Central South University

Polo: Receiver-Driven Congestion Control for Low Latency over Commodity Network Fabric

Chang Ruan,
Jianxin Wang, Wanchun Jiang, Tao Zhang

Outline

- Introduction
- Motivation
- Design
- Evaluation

INTRODUCATION

- ➢ Many applications require low latency in data center networks
  - Web search, retail recommendation system
- ➢ Existing low latency protocols
  - Sender-driven protocols, e.g, DCTCP

    timeout due to highly concurrent flows, queueing at switch
  - Receiver-driven protocols , e.g, NDP and Homa (show better performance)

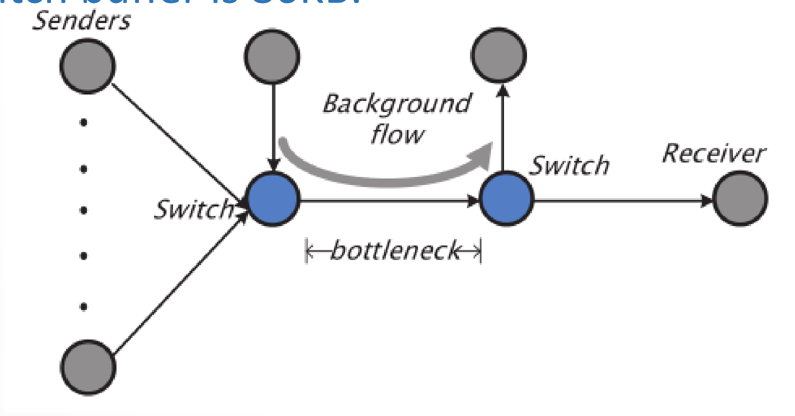
    Assuming the core layer has no congestion or requiring switch modification

MOTIVATION

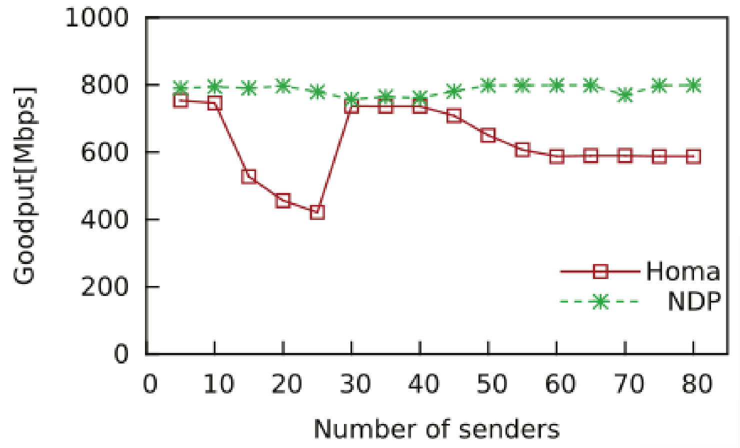
Problem 1: Core layer is congested (oversubscription, switch failure)

➢ Intermediate link (at the core layer) is the bottleneck

- The link bandwidth is 1Gbps, background flow occupies 200Mbps, RTT is 50us. Many Senders send data to one receiver. Switch buffer is 86KB.



- Goodput of Homa and NDP



- Cause:

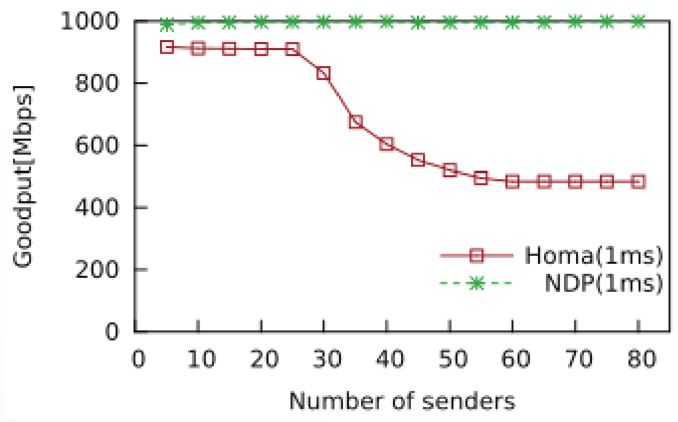
  Senders send data at the line rate, which is larger than the bottleneck link rate. Besides, the receiver sends grants packets back also with the line rate. Homa relies on timeout retransmission, while NDP can retransmit the packets quickly after trimming the packet to the header.

MOTIVATION

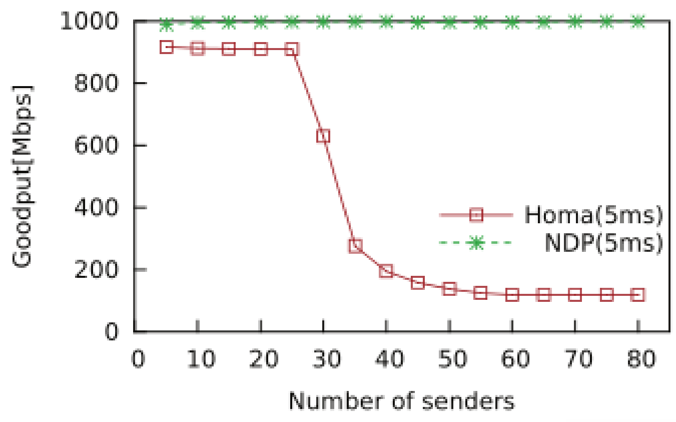
Problem 2: Edge layer can be also congested

➢ Synchronized senders send data

- The link bandwidth is 1Gbps, RTT is 50us. Many Senders send data to one receiver. Switch buffer is 86KB.
- Goodput of Homa and NDP



Timeout retransmission time is 1ms



Timeout retransmission time is 5ms

- Cause:

  Packet losses occur at the last hop due to the highly concurrent flows. Homa relies on timeout retransmission, while NDP can retransmits the lost packets quickly after trimming the packet to the header.

Design

- How do we achieve the low latency in these cases for a receiver-driven protocol?

- Recovering the lost packet quickly
- As the bottleneck link bandwidth is not known a priori, the protocol should keep the queue at the bottleneck switch small to reduce the queueing delay
- Specially pay attention for highly concurrent communication pattern, where a flow may not send an entire packet in a RTT
- Readily deployment.

Design

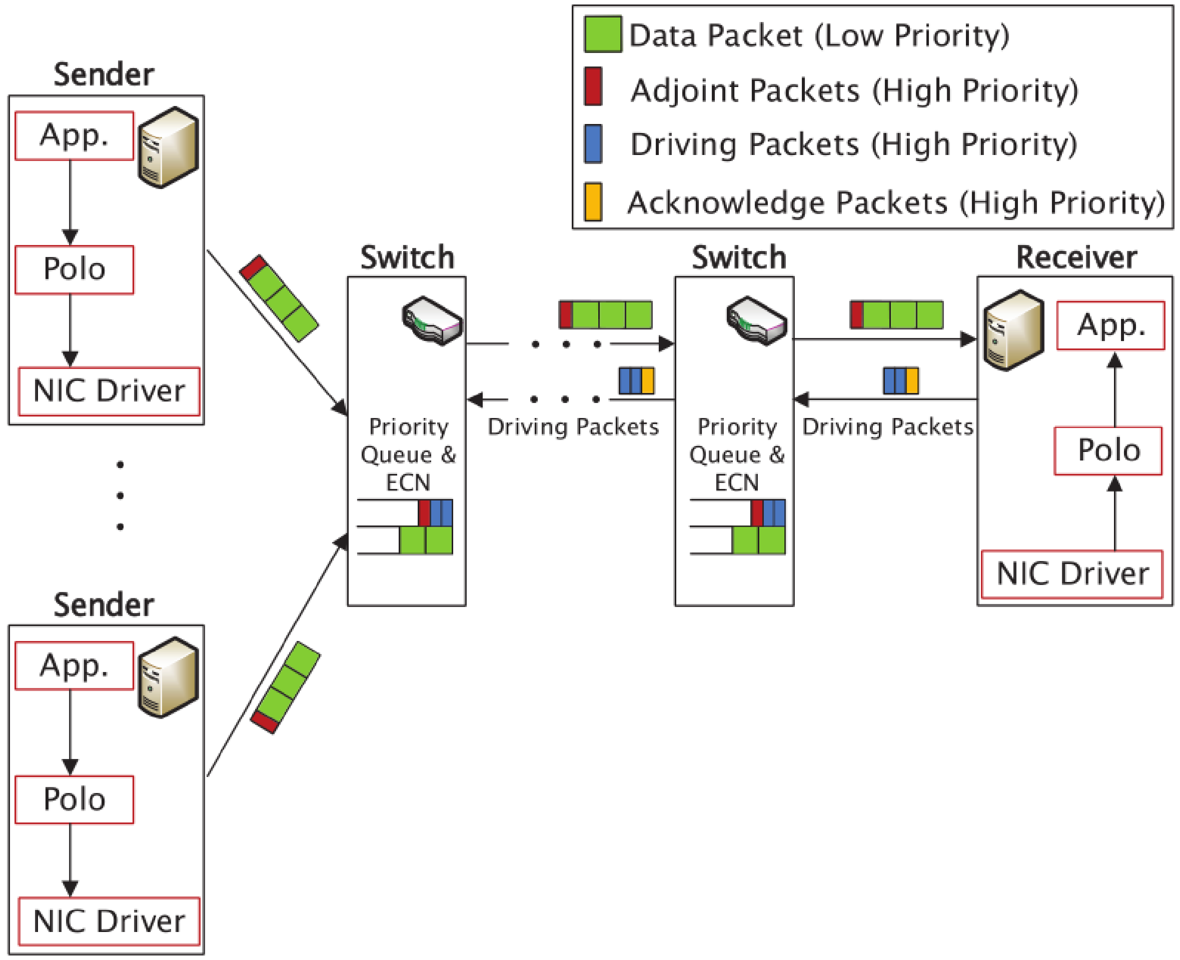
➢ Our solution: Polo, a receiver-driven protocol. We use priority queue and ECN to achieve the target.

- Priority queue is used for:
  - i) Recovering the lost packet quickly
  - ii) Optimization for highly concurrent communication pattern
- ECN is used for:
  - keep the small queue by adjusting the number of driving packets dynamicall

Supported by commodity switches

Design

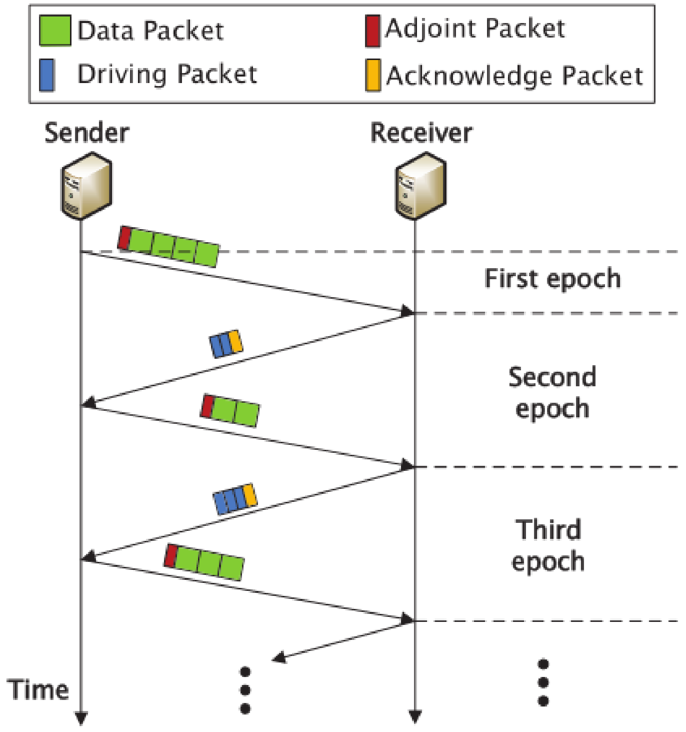
➢ Overview



- Each sender sends data packet at the line rate, following by an adjoint packet with high priority
- Switch marks the packet with ECN if the queue length surpasses a threshold
- Receiver feeds a driving packet back for the received data packet and the total number of the driving packets is adjusted dynamically .

Design

➢ Determining the adjustment epoch



- Each sender sends an adjoint packet with the packet header size and the receiver feeds an acknowledge packet back corresponding to this adjoint packet. The ping-pong packets define the adjustment epoch of dynamically adjusting the number of driving packets.

Design

➢ Adjusting the number of driving packets

- The Polo receiver maintains a variable *D* to record the number of driving packets in each epoch.

  In the initialization, *D*=0 .

Increase:

- In the first epoch

  D ← D+1 for each packet without ECN mark

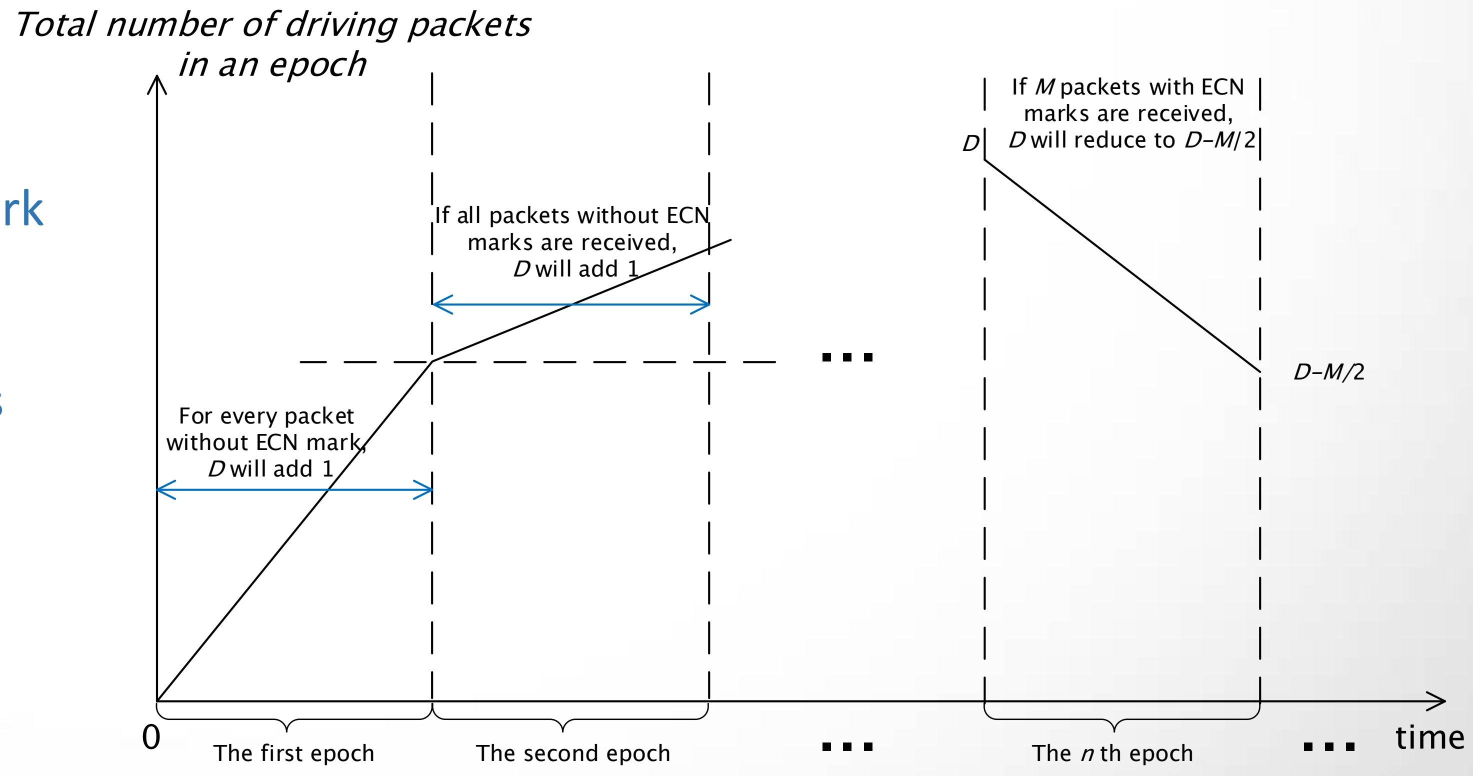
- In the second or subsequent epoch

  D ← D+1 if all packets carry no ECN marks

Decrease:

- In an epoch:

  D ← D-M/2 if M packets carry ECN marks



Total number of driving packets in an epoch
If all packets without ECN marks are received, D will add 1
For every packet without ECN mark, D will add 1
If M packets with ECN marks are received, D will reduce to D−M/2
D
D−M/2
...
0
The first epoch
The second epoch
The n th epoch
time

We want to mimic the AIMD principle of TCP for steadily

Design

➢ Recovering the lost packet quickly

- **Recovery mechanism 1:** relies on the sequence gap

  The receiver detects packet loss according to the gap between max seen sequence and the expected next sequence. If there is a gap, Polo returns a loss packet to the corresponding sender for retransmitting the lost packet.

- **Recovery mechanism 2**: relies on the epoch determined by the adjoint packet

  Polo will send a loss packet to a random active flow if two epochs pass and the receiver does not receive any data packet of all active flows.

- **Recovery mechanism 3**: relies on timeout retransmission

  If the adjoint packet is lost, Polo relies on the timeout retransmission, e.g., 1ms.

Design

Optimization for highly concurrent flows

Problem:

In the Incast scenario, even if each of these flows only send one packet, the switch buffer will overflow.

Method:

Polo designs the pause mechanism to suspend the sending of part of flows

- Active flow:

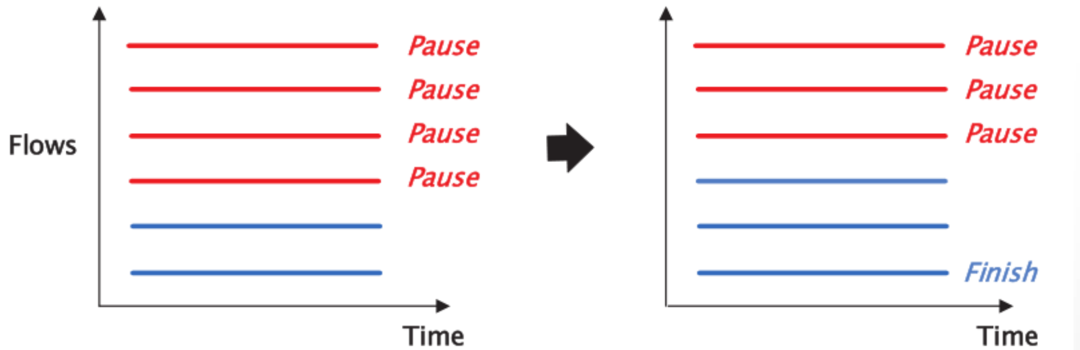
  In the beginning, before the receiver receives any packet with ECN mark, the flow whose packet arrives at the receiver is called the active flow.

- Inactive flow:

  Other than active flows, other flows are called inactive flows. They are paused temporally.

Inactive flow → Active flow

The, if an active flow is finished, an inactive flow is switched to the new active flow.



Evaluation

- **Scheme**.

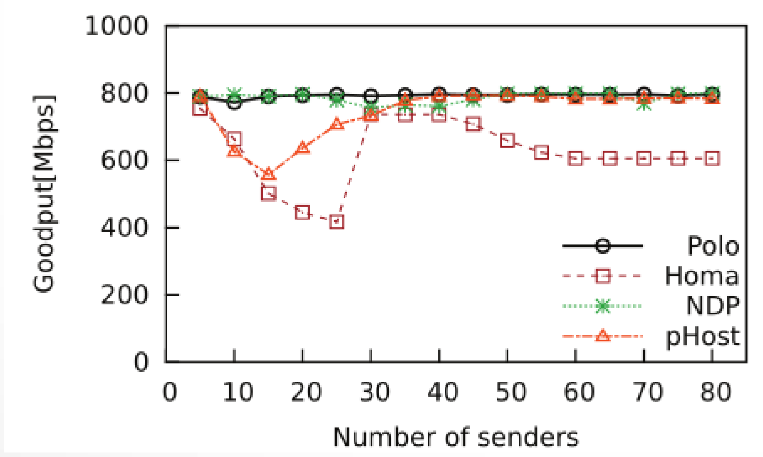
  Homa: uses 8 priority queues, the degree of overcommitment is 2. Packet spraying is used for packet forwarding. Timeout retransmission time is 1ms. RTTBytes is 12.

  NDP: uses 2 priority queues. Timeout retransmission time is 1ms. Initial window is 12.

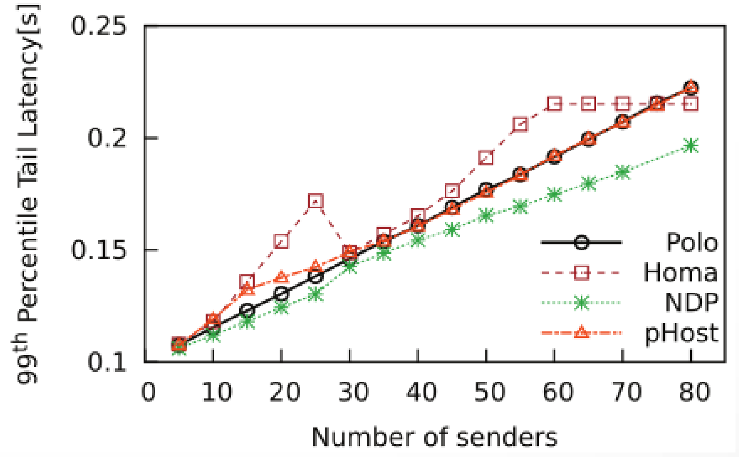
  pHost: schedules flow in a round robin way. free number of tokens is 12.

➢ Microbenchmark

- **Intermediate link is the bottleneck**



goodput

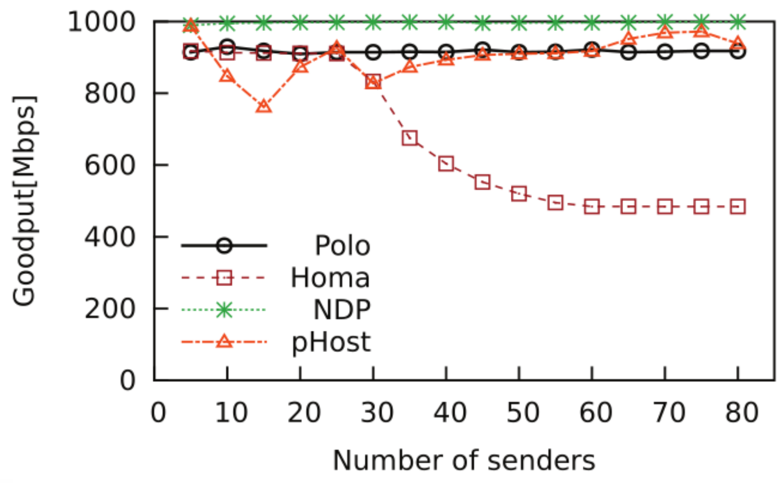


99th percentile tail latency

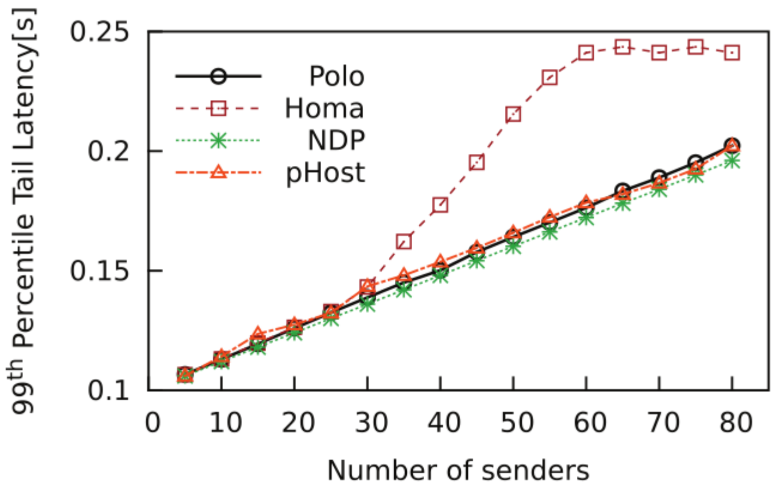
With the number of senders increasing, Polo still has goodput close to full available bandwidth and 99th percentile tail latency is close to pHost

Evaluation

- Edge link is the bottleneck



goodput

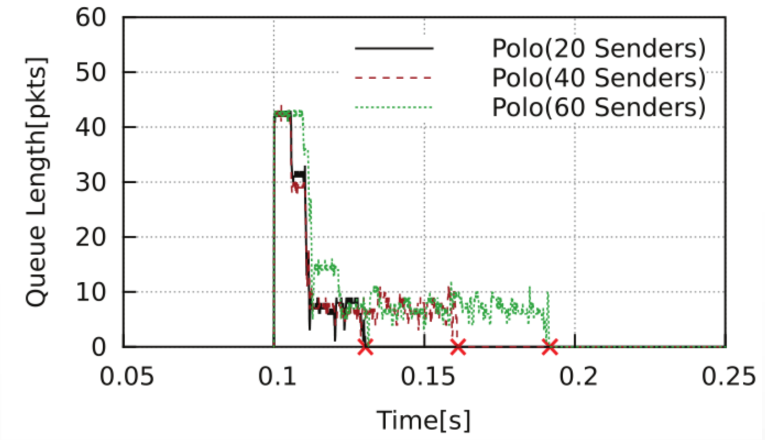


99th percentile tail latency

With the number of senders increasing, Polo's goodput is between NDP and Homa. Its 99th percentile tail latency has the same trend.

- Controlling the queue well

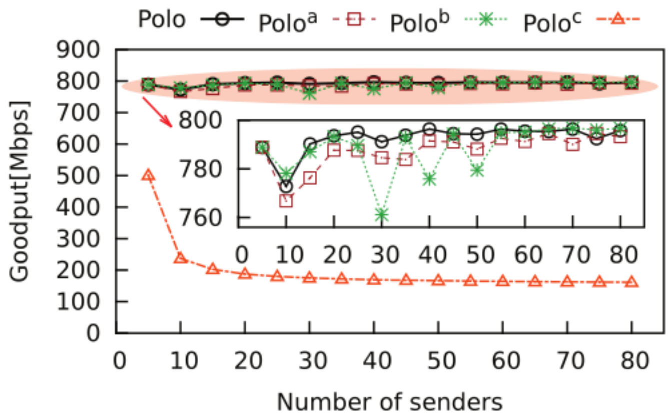
  ✓ Many-to-one scenario, 1Gbps link
  ✓ ECN threshould is 5
  ✓ Each flow has 100KB, starting at 0.1s



Evaluation

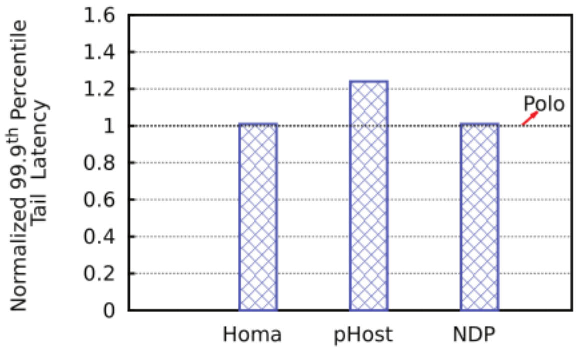
- Role of recovery mechanism

✓ Polo^b denotes Polo without the recovery mechanism 2
✓ Polo^c denotes Polo without the recovery mechanism 1
✓ Polo^a denotes Polo without the optimization mechanism for the wasted driven packets

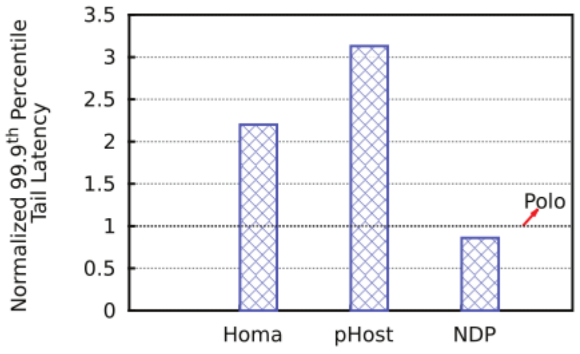


➢ Larger scale simulation

✓ Leaf-spine topology , network with the over-subscribed bandwidth, each leaf switch connect to 25 hosts
✓ Data mining and web search
✓ Workload is 0.5



(a) Web search.

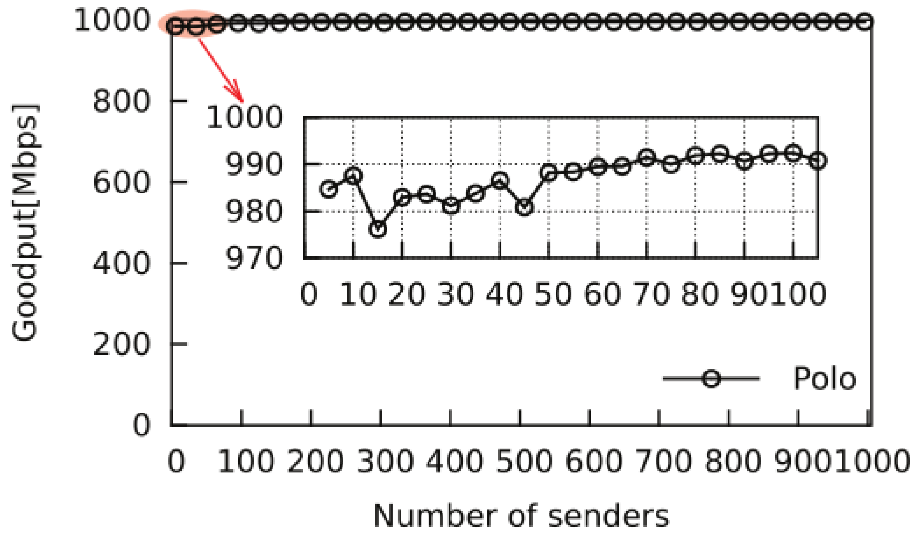


(b) Data mining.

Since Polo can recover the lost packet faster than Homa and pHost, Polo improves the tail latency by 2.2× and 3.1×, respectively.

Evaluation

- ➢ Large scale Incast
  - ✓ 1Gbps link
  - ✓ Each flow has 100KB size



Since Polo can pause flows, Polo always keeps high goodput until 1000 senders or even larger number of senders.

Thank you for your attention