- Different Workloads: NVMe-oF and GPU-Direct



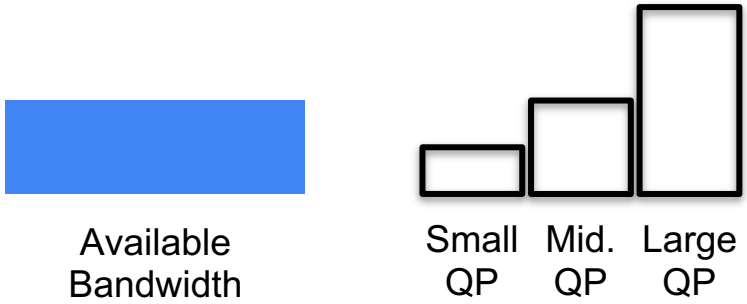
Use Cases: Spectrum of New Customizations

- Different Domains: Host/Fabric/Peer
- Different Workloads: NVMe-oF and GPU-Direct



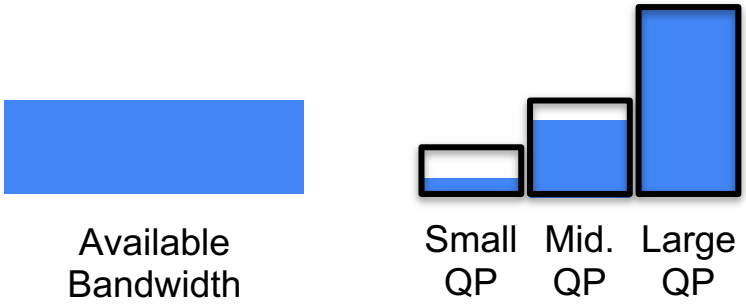
Use Case #1: Fair QP Scheduler

- Bandwidth Allocation Problem
  - Multi QPs: small/middle/large message sizes
  - Metrics: Fairness and Utilization
  - SCR can express three example algorithms



Use Case #1: Fair QP Scheduler

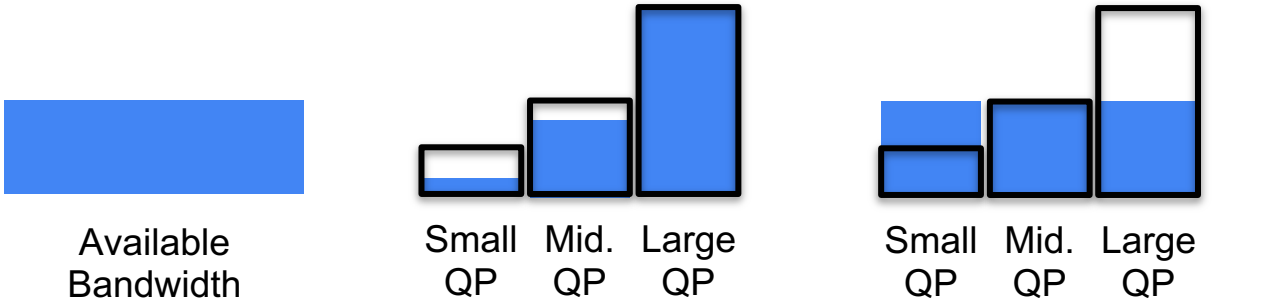
- Bandwidth Allocation Problem
  - Multi QPs: small/middle/large message sizes
  - Metrics: Fairness and Utilization
  - SCR can express three example algorithms



Default RNIC
Unfair for Small QP

Use Case #1: Fair QP Scheduler

- Bandwidth Allocation Problem
  - Multi QPs: small/middle/large message sizes
  - Metrics: Fairness and Utilization
  - SCR can express three example algorithms



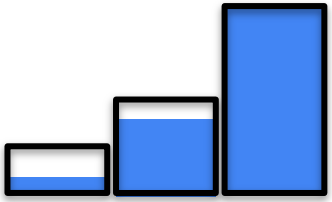
Default RNIC
Unfair for Small QP

Static Allocation
Small QP wastes BW

Use Case #1: Fair QP Scheduler

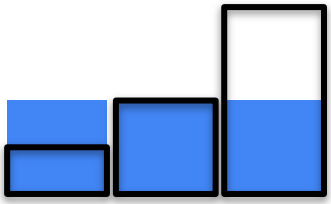
- Bandwidth Allocation Problem
  - Multi QPs: small/middle/large message sizes
  - Metrics: Fairness and Utilization
  - SCR can express three example algorithms

Available Bandwidth



Small QP | Mid. QP | Large QP

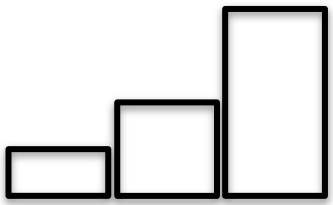
Default RNIC
Unfair for Small QP



Small QP | Mid. QP | Large QP

Static Allocation
Small QP wastes BW

Remaining BW



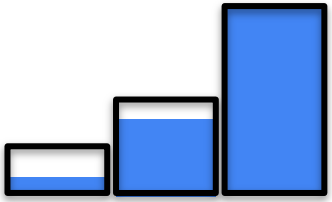
Small QP | Mid. QP | Large QP

Water Filling
Fair and High Util.

Use Case #1: Fair QP Scheduler

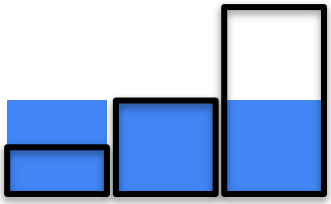
- Bandwidth Allocation Problem
  - Multi QPs: small/middle/large message sizes
  - Metrics: Fairness and Utilization
  - SCR can express three example algorithms

Available Bandwidth



Small QP | Mid. QP | Large QP

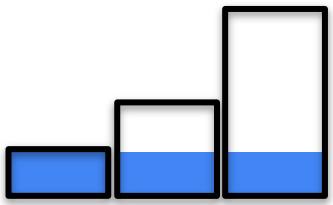
Default RNIC
Unfair for Small QP



Small QP | Mid. QP | Large QP

Static Allocation
Small QP wastes BW

Remaining BW



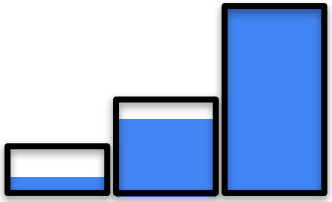
Small QP | Mid. QP | Large QP

Water Filling
Fair and High Util.

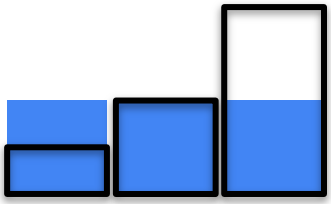
Use Case #1: Fair QP Scheduler

- Bandwidth Allocation Problem
  - Multi QPs: small/middle/large message sizes
  - Metrics: Fairness and Utilization
  - SCR can express three example algorithms

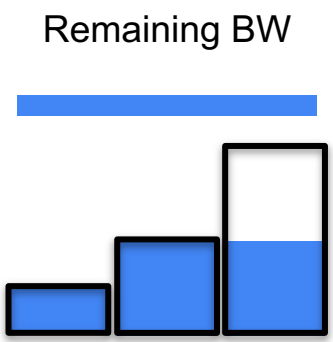
Available Bandwidth



Default RNIC
Unfair for Small QP



Static Allocation
Small QP wastes BW

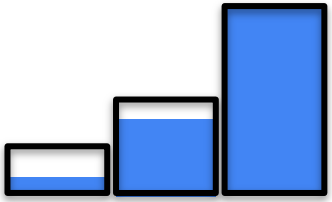


Water Filling
Fair and High Util.

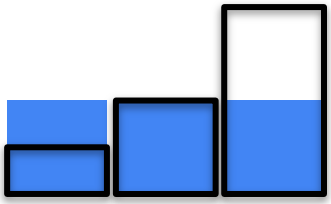
Use Case #1: Fair QP Scheduler

- Bandwidth Allocation Problem
  - Multi QPs: small/middle/large message sizes
  - Metrics: Fairness and Utilization
  - SCR can express three example algorithms

Available Bandwidth

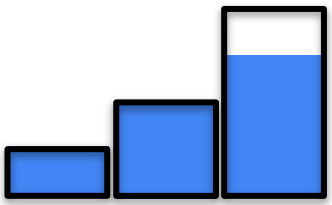


Default RNIC
Unfair for Small QP



Static Allocation
Small QP wastes BW

Remaining BW Zero



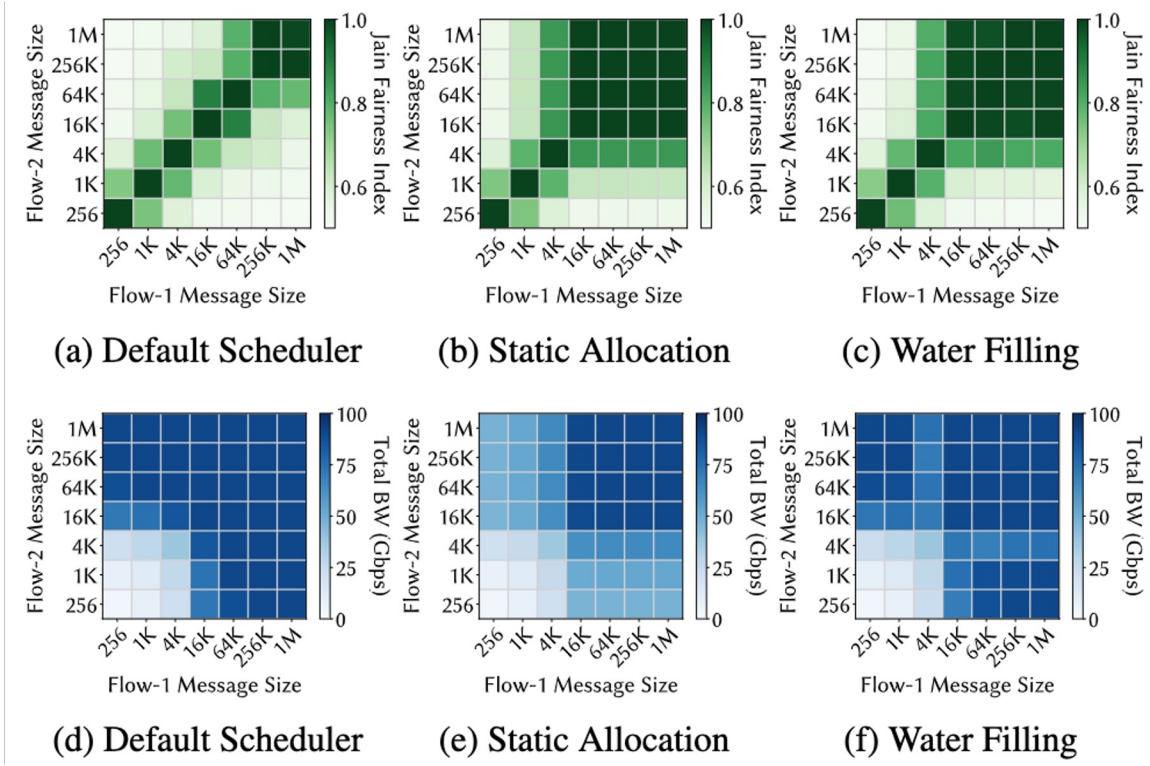
Water Filling
Fair and High Util.

Use Case #1: Fair QP Scheduler

- E2E: Two QP Compete

Use Case #1: Fair QP Scheduler

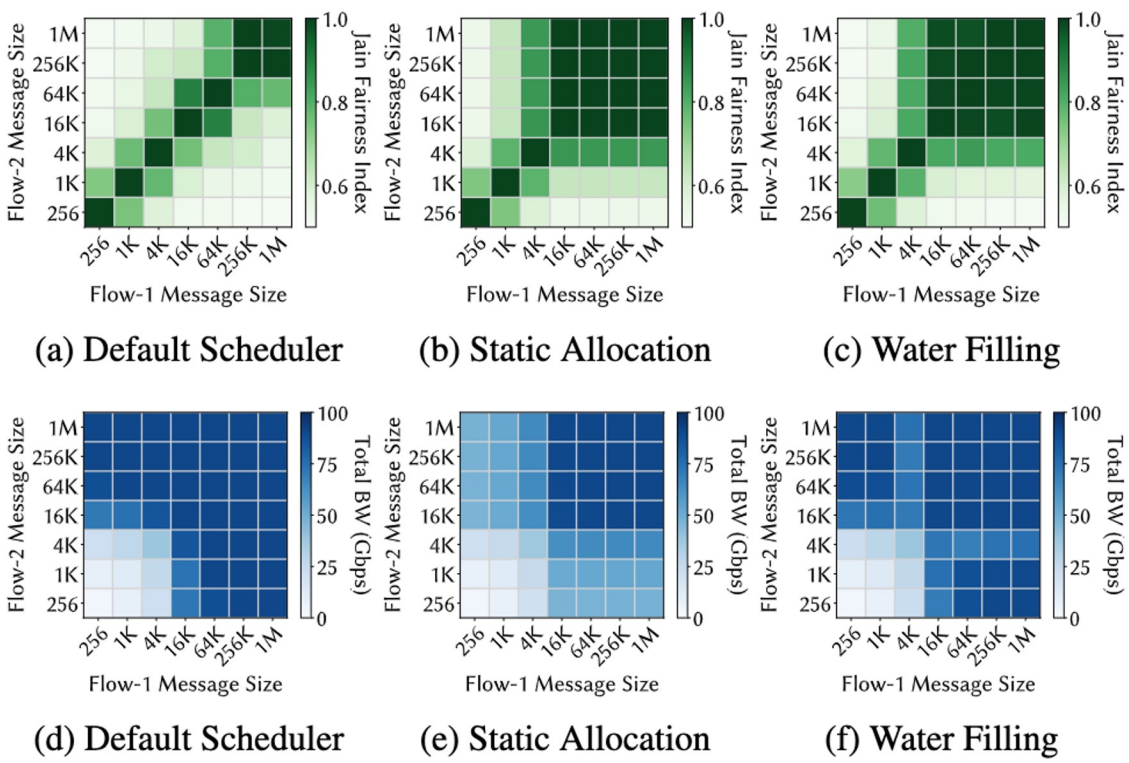
- E2E: Two QP Compete
- Deeper is more fair
- Deeper is higher util.



Use Case #1: Fair QP Scheduler

- E2E: Two QP Compete
- Deeper is more fair
- Deeper is higher util.

[Takeway]
SCR enables better algorithms!



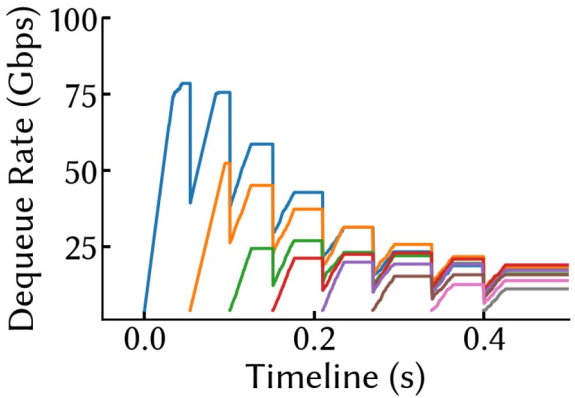
Use Case #1: Further Customize Water Filling

Use Case #1: Further Customize Water Filling

- How to reset existing flows when a new flow joins?

Use Case #1: Further Customize Water Filling

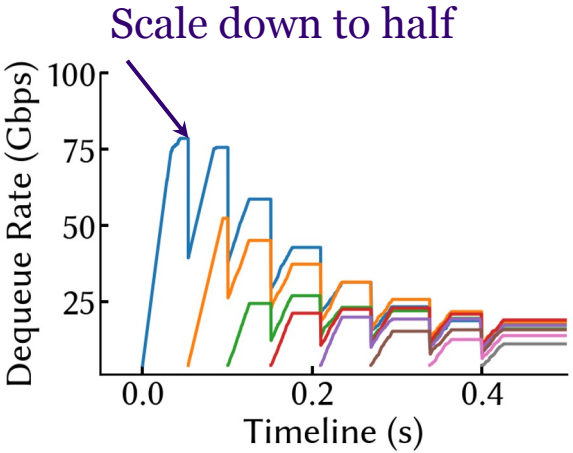
- How to reset existing flows when a new flow joins?



Heuristic Stateless Strategy

Use Case #1: Further Customize Water Filling

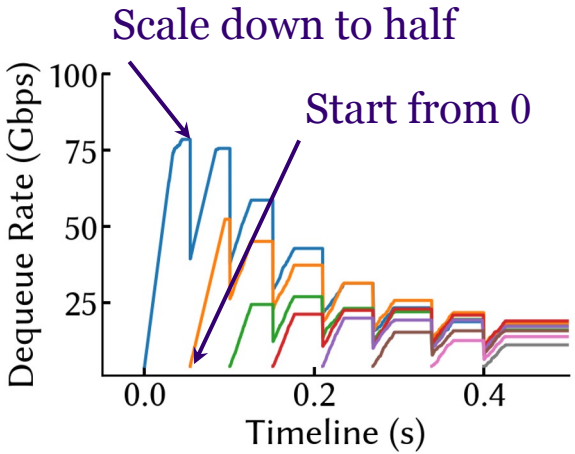
- How to reset existing flows when a new flow joins?



Heuristic Stateless Strategy

Use Case #1: Further Customize Water Filling

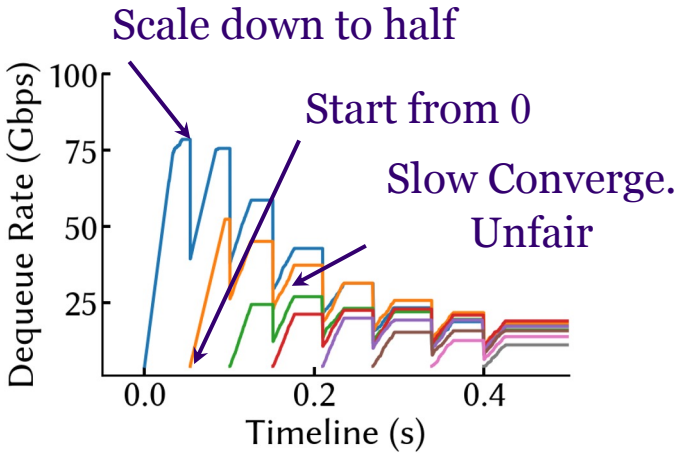
- How to reset existing flows when a new flow joins?



Heuristic Stateless Strategy

Use Case #1: Further Customize Water Filling

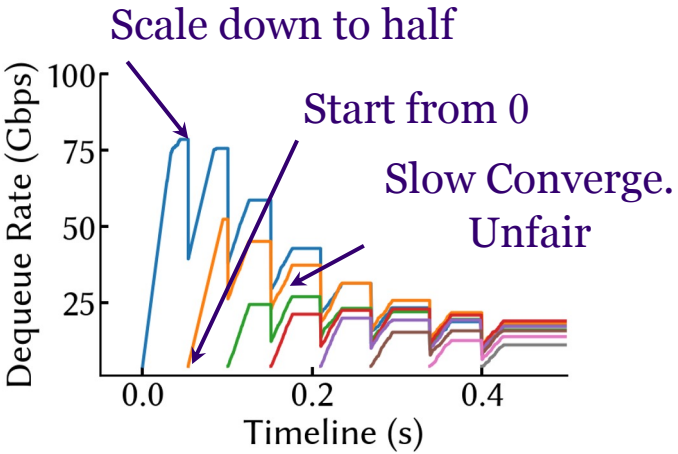
- How to reset existing flows when a new flow joins?



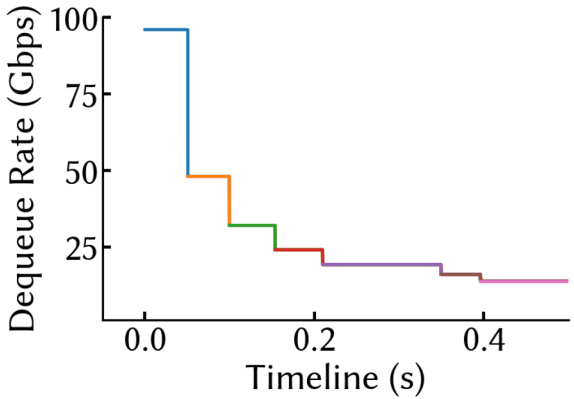
Heuristic Stateless Strategy

Use Case #1: Further Customize Water Filling

- How to reset existing flows when a new flow joins?



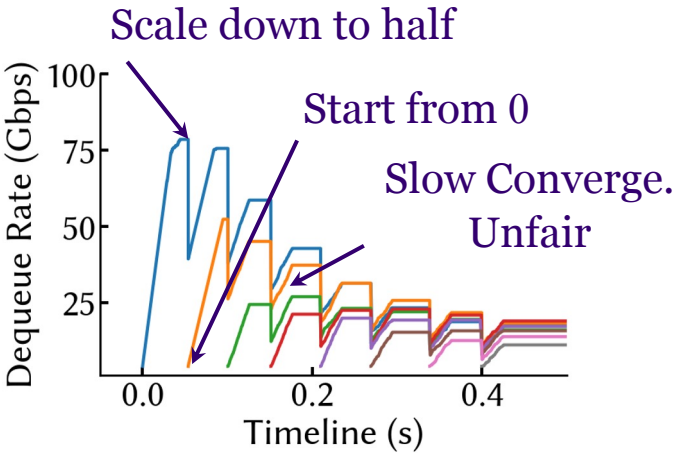
Heuristic Stateless Strategy



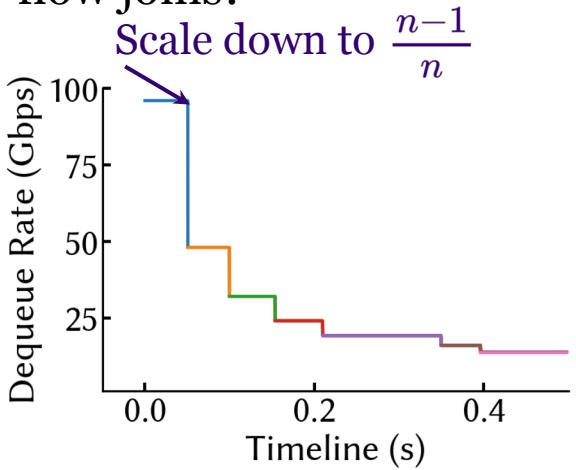
With SCR tracking states (*e.g.*, flow_num *n*)

Use Case #1: Further Customize Water Filling

- How to reset existing flows when a new flow joins?



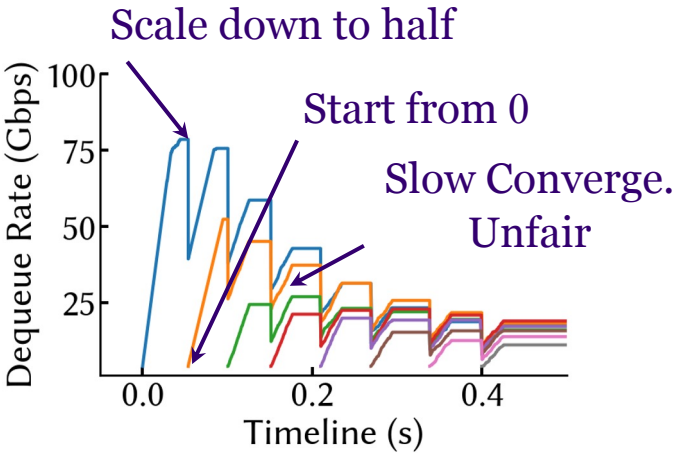
Heuristic Stateless Strategy



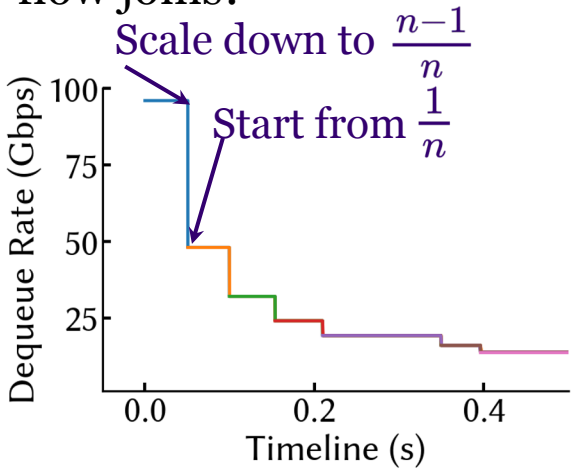
With SCR tracking states (*e.g.*, flow_num n)

Use Case #1: Further Customize Water Filling

- How to reset existing flows when a new flow joins?



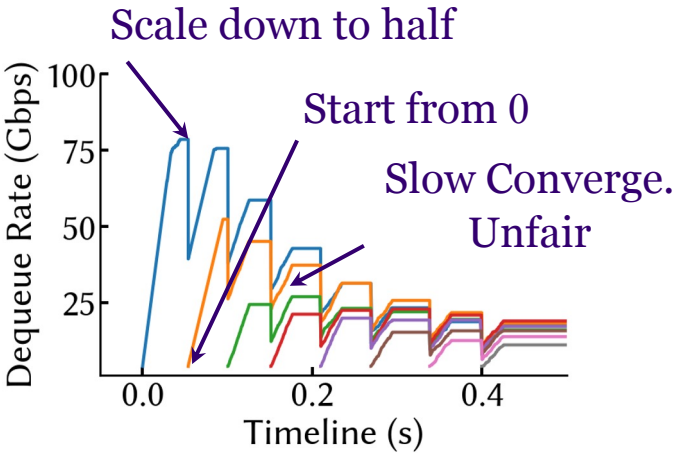
Heuristic Stateless Strategy



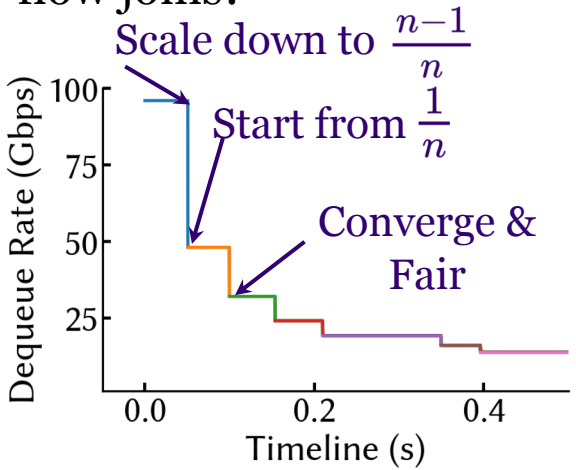
With SCR tracking states (*e.g.*, flow_num n)

Use Case #1: Further Customize Water Filling

- How to reset existing flows when a new flow joins?



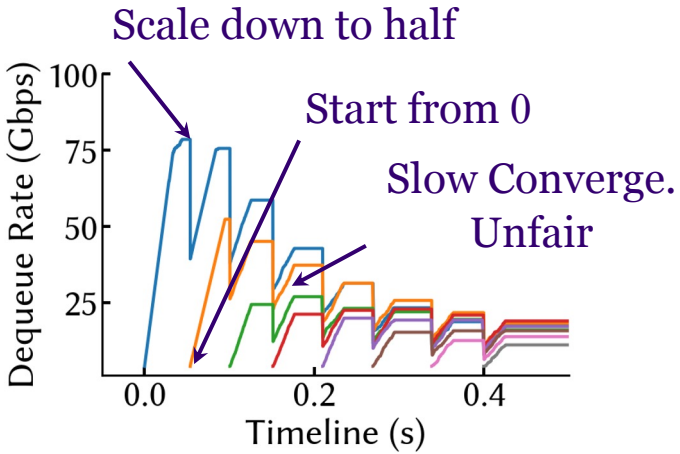
Heuristic Stateless Strategy



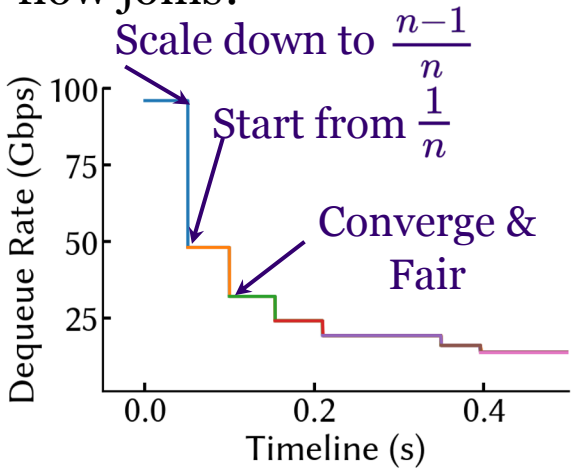
With SCR tracking states (*e.g.*, flow_num n)

Use Case #1: Further Customize Water Filling

- How to reset existing flows when a new flow joins?



Heuristic Stateless Strategy



With SCR tracking states (*e.g.*, flow_num n)

[Takeway] SCR enables stateful customizability!

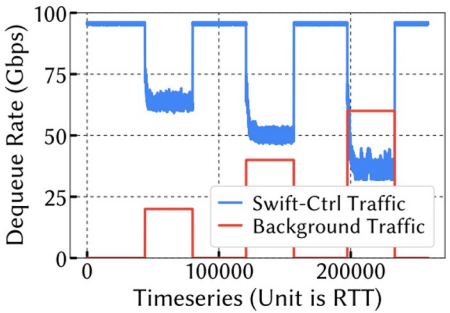
More Use Cases

More Use Cases

- Case#1: GPU-Direct/NVMe-oF workloads

More Use Cases

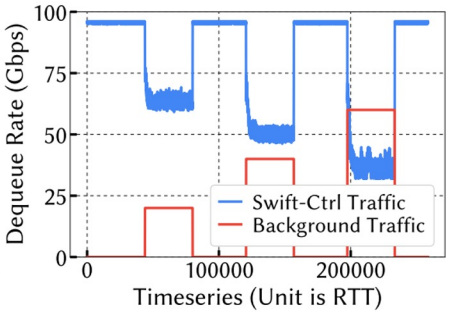
- Case#1: GPU-Direct/NVMe-oF workloads
- Case#2: RTT-based congestion control (Swift Algorithm)



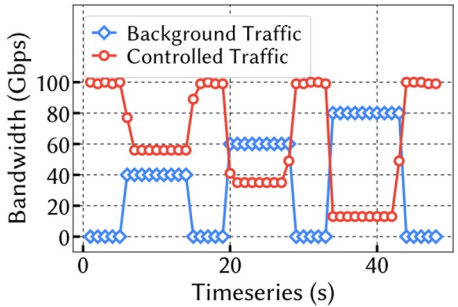
[Case#2]

More Use Cases

- Case#1: GPU-Direct/NVMe-oF workloads
- Case#2: RTT-based congestion control (Swift Algorithm)
- Case#3: Receiver-driven flow control to accurately allocate bandwidth



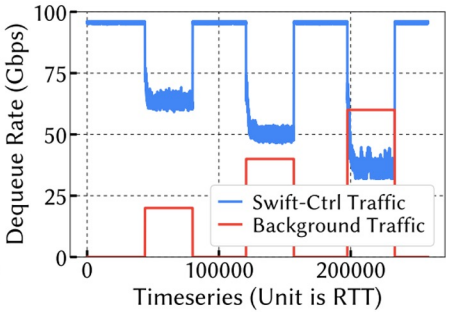
[Case#2]



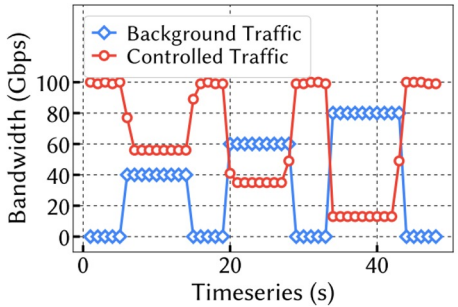
[Case#3]

More Use Cases

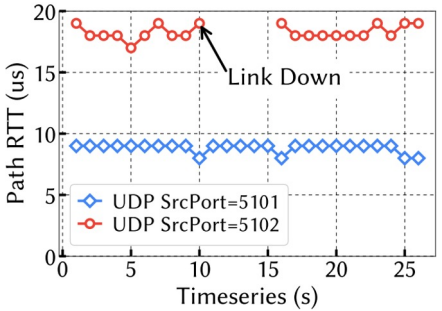
- Case#1: GPU-Direct/NVMe-oF workloads
- Case#2: RTT-based congestion control (Swift Algorithm)
- Case#3: Receiver-driven flow control to accurately allocate bandwidth
- Case#4: Multipath monitoring to help make routing decision



[Case#2]



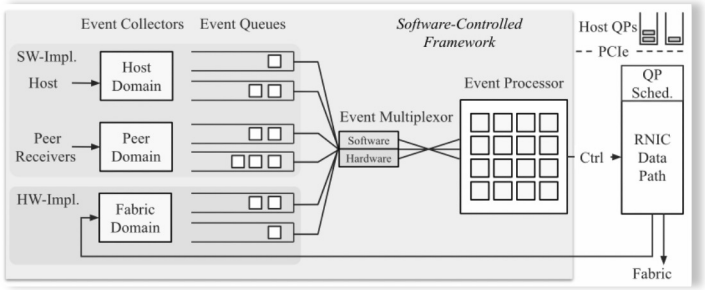
[Case#3]

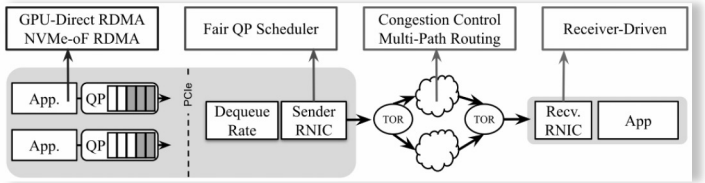


[Case#4]

Conclusion

- Whiteboxing RDMA Methodology
  - SW Ctrl; HW Data
- Software-Controlled RDMA (SCR)
  - Flexibility
  - Packet-Granularity
  - Scalability
- Land New Customizations
  - Multi-Tenant Isolation
  - Congestion Control
  - Routing
  - Flow Control





UNIVERSITY of WASHINGTON

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON