Mitigating Scalability Walls of RDMA-based Container Networks

Wei Liu, Kun Qian, Zhenhua Li, Feng Qian, Tianyin Xu, Yunhao Liu, Yu Guan, Shuhong Zhu, Hongfei Xu, Lanlan Xi, Chao Qin, Ennan Zhai

清華大學 Tsinghua University · Alibaba Cloud · USC University of Southern California · University of Illinois Urbana-Champaign

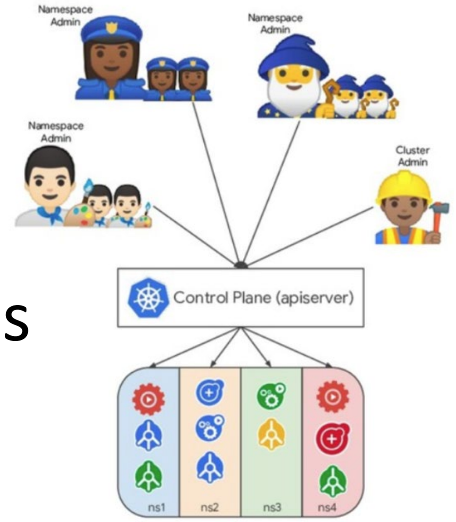
1. Background

Container is a pivotal technique

- Building microservices
- Deploying LLM training clusters at scale
- Facilitating DevOps and CI/CD pipelines

Container networks enable seamless communications

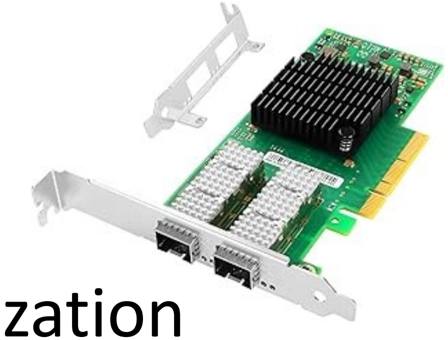
- Multi-tenancy
- Network virtualization
- High-performance container-to-container communications

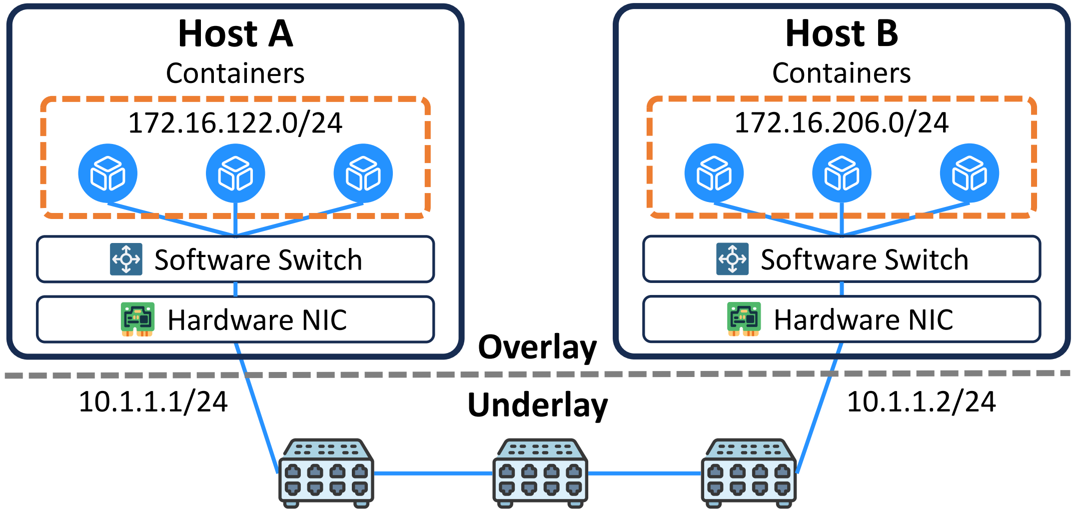


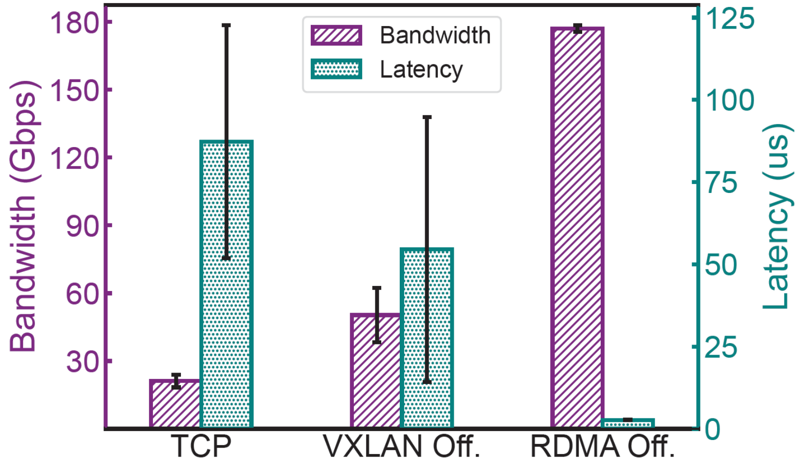
1. Background

RDMA-offloaded Container Network (RCN)

- RDMA support
- Hardware accelerated (through RDMA NICs)
- Offloading packet switching for network virtualization





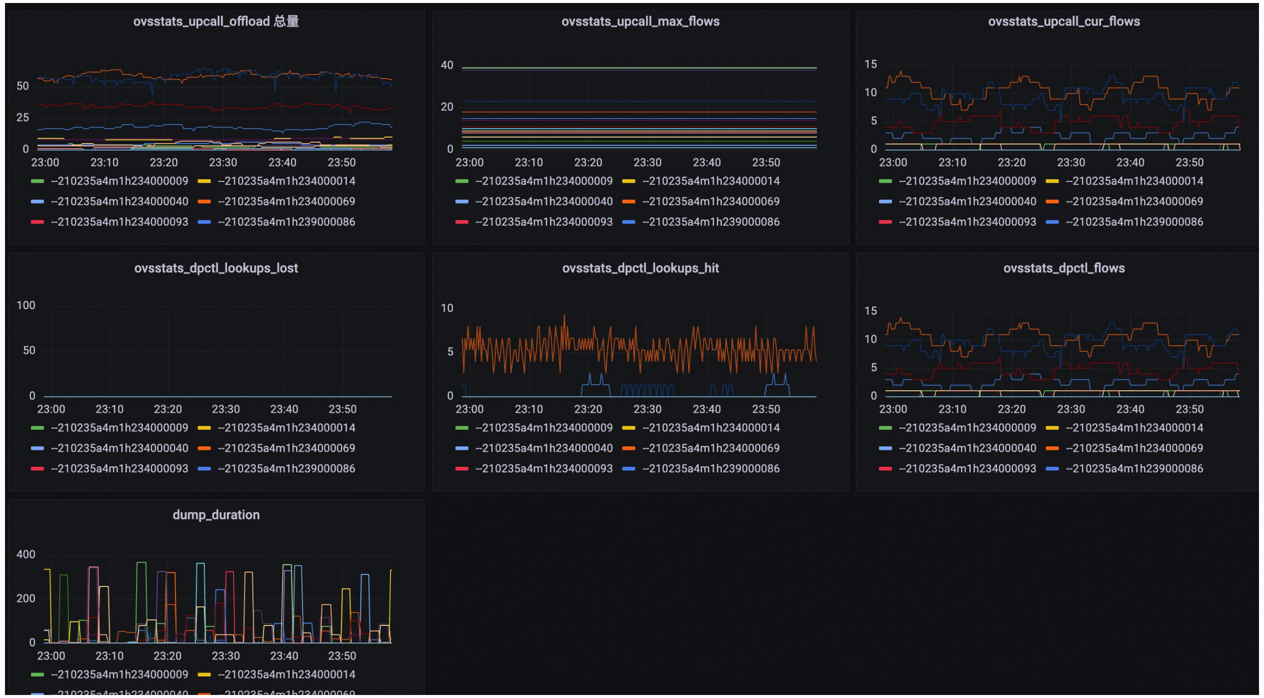


*At least **3X** performance improvement*

2. Motivation

An RCN in production has “scalability walls”

- A typical RCN cluster in Alibaba Cloud
  - 8K hosts
  - 40K RNICs
  - 0.5M active containers
- 0.4 M → 0.8 M containers
  - Bandwidth ↓87%
  - Latency ↑34X



2. Motivation

Measurement Study

- Apr. 15, 2023 — Apr. 15th, 2024
- Each host is equipped with 1 to 8 RNICs (mostly NVIDIA CX/BF series)
- Data collection
  - RNIC and kernel statistics
  - Open vSwitch (OVS) status
  - Container network interface (Nimitz CNI in Alibaba) events

2. Motivation

Measurement Findings

Most are RNIC-related

- Concrete symptoms of “scalability walls”:

Symptom	Layer	Ratio
Repetitive flow re-offloading	Virtual Switch	17.1%
Kernel stagnation	RNIC driver	5.9%
Kernel crash on new flows	RNIC driver	5.2%
Slow flow state maintenance	RNIC hardware	11.4%
Intermittent software forwarding	RNIC hardware	15.3%
Poor performance of specific flows	RNIC hardware	29.9%
PCIe link down when unbinding VFs	RNIC hardware	8.4%
RNIC unresponsiveness	RNIC hardware	6.8%

2. Motivation

Symptom——Repetitive Flow Offloading

- OVS bears considerable lookup miss/lost events

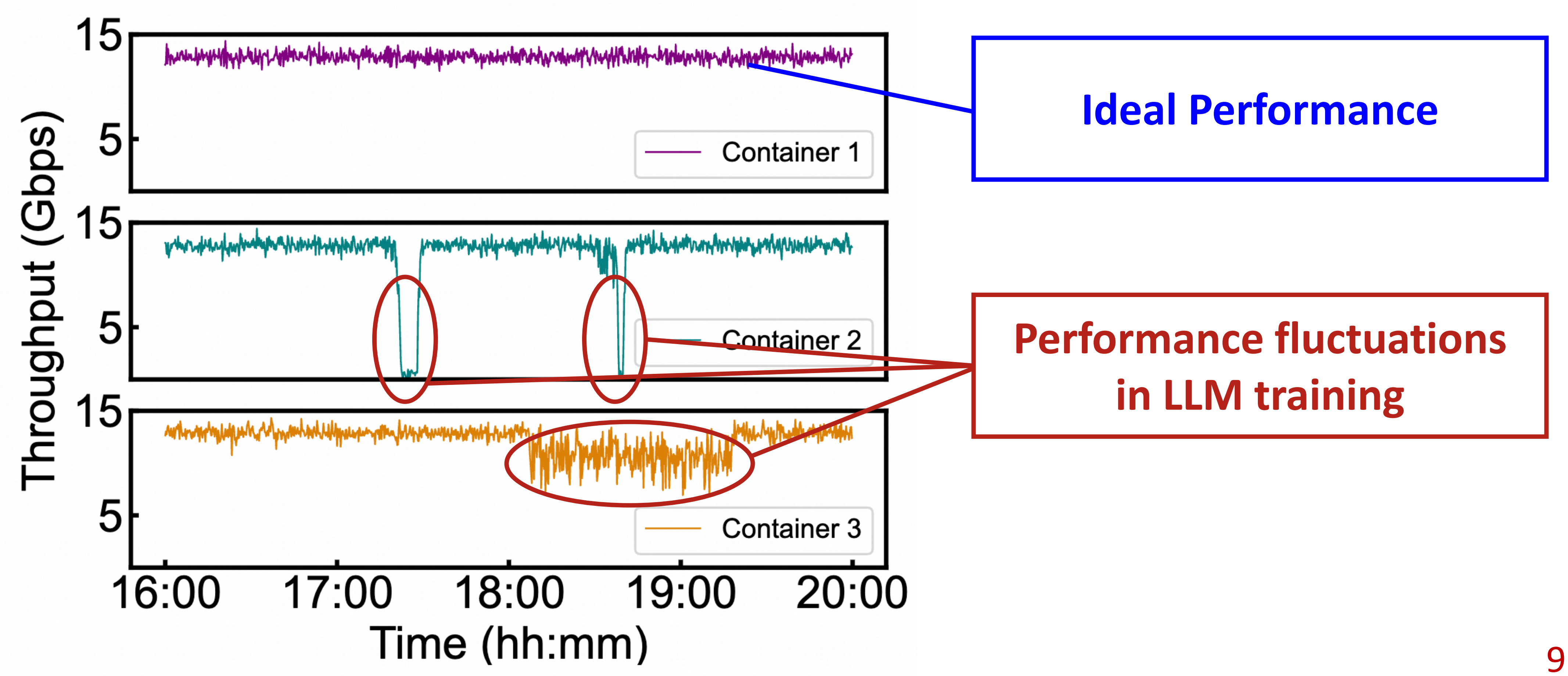
Flow offloading is unstable

Lookup miss/lost events in CNI

2. Motivation

Symptom——Repetitive Flow Offloading

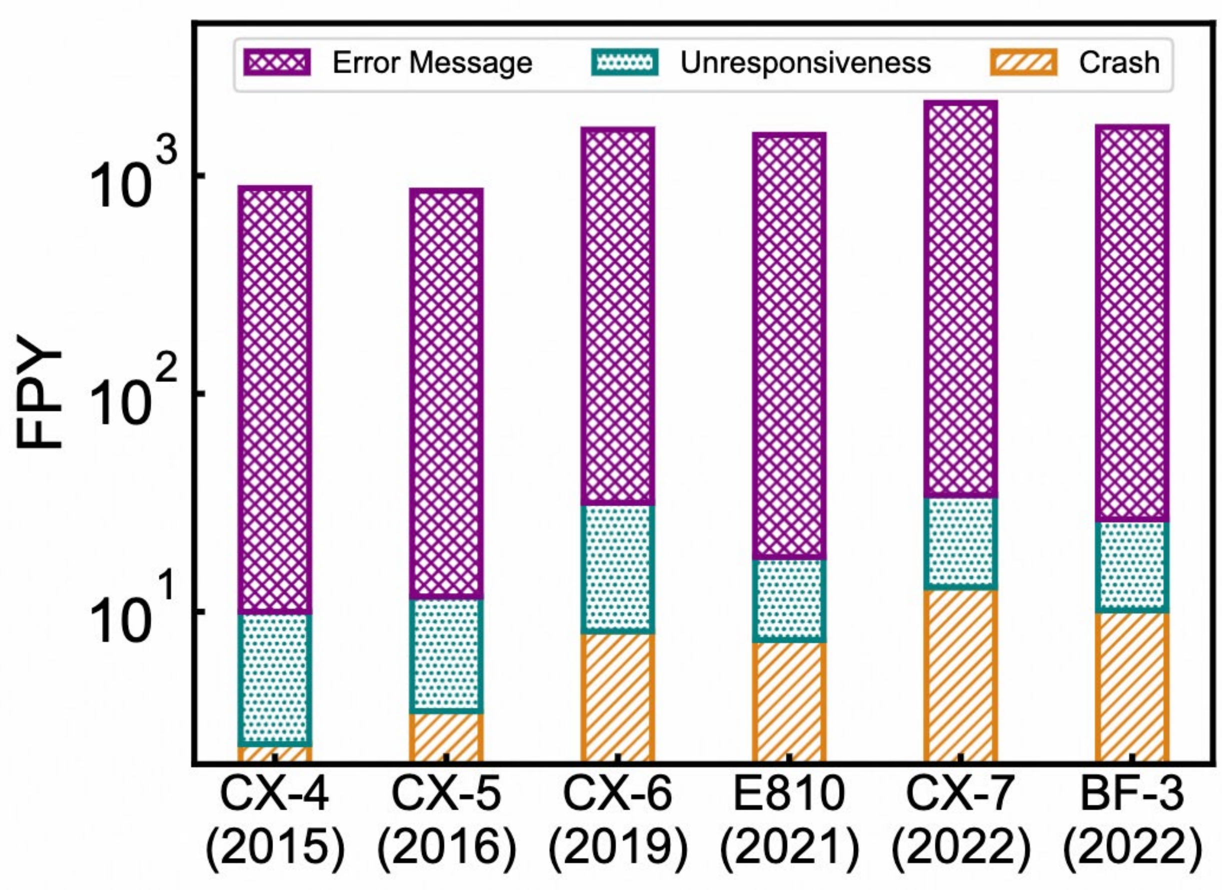
- Flows are intermittently processed by the software stack



2. Motivation

Symptom——RNIC Driver Defects

- The offloading of some flows in driver lead to kernel crash

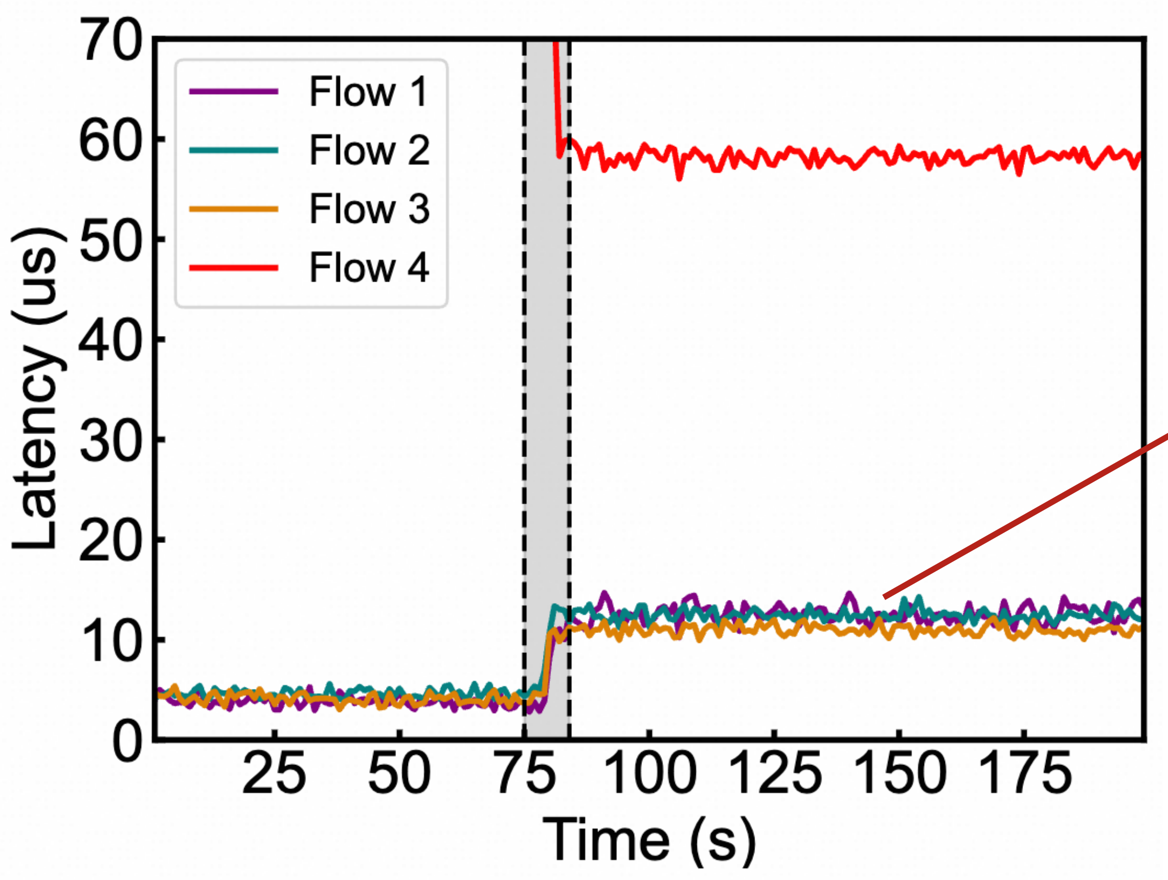


Newer RNICs present higher failures per year (FPY)

2. Motivation

Symptom——RNIC's Unexpected Behaviors

- Poor performance of specific flows

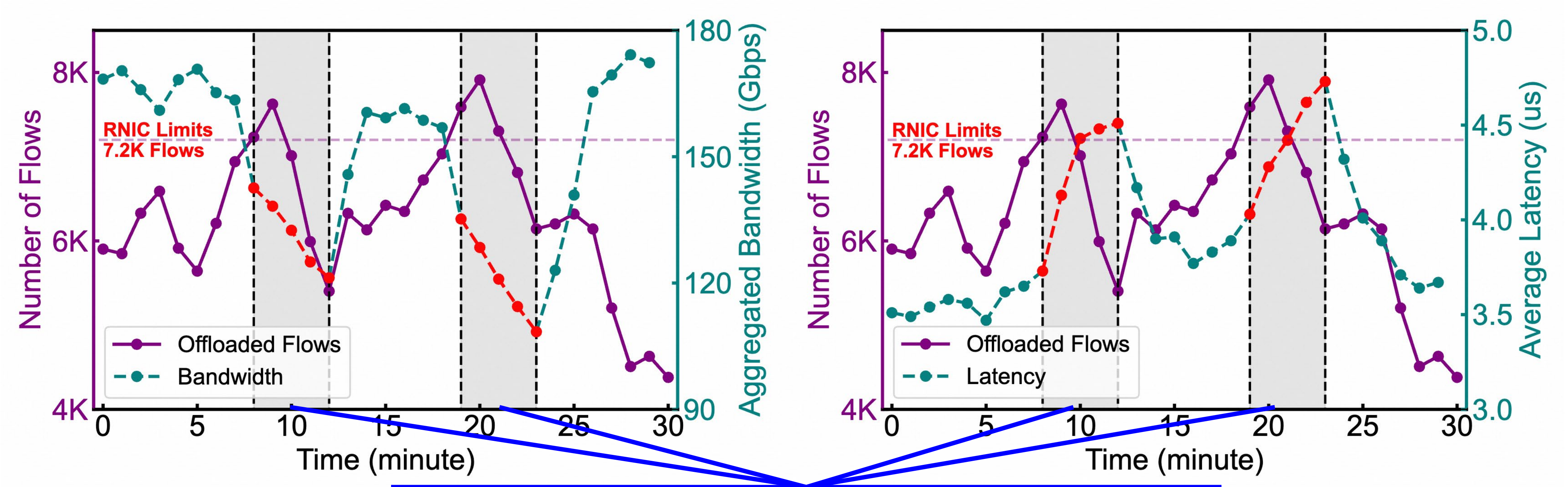


Performance degradation after RNIC flow offloading

2. Motivation

Symptom——RNIC's Unexpected Behaviors

- Unrecoverable poor performance



The performance cannot recover from the past high workload

2. Motivation

Challenges

- A commodity RNIC is a *blackbox*
  - Limited visibilities into its internals
  - Its *micro behaviors* are not recorded in its datasheet
- Difficult for RNIC vendors to reproduce our encountered issues
  - Cannot share real-world workloads
  - Our workloads push the RNICs to the extreme

3. Design

ScalaCN——Performance Testing and Modeling

- Key idea—***infer*** the RNICs' architecture model and performance model
- Today's RNICs in an RCN provide:
  - **RDMA verb interface** for bypassing software stack
  - **eSwitch interface** for efficient packet forwarding and transformation

↓

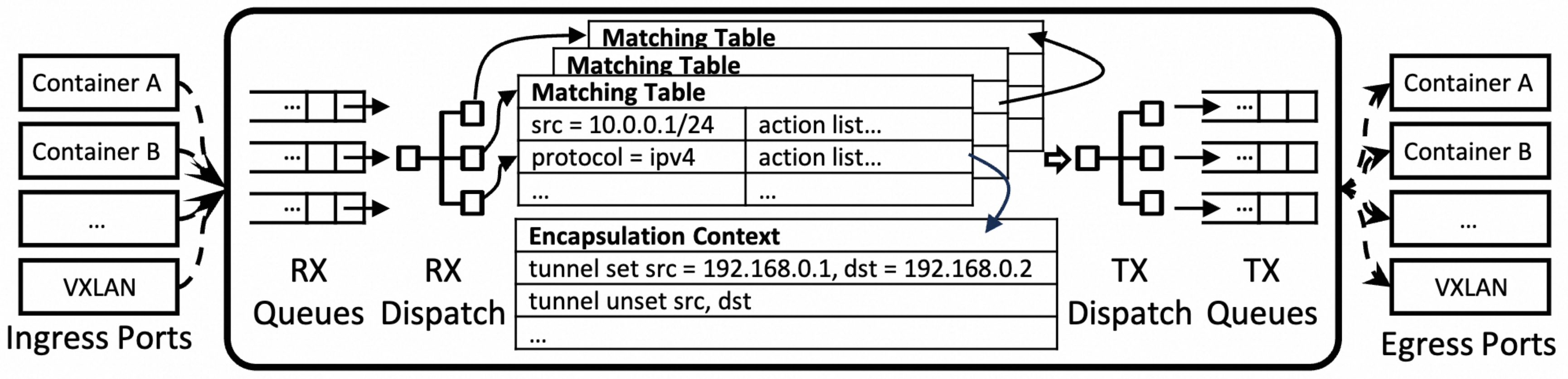
Use RNICs' **common abstractions** for performance testing

A greybox-based approach

3. Design

ScalaCN——Greybox RNIC Testing and Optimization

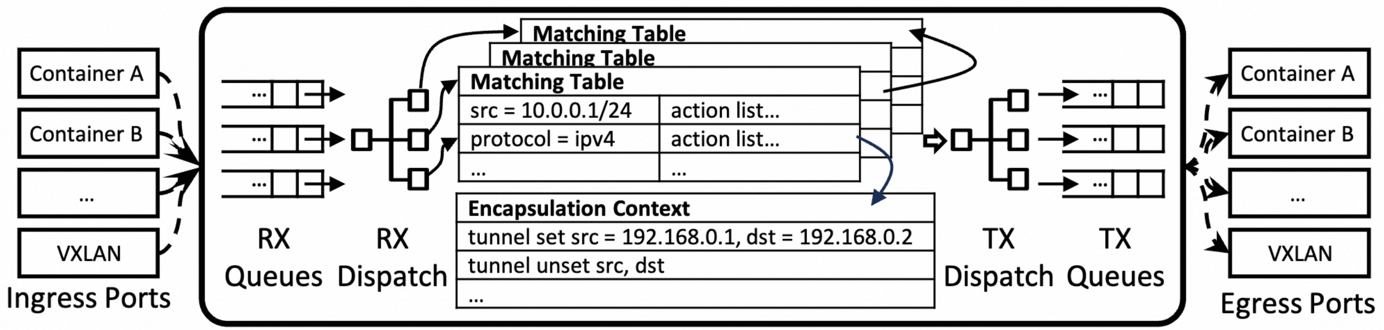
- Hardware abstractions in RDMA verbs
  - Queue Pair (QP), Completion Queue (CQ), Work Queue (WQ)
- eSwitch: embedded switch in RNICs
  - Conform to the *switchdev* model of Linux kernel
  - In a match-action manner



3. Design

ScalaCN——Greybox RNIC Testing and Optimization

- **Exponential search space**
  - RNIC is highly configurable
  - Examples
    - NVIDIA CX series has at least 192 bits for packet matching
    - Different QP, CQ, and WQ combinations




```
struct mlx5dr_match_spec {
 u32 smac_47_16;
 u32 smac_15_0:16;
 u32 ethertype:16;
 u32 dmac_47_16;
 u32 dmac_15_0:16;
 u32 first_prio:3;
 u32 first_cfi:1;
 u32 first_vid:12;
 u32 ip_protocol:8;
 u32 ip_dscp:6;
 u32 ip_ecn:2;
 u32 cvlan_tag:1;
 u32 svlan_tag:1;
 u32 frag:1;
 u32 ip_version:4;
  ...
};
```

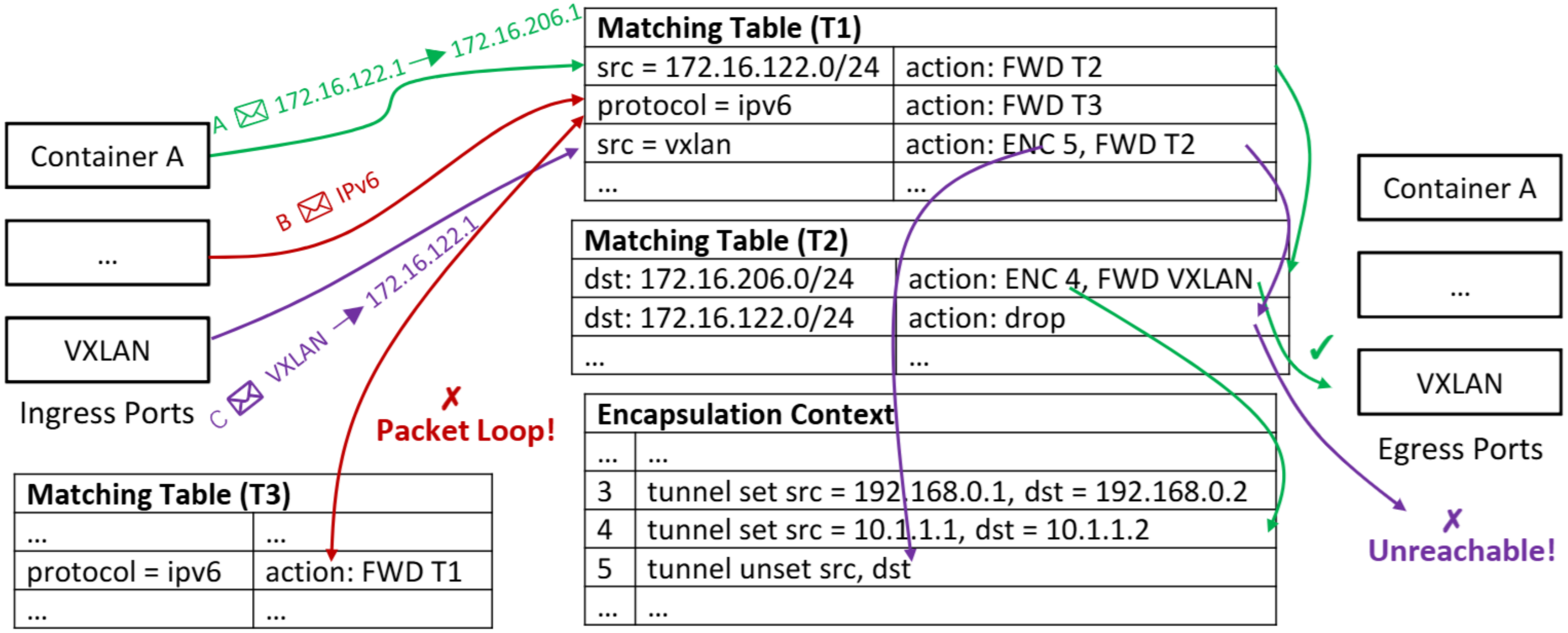
3. Design

ScalaCN——Combinatorial Causal Testing

- Efficient testing with **topological restrictions**

60X faster

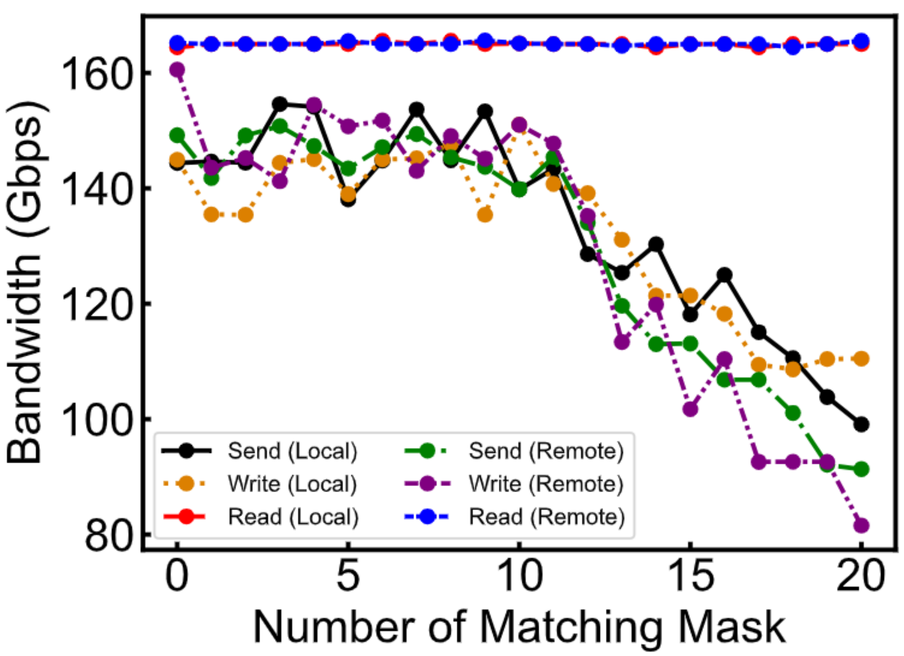
  - Key idea: leverage the input dependencies to filter out the combinations that lead to **packet loops** or **unreachability**

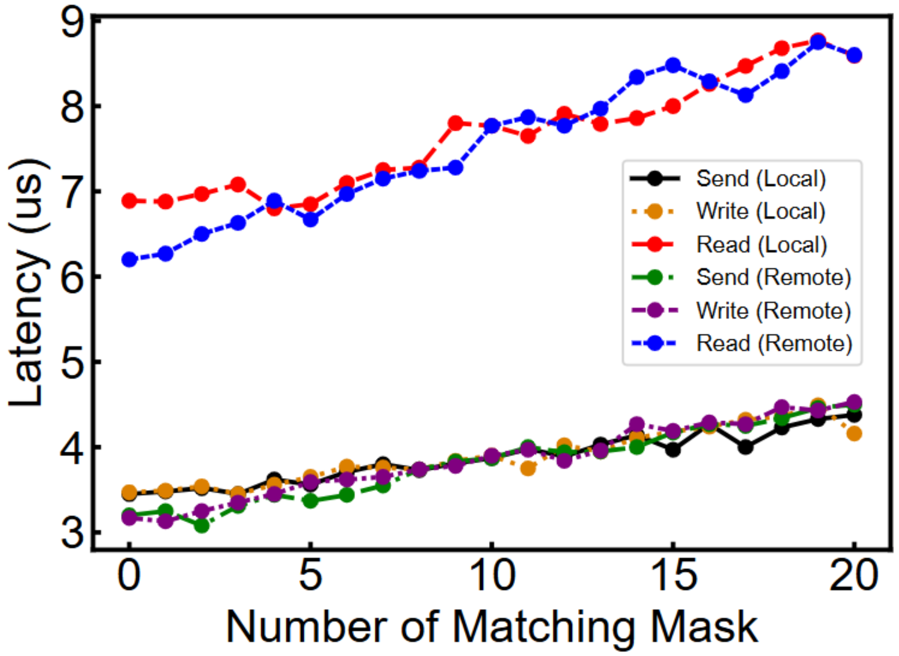


3. Design

ScalaCN——Causal Inference

- Identify concrete configurations that lie in the critical path
- Refine the critical path through permutation removal and sensitivity analysis

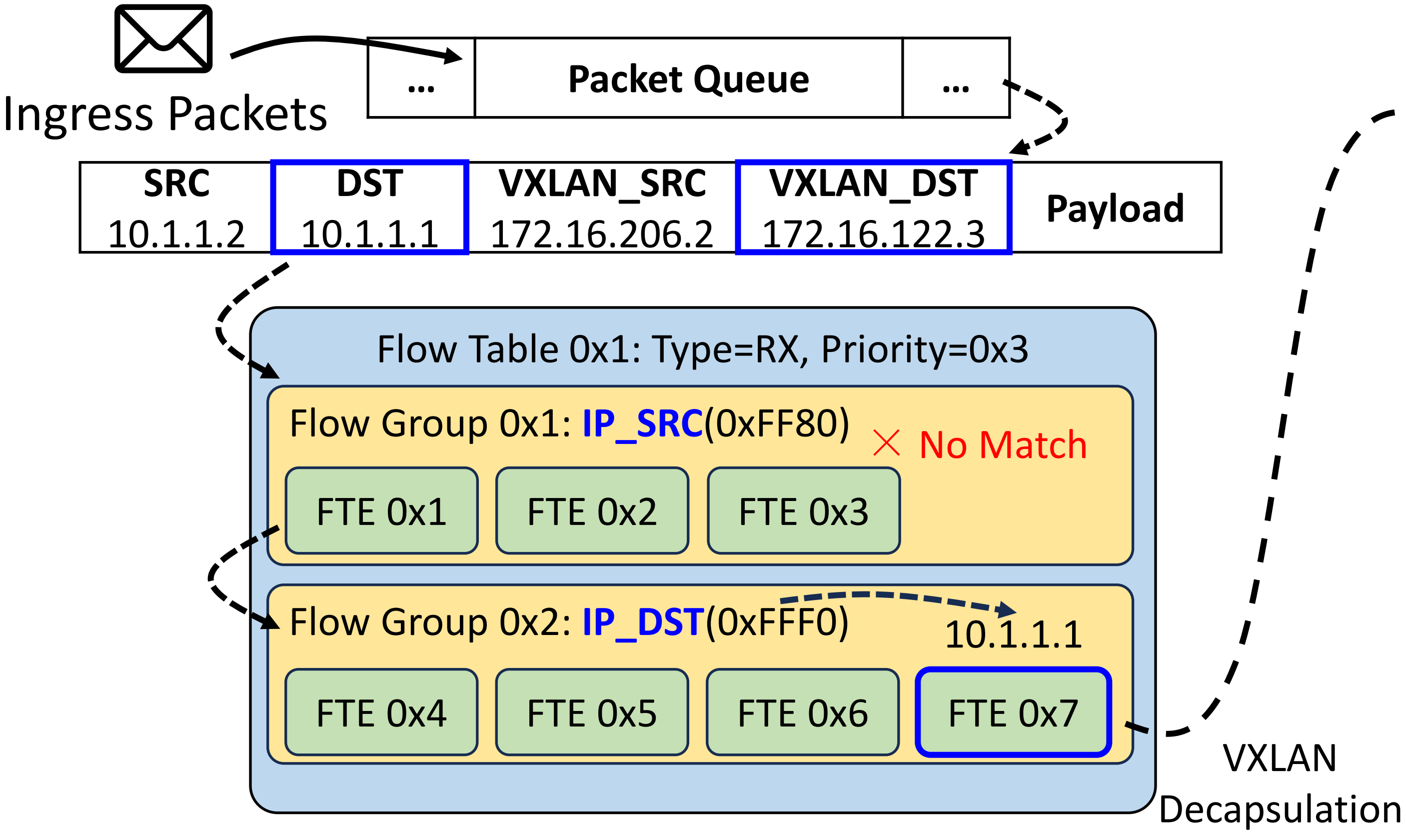


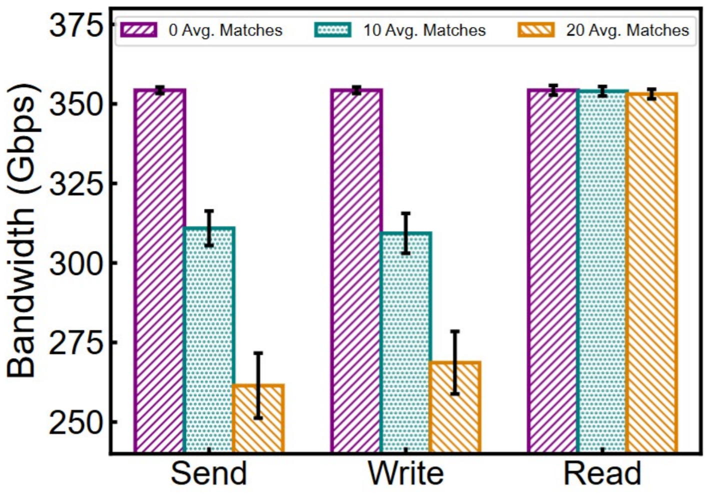


3. Design

ScalaCN——Performance Interpretation & Prediction

- Sequential matching mask query →Poor performance of specific flows





3. Design

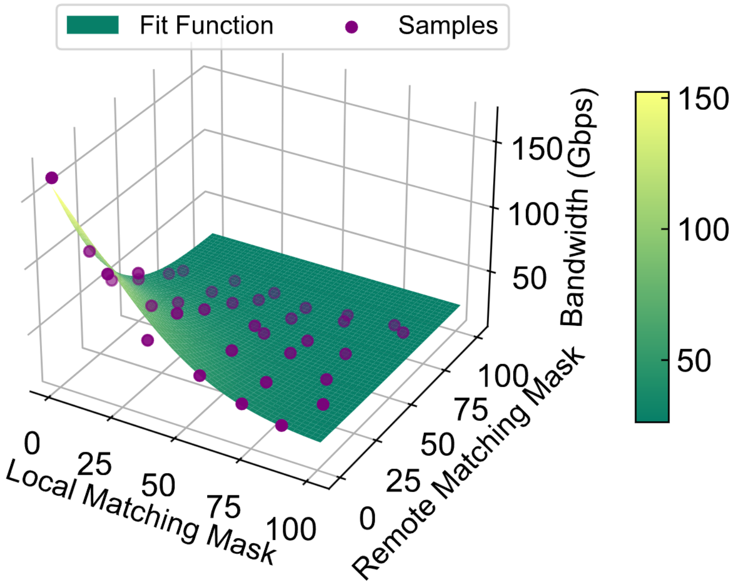
- **ScalaCN——Performance Interpretation & Prediction**
  - Sequential mask query →Poor performance of specific flows
  - QP contention→Proportional performance degradation
  - Inconsistent flow counter→Flow re-offloading
  - VXLAN context overflow→Kernel stagnation

3. Design

ScalaCN——Performance Interpretation & Prediction

- The performance of CX series RNICs can be predicted as

$$BW(Q_l, Q_r) = \mathrm{u} \cdot e^{\frac{-(Q_l - m)^2 + (Q_r - n)^2}{2 \cdot v^2}} + w \qquad LAT(Q_l, Q_r) = \mathrm{a} \cdot Q_l + \mathrm{a} \cdot Q_r + c$$



RNIC	u	v	m	n	w	a	b	c	GD
CX-4	78.86	39.14	-28.25	-43.19	4.91	0.049	0.063	5.02	0.94
CX-5	148.97	42.34	-29.37	-45.14	12.43	0.051	0.074	4.93	0.93
CX-6	324.47	42.13	-26.92	-49.71	26.28	0.047	0.068	2.69	0.93
CX-7	739.52	48.66	-33.53	-52.88	29.32	0.036	0.045	2.57	0.94
BF-3	748.52	48.01	-33.40	-52.42	30.65	0.037	0.043	2.56	0.94
E810	335.64	43.54	-27.01	-49.55	25.16	0.042	0.069	2.74	0.92

3. Design

ScalaCN——Proactive Performance Optimization

- **Packet matching forms the major bottleneck in production**

Hyper Matching Mask

Mask 0x10: All Header Fields
FE 0x1: IP_SRC=172.16.122.1, IP_DST= 172.17.206.2, PORT=4789, …
FE 0x2: IP_SRC=172.10.31.98, IP_DST=172.8.30.1, PORT=8472, …
…

Hyper FE Activation

Cascading Matching Masks

Mask 0x12: IP_SRC (0xFFF0), IP_DST (0xFF00)
FE 0x21: IP_SRC=172.16.122.0/**24**, IP_DST= 172.17.0.0/**16**
FE 0x22: IP_SRC=172.10.31.0/**24**, IP_DST=172.8.0.0/**16**

Mask 0x13: IP_SRC (0xFFF0)
FE 0x31: IP_SRC=172.16.122.0/**24**
FE 0x32: IP_SRC=172.10.31.0/**24**

Mask 0x14: IP_DST (0xFF00)
FE 0x41: IP_DST= 172.17.0.0/**16**
FE 0x42: IP_DST=172.8.0.0/**16**

4. Evaluation

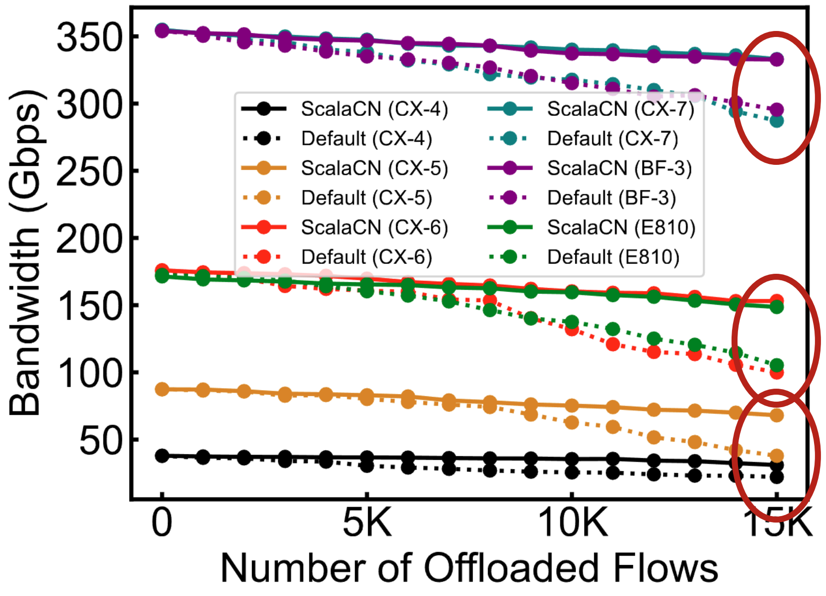
Testbed

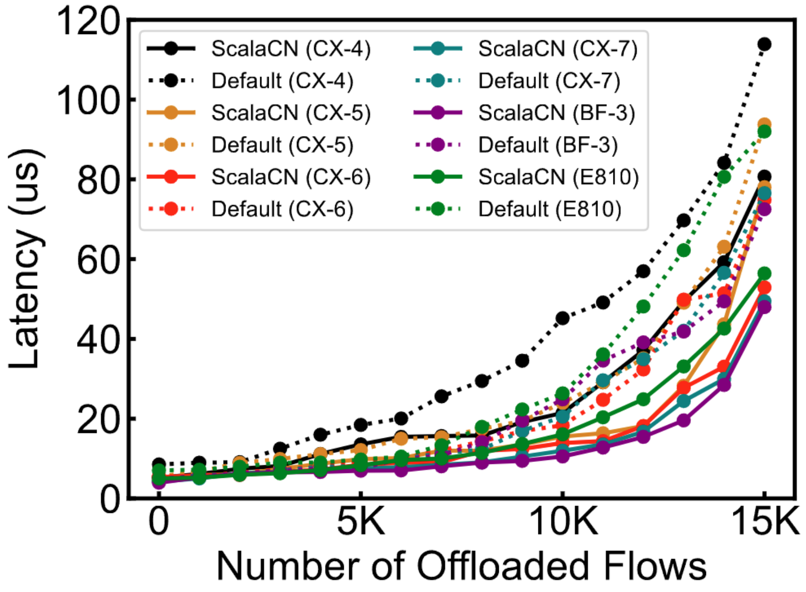
- Middle-Scale RCN Configuration
  - 50-node cluster with heterogeneous RNICs (CX-4/CX-5/CX-6/CX-7/BF-3/E810)
  - 4 RNICs per host
  - Avoid inter-operations of different RNIC models
- Real-World Traffic Generation
  - Large model training workloads
  - O(4M) daily flows, 150-400 Gbps throughput per RNIC
- Metrics
  - Packet forwarding bandwidth
  - End-to-end latency

4. Evaluation

Major results

Microbenchmarks



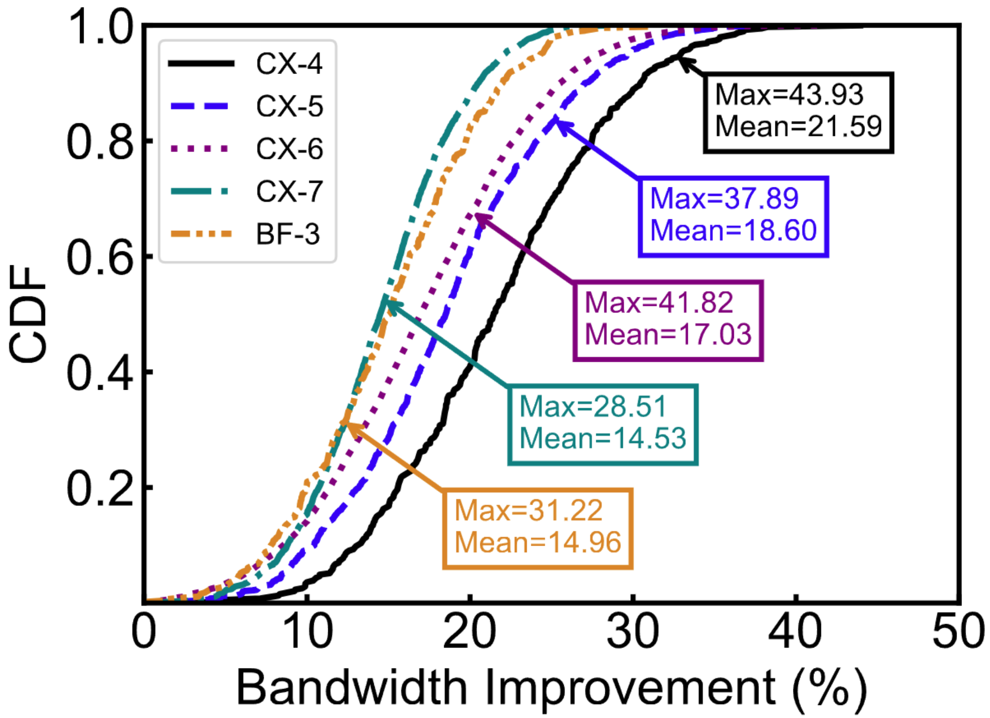


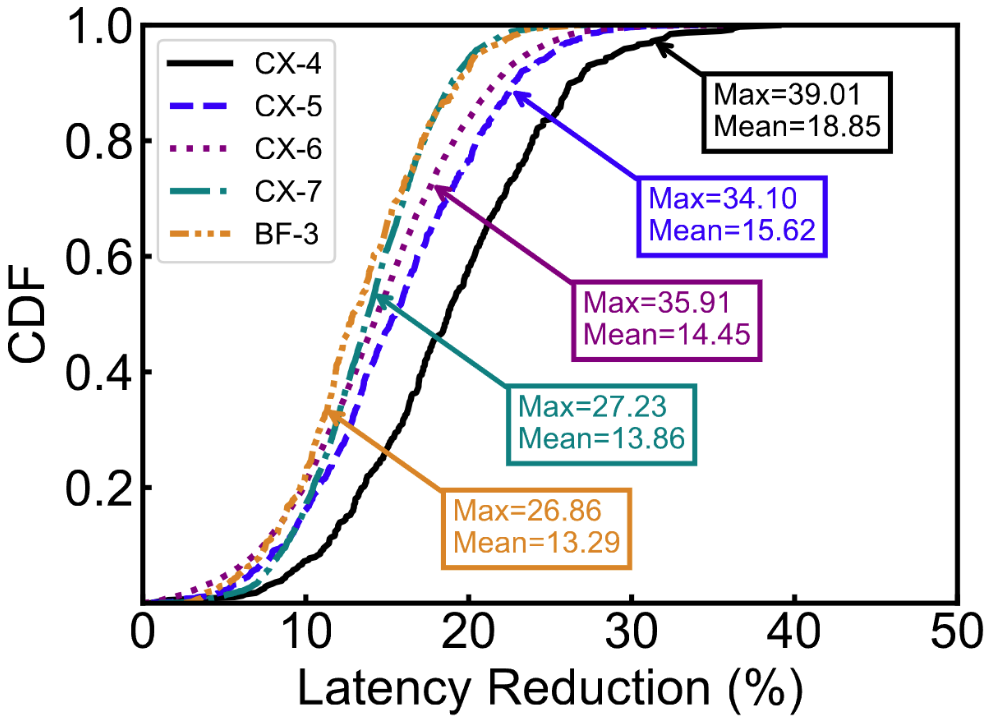
ScalaCN significantly improves the performance by ~40% under heavy workloads

4. Evaluation

Major results

- ScalaCN in production



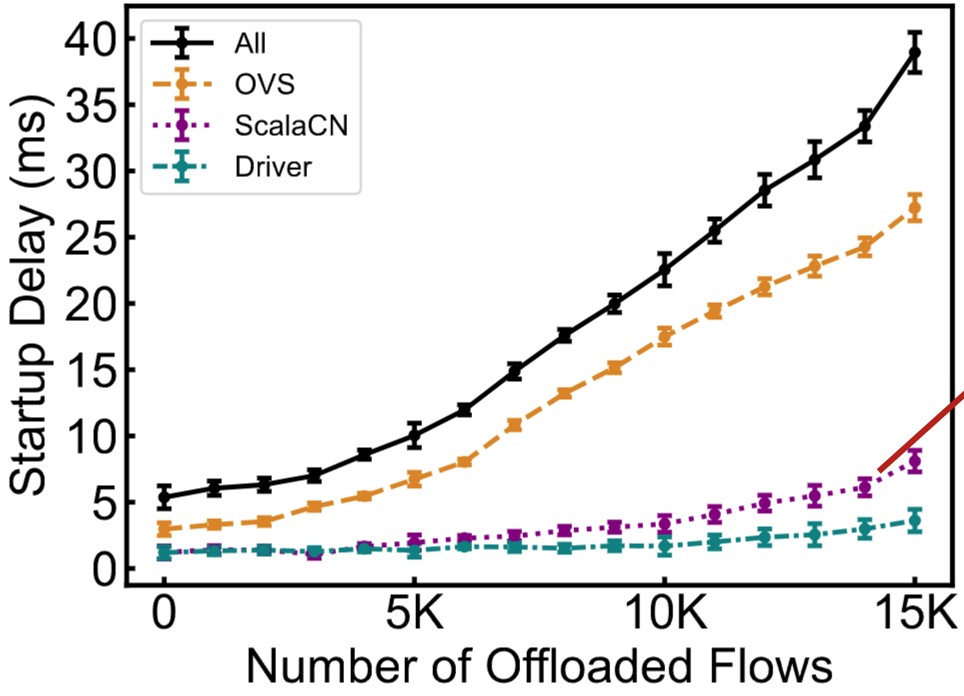


ScalaCN significantly improves the performance by ~17% on average in production

4. Evaluation

Beyond performance improvements

- Flow startup delay



ScalaCN incurs negligible side effects on startup performance

5. Conclusion

- We conduct the first study to uncover the scalability wall in a large-scale RCN, and pinpoint the culprit to be RNICs.
- We devise combinatorial causal testing based on RNICs' common abstractions, so as to efficiently approximate RNICs' internals and infer the root causes of performance issues.
- We devise an effective method to accommodate RNICs to RCN scaling. Evaluation on real-world workloads and the feedback from vendors confirm its efficacy. We are now gradually deploying ScalaCN over the production RCN.
- Code and data are released at https://scala-cn.github.io/

Thanks!
Q & A

Special thanks to Tsinghua Deng Feng Fund!