THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

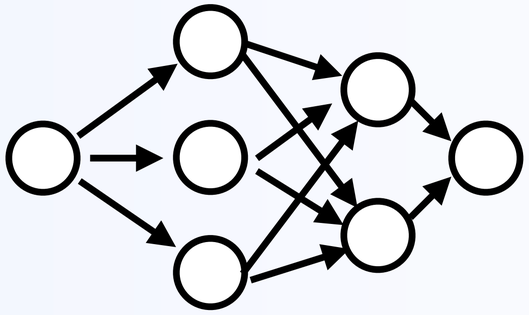
CLUSTAR 星云

Berkeley UNIVERSITY OF CALIFORNIA

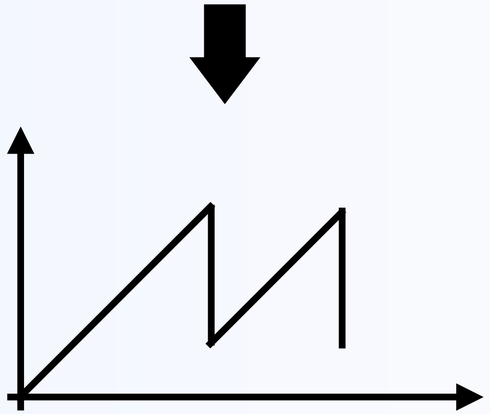
LiteFlow: Towards High-performance Adaptive Neural Networks for Kernel Datapath

Junxue Zhang, Chaoliang Zeng, Hong Zhang, Shuihai Hu, and Kai Chen

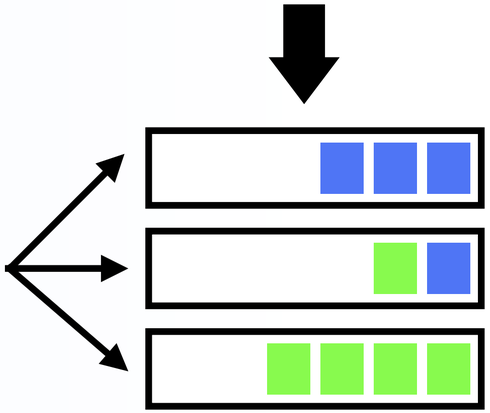
Background



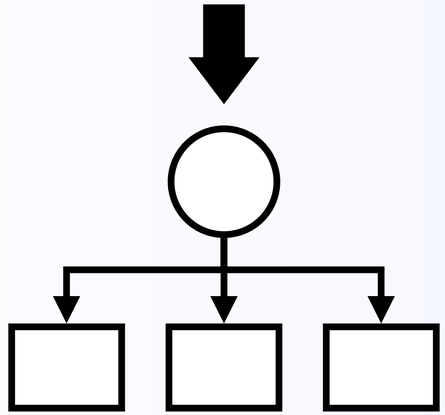
1. Adaptive neural networks (NN) have been used to optimize various networking datapath functions
2. Adaptive NNs have achieved superb performance since it combines model inference and model tuning as a whole



Congestion Control
Aurora, ICML19; MOCC, EuroSys 22



Flow Scheduling
AUTO, SIGCOMM 18

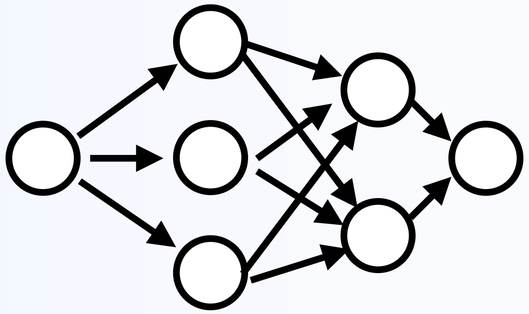


Packet Routing
Learning to Route, HotNets 17

Background

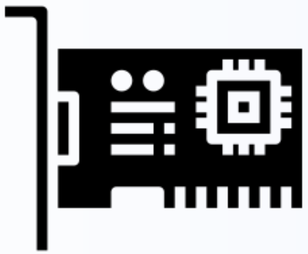
Taking CC as an example

② Performing Inference
+ Tuning the NN



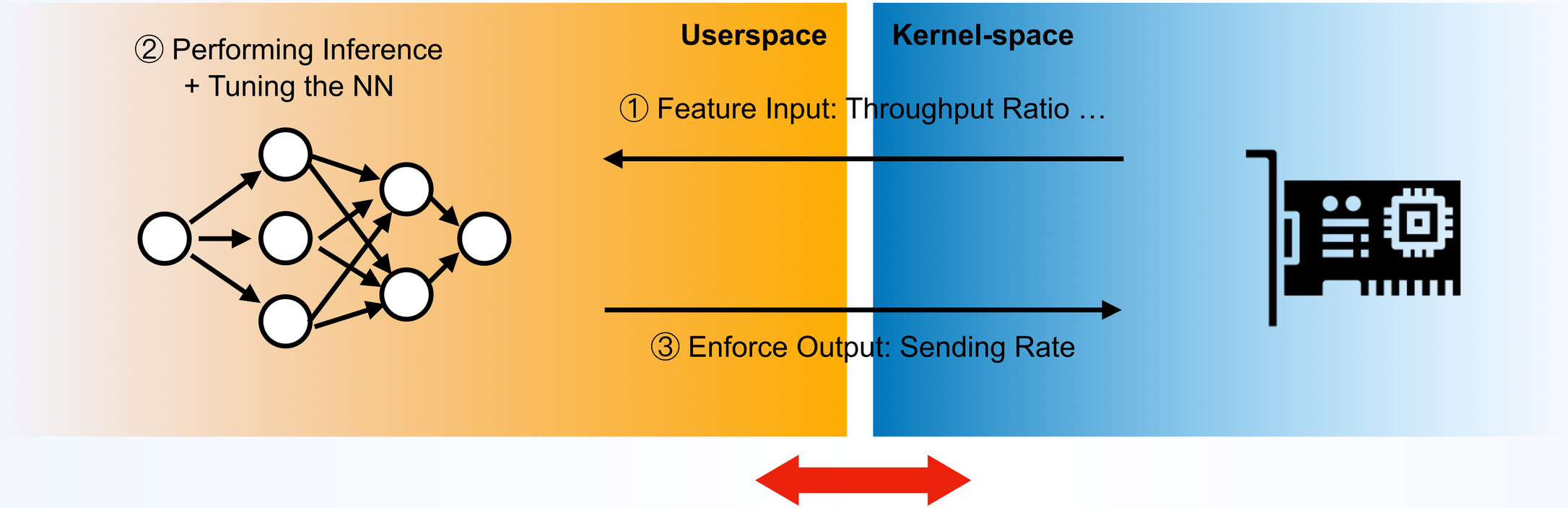
① Feature Input: Throughput Ratio …

③ Enforce Output: Sending Rate



Motivation

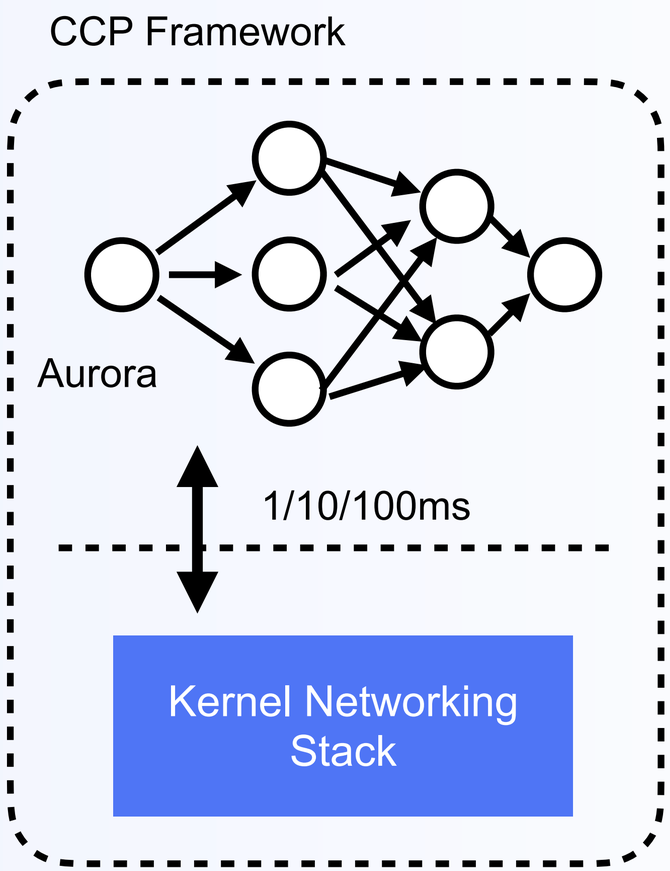
Taking CC as an example



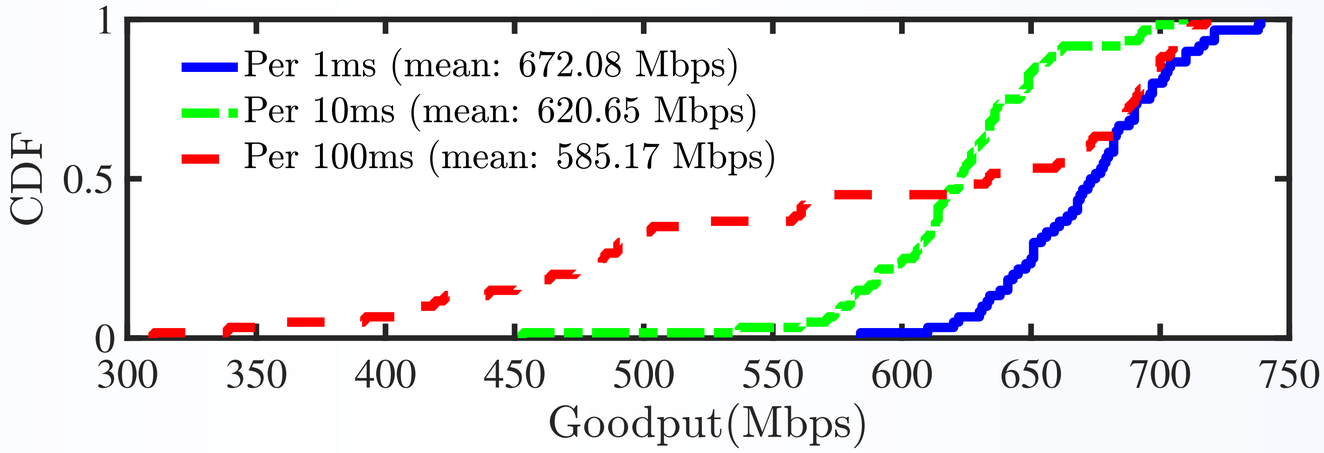
Cross-space communication exists! How to set the communication interval?

Motivation

Fine-grained cross-space communication is necessary for achieving ideal network performance.



Case 1: Single flow with Aurora+CCP. UDP Background Traffic.

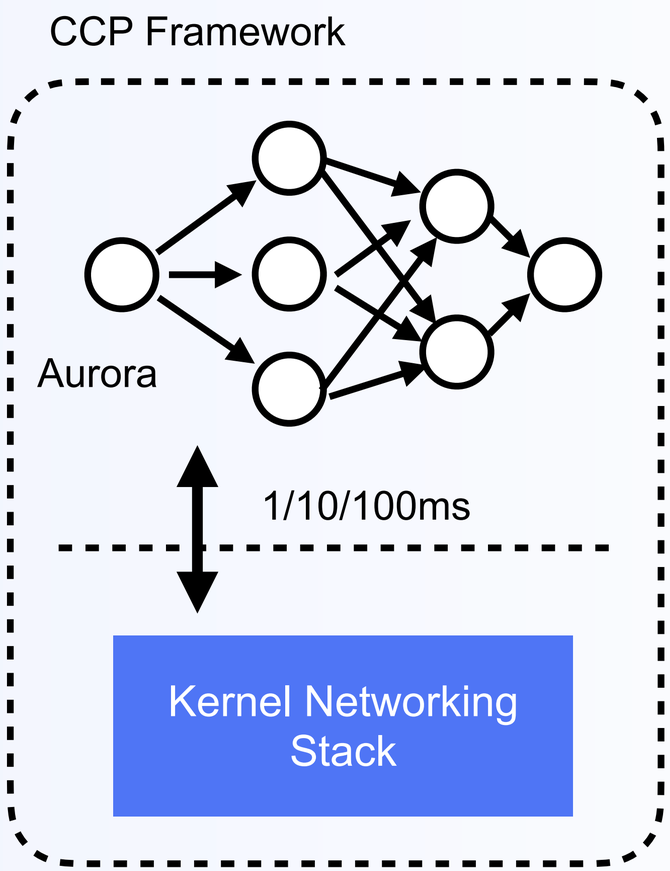


Communication Interval:100ms →10ms →1ms

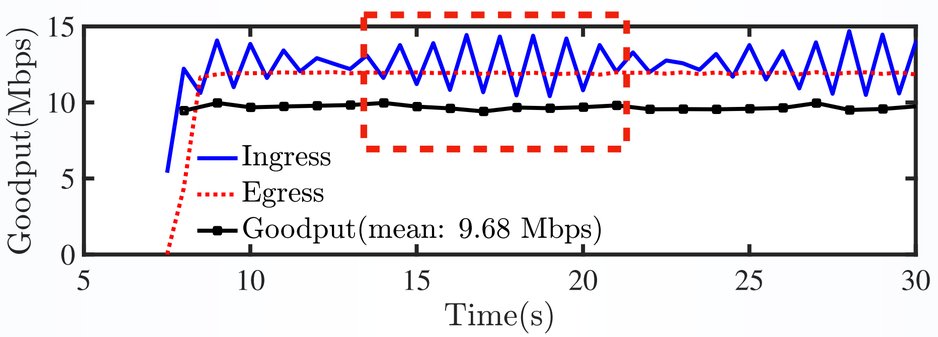
Flow Goodput: 585.17Mbps→ 620.65Mbps → 670.08Mbps

Motivation

Fine-grained cross-space communication is necessary for achieving ideal network performance.



Why? We use a toy experiment to visualize the phenomenon.



10ms communication interval

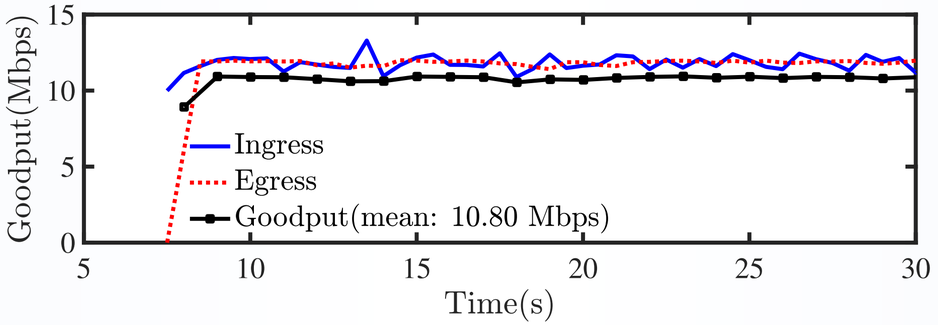
A large control interval makes the flow cannot even converge!

↓

Dramatic queue oscillation

→ Packet loss

→ Degraded goodput

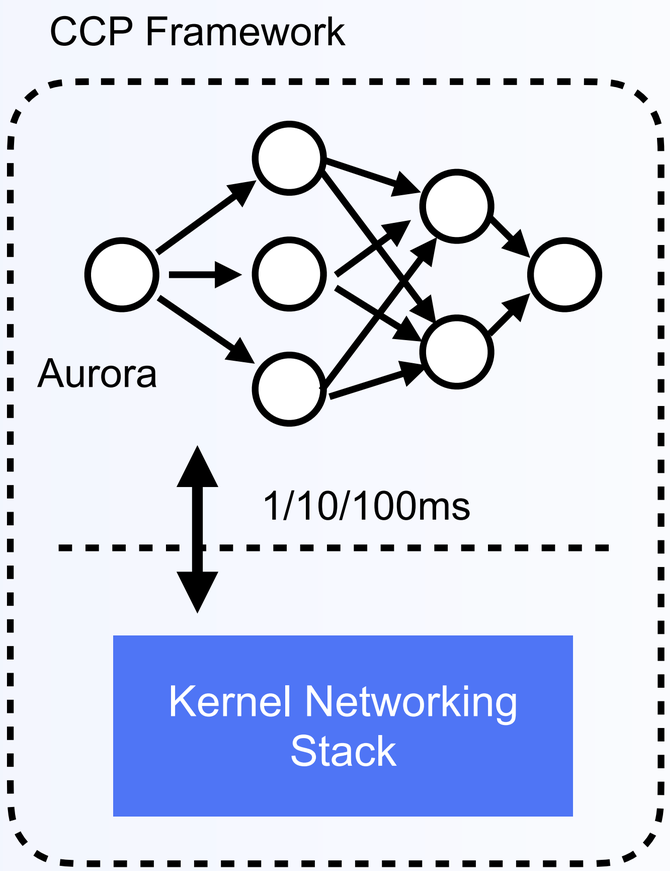


2.5ms communication interval

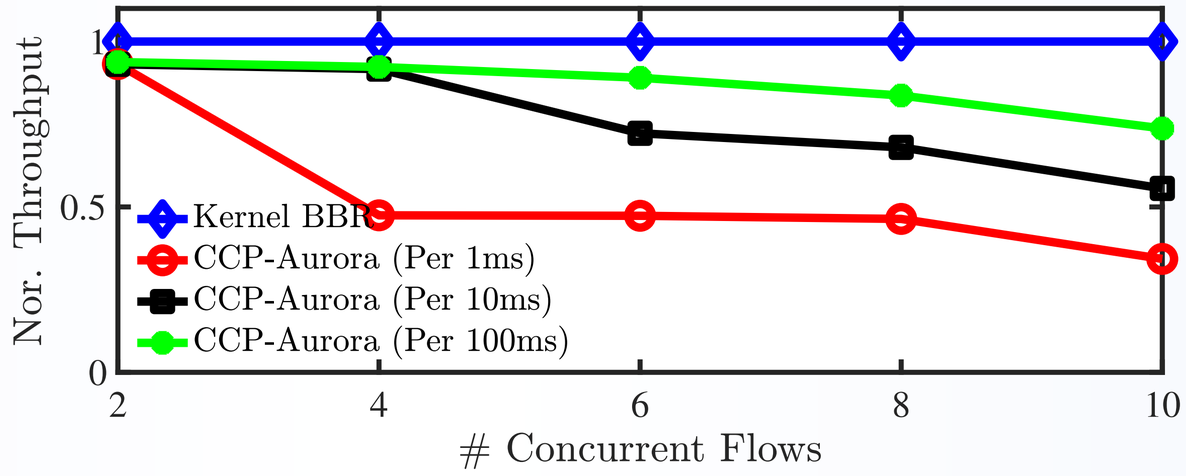
A small control interval does not cause the problem

Motivation

Fine-grained cross-space communication suffers high overhead.



Case 2: Multiple concurrent flow with Aurora+CCP. No Background Traffic. Pure kernel implementation is used as a baseline.

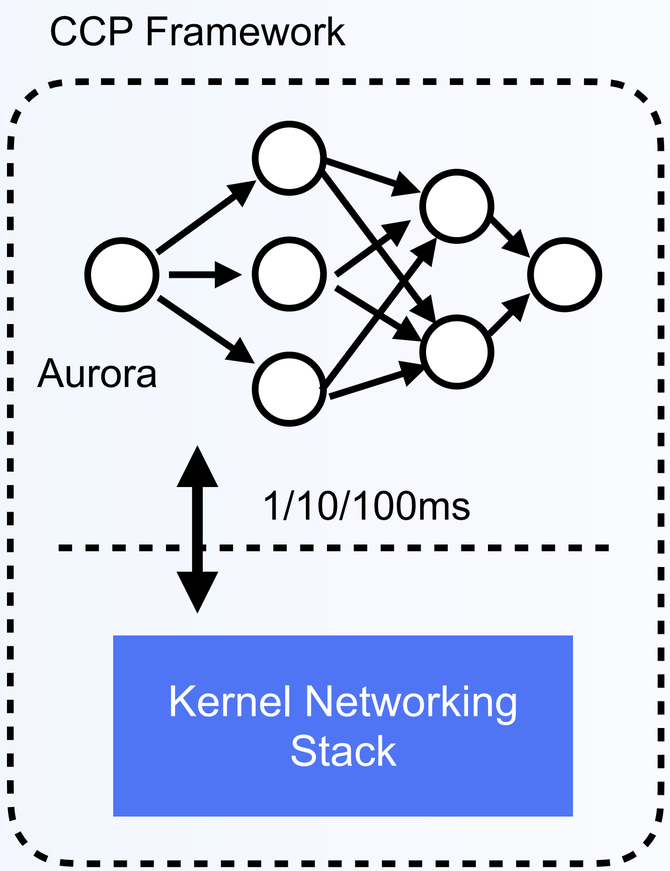


Communication Interval:100ms →10ms →1ms

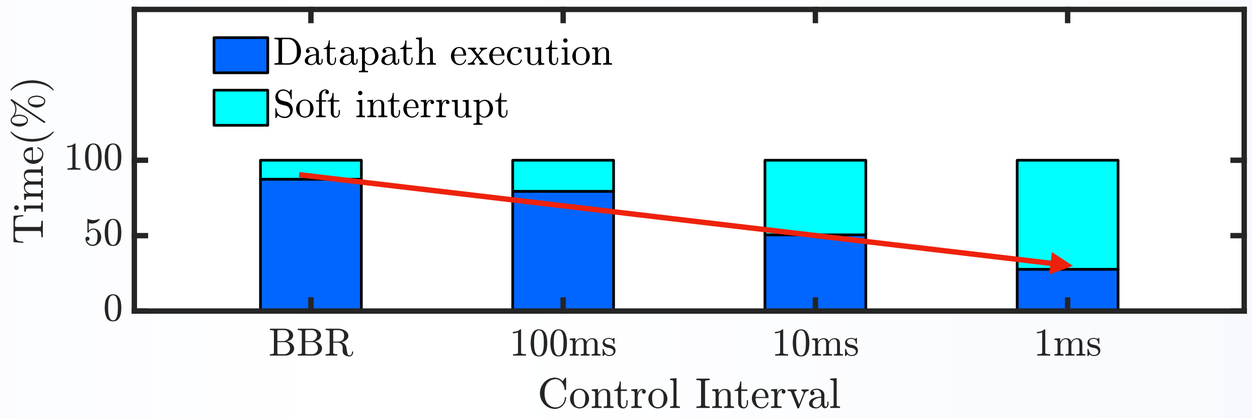
Throughput Degradation: 26.2%→ 44.4% → 65.7%

Motivation

Fine-grained cross-space communication suffers high overhead.



Why? We use *mpstat* to explain the root cause.



Communication Interval: Pure Kernel→100ms →10ms →1ms

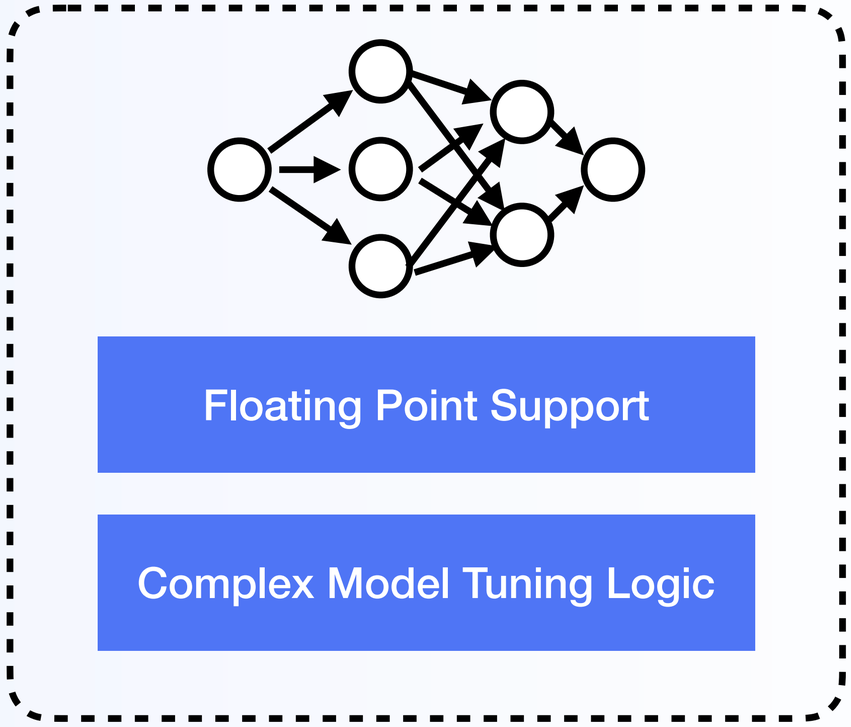
SIRQ/Total execution time: 12.6%→ 20.6% → 49.5% → 72.3%

The userspace deployment of NN suffers from either performance degradation or large overhead

Motivation

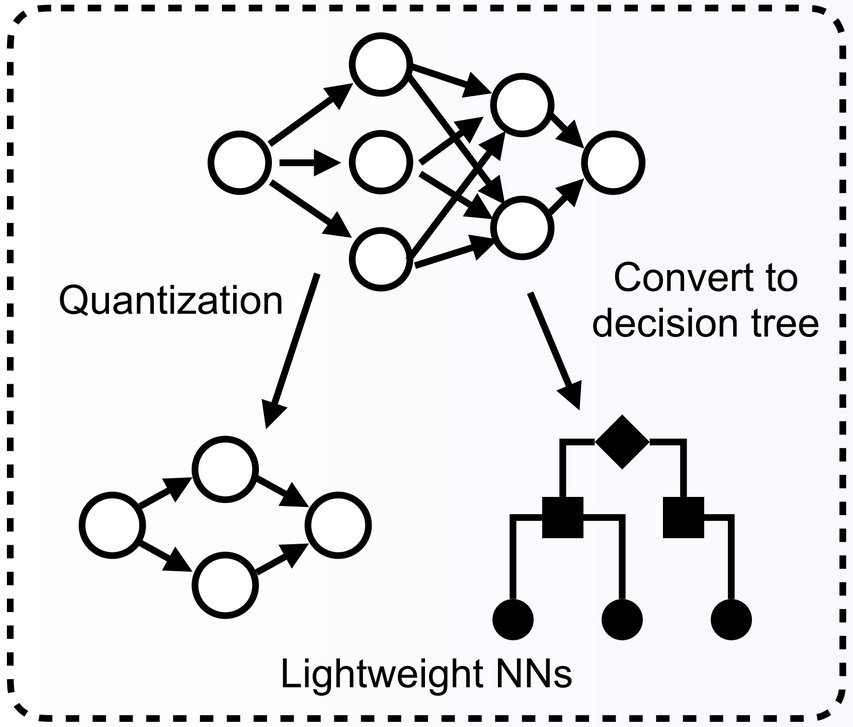
Shall we directly put the NN in the kernel space?

Direct developing adaptive NN in the kernel space



Suffer from enlarged development overhead & performance degradation

Convert the adaptive NN into a lightweight one



Lose the adaptation capacity

Directly deploying adaptive NNs in the kernel space also leads to suboptimal performance due to either large implementation complexity/overhead or lack of adaptation.

Can we design high-performance adaptive NNs for kernel datapath to optimize function performance?

LiteFlow: Towards High-performance Adaptive Neural Networks for Kernel Datapath

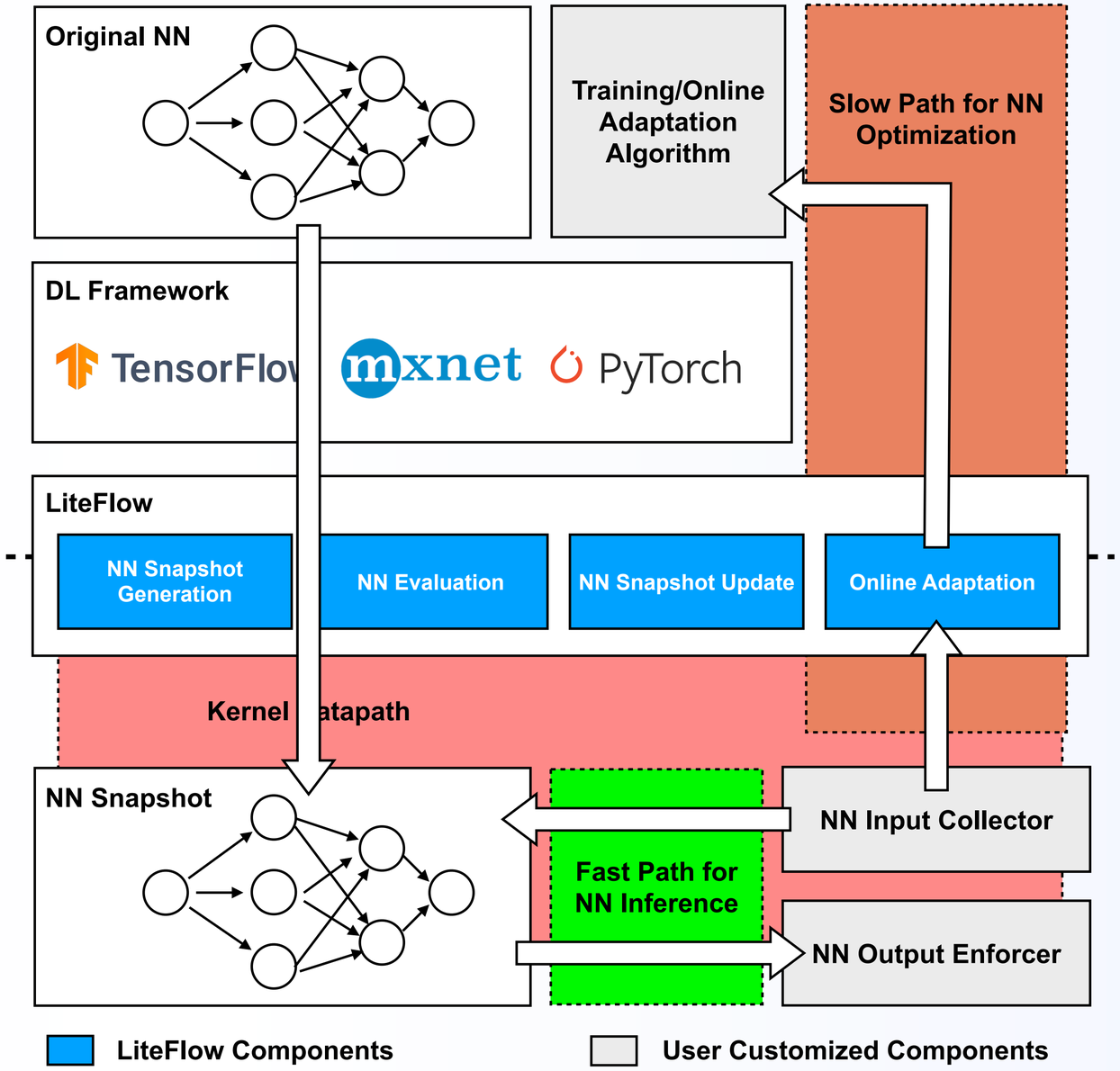
Core idea: LiteFlow decouples model inference from model tuning into two paths

Challenges of decoupling:

- The decoupling method requires two NNs of different design targets
- The NN deployed in the kernel-space cannot timely adapt to the changing network environment, degrading the function performance
- Tuning the userspace-deployed NN requires continuously delivering data from kernel space to userspace, which again causes massive cross-space communication

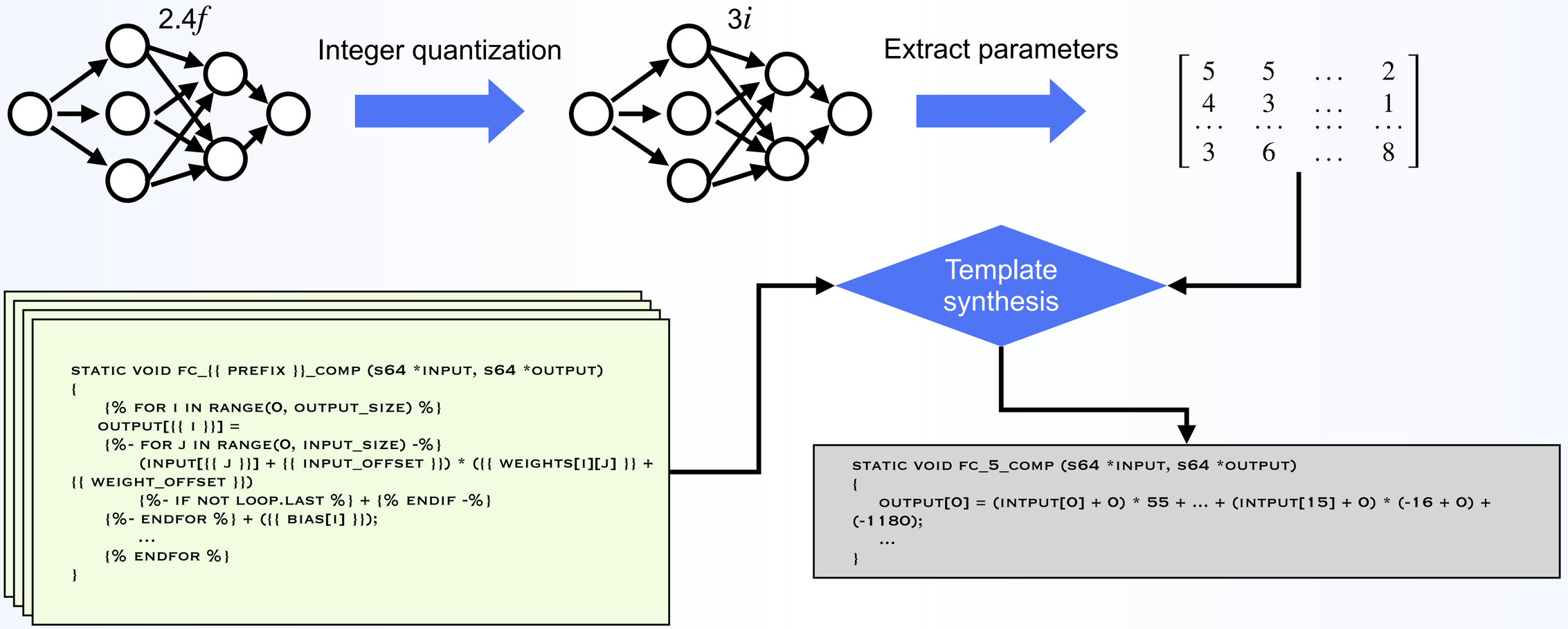
LiteFlow in One Slide

- A cross-space *hybrid* solution
- LiteFlow *automatically* generates a NN snapshot to deploy in the kernel space.
- The snapshot is used for inference only.
- LiteFlow *batch-wisely* re-trains the userspace-deployed NNs
- LiteFlow *conservatively* update the NN by evaluating both the correctness and necessity

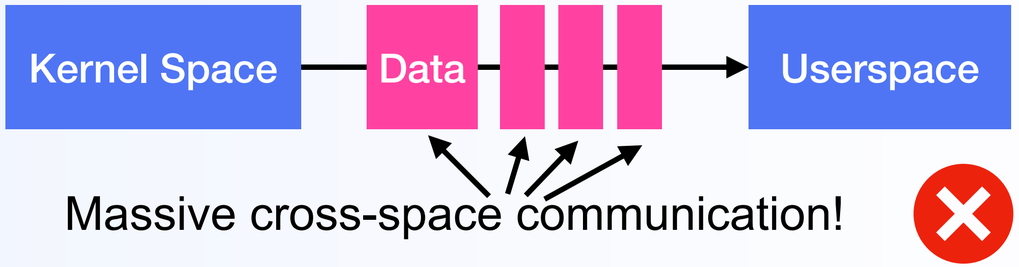


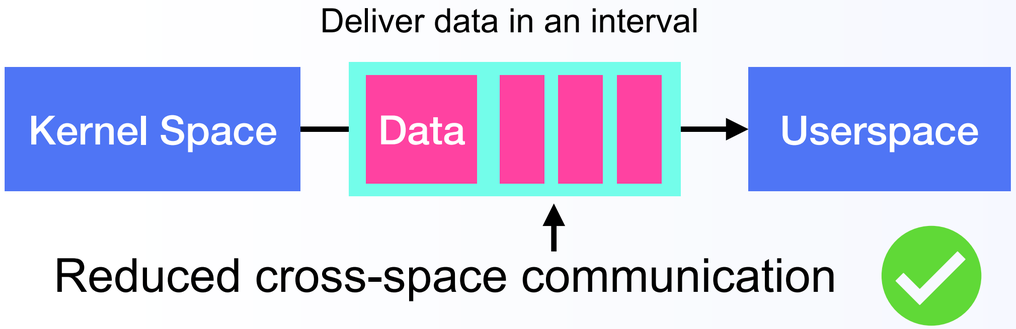
LiteFlow — Snapshot Generation

1. To avoid the overhead of using SIMD/FP instructions → Integer quantization with high accuracy
2. To make the snapshot kernel-compatible → Code translation to convert the NN into a kernel module



LiteFlow — Online Adaptation

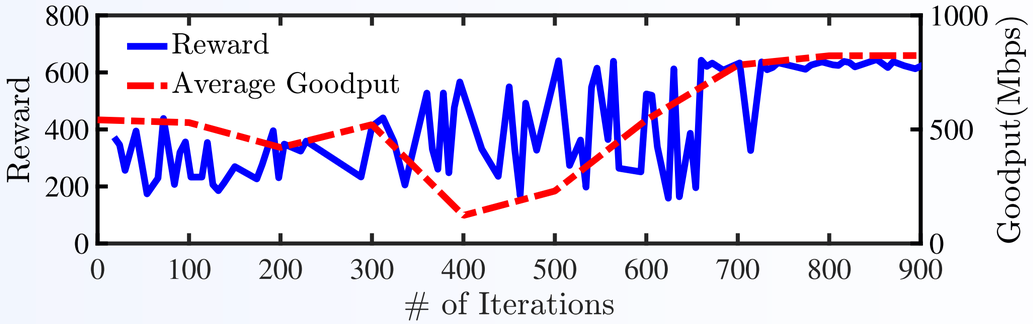




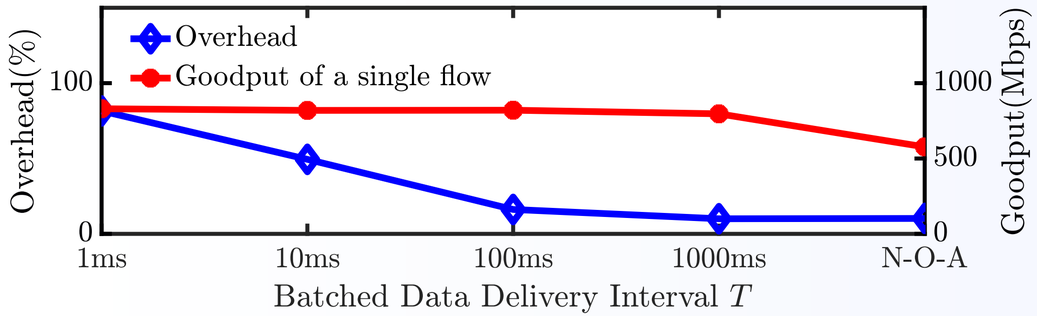
Q1: Why it works?

Observation 1: The network characteristics usually do not change in a sub-second manner → Do not need very frequent NN update

Observation 2: When the NN is performing online adaptation, it does not always lead to better performance before convergence → Periodically update the NN leads to better/stable performance



Q2: How to determine the interval?



A small interval → large overhead caused by cross-space overhead that is similar to previous discussion

A large interval → the NN loses the ability to learn and adapt

Experiment results show 100ms ∼ 1000ms is a good interval for most cases

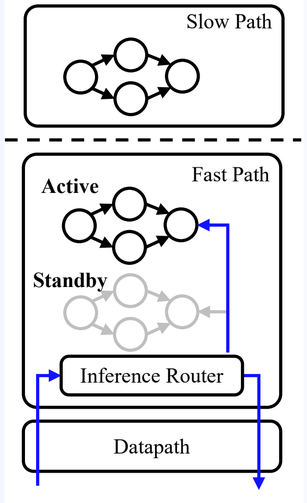
LiteFlow — NN Evaluation & Update

When should we update the NN?

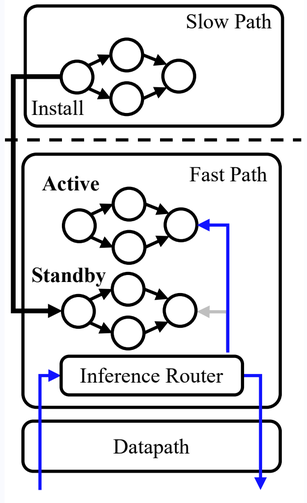
- Correct → After NN tuning converges
- Necessary
  - The difference between snapshot & NN in the userspace is large enough
  - Fidelity loss: $L(x) = |f(x) - f'(x)|$

How to update the NN?

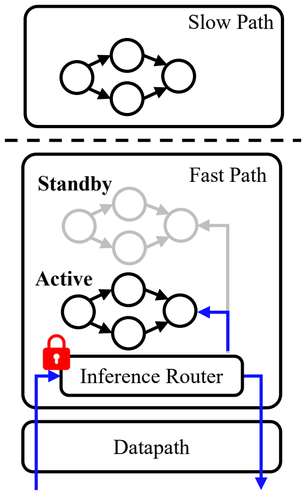
- Properly handle kernel locks → Avoid potential performance interference during NN update



Maintains both an active and standby snapshot



Install a new snapshot by replacing the standby one



Change the role of the two snapshot to avoid long lock

Please refer to our paper for more details: parameter choice & flow consistency

LiteFlow — Implementation

Userspace implementation

- A set of Python interfaces for users to implement

NN Freezing Interface:

- To implement the interface, users should save the model and return the path to the saved model.

NN Evaluation Interface:

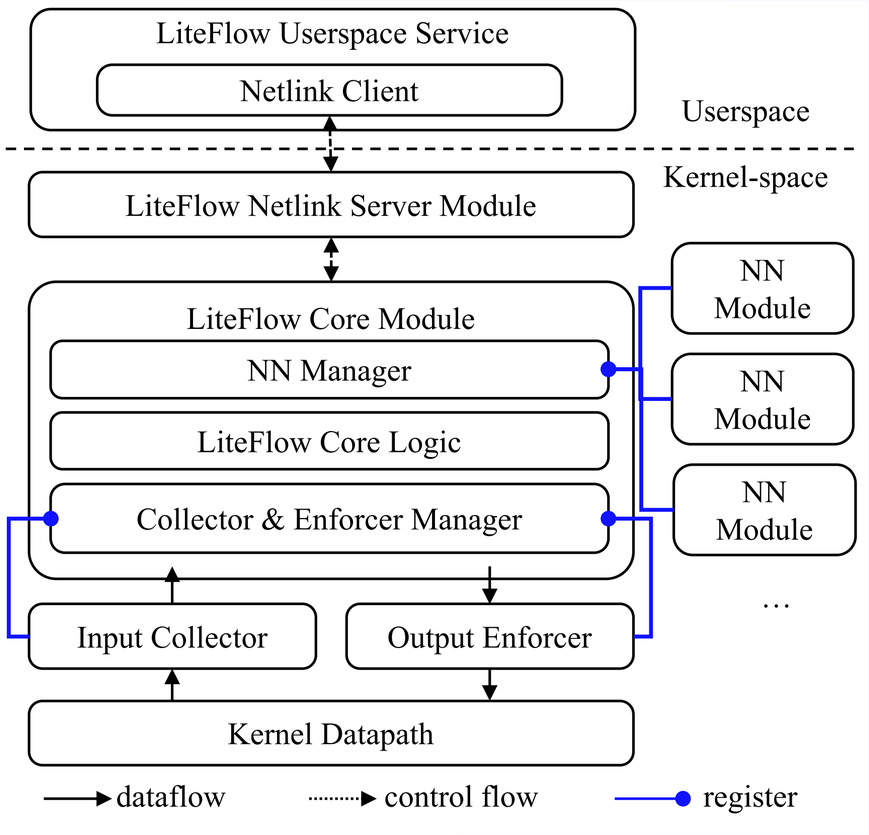
- The interface requires LiteFlow users to (1) return the stability value and (2) calculate the output when given a set of input data.

NN Online Adaptation Interface:

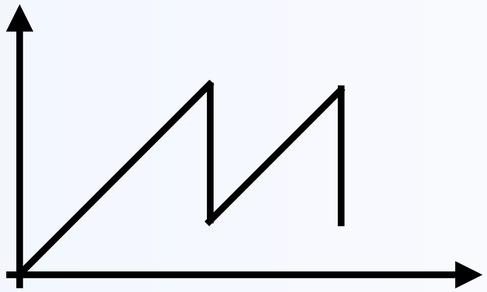
- To implement the interface, LiteFlow users have to include the scripts/programs to tune the NNs.

Flexible to be extended to support various NN-optimized datapath functions

Kernel-space implementation



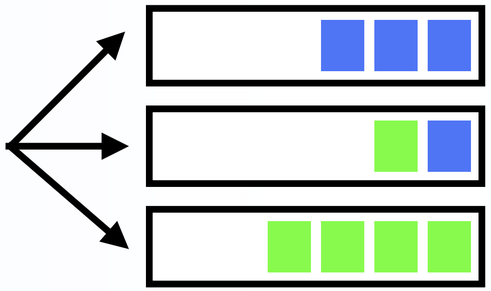
LiteFlow Application



Congestion Control

Aurora, ICML19; MOCC, EuroSys 22

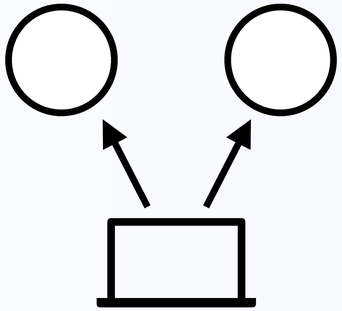
The NN is used to decide flow sending rate to handle network congestion



Flow Scheduling

FFNN, NSDI 19

The NN is used to predict the flow size, which will be used in flow scheduling

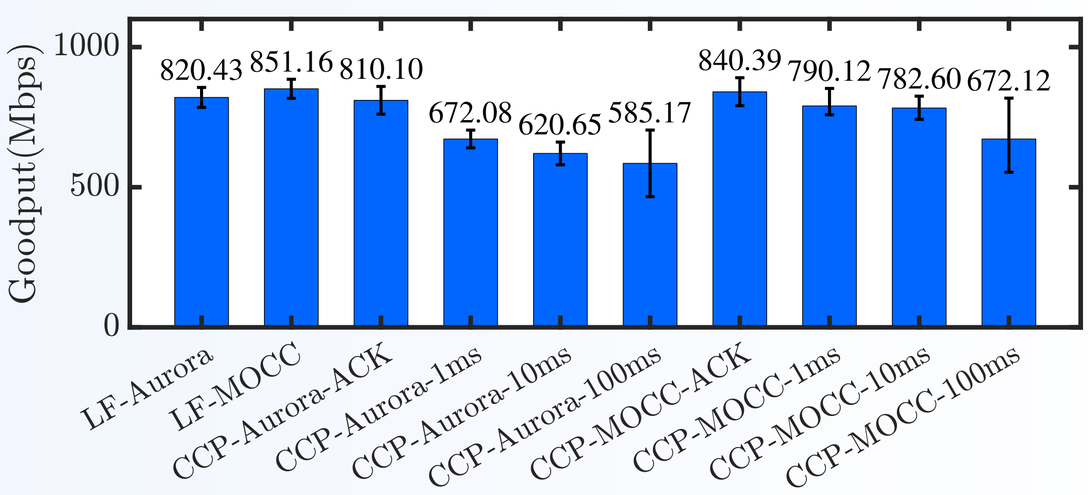


Load Balance

We design a MLP NN

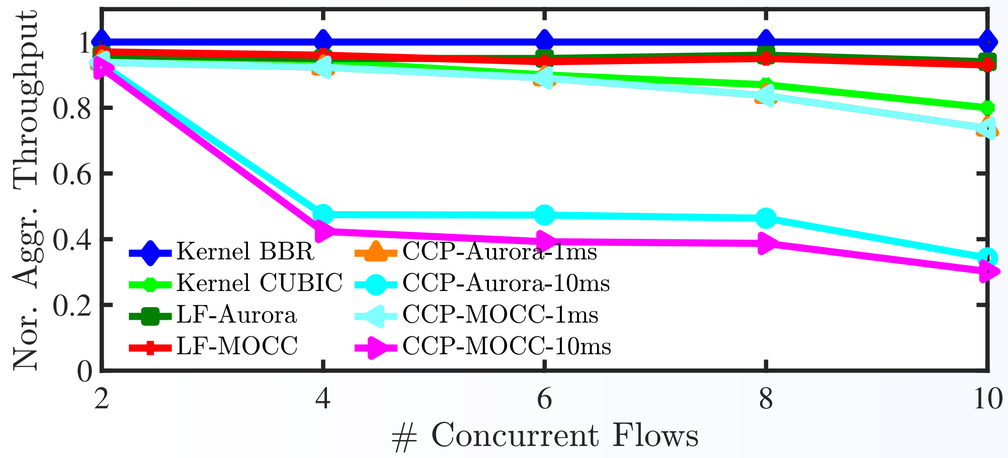
The NN is used to select the path at the endhost to achieve load balancing

LiteFlow Application — Congestion Control



- Flows controlled by LF-Aurora and LF-MOCC achieve higher goodput than those controlled by CCP-Aurora and CCP-MOCC
- The standard deviation of goodput achieved by LF-Aurora and LF-MOCC is much smaller than CCP-Aurora and CCP-MOCC with large communication intervals

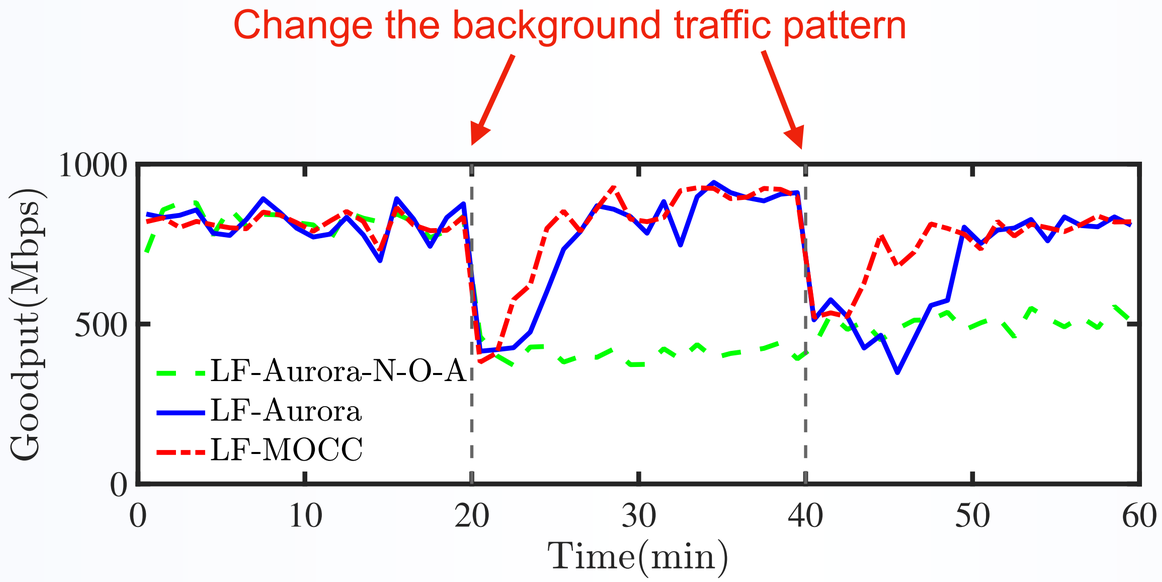
By eliminating the cross-space communication, LiteFlow makes the NNs more responsive, thus leading to better and more stable performance



- LF-Aurora and LF-MOCC achieve comparable performance to kernel BBR (the performance loss is <5%) and outperform CUBIC by 17.5%
- LF-Aurora and LF-MOCC largely outperform CCP-Aurora and CCP-MOCC by up to 63.5%

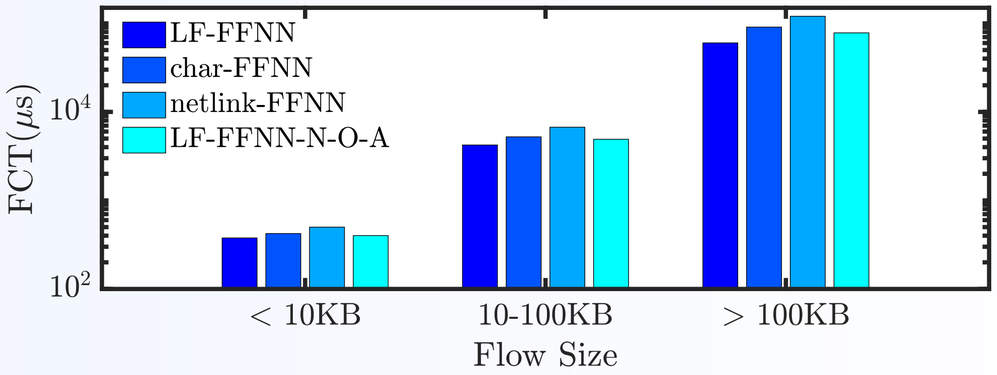
By eliminating the overhead caused by cross-space communication, LiteFlow with NNs can achieve comparable performance to pure kernel-space implementations

LiteFlow Application — Congestion Control

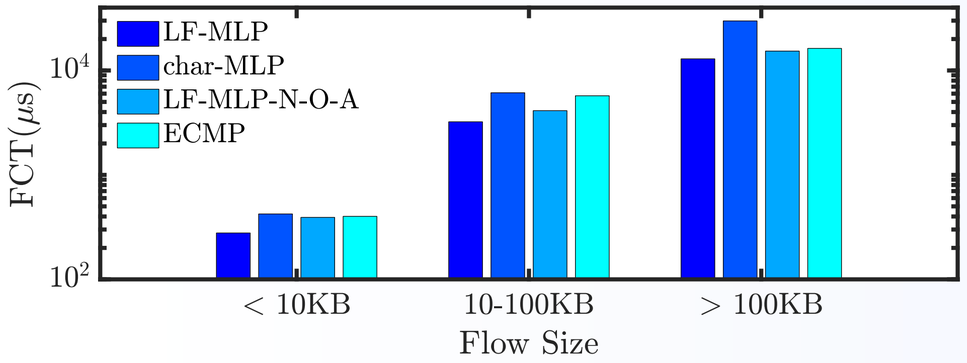


- Compared to LF-Aurora-N-O-A (No Online Learning), which has a dramatic goodput drop when the environment changes, LF-Aurora and LF-MOCC can learn and adapt to such dynamics, thus achieving significantly better goodput

LiteFlow Application — Flow Scheduling & Load Balance



Flow Scheduling



Load Balance

- For flow scheduling, LiteFlow can largely outperform the other two userspace-deployed solutions in all cases and achieve up to10.9% and 33.7% better FCT for short flows and long flows respectively
- For load balance, LiteFlow can largely outperform the other two userspace-deployed solutions in all cases and achieve up to 34.3% and 56.7% better FCT for short flows and long flows respectively
- Please see our paper for more details

Conclusion

Problem: We point out that existing deployment of adaptive NN for kernel datapath suffers from inevitable performance degradation

Solution: To solve the problem, we propose LiteFlow, a hybrid solution to deploy the adaptive NN for the kernel datapath, thus both NN inference & tuning can be at the right place

Application: We have applied LiteFlow in 3 real-world applications and evaluation results show that LiteFlow can meet all its design goals

Thank You!

Contact: jzhangcs@connect.ust.hk