RDMA at Hyperscale: Experience and Future Directions

Wei Bai

Microsoft Research Redmond

TL;DR

We use RDMA for storage traffic

- RoCEv2 over commodity Ethernet/IP network
- Enabled for speeds > 100gbps, over distances up to 100km
- ~ 70% Azure traffic (bytes and packets)

Made possible by many innovations

- DCQCN
- RDMA Estats
- PFC watchdog
- Carefully tuned libraries… and much more

Many opportunities ahead!

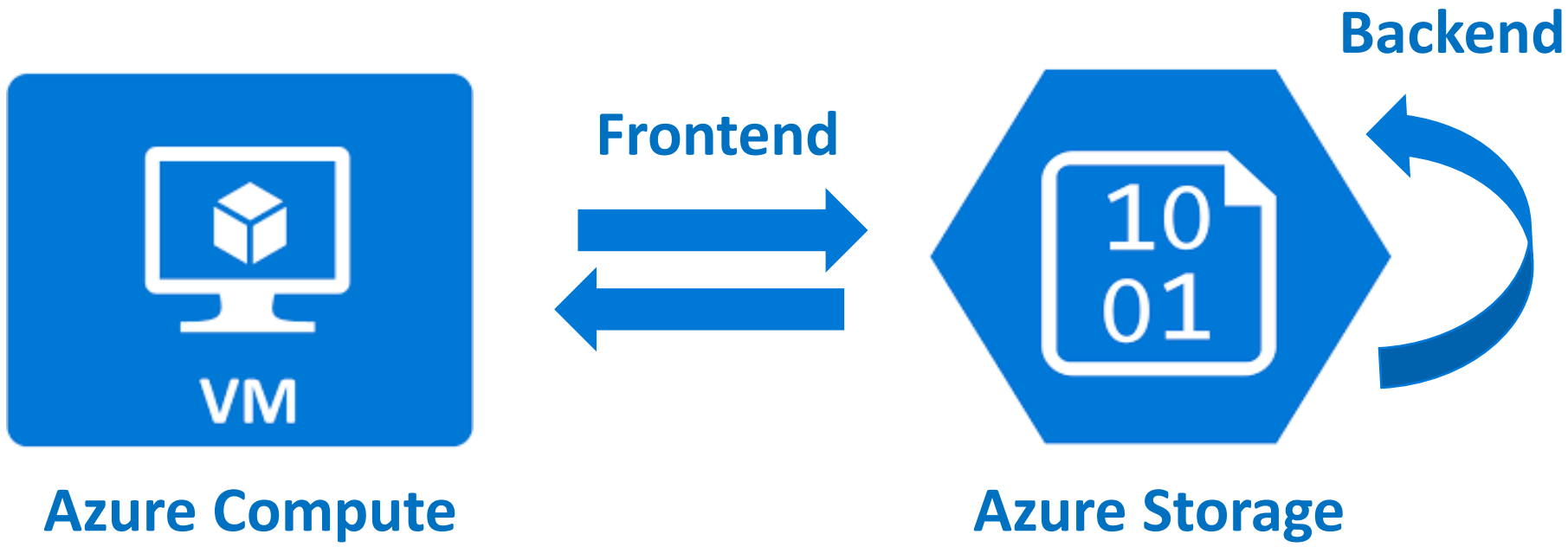
Azure storage overview

Disaggregated storage in Azure

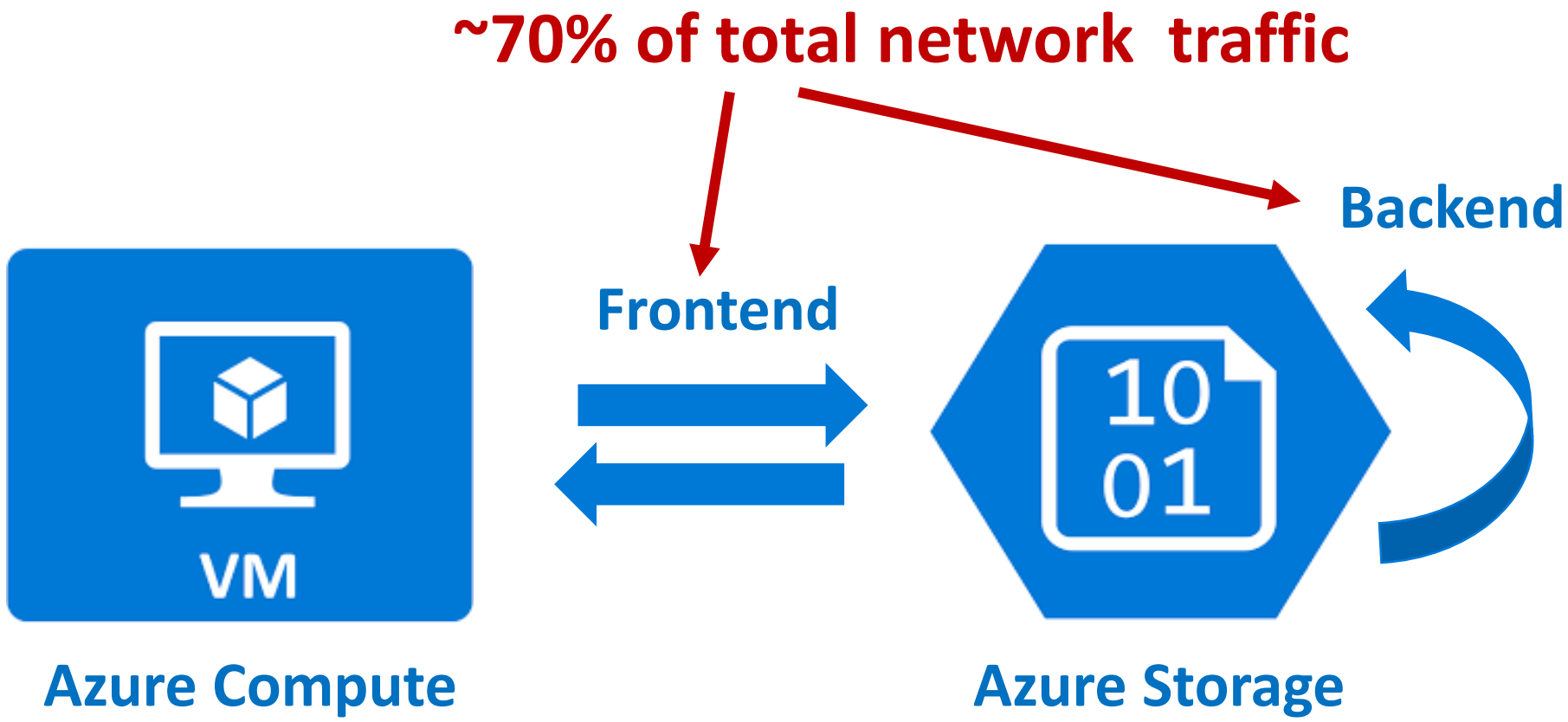
Azure Compute

Azure Storage

Disaggregated storage in Azure



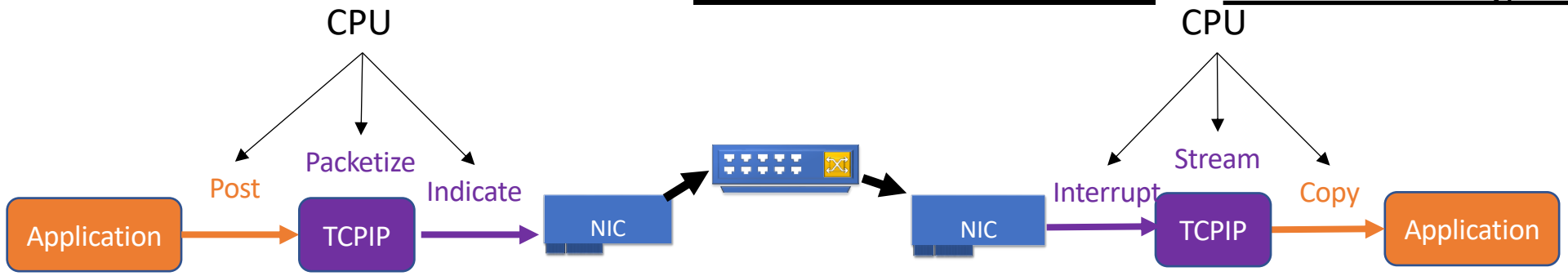
Disaggregated storage in Azure



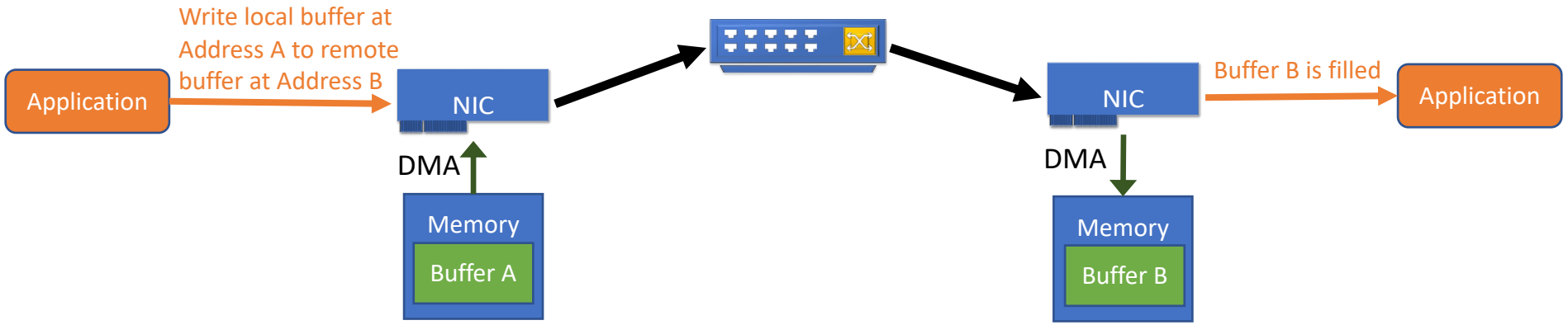
Majority of network traffic generated by VMs is related to storage I/O

Use TCP or RDMA?

Application: “Write the block of data at local disk w, address x to remote disk y, address z”



TCP: Waste of CPU, high latency due to CPU processing



RDMA: NIC handles all transfers via DMA

RDMA Benefits

For Azure

Core savings

Sell freed-up CPU cores in compute

Buy cheaper servers in storage

For Customers

Lower IO latency and jitter

Two main RDMA networking techniques

InfiniBand (IB)

- The original RDMA technology
- Custom stack (L1-L7), incompatible with Ethernet/IP
- Hence expensive
- Scaling problems

RoCE (v2)

- RDMA over converged Ethernet
- Compatible with Ethernet/IP
- Hence cheap
- Can scale to DC and beyond

✓ **Our choice**

There is also iWarp


HONDA
Honda of Kirkland
KIRKLAND

TCP


Marlboro
Marlboro
HONDA
Marlboro
HONDA
12
Shell

RDMA

Problems …

- Tailor storage code for RDMA? (not socket programming!)
- How do we monitor performance? (not TCP!)
- What network configuration do we need?
- Congestion control?
- And many others ….

Solutions …

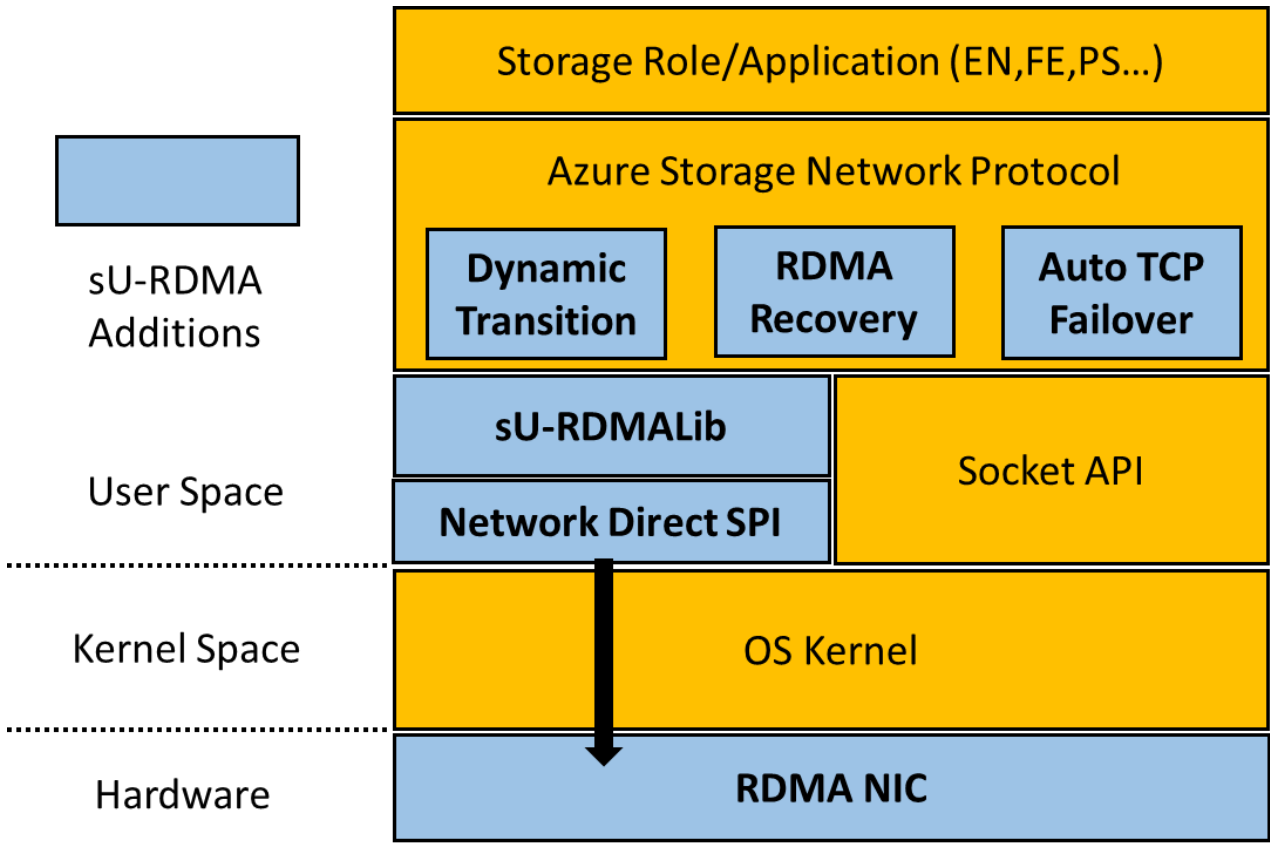
- Custom libraries: sU-RDMA and sK-RDMA
- Host monitoring: RDMA Estats
- SONiC, PFC WD, careful buffer tuning …
- DCQCN
- And much more

Tailor storage code for RDMA?

Two libraries: user-space, and kernel-space (extends SMB-direct)

RDMA operations optimizations, plus TCP fallback if needed

sU-RDMA* for storage backend

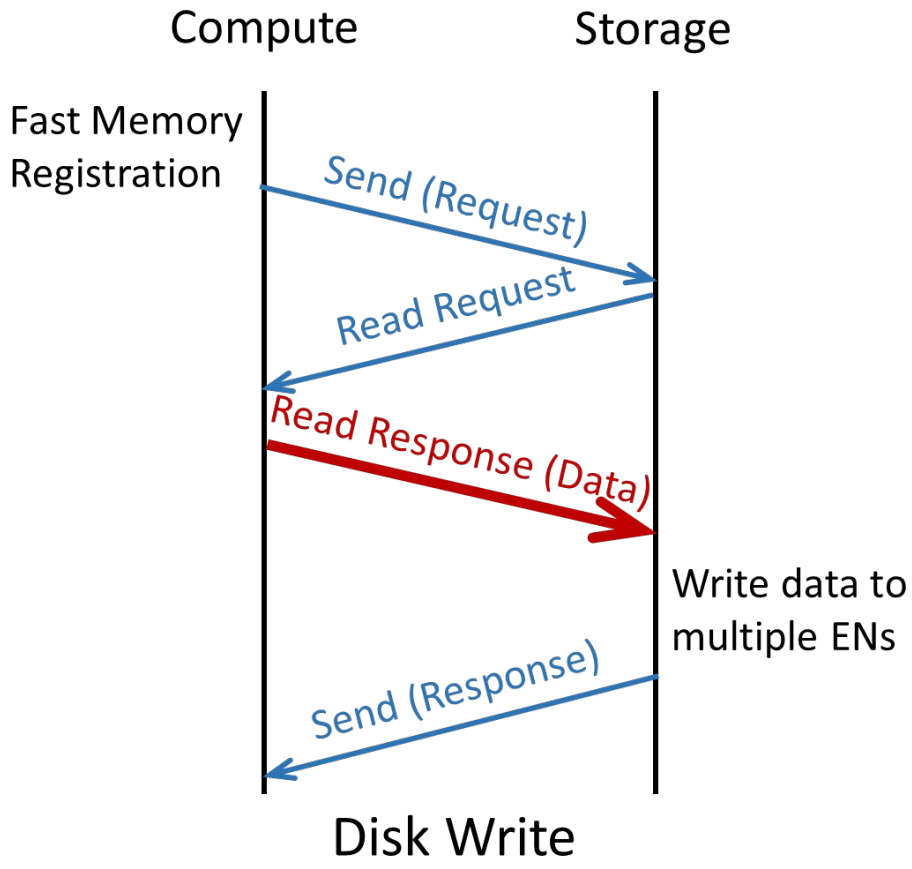


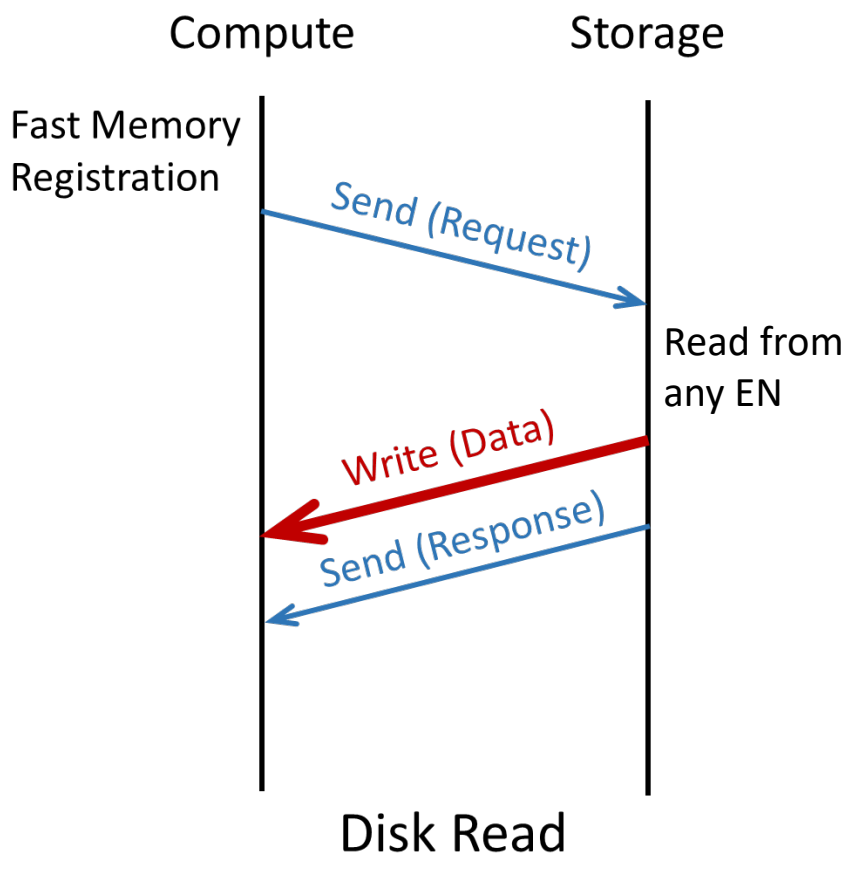
- Translate Socket APIs to RDMA verbs
  - Three transfer modes based on message sizes
- TCP and RDMA transitions
  - If RDMA fails, switch to TCP
  - After some time try reverting back to RDMA.
- Chunking: static window

*__sU__-RDMA = __s__torage __u__ser space RDMA

sK-RDMA* for storage frontend

- Extend SMB Direct. Run RDMA in kernel space.





*__sK__-RDMA = __s__torage __k__ernel space RDMA

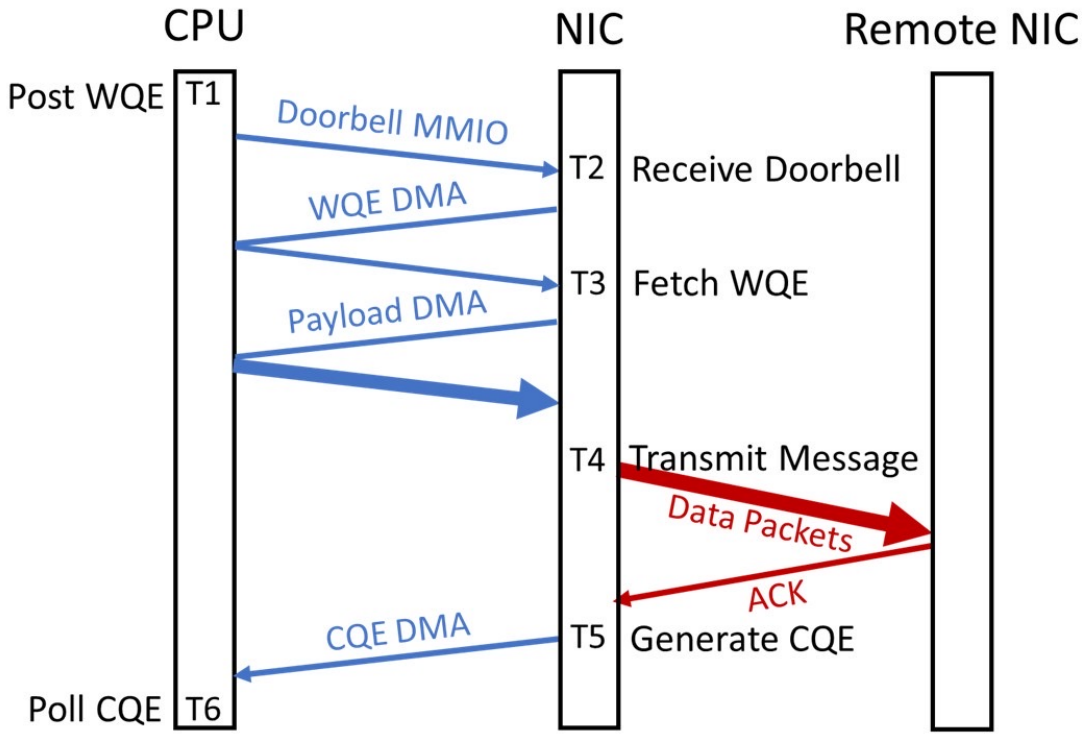
How do we monitor RDMA traffic?

On host: RDMA Estats and NIC counters

On routers: standard router telemetry (PFCs sent and received, per-queue traffic and drop counters, special PFC WD summary)

RDMA Estats

- Akin to TCP Estats
- Provides fine-grained latency breakdown for each RDMA operation
- Data aggregated over 1 minute



T6 - T1: client latency

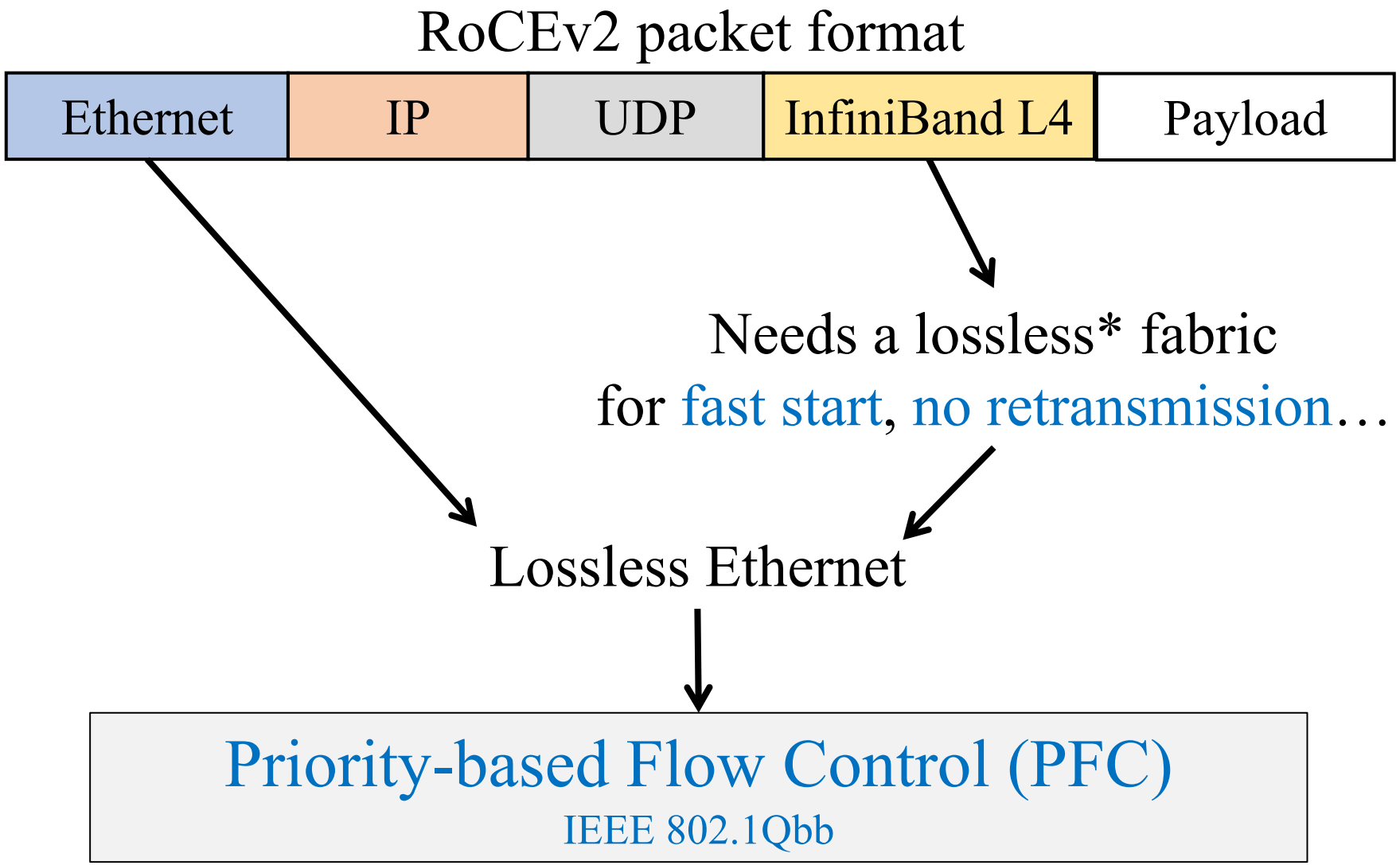
T5 - T1: hardware latency

Must sync CPU and NIC clocks!

What special network configuration do we need and why?

Need lossless Ethernet

Sample config with SONiC

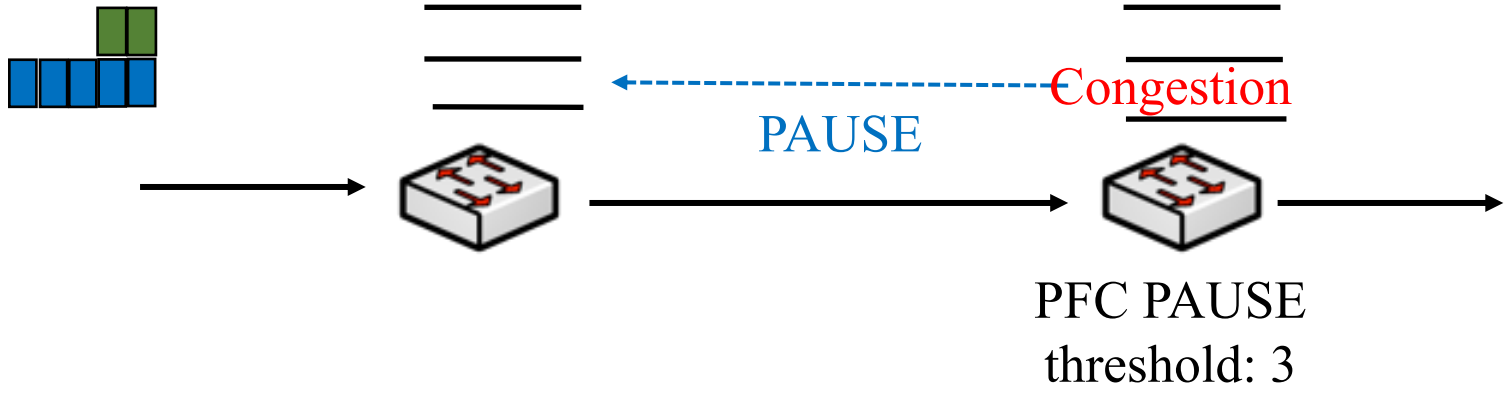


*Lossless: no congestion packet drop

Why lossless?

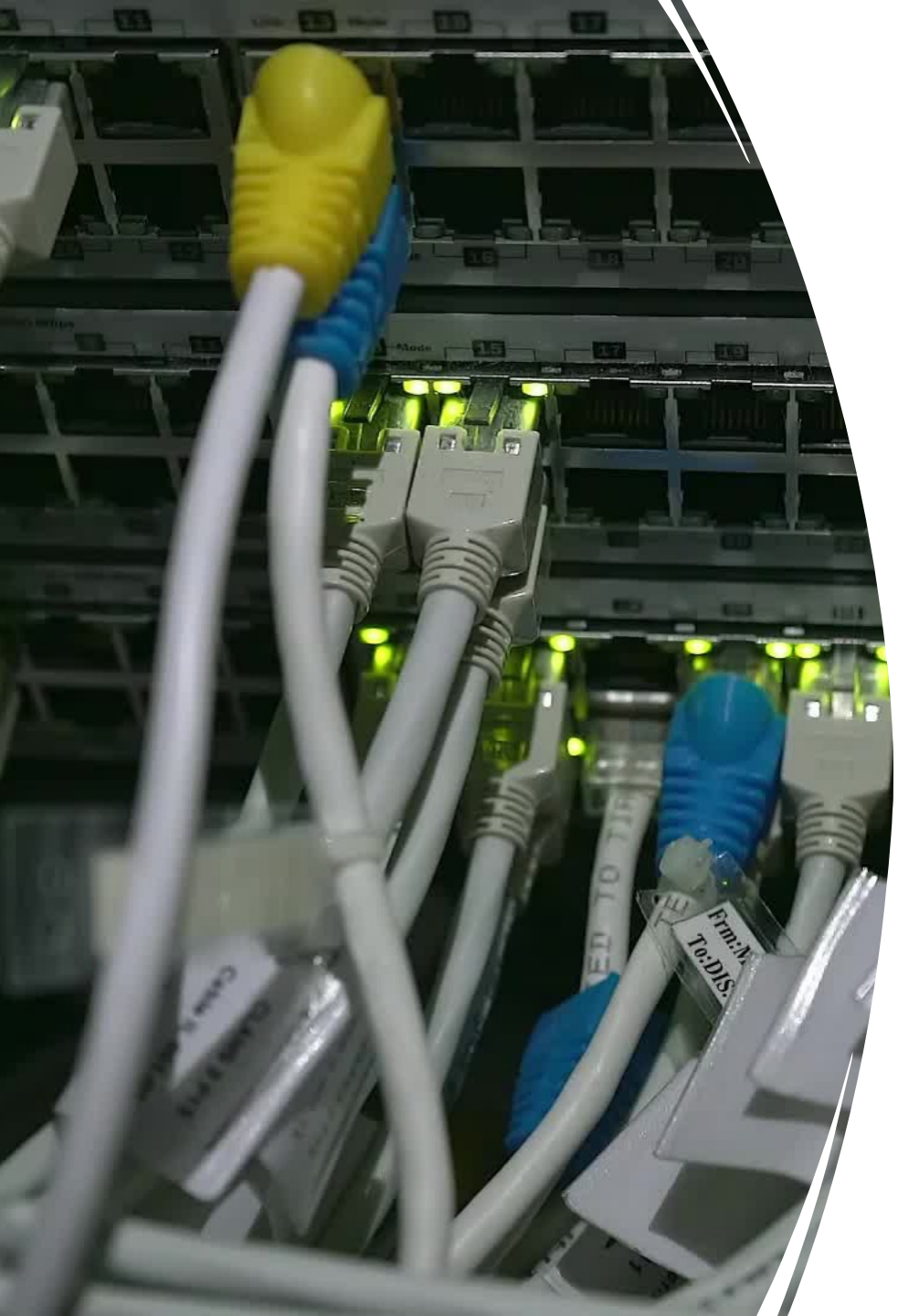
- Fast start
  - Reduces latency in common cases
- No retransmission due to congestion drop
  - Reduce tail latency
  - Simplifies transport protocol
- ACK consolidation
  - Improve efficiency
- Older NICs were inefficient at retransmission

Priority-based Flow Control (PFC)



Pause upstream neighbor when queue over XOFF limit

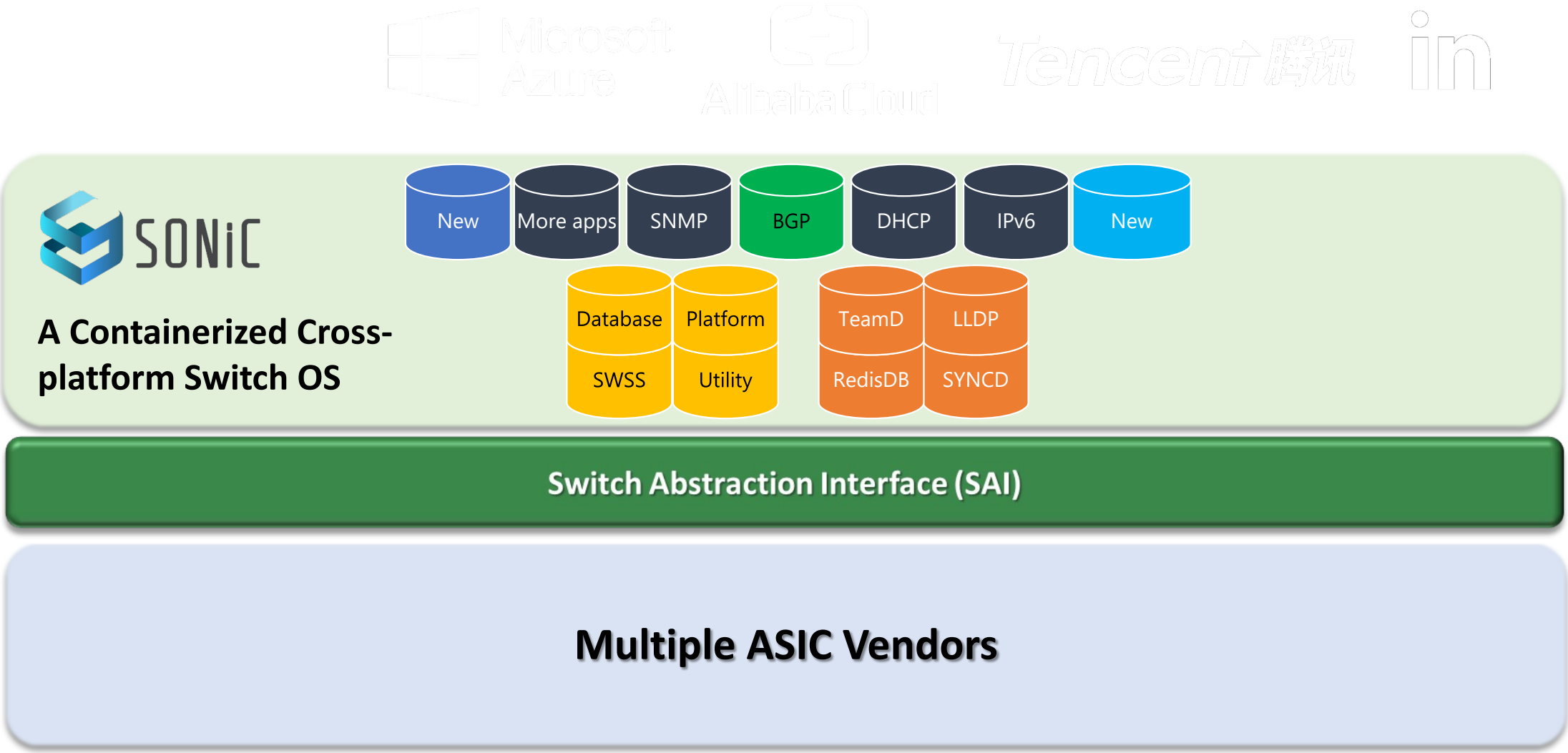
Ensures no congestion drop



PFC and associated buffer configuration is non-trivial

- Headroom – which depends on cable length
- Joint ECN-PFC config – ECN must trigger before PFC (more on this later)
- Configure PFC watchdog to guard against hardware faults
- .. And all this on heterogenous, multi-generation routers

SONiC: Software for Open Networking in the Cloud



SONiC: Software for Open Networking in the Cloud

Cloud Provider using SONiC

Open Software

+

Standardized Hardware Interfaces

RDMA features in SONiC

- ECN
- Shared buffer
- PFC
- PFC headroom pool
- **PFC watchdog**
- Various counters

```
"BUFFER_POOL": {
    "ingress_pool": {
        "size": "18000000",
        "type": "ingress",
        "mode": "dynamic",
        "xoff": "6000000"
    },
    "egress_lossy_pool": {
        "size": "14000000",
        "type": "egress",
        "mode": "dynamic"
    },
    "egress_lossless_pool": {
        "size": "24000000",
        "type": "egress",
        "mode": "static"
    }
}

"BUFFER_PROFILE": {
    "pg_lossless_profile" : {
        "pool": "[BUFFER_POOL|ingress_pool]",
        "size": "1248",
        "dynamic_th": "-3",
        "xoff": "96928",
        "xon"    "1248",
        "xon_offset"    "2496"
    },
    "ingress_lossy_profile": {
        "pool": "[BUFFER_POOL|ingress_pool]",
        "size": "0",
        "static_th":"24000000"
    },
    "egress_lossless_profile": {
        "pool": "[BUFFER_POOL|egress_lossless_pool]",
        "size": "0",
        "static_th": "24000000"
    },
    "egress_lossy_profile": {
        "pool": "[BUFFER_POOL|egress_lossy_pool]",
        "size": "1664",
        "dynamic_th": "-1"
    }
}
```

A SONiC Buffer Template Example

If a queue has been in paused state for very long time, drop all its packets

Congestion control in heterogenous environment

DCQCN

Why do we need congestion control?

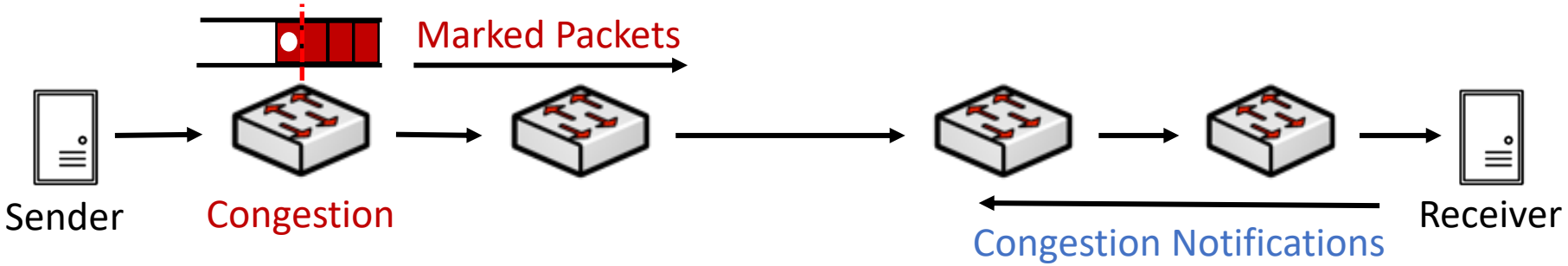
PFC operates on a per-port/priority basis, not per-flow.
It can spread congestion, is unfair, and can lead to deadlocks!

Zhu et al. [SIGCOMM'15], Guo et al. [SIGCOMM'16], Hu et al. [HotNets'17]

Solution

- Minimize PFC generation using per-flow E2E congestion control
- BUT keep PFC to allow fast start and lower tail latency
- In other words, use PFC as a last-resort

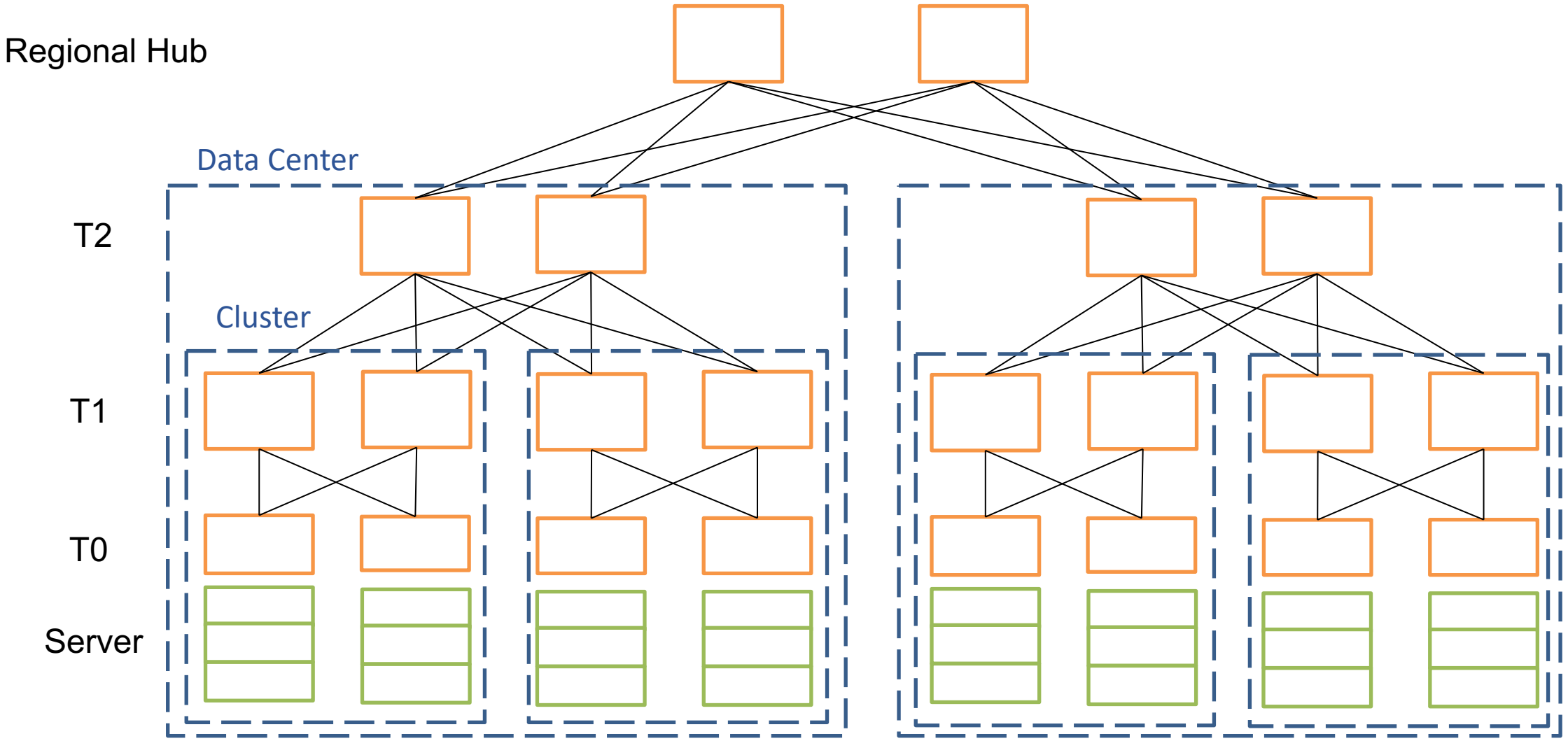
DCQCN



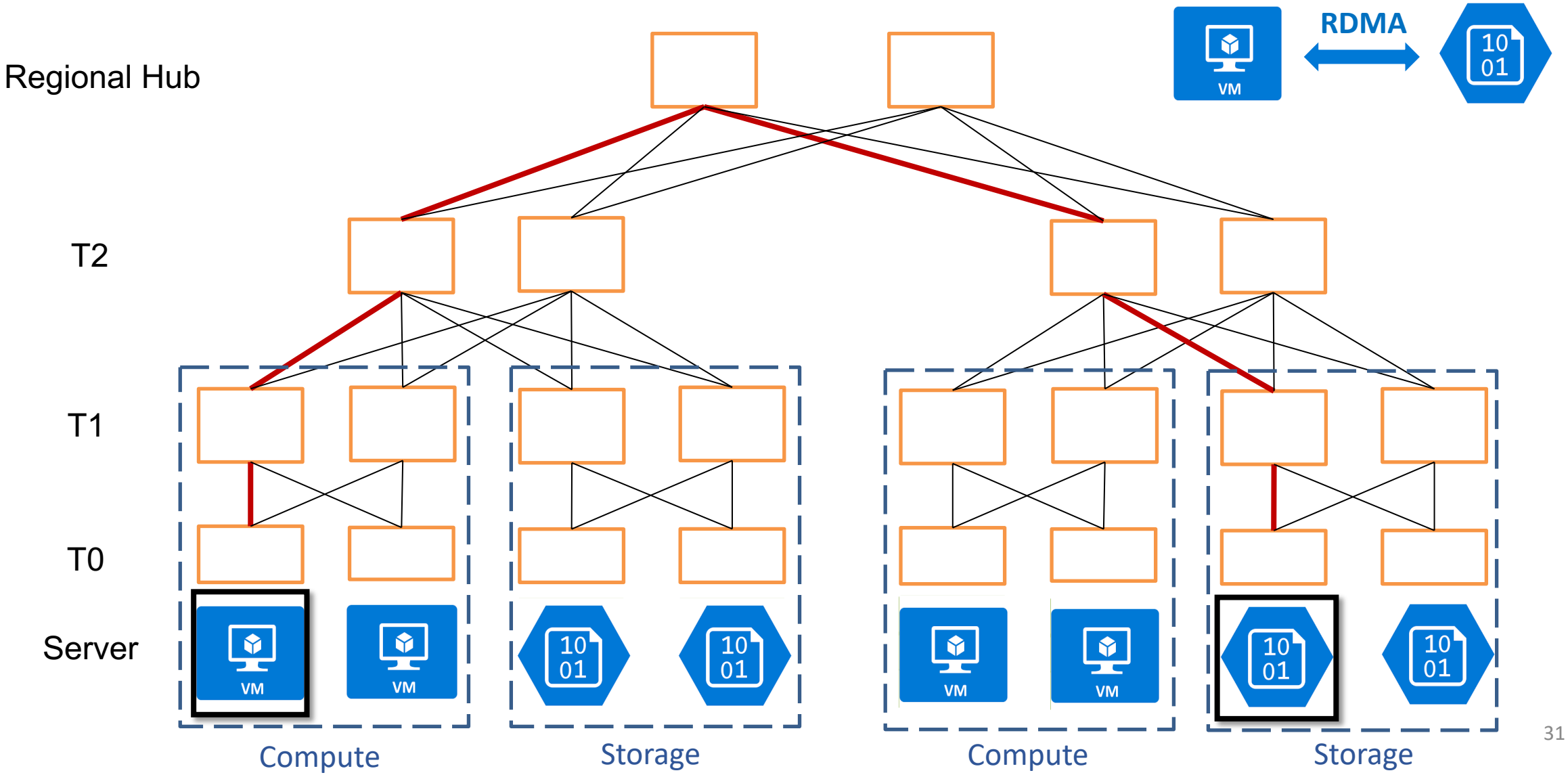
- Sender lowers rate when notified of congestion
- Rate based, RTT-oblivious congestion control
- ECN as congestion signal (better stability than delay)

Congestion control for large-scale RDMA deployments, SIGCOMM 2015

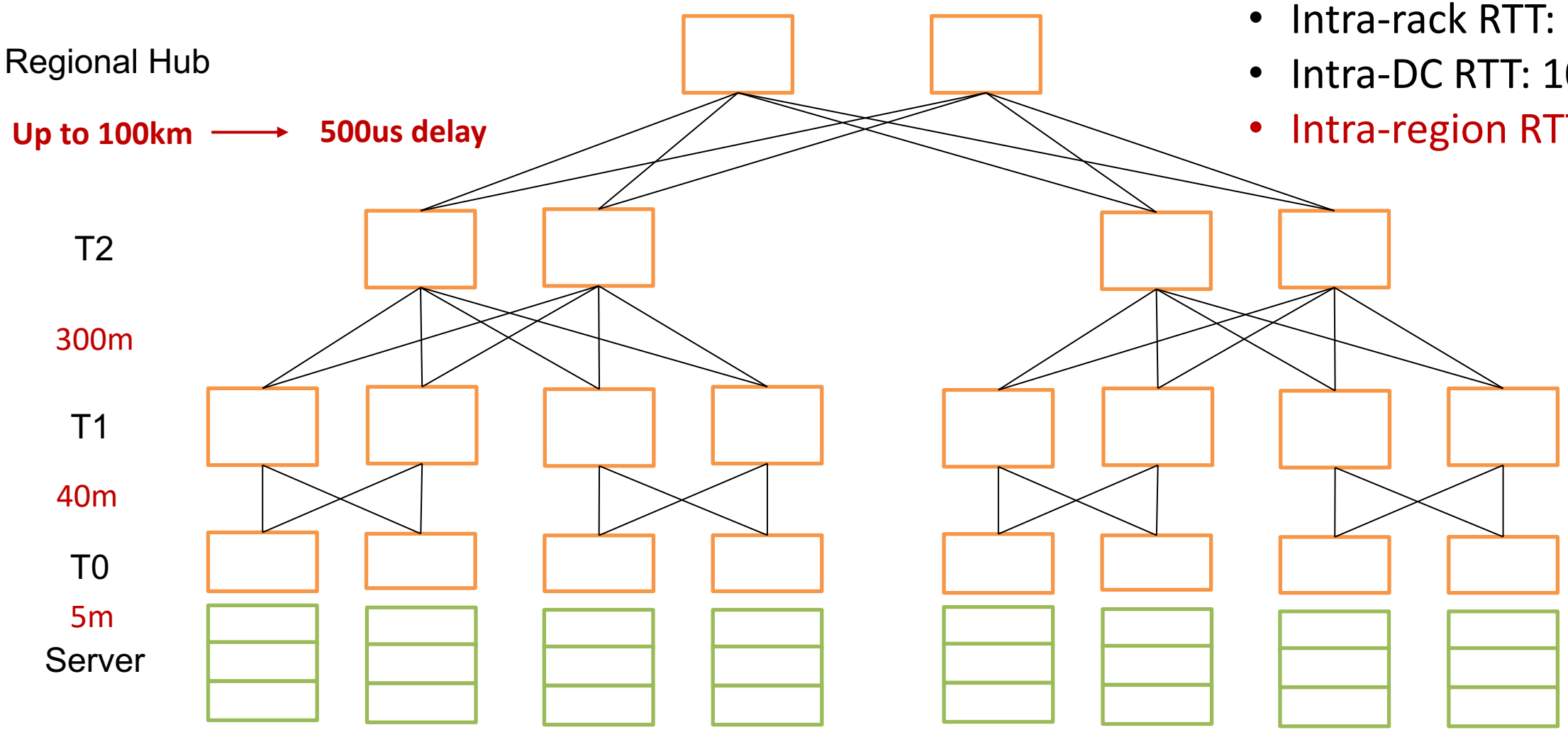
Network architecture of an Azure region



Azure needs intra-region RDMA



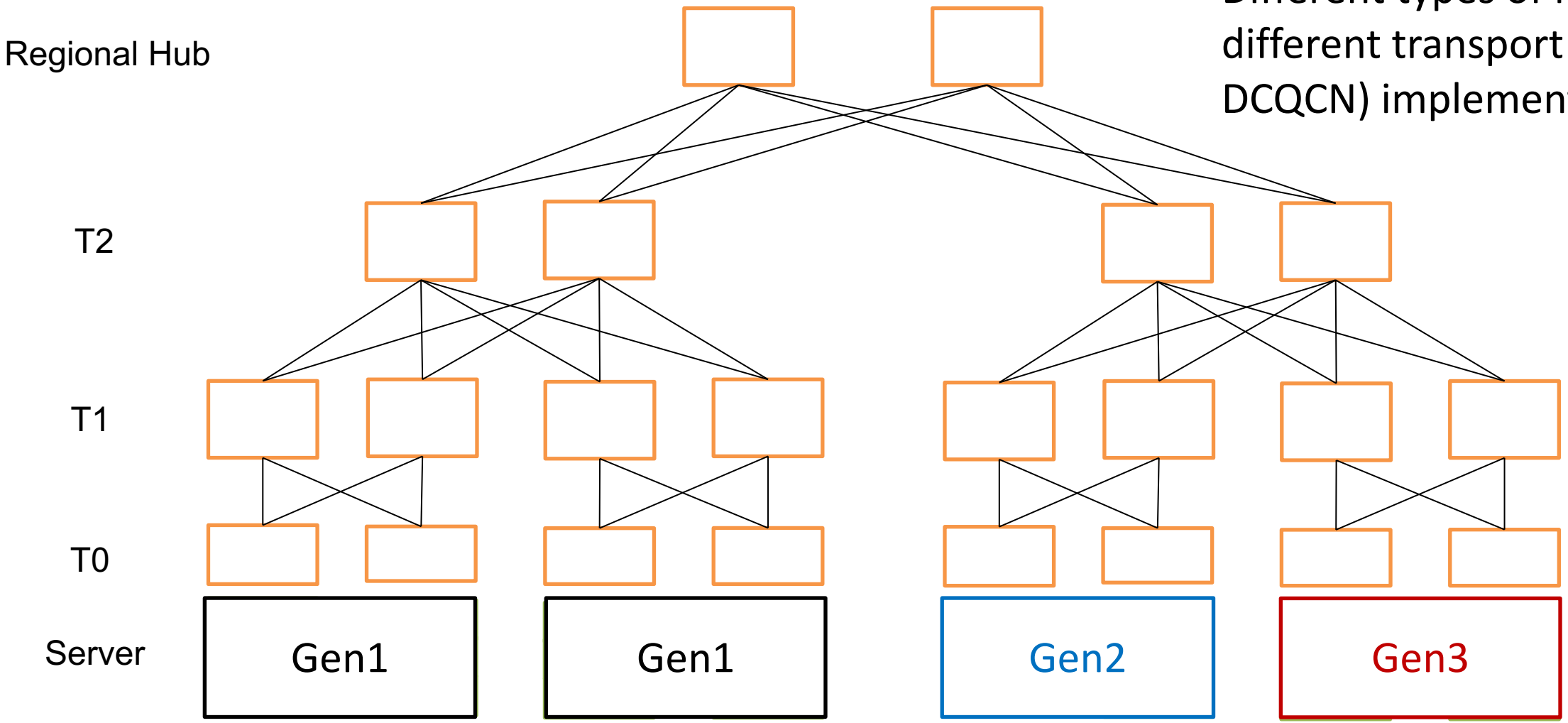
Large and variable latency end-to-end



- Intra-rack RTT: 2us
- Intra-DC RTT: 10s of us
- Intra-region RTT: 2ms

Heterogenous NICs

Different types of NICs have different transport (e.g., DCQCN) implementations

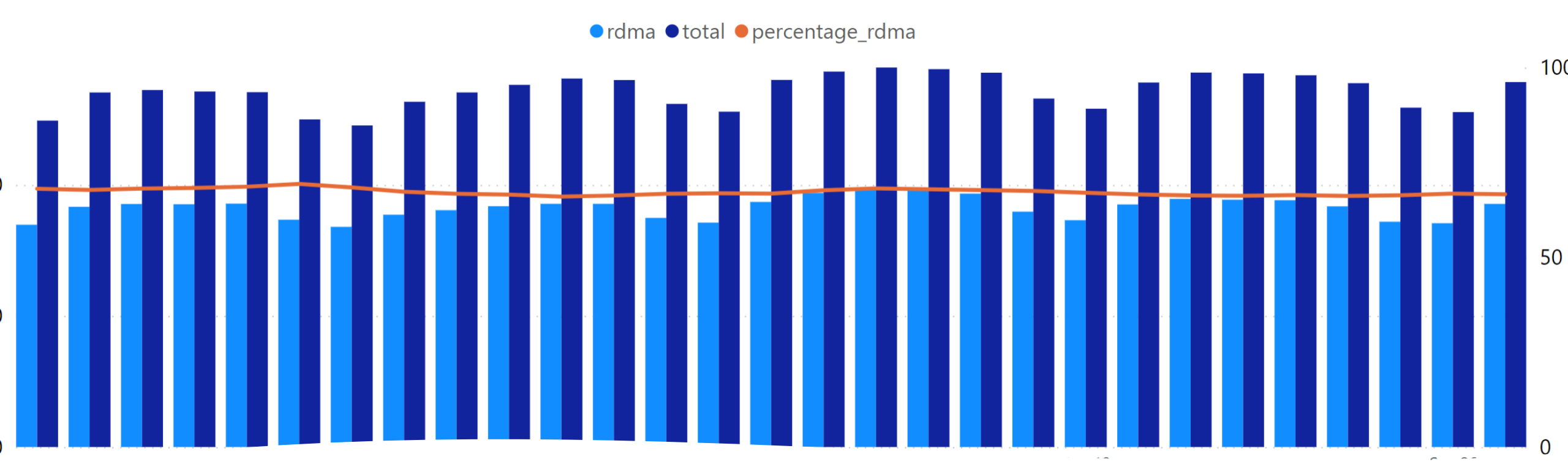


DCQCN in production

- Different DCQCN implementations on NIC **Gen1/2/3**
  - **Gen1** has DCQCN implemented in firmware (slow) and uses a different congestion notification mechanism.
  - Solution: modify firmware of **Gen2** and **Gen3** to make them behave like **Gen1**

- Careful DCQCN tuning for a wide range of RTTs
  - Interesting observation: DCQCN has near-perfect RTT fairness

- Careful buffer tuning for various switches
  - Ensure that ECN kicks in before PFC



100x100

100 Gbps over 100 km

- **$O(10^{18})$ bytes / day, $O(10^{9})$ IOs / day**
- **~70% of bytes. *TCP distant second.***
- **Significant perf improvements and COGS savings**
- **Possibly largest RDMA deployment in the world**

Problems fixed, lessons and open problems

Many new research directions

Problems discovered and fixed

- Congestion leaking
  - Flows sent by the NIC always have near identical sending rates regardless of their congestion degrees
- PFC and MACsec
  - Different switches have no agreement on whether PFC frames sent should be encrypted or what to do with arriving encrypted PFC frames
- Slow receiver due to loopback RDMA traffic
  - Loopback RDMA traffic and inter-host RDMA traffic cause host congestion
- Fast Memory Registration (FMR) hidden fence
  - The NIC processes the FMR request only after the completions of previously posted requests

Lessons and open problems

Failovers are very expensive for RDMA

Lessons and open problems

Failovers are very expensive for RDMA

Host network and physical network should be converged

Lessons and open problems

Failovers are very expensive for RDMA

Host network and physical network should be converged

Switch buffer is increasingly important and needs more innovations

Lessons and open problems

- Failovers are very expensive for RDMA
- Host network and physical network should be converged
- Switch buffer is increasingly important and needs more innovations
- Cloud needs unified behavior models and interfaces for network devices

Lessons and open problems

Failovers are very expensive for RDMA

Host network and physical network should be converged

Switch buffer is increasingly important and needs more innovations

Cloud needs unified behavior models and interfaces for network devices

Testing new network devices is critical and challenging

Lessons and open problems

Failovers are very expensive for RDMA

Host network and physical network should be converged

Switch buffer is increasingly important and needs more innovations

Cloud needs unified behavior models and interfaces for network devices

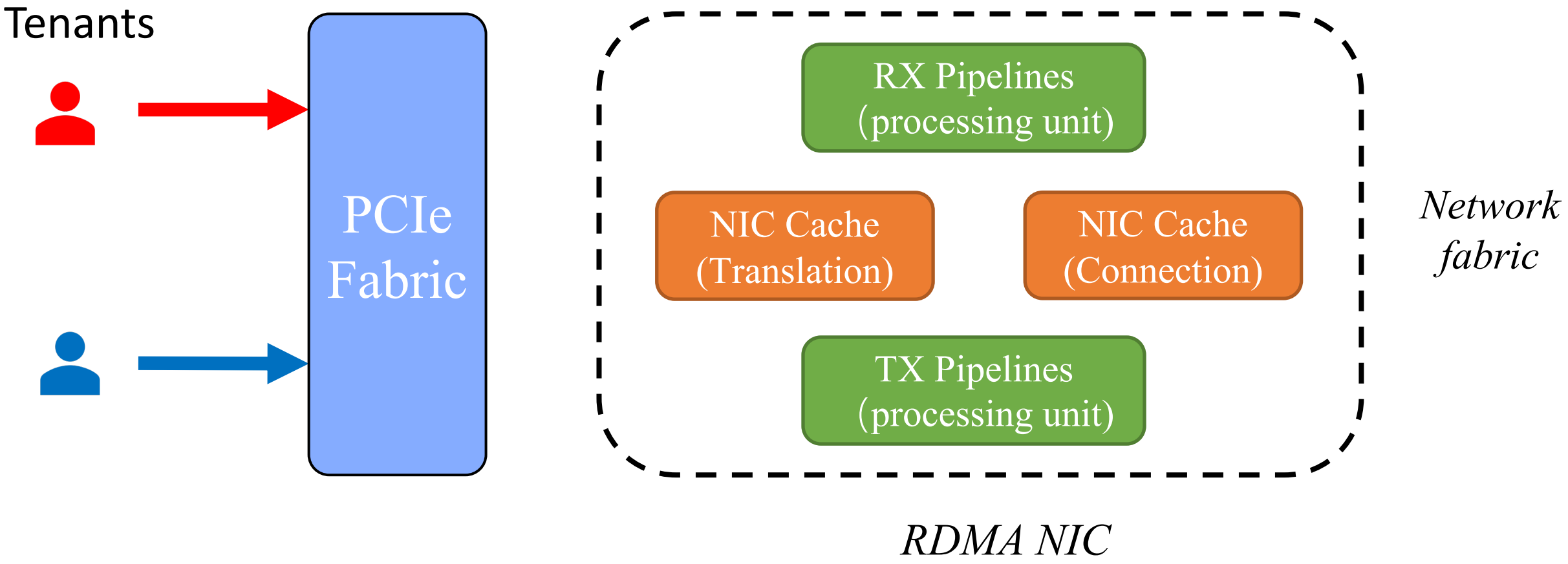
Testing new network devices is critical and challenging

Husky

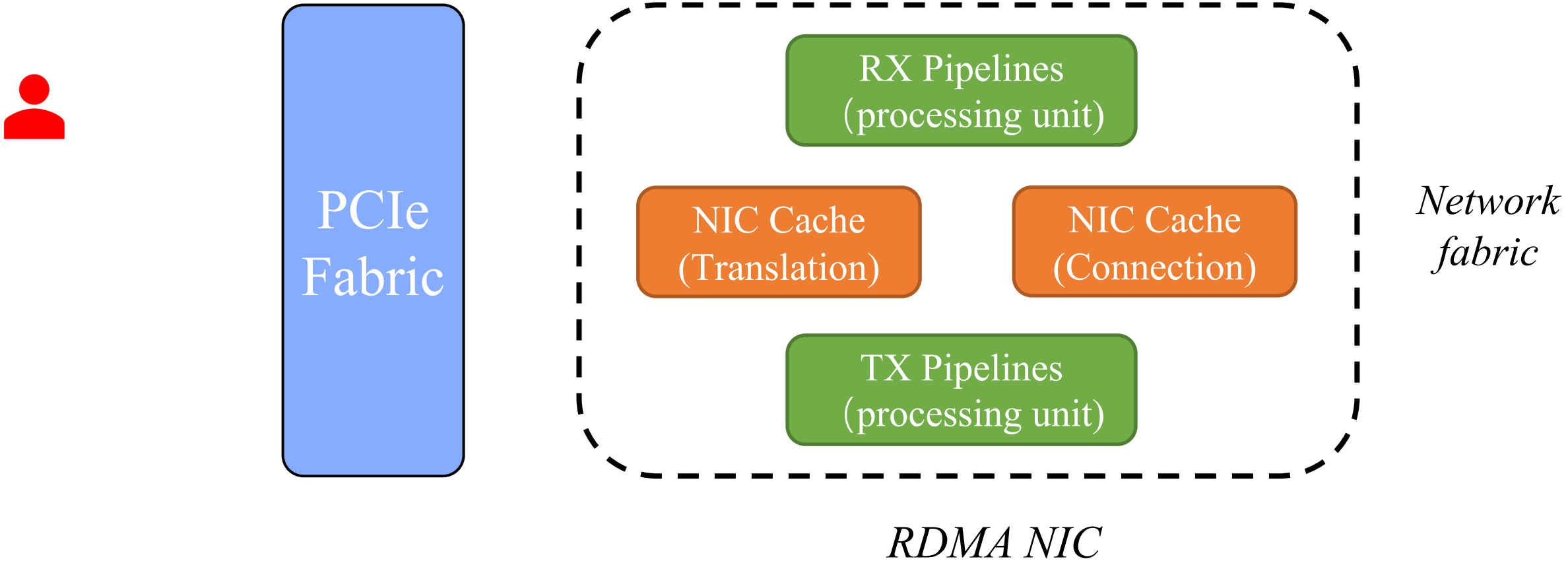
Understanding RDMA Microarchitecture Resources for Performance Isolation

Credit of slides: Xinhao Kong

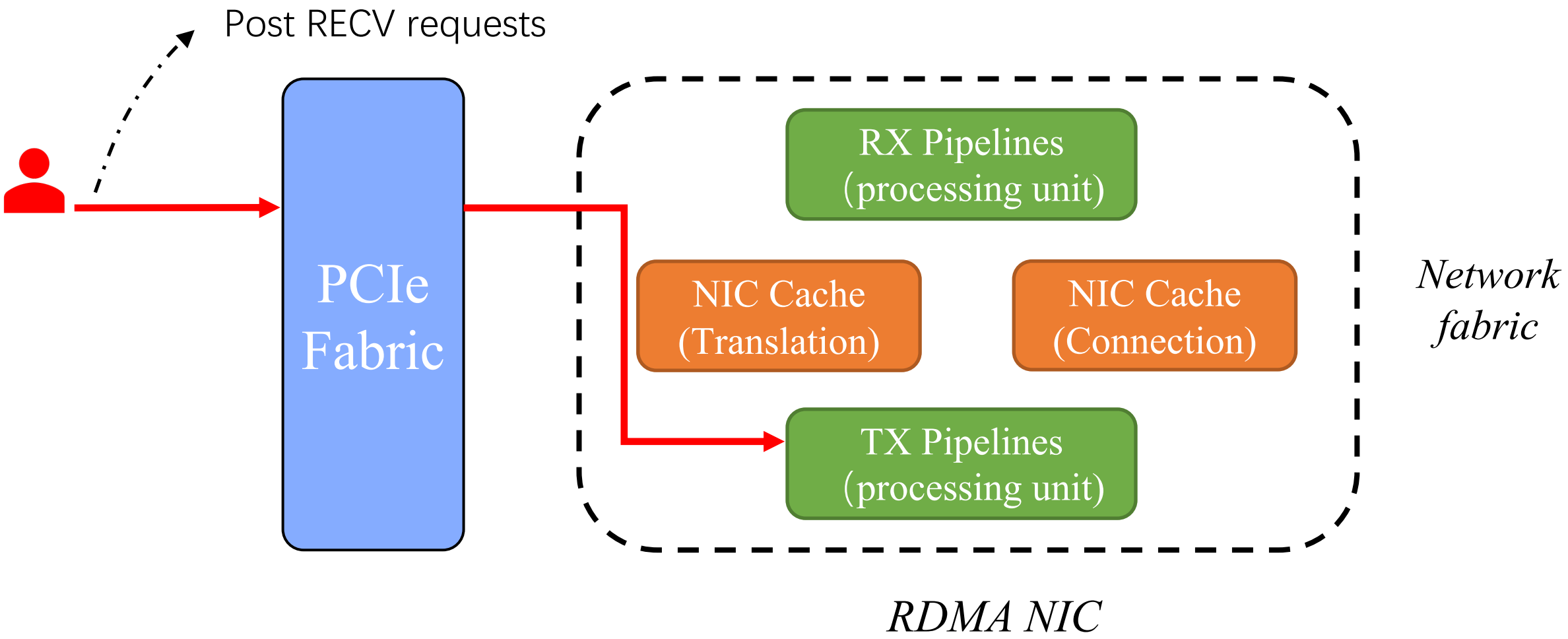
RDMA microarchitecture resource contention



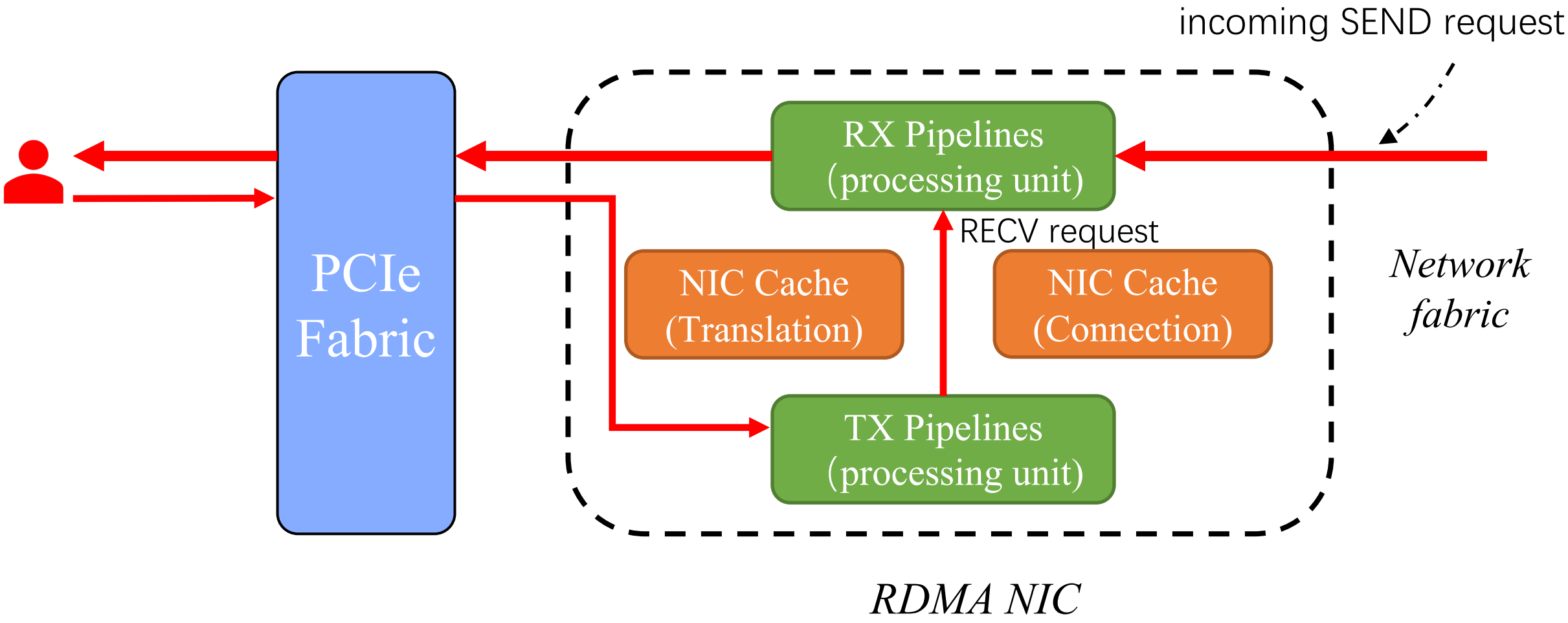
SEND/RECV error handling exhausts RX pipelines



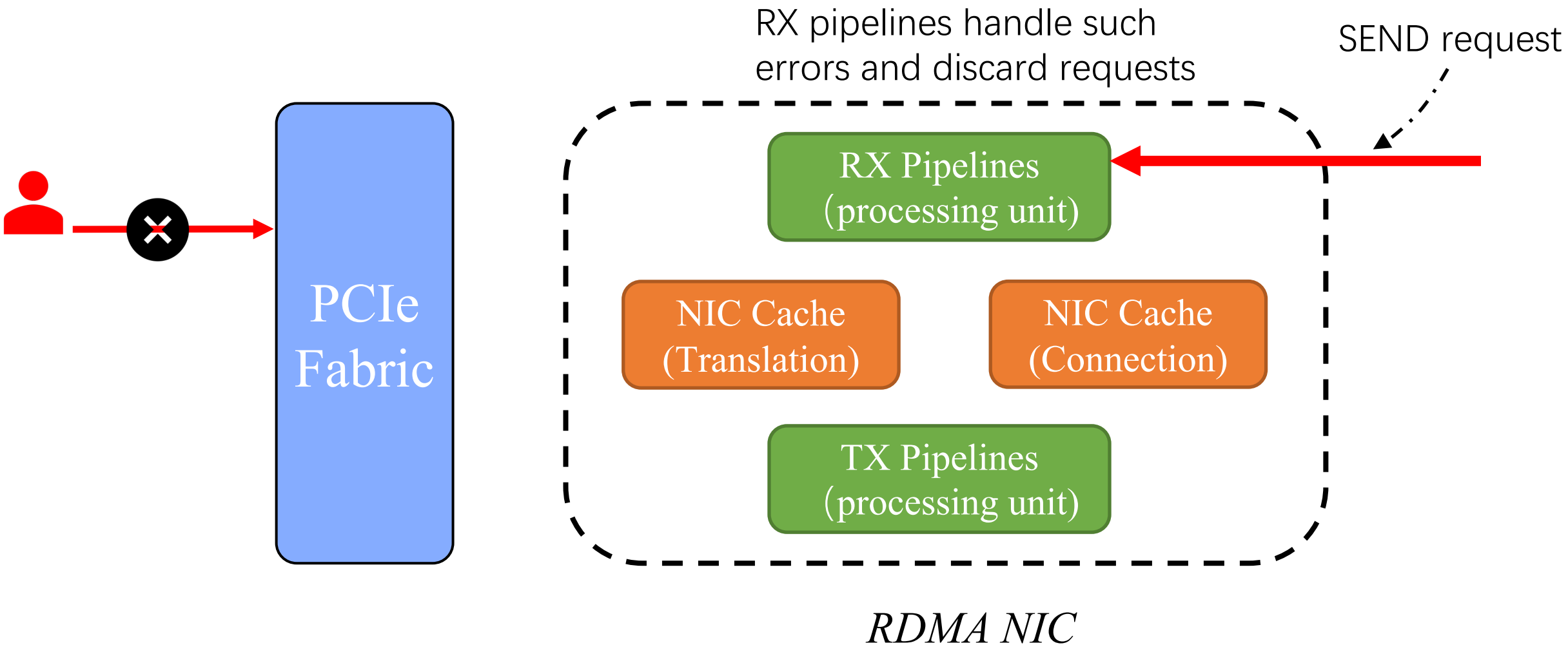
SEND/RECV needs prepared receive requests



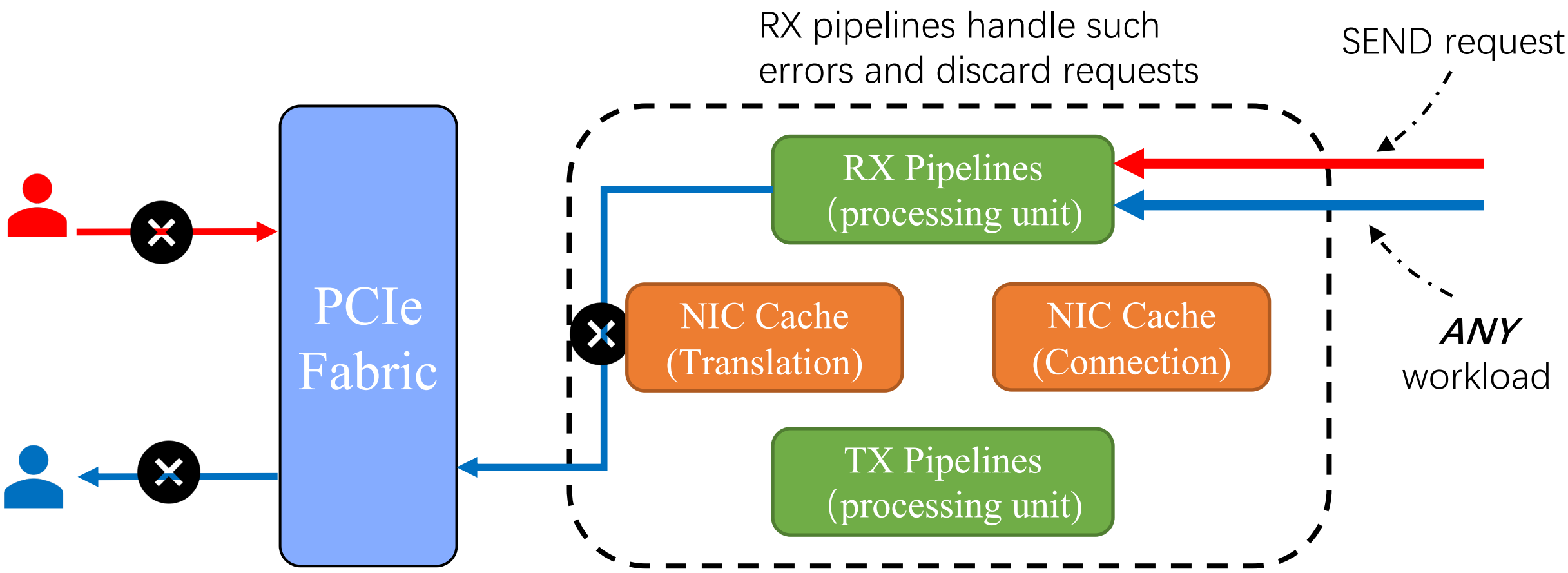
SEND/RECV needs prepared receive requests



Receive-Not-Ready (RNR) consumes RX pipelines



RNR causes severe RX pipelines contention



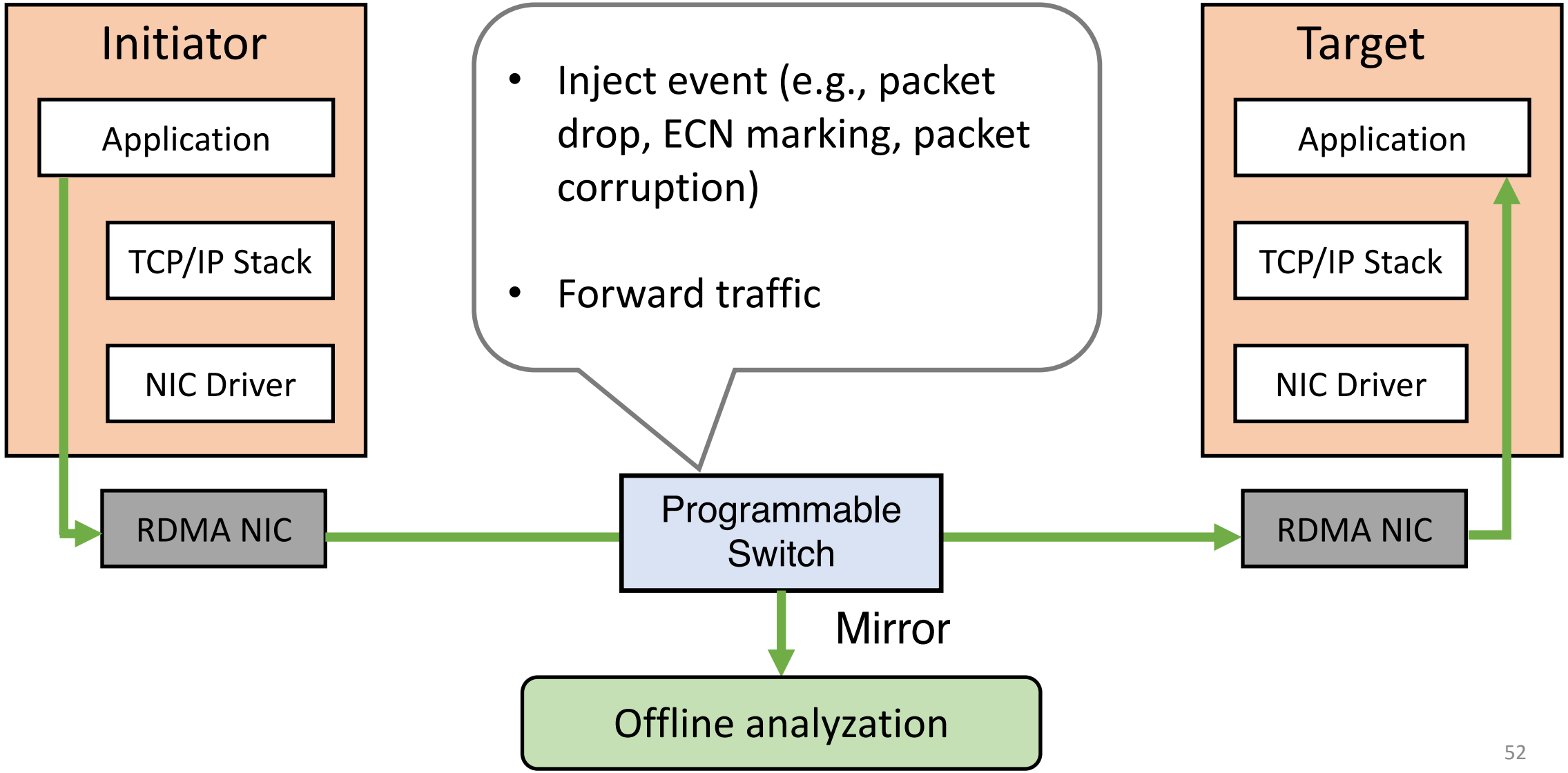
	Victim Bandwidth	Attacker Bandwidth
w/o RNR error	97.07 Gbps	\
w/ RNR error	0.018 Gbps	0 Gbps

Lumina

Understanding the Micro-Behaviors of Hardware Offloaded Network Stacks

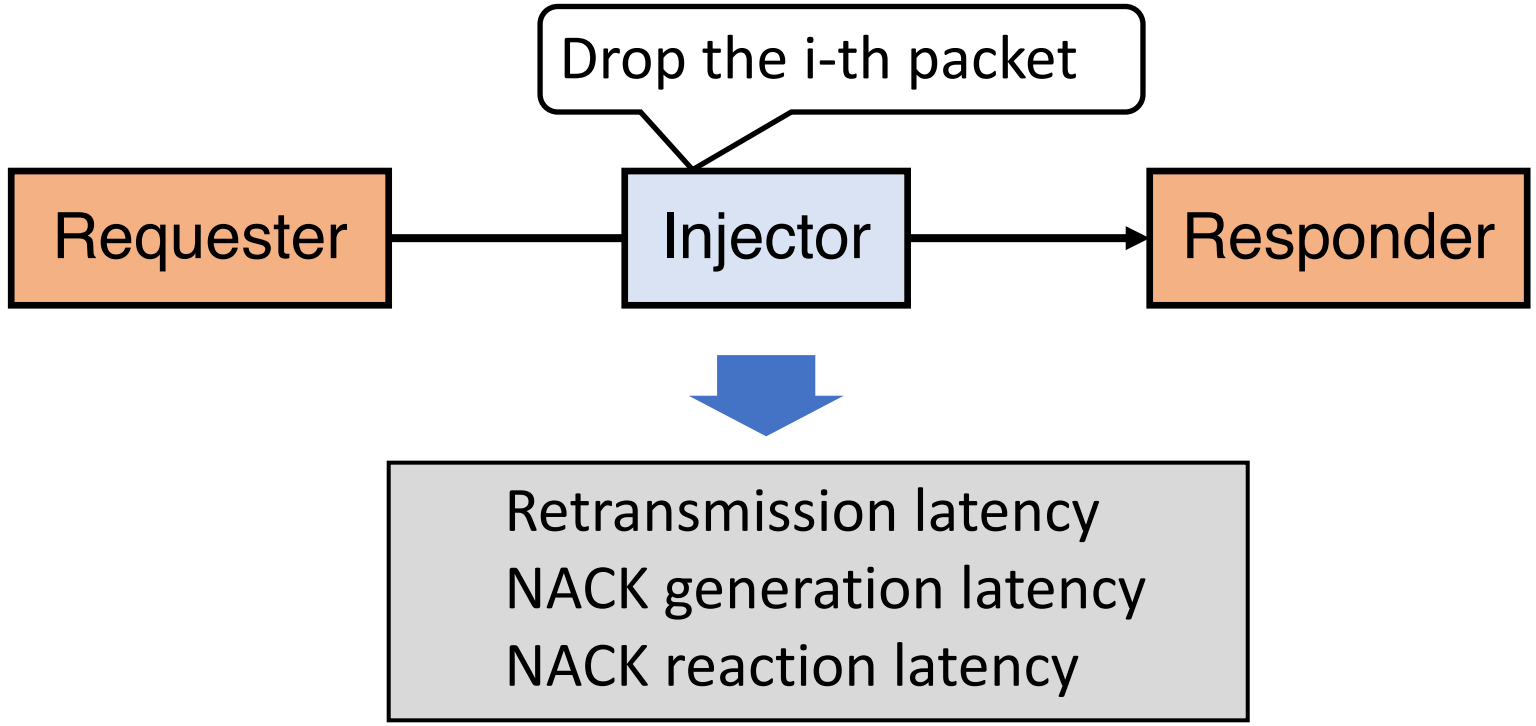
Credit of slides: Zhuolong Yu

Lumina: an in-network solution

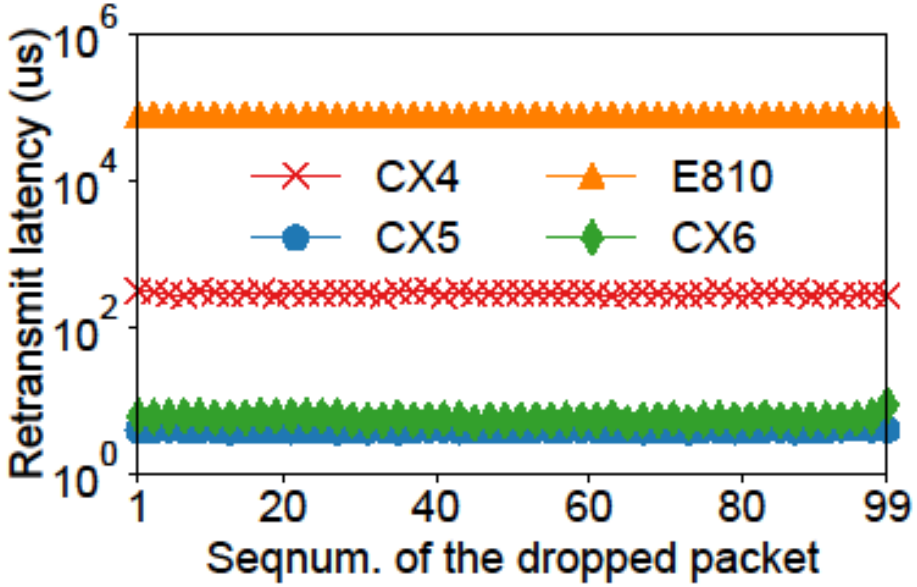


Experiment setting

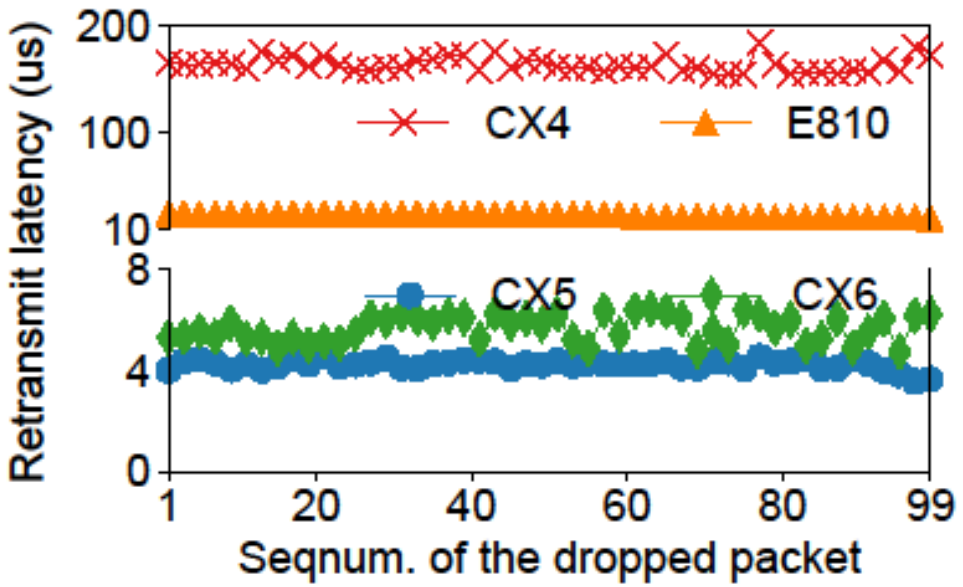
- RDMA Verb: WRITE and READ
- Message size = 100KB, MTU=1KB
- For each message, drop i-th packet in each message
- RNICs under test: NVIDIA CX4 Lx, CX5, CX6 Dx, and Intel E810



Retransmission latency



READ



WRITE

Significant improvement from NVIDIA CX4 Lx to CX5 and CX6 Dx

Intel E810 cannot efficiently recover lost READ packets

Project contributors

“**Empowering Azure Storage with RDMA**” NSDI '23 (Operational Systems Track)
Wei Bai, Shanim Sainul Abdeen, Ankit Agrawal, Krishan Kumar Attre, Paramvir Bahl, Ameya Bhagat, Gowri Bhaskara, Tanya Brokhman, Lei Cao, Ahmad Cheema, Rebecca Chow, Jeff Cohen, Mahmoud Elhaddad, Vivek Ette, Igal Figlin, Daniel Firestone, Mathew George, Ilya German, Lakhmeet Ghai, Eric Green, Albert Greenberg, Manish Gupta, Randy Haagens, Matthew Hendel, Ridwan Howlader, Neetha John, Julia Johnstone, Tom Jolly, Greg Kramer, David Kruse, Ankit Kumar, Erica Lan, Ivan Lee, Avi Levy, Marina Lipshteyn, Xin Liu, Chen Liu, Guohan Lu, Yuemin Lu, Xiakun Lu, Vadim Makhervaks, Ulad Malashanka, David A. Maltz, Ilias Marinos, Rohan Mehta, Sharda Murthi, Anup Namdhari, Aaron Ogus, Jitendra Padhye, Madhav Pandya, Douglas Phillips, Adrian Power, Suraj Puri, Shachar Raindel, Jordan Rhee, Anthony Russo, Maneesh Sah, Ali Sheriff, Chris Sparacino, Ashutosh Srivastava, Weixiang Sun, Nick Swanson, Fuhou Tian, Lukasz Tomczyk, Vamsi Vadlamuri, Alec Wolman, Ying Xie, Joyce Yom, Lihua Yuan, Yanzhao Zhang, Brian Zill, and many others

“**Understanding RDMA Microarchitecture Resources for Performance Isolation**” NSDI '23
Xinhao Kong, Jingrong Chen, Wei Bai, Yechen Xu, Mahmoud Elhaddad, Shachar Raindel, Jitendra Padhye, Alvin R. Lebeck, Danyang Zhuo

“Understanding the Micro-Behaviors of Hardware Offloaded Network Stacks with Lumina” SIGCOMM '23
Zhuolong Yu, Bowen Su, Wei Bai, Shachar Raindel, Vladimir Braverman, Xin Jin

Technical support from Arista Networks, Broadcom, Cisco, Dell, Intel, Keysight, NVIDIA

Thank You!