

Advanced Computer Networks

Flow Scheduling in Data Center Networks (II)

<https://pages.cs.wisc.edu/~mgliu/CS740/F26/index.html>

Ming Liu
mgliu@cs.wisc.edu

Outline

- Last lecture
 - Flow scheduling in data center networks (I)
- Today
 - Flow scheduling in data center networks (II)
- Announcements
 - Project proposal due 10/12/2026 11:59 PM
 - Lab1 due 10/09/2026 11:59 PM

Unsolved Issues in Hedra

- #1: Networking bandwidth capacity (supply) is ephemeral!
- #2: Flow demand and flow # are unpredictable!
- #3: Real-time global view is hard to maintain!

How does CONGA help?

CONGA Technique #1

- #1: Networking bandwidth capacity (supply) is ephemeral!
- #2: Flow demand and flow # are unpredictable!
- #3: Real-time global view is hard to maintain!

Discounting Rate Estimator (DRE)

Discounting Rate Estimator (DRE)

- Per-egress port register (R)
 - Additive increase: $R = R + \text{packet_size}$, triggered every packet
 - Multiplicative decrease: $R = R \times (1 - \alpha)$, triggered periodically (T_{dre})

Discounting Rate Estimator (DRE)

- Per-egress port register (R)
 - Additive increase: $R = R + \text{packet_size}$, triggered every packet
 - Multiplicative decrease: $R = R \times (1 - \alpha)$, triggered periodically (T_{dre})
- First-order low pass filter applied to packet arrival
- React quickly to traffic bursts
 - Increments take immediately upon packet arrival

CONGA Technique #2

- #1: Networking bandwidth capacity (supply) is ephemeral!
- #2: Flow demand and flow # are unpredictable!
- #3: Real-time global view is hard to maintain!

CONGA Technique #2

- #1: Networking bandwidth capacity (supply) is ephemeral!

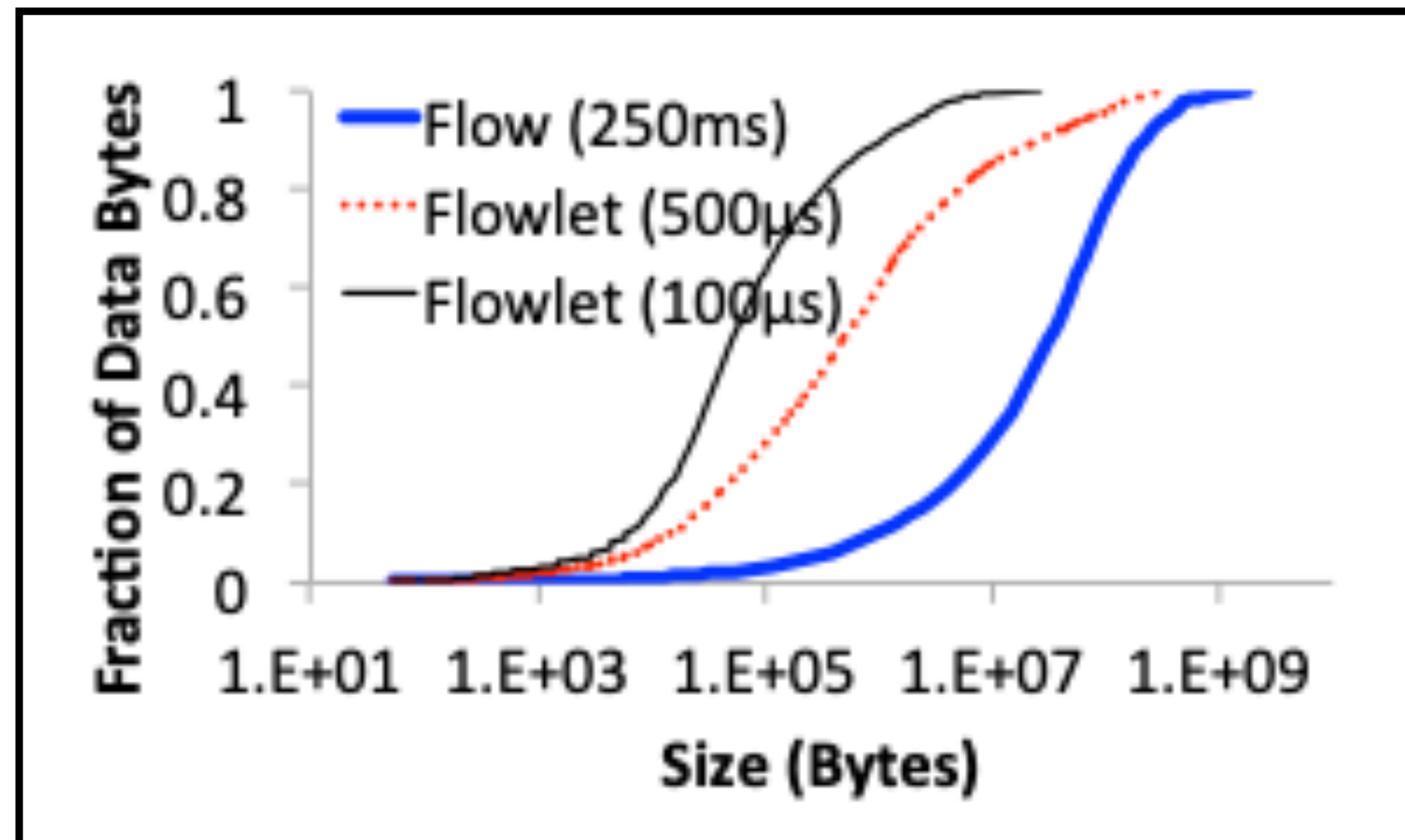
- # But,
 - How large is a flow?
 - What is a flow size distribution?
- #
 - How many concurrent flows?
 - ...

Flowlet

- Bursts of packets from a flow separated by large enough gaps
 - “gap” is defined by time

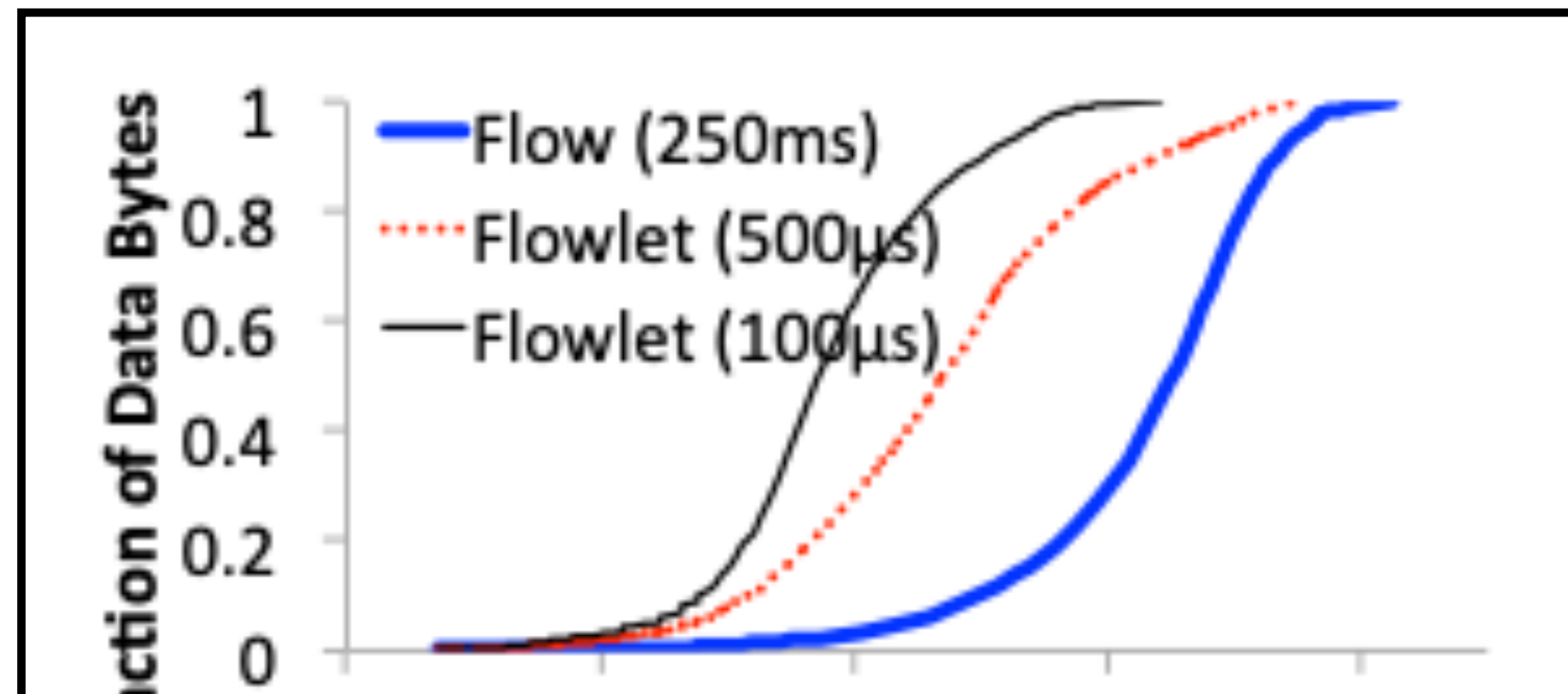
Flowlet

- Bursts of packets from a flow separated by large enough gaps
 - “gap” is defined by time



Flowlet

- Bursts of packets from a flow separated by large enough gaps
 - “gap” is defined by time



- 250ms: 50% of bytes are in a flow (> ~30MB)
- 500us: 50% of bytes are in a flow (> ~500KB)
- Fine-grained load balancing is possible

How do we detect a FlowLet?

Flowlet Detection

- “Smart” edge switch
 - FlowLet ID = Hash (5-tuple)
 - Port Number: forwarding port
 - Valid Bit: 1 if the FlowLet is active
 - Age Bit: 1 if the entry is expired

FlowLet ID	Port Number	Valid Bit	Age Bit
1234	4	1	1
5678	5	0	0

Flowlet Detection

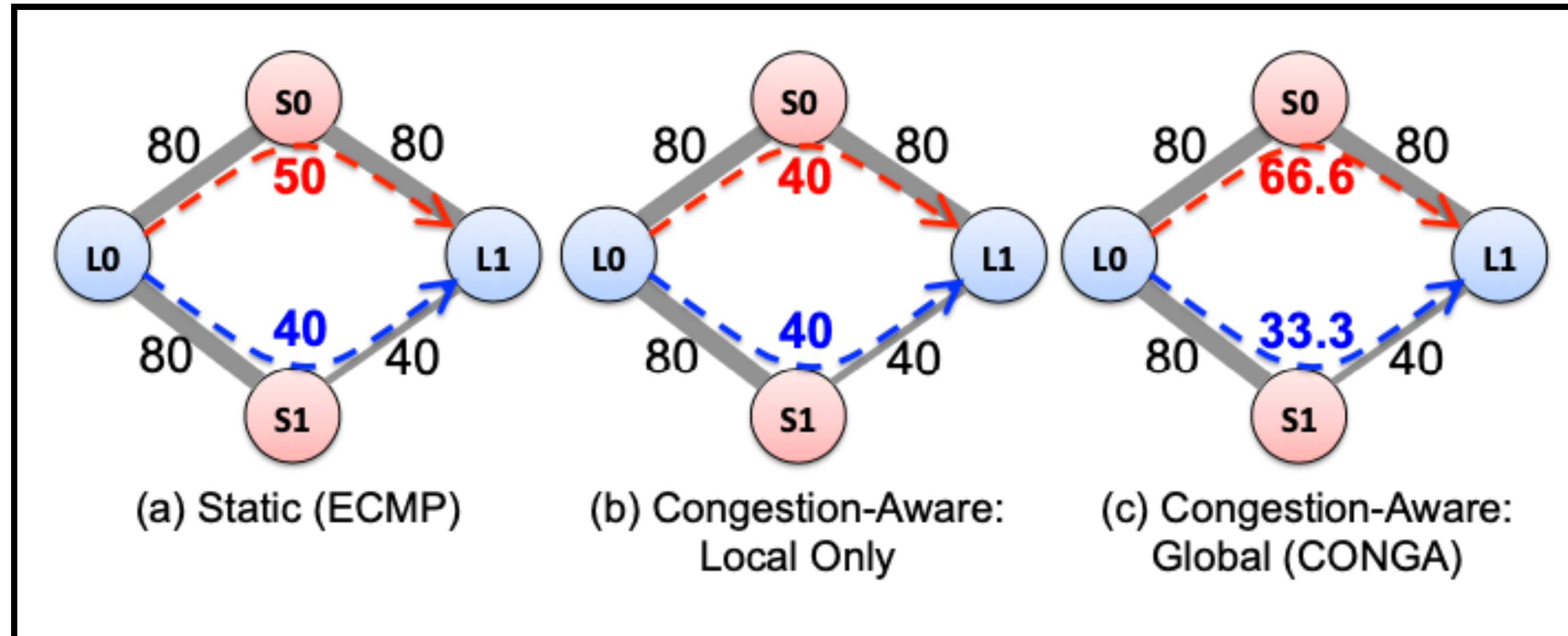
- “Smart” edge switch
 - Flowlet ID - Hash (5 tuple)

Is this enough?

- Age Bit: 1 if the entry is expired

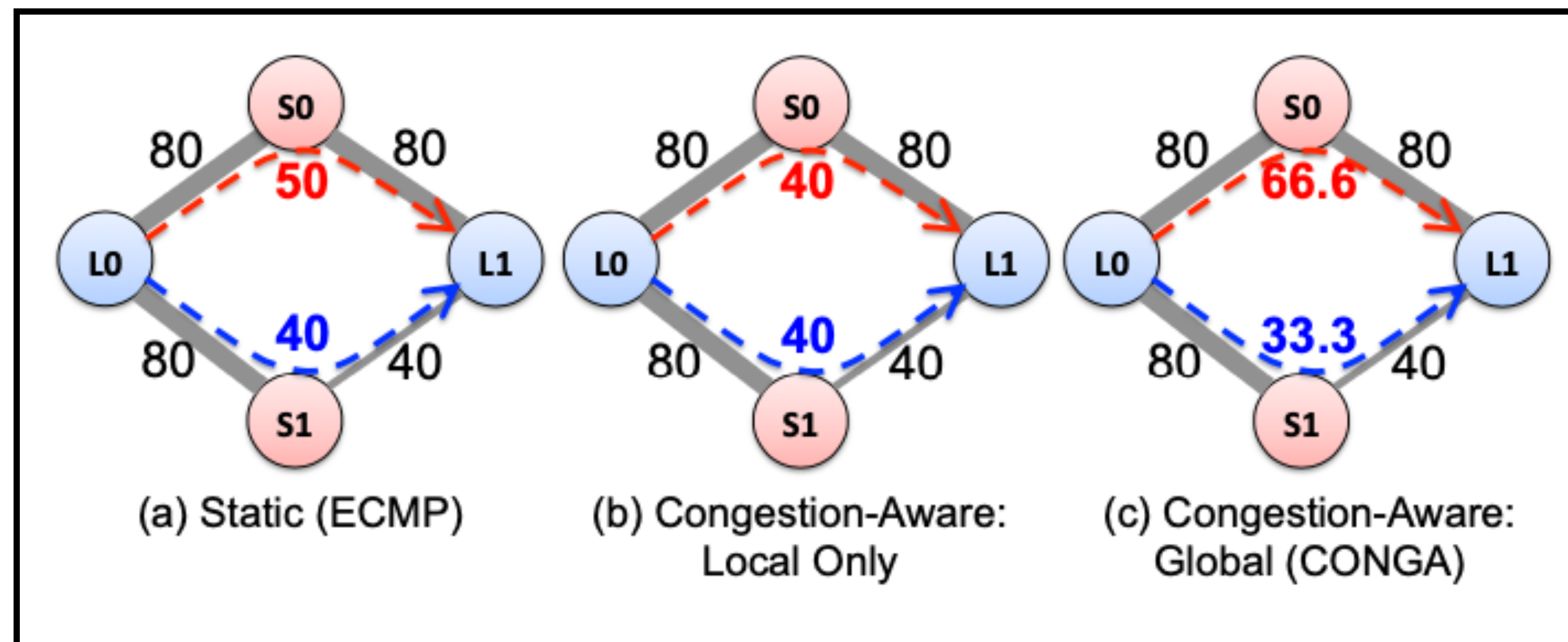
FlowLet ID	Port Number	Valid Bit	Age Bit
1234	4	1	1
5678	5	0	0

Congestion Awareness



Congestion Awareness

- Timely link utilization
 - Based on DRE
 - DRE Register / Link Capacity, quantized into 3-bits
- Congestion should be global



CONGA Technique #3

- #1: Networking bandwidth capacity (supply) is ephemeral!
- #2: Flow demand and flow # are unpredictable!
- #3: Real-time global view is hard to maintain!

Distributed Load Balancing and In-Network

Distributed Load Balancing and In-Network

- Why distributed, not central?
 - Data center traffic is very bursty and unpredictable
 - Data center topology is regular

Distributed Load Balancing and In-Network

- Why distributed, not central?
 - Data center traffic is very bursty and unpredictable
 - Data center topology is regular
- Why in-network, not host?
 - The endhost cannot capture Burstiness
 - The transport stack is already fat with many functionalities
 - Kernel-bypass I/O systems emerge

Distributed Load Balancing and In-Network

- Why distributed, not central?
 - Data center traffic is very bursty and unpredictable
 - Data center topology is regular
- Why in-network, not host?
 - The endhost cannot capture Burstiness
 - The transport stack is already fat with many functionalities
 - Kernel-bypass I/O systems emerge

Make the load-balancing decision for the first packet of a FlowLet at the edge switch

Combine Everything Together

- #1: Networking bandwidth capacity (supply) is ephemeral! ✓
- #2: Flow demand and flow # are unpredictable! ✓
- #3: Real-time global view is hard to maintain! ✓

How does Conga work?

CONGA — Host Server

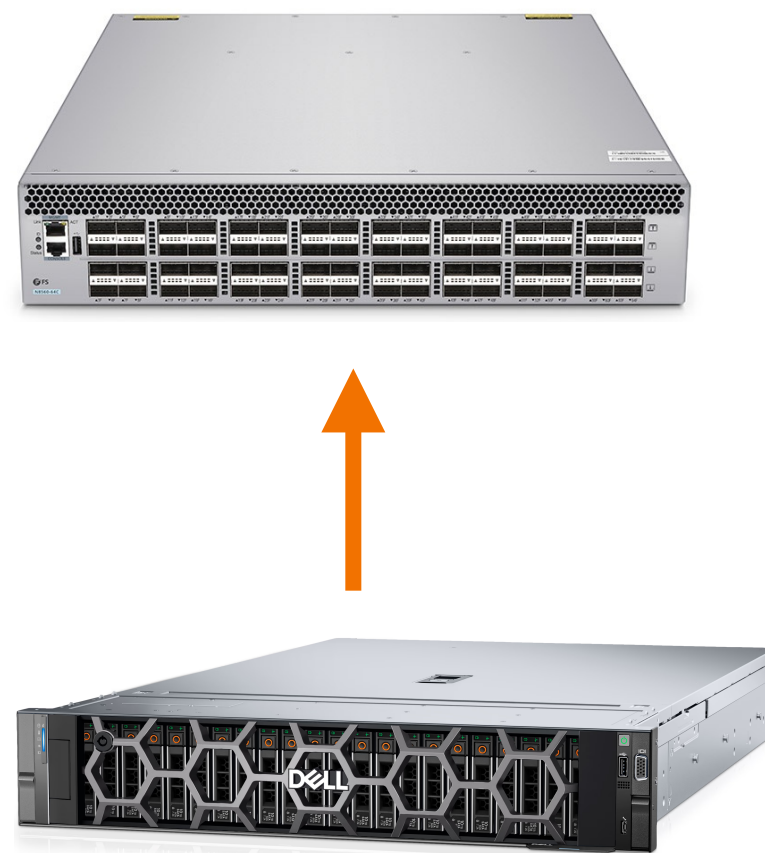


CONGA — Host Server

- Hosts send TCP/IP traffic
 - No load balancing decision is made



CONGA — Source Leaf Switch



CONGA — Source Leaf Switch

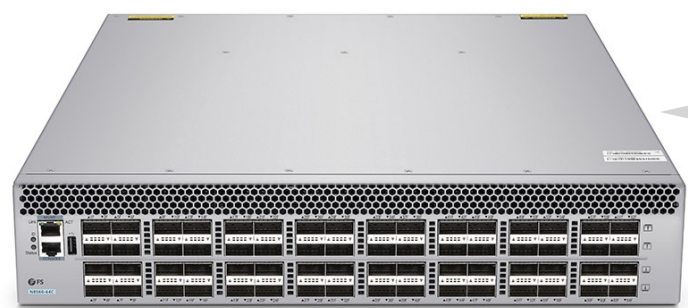
FlowLet Table

FlowLet	Port	Valid	Age
1234	4	1	1
5678	5	0	0



1. FlowLet detection

CONGA — Source Leaf Switch



- 1. FlowLet detection
- 2. Choose the path (load balancing)

FlowLet Table

FlowLet	Port	Valid	Age
1234	4	1	1
5678	5	0	0

Congestion-To-Leaf Table

Dst Leaf	Path 1	Path 2	Path k
1	0b000	0b000	0b111
2	0b111	0b110	0b101

CONGA — Source Leaf Switch



- 1. FlowLet detection
- 2. Choose the path (load balancing)

FlowLet Table

FlowLet	Port	Valid	Age
1234	4	1	1
5678	5	0	0

Congestion-To-Leaf Table

Dst Leaf	Path 1	Path 2	Path k
1	0b000	0b000	0b111
2	0b111	0b110	0b101

CONGA — Source Leaf Switch

- Source Leaf switches forward traffic
 - Setup the FlowLet table
 - Perform load balancing, i.e., minimal load
 - Update the link load

FlowLet Table

FlowLet	Port	Valid	Age
1234	4	1	1
5678	5	0	0

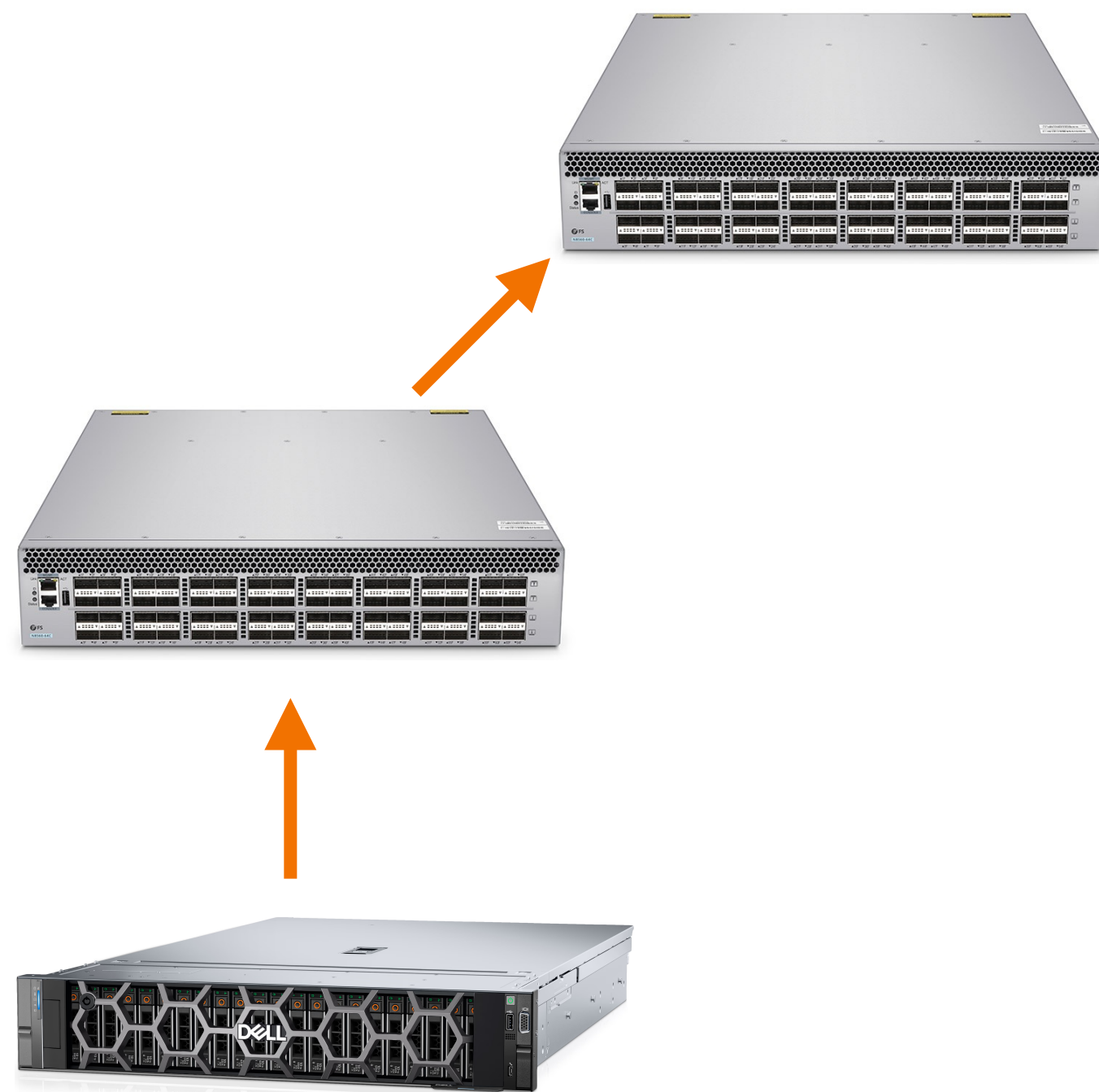
Congestion-To-Leaf Table

Dst Leaf	Path 1	Path 2	Path k
1	0b000	0b000	0b111
2	0b111	0b110	0b101

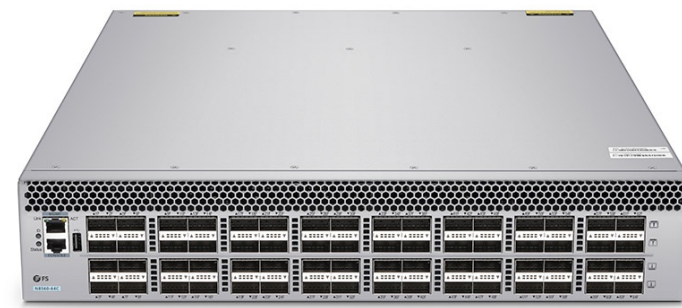


1. FlowLet detection
2. Choose the path (load balancing)
3. Update the link load

CONGA — Spine Switch

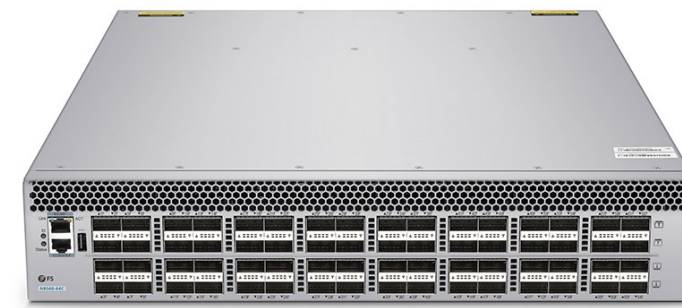


CONGA — Spine Switch

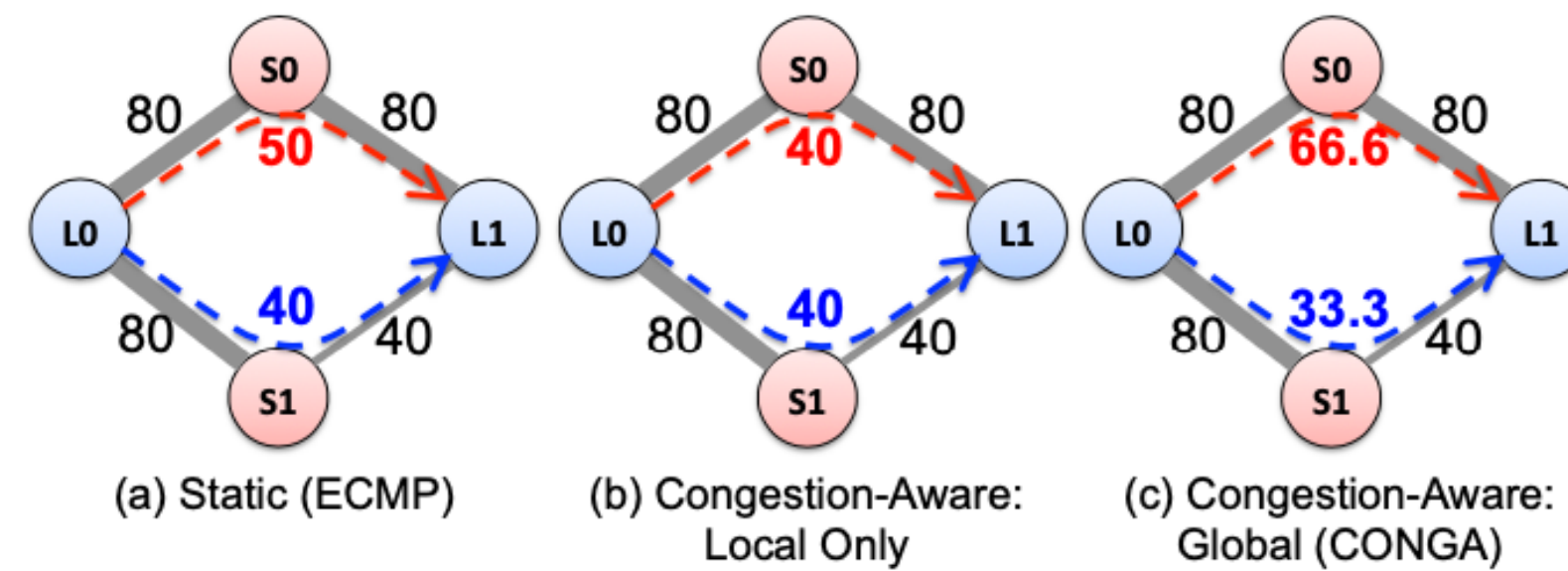


Do we perform load balancing again?

CONGA — Spine Switch

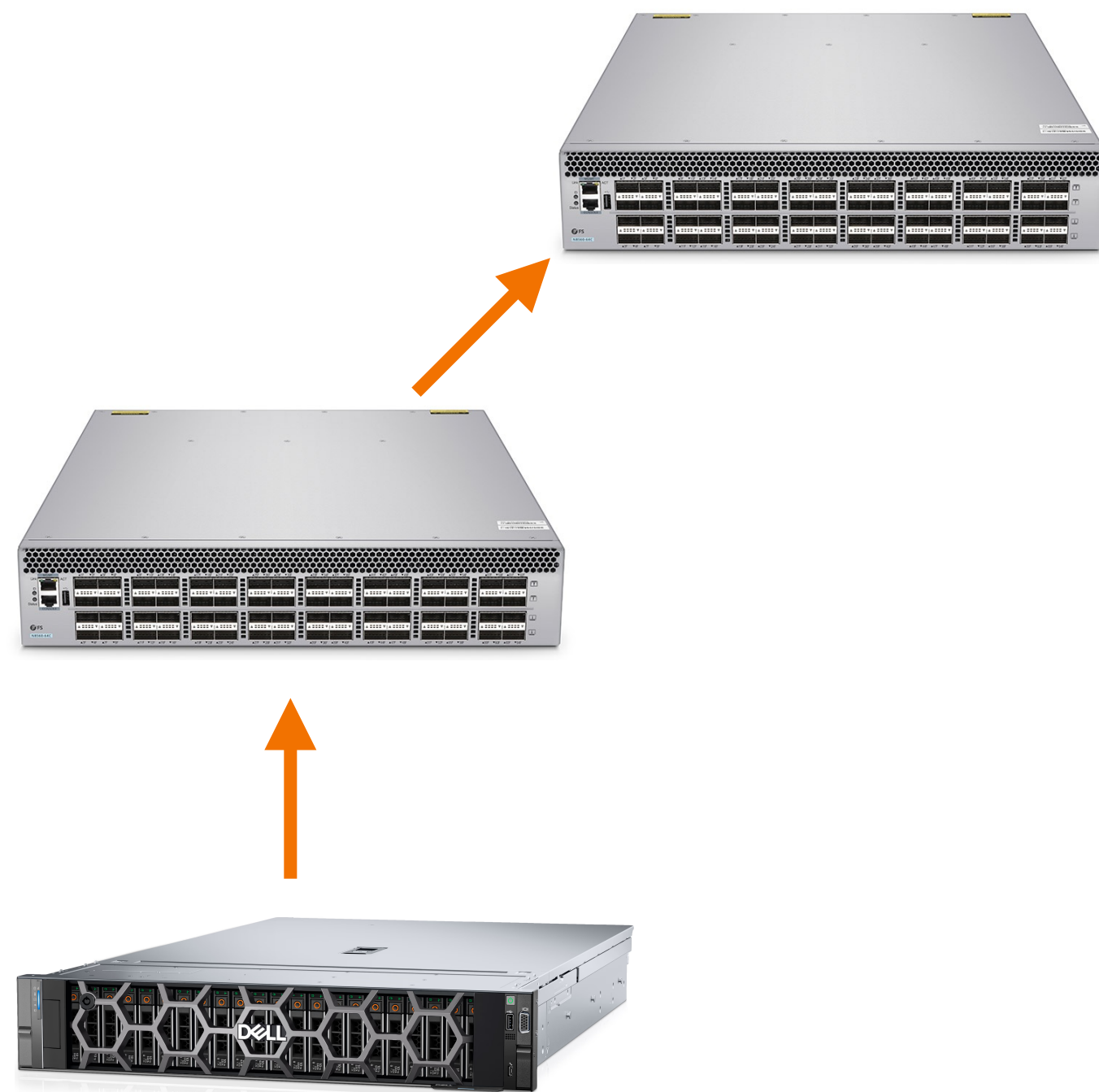


Do we perform load balancing again?



CONGA — Spine Switch

- Spine switches forward traffic
 - Directly choose the next hop based on the existing LB choice
 - Bookkeep the link load



1. Reuse the load-balancing decision
2. Encode the link load

CONGA — Spine Switch

- Spine switches forward traffic
 - Directly choose the next hop based on the existing LB choice
 - Bookkeep the link load



1. Reuse the load-balancing decision
2. Encode the link load

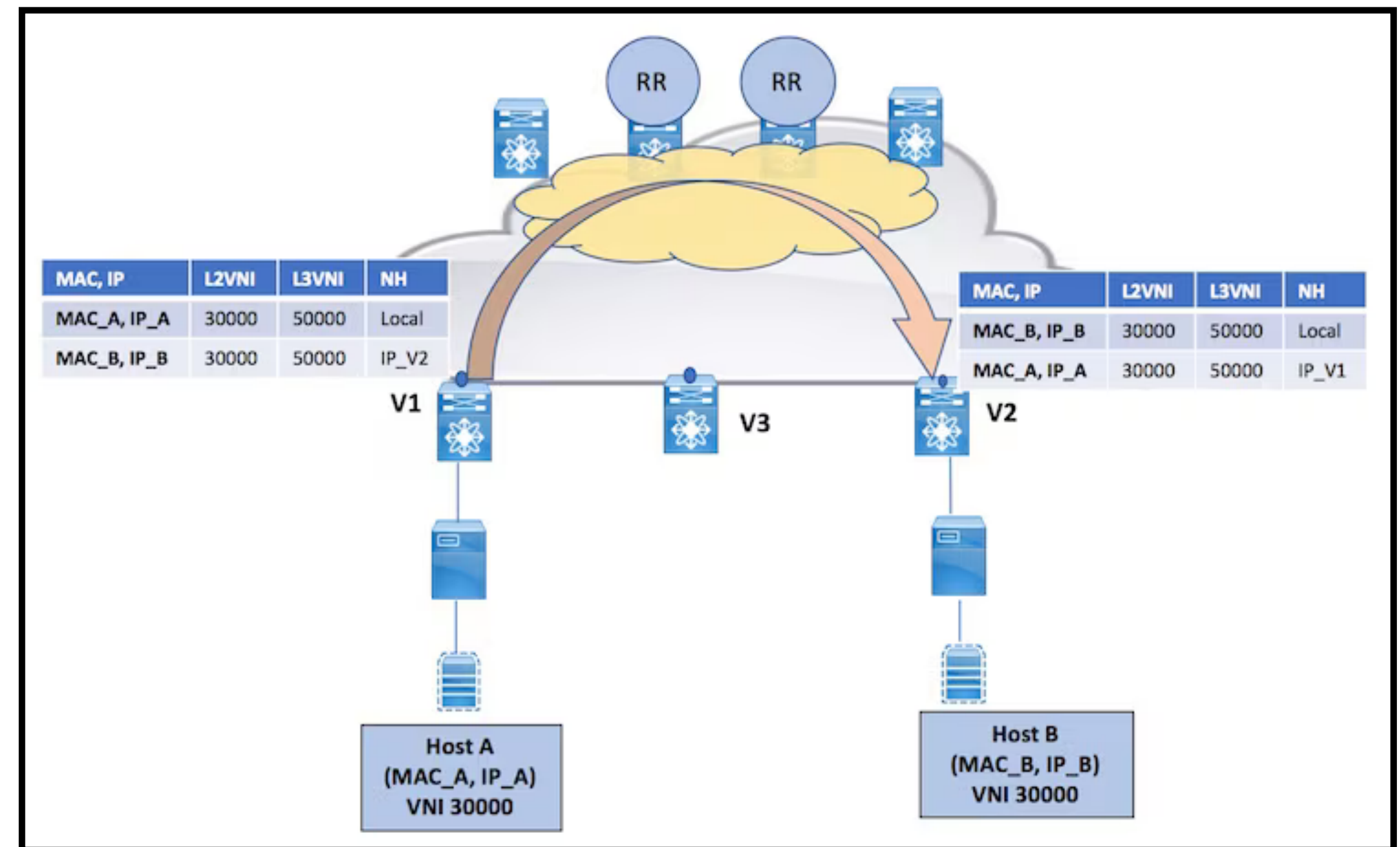
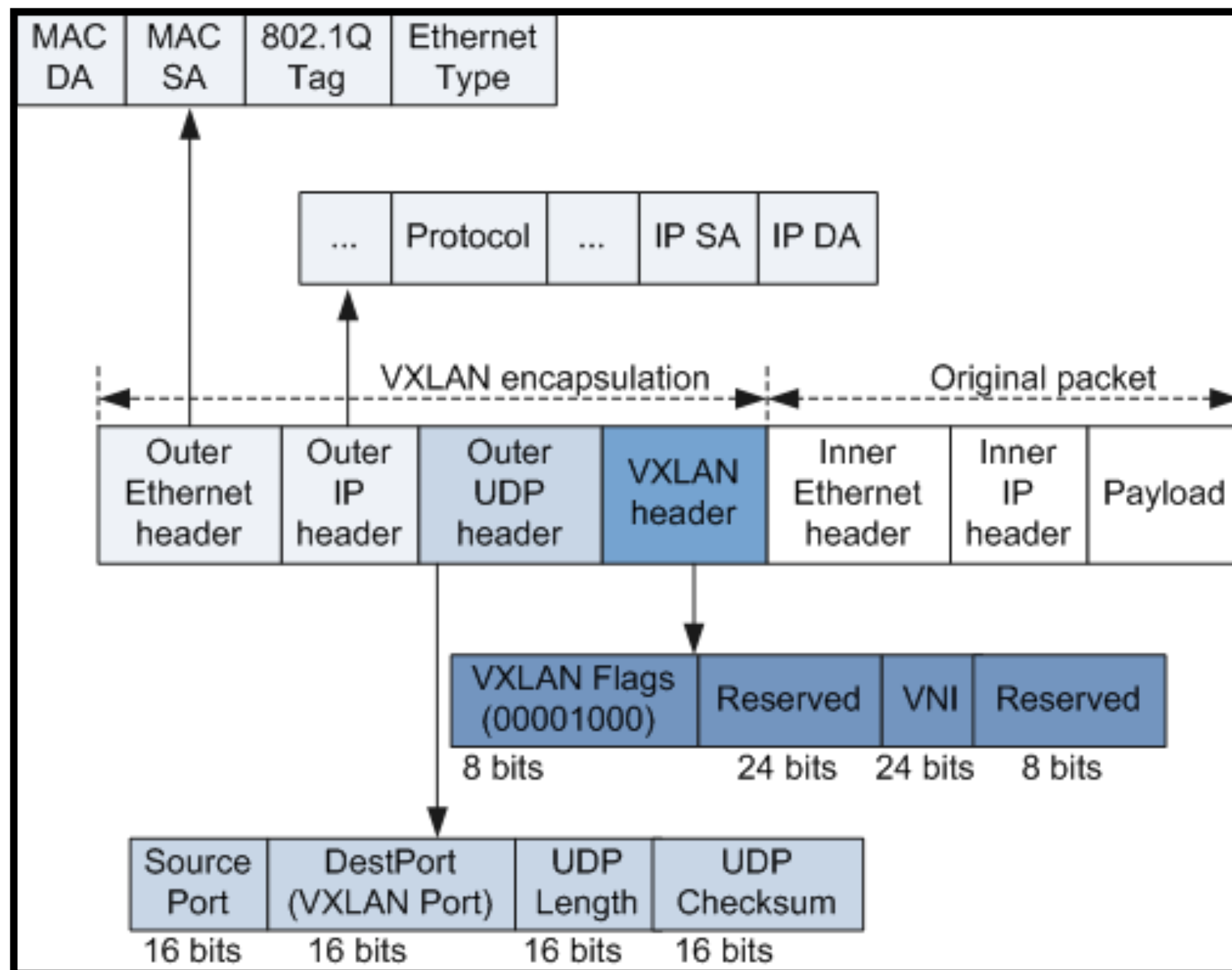
But how?

Path Revisit: Server —> Leaf —> Spine

How does CONGA perform addressing and routing?

Path Revisit: Server → Leaf → Spine

- VXLAN tunneling between source and destination leaf



Special Packet Header

- LBTAG (4 bits)
 - Set by the source leaf when making the LB decision
- CE (3 bits)
 - Indicate the path congestion extent
 - Set by every traversed switch along the path
- FB_LBTAG (4 bits)
 - Used by the destination leaf switch to piggyback which LBTAG is taken
- FB_Metric (3 bits)
 - Used by the destination leaf switch to piggyback the path CE value

Revisiting CONGA — Host Server

- Hosts send TCP/IP traffic
 - No load balancing decision is made
 - Build the VXLAN packet header



Revisit CONGA — Source Leaf Switch

- Source Leaf switches forward traffic
 - Setup the FlowLet table
 - Perform load balancing, i.e., minimal load
 - Update the link load
 - **Modify the conga packet header**

FlowLet Table

FlowLet	Port	Valid	Age
1234	4	1	1
5678	5	0	0

Congestion-To-Leaf Table

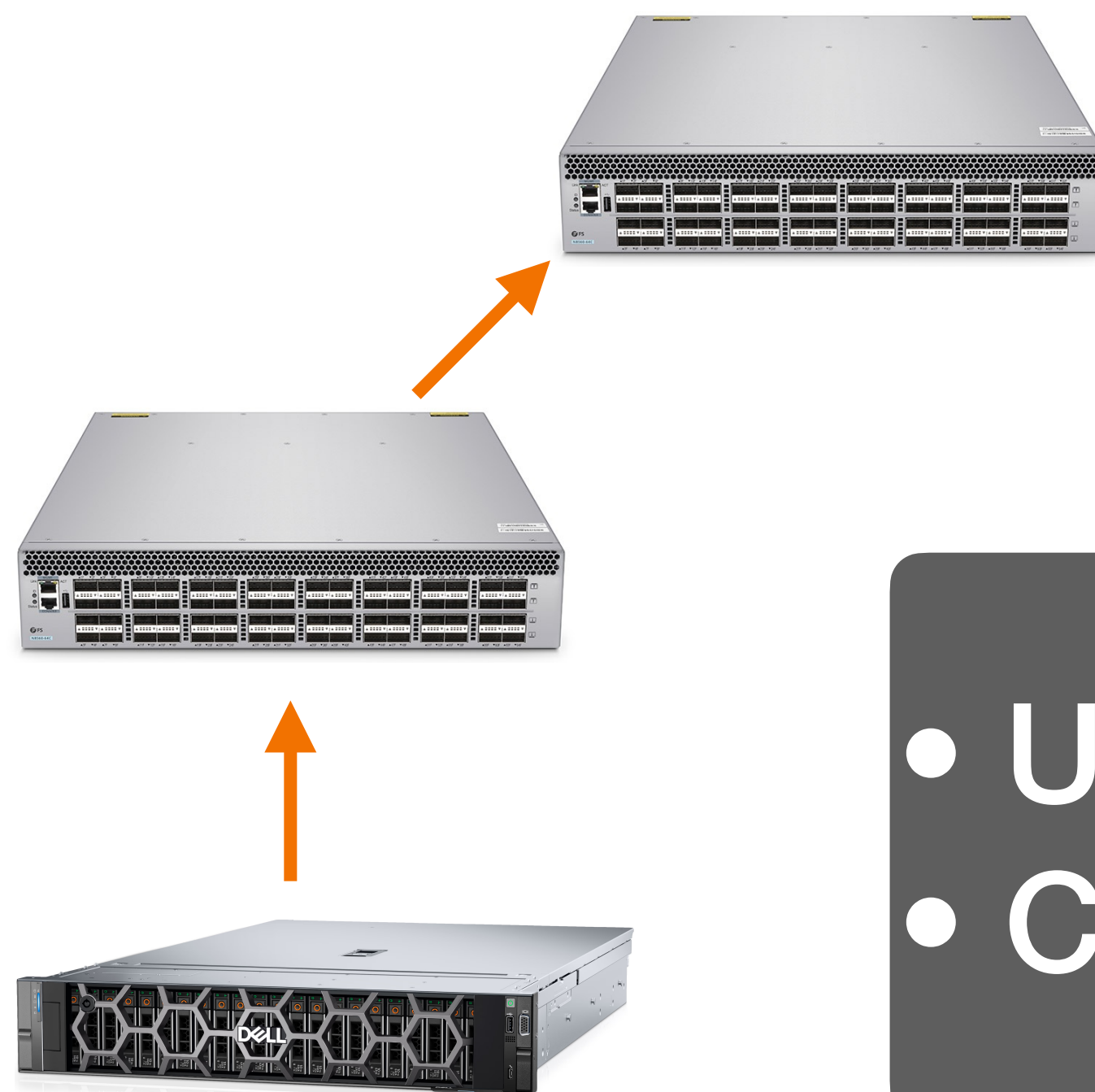
Dst Leaf	Path 1	Path 2	Path k
1	0b000	0b000	0b111
2	0b111	0b110	0b101



1. FlowLet detection
2. Choose the path (load balancing)
3. Update the link load
4. Encode the LB decision

Revisit CONGA — Spine Switch

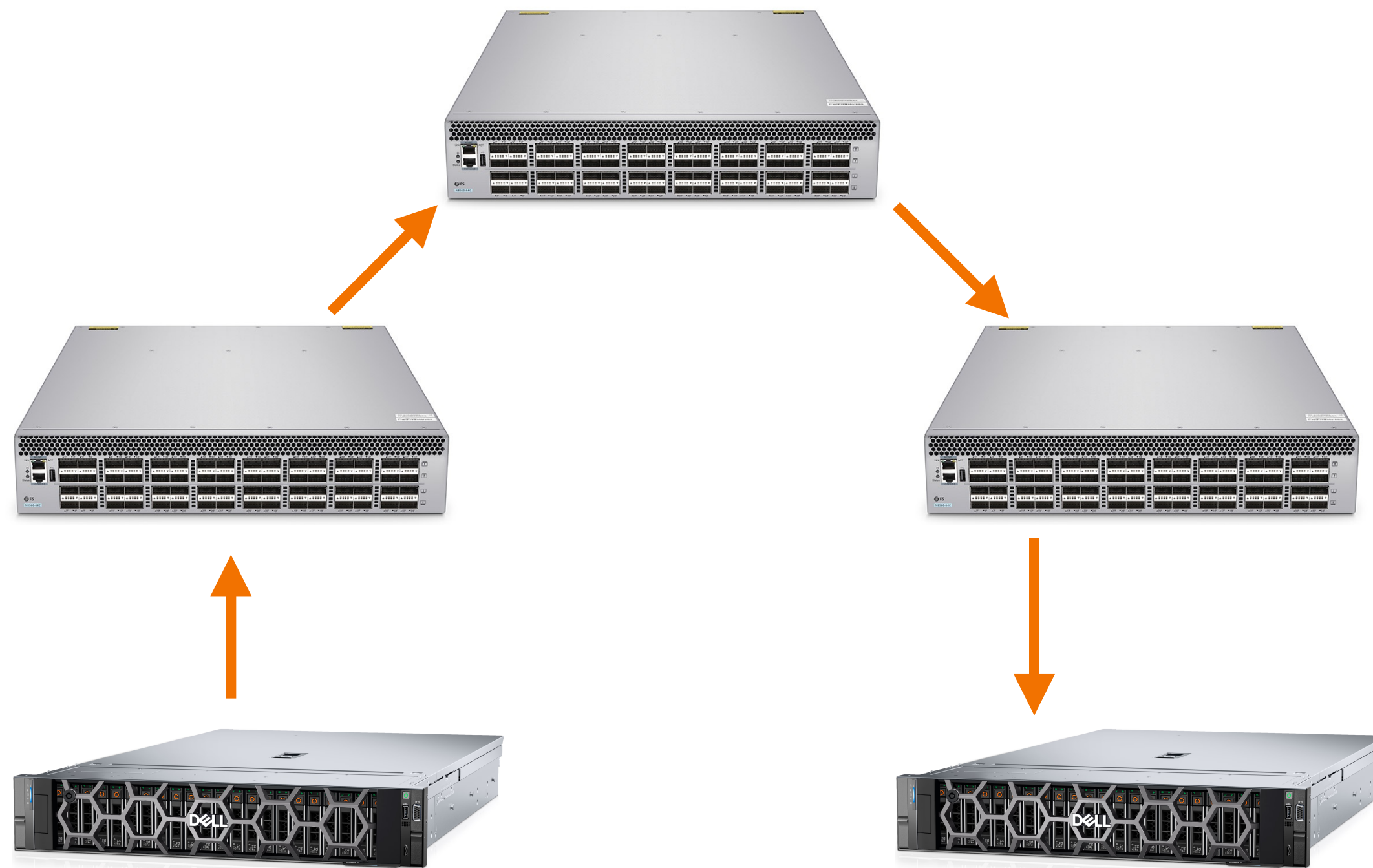
- Spine switches forward traffic
 - Directly choose the next hop based on the existing LB choice
 - Bookkeep the link load



1. Reuse the load-balancing decision
2. Encode the link load

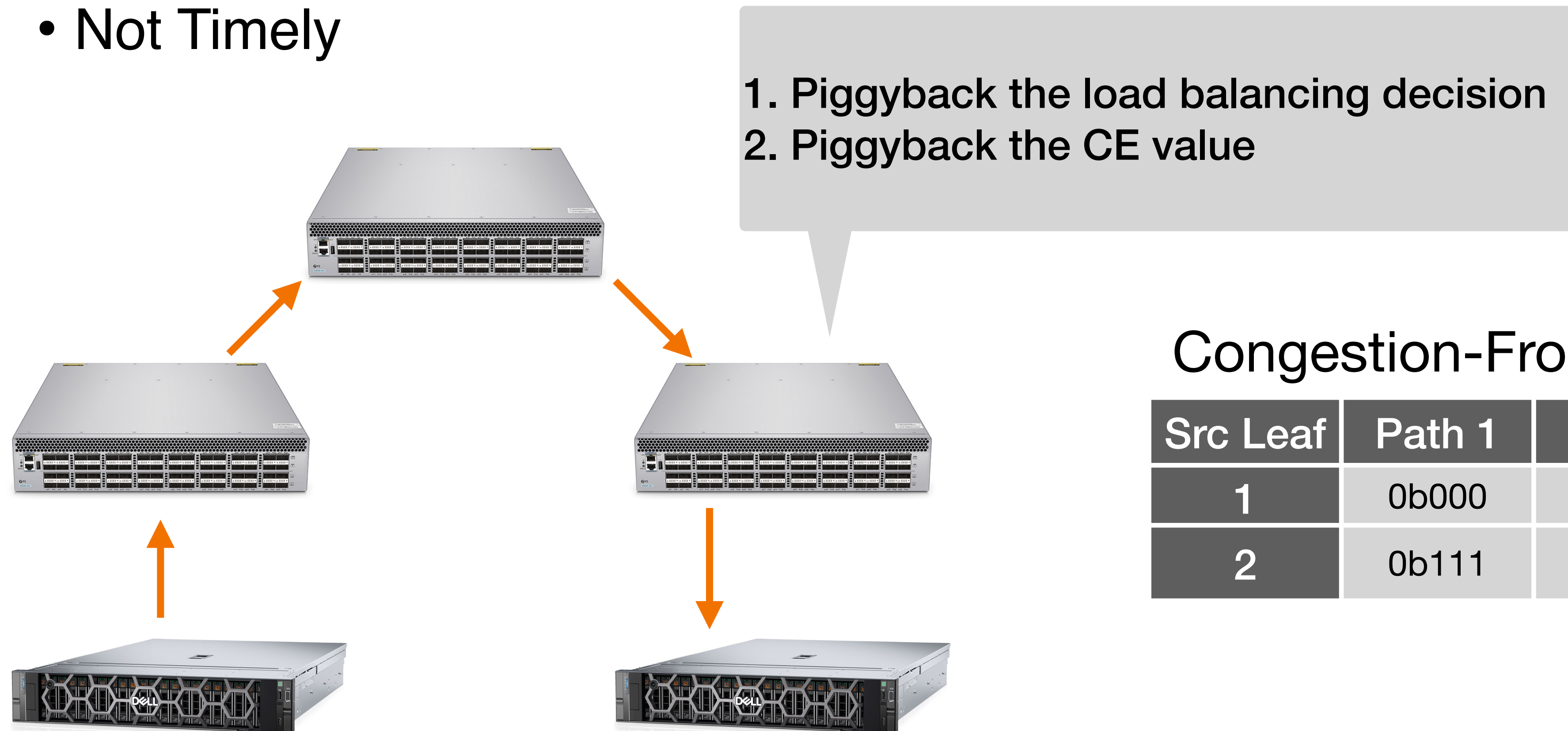
- Use the LBTag to forward
- Conditional update the CE value

CONGA — Destination Leaf Switch



CONGA — Destination Leaf Switch

- Destination leaf switches return the load status opportunistically
 - Reuse the ACK packets
 - Not Timely



Congestion-From-Leaf Table

Src Leaf	Path 1	Path 2	Path k
1	0b000	0b000	0b111
2	0b111	0b110	0b101

CONGA — Destination Leaf Switch

- Destination leaf switches return the load status opportunistically
 - Reuse the ACK packets
 - Not Timely

Why not update the CE value along the reply path?



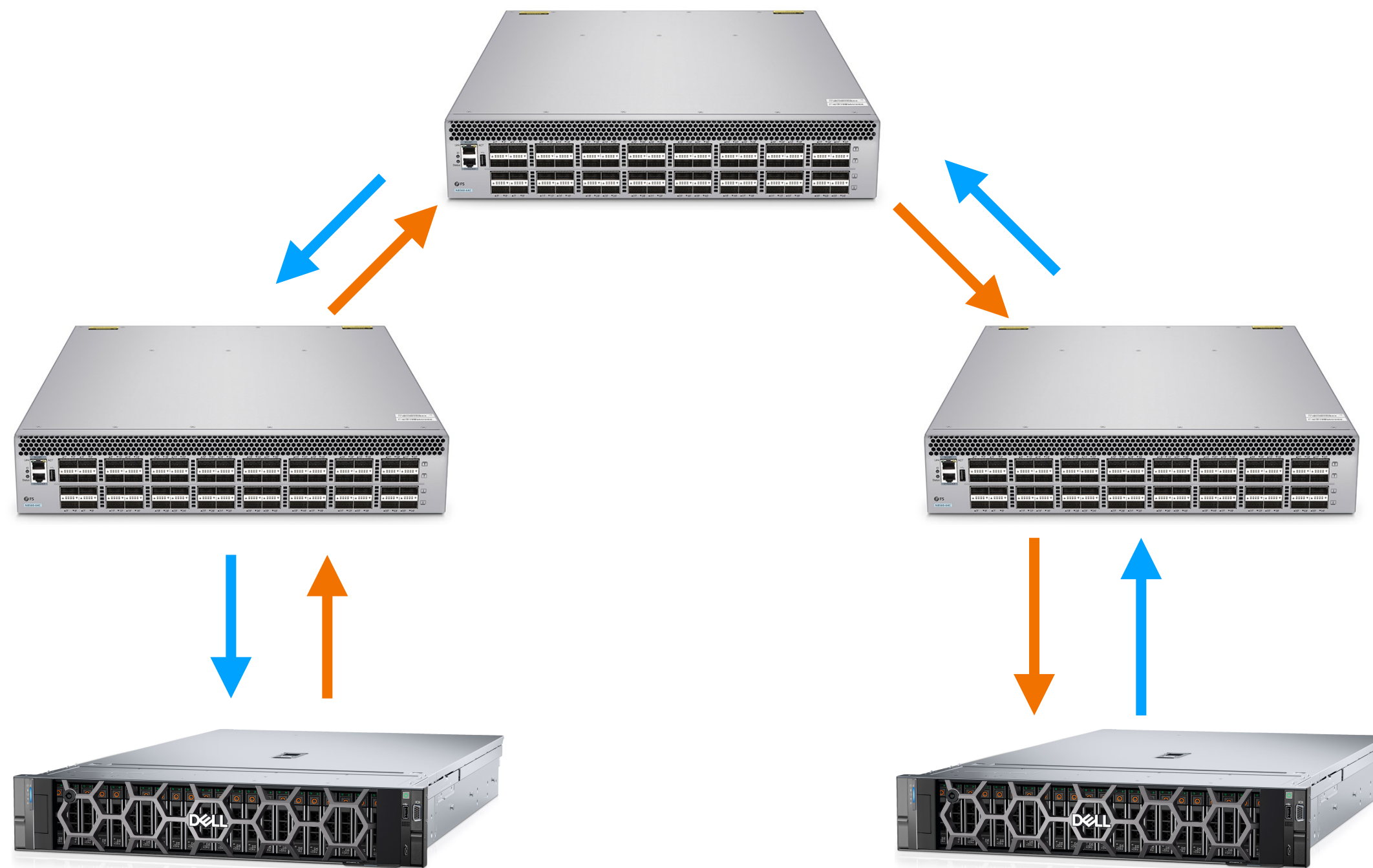
CONGA — Destination Leaf Switch

- Destination leaf switches return the load status opportunistically
 - Reuse the ACK packets
 - Not Timely

Why not update the CE value along the reply path?
SRC->DST and DST->SRC are not the same!

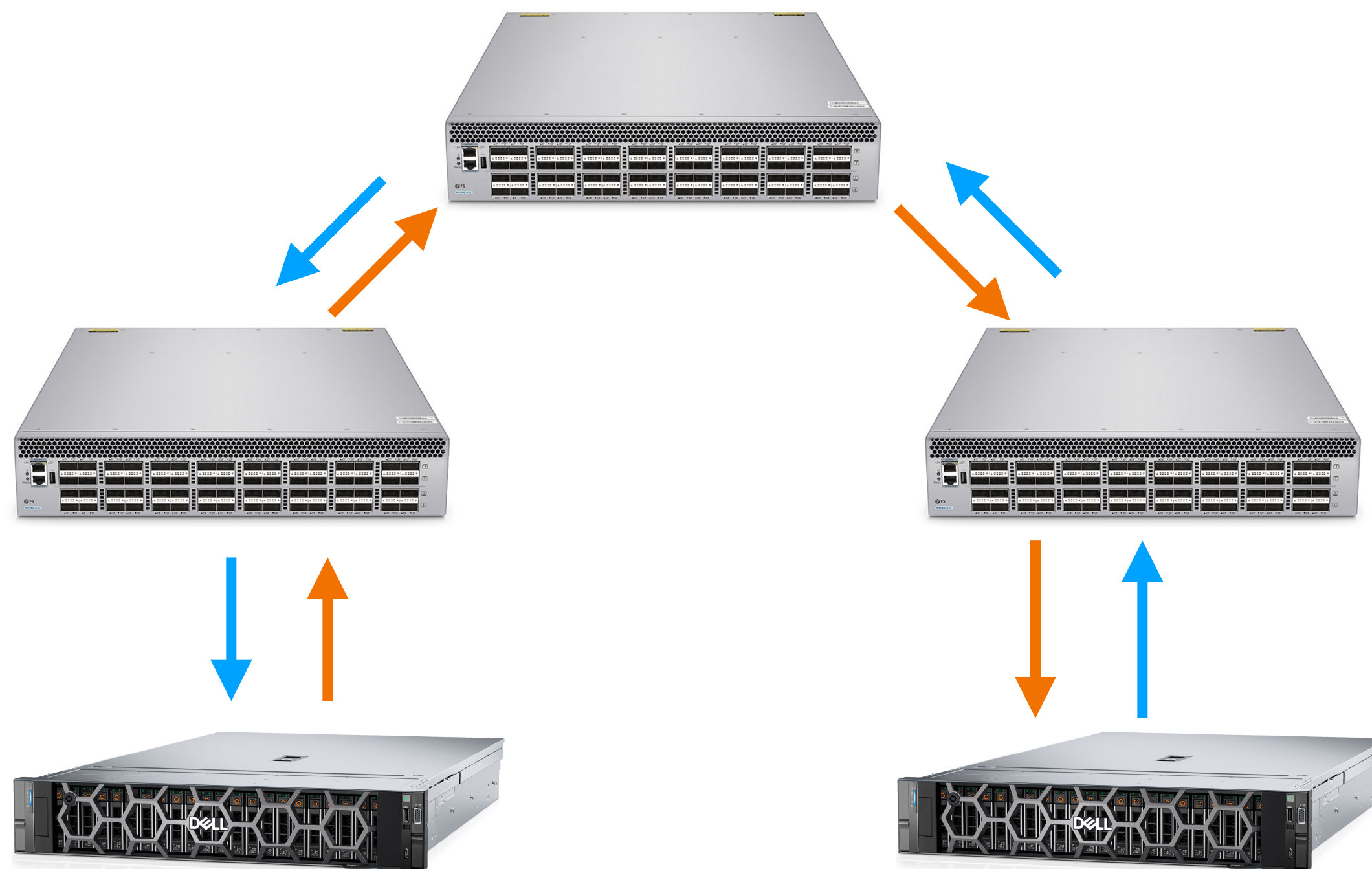


CONGA — Leaf/Spine Switch



CONGA — Leaf/Spine Switch

- Source leaf switches update the congestion-to-leaf table



Congestion-To-Leaf Table

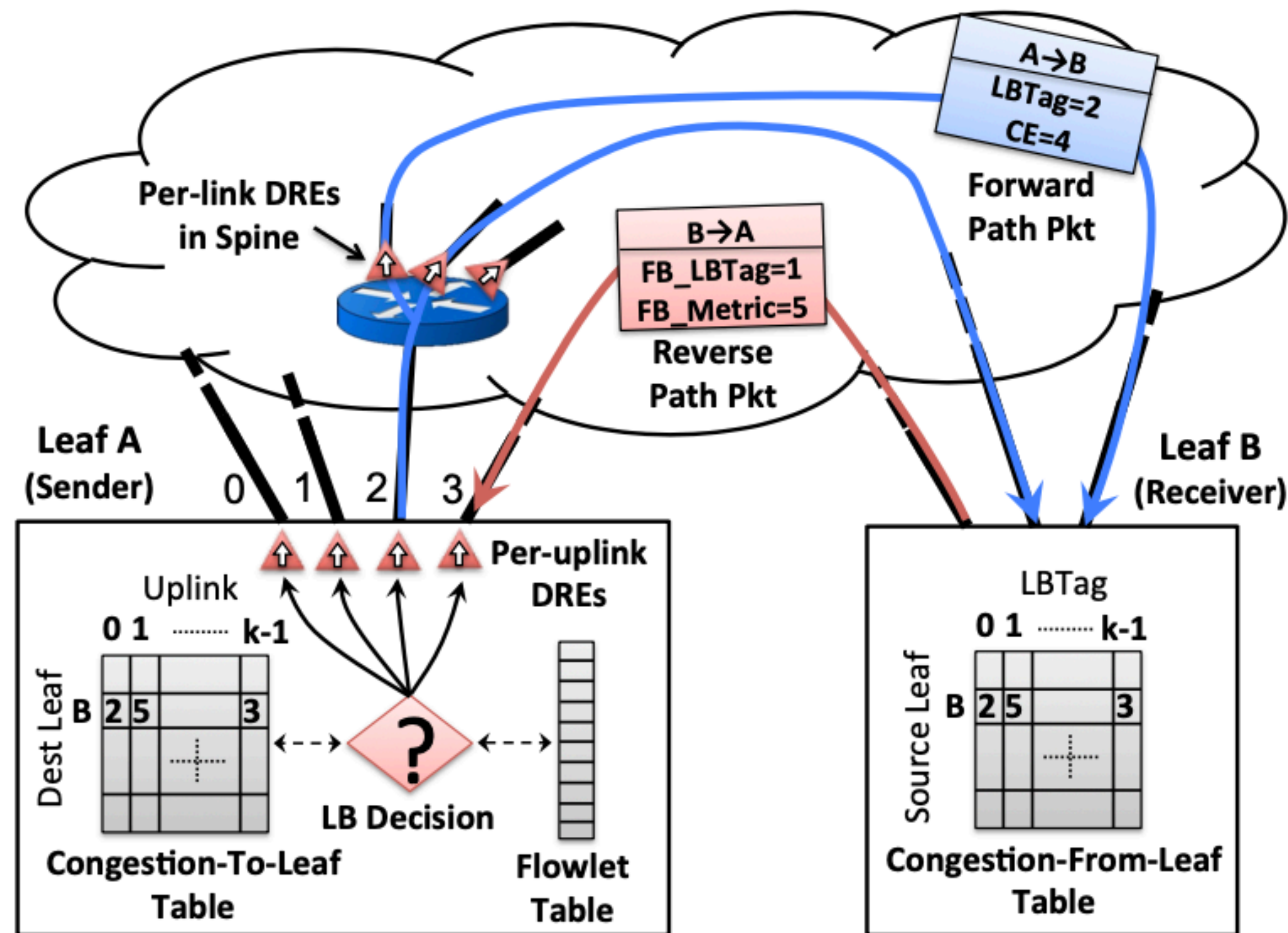
Dst Leaf	Path 1	Path 2	Path k
1	0b000	0b000	0b111
2	0b111	0b110	0b101

Communication Path in CONGA

- Hosts send TCP/IP traffic
 - Build the VXLAN packet header
- Source Leaf switches forward traffic
 - Setup the FlowLet table
 - Perform load balancing, i.e., minimal load
 - Update the link load and congestion-to-leaf table
- Spine switches forward traffic
 - Directly choose the next hop based on the existing LB choice
 - Bookkeep the link load
- Destination leaf switches return the load status opportunistically
 - Reuse the ACK packets

Communication Path in CONGA

- Hosts send
 - Build the path
- Source Leaf
 - Setup the path
 - Perform congestion control
 - Update the path
- Spine switch
 - Directly
 - Bookkeeping
- Destination
 - Reuse the path



choice

opportunistically

CONGA is a distributed, in-network, global congestion-aware load balancer based on leaf-to-leaf feedback and flowlet switching.

- React fast
- DCN is a regular fabric

CONGA is a distributed, in-network, global congestion-aware load balancer based on leaf-to-leaf feedback and flowlet switching.

- Avoid transport stack modifications

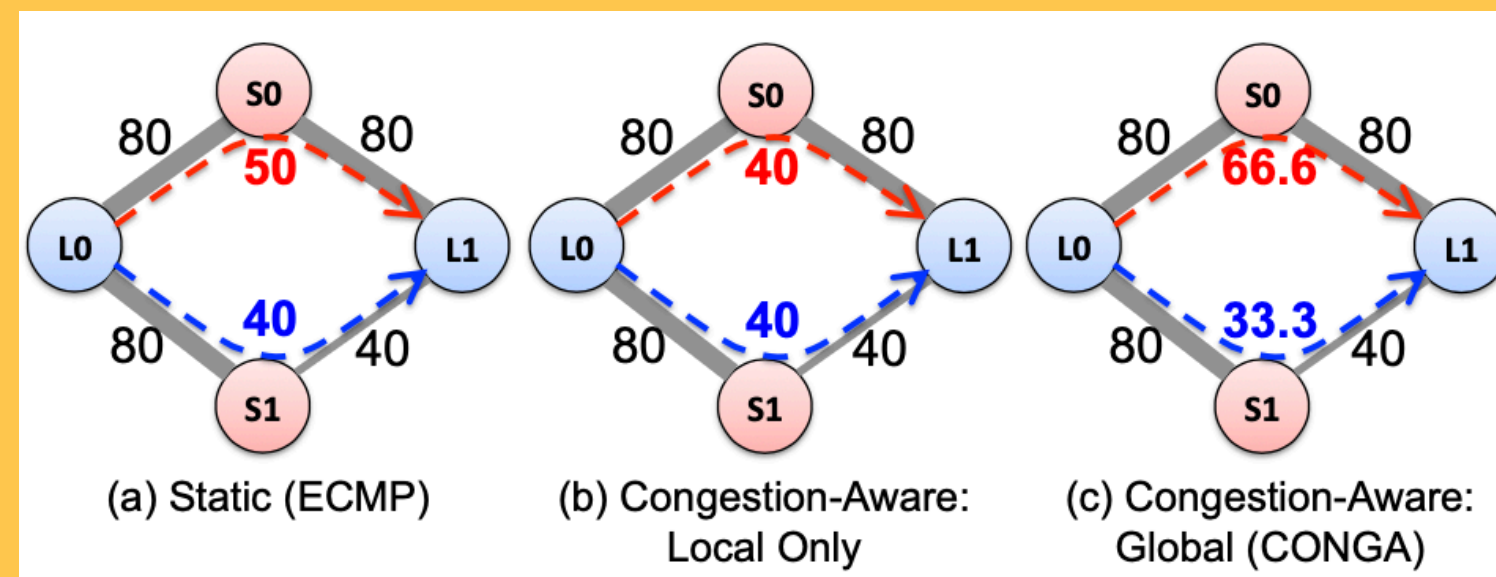
- React fast
- DCN is a regular fabric

CONGA is a distributed, in-network, global congestion-aware load balancer based on leaf-to-leaf feedback and flowlet switching.

- Avoid transport stack modifications

- React fast
- DCN is a regular fabric

CONGA is a distributed, in-network, global congestion-aware load balancer based on leaf-to-leaf feedback and flowlet switching.

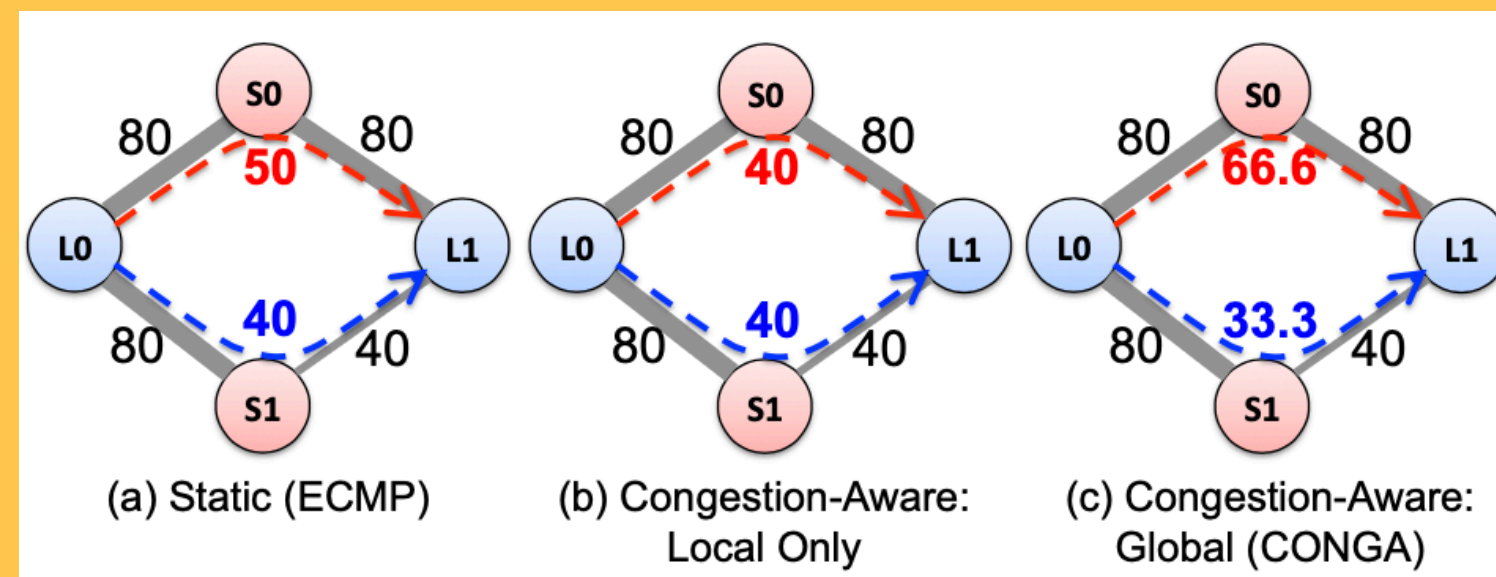


- Avoid transport stack modifications

- Overlay network

- React fast
- DCN is a regular fabric

CONGA is a distributed, in-network, global congestion-aware load balancer based on leaf-to-leaf feedback and flowlet switching.

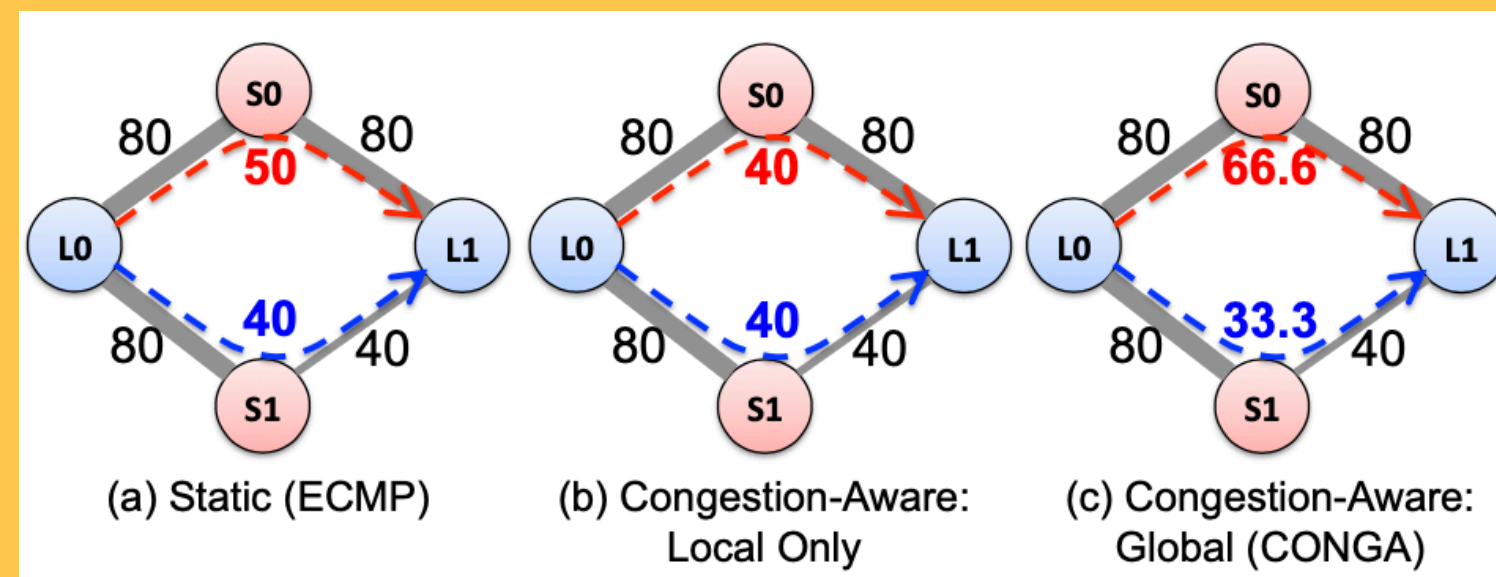


- Avoid transport stack modifications

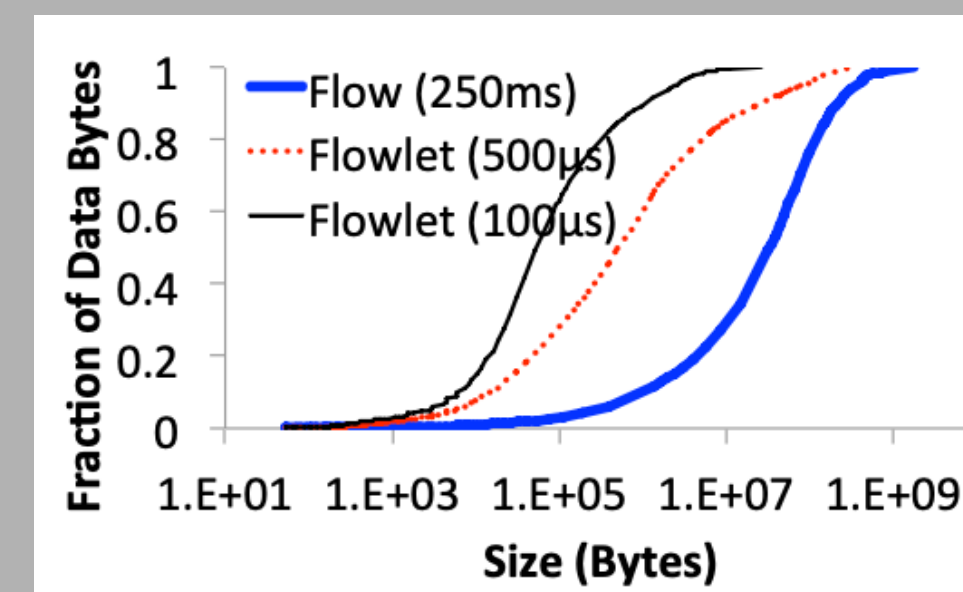
- Overlay network

- React fast
- DCN is a regular fabric

CONGA is a distributed, in-network, global congestion-aware load balancer based on leaf-to-leaf feedback and flowlet switching.



- Flowlets are bursts of packets from a flow that are separated by large enough gaps



Flow Scheduling Problem Solved?

Flow Scheduling Problem Solved?

- #1: No-trivial hardware support
- #2: Micro-second traffic under 100+Gbps requires timely reaction
- #3: VXLAN has poor flexibility and requires lots of manual efforts

CONGA Today: Works or not?

ByteDance SGLB (Sigcomm'25)

- AI training traffic
 - Low entropy, high burstiness, long-running jobs

SGLB: Scalable and Robust Global Load Balancing in Commodity AI Clusters

Chenchen Qi¹, Wenfei Wu², Yongcan Wang¹, Keqiang He³, Yu-Hsiang (Sean) Kao¹, Zongying He⁴,
Chen-Yu Yen¹, Zhuo Jiang¹, Feng Luo¹, Surendra Anubolu⁴, Yanjin Gao¹, Bingfeng Lin¹,
Wenda Ni¹, Yiming Yang¹, Donglin Wei¹, Boyang Zhou¹, Jian Wang¹, Shan Ding¹
¹ ByteDance, ² Peking University, ³ Shanghai Jiao Tong University, ⁴ Broadcom

ABSTRACT

Internet companies are constructing large-scale AI clusters with commodity Ethernet switches for AI model training to support their businesses. AI training workloads impose stringent network requirements, mandating that cluster networks deliver high peak throughput while maintaining robustness and resilience in the face of link failures. We present SGLB, a distributed, global congestion-aware load balancing system for AI clusters. SGLB operates a control-plane protocol, SyncMesh, to enable a new load balancing abstraction in modern commodity switches—Global Load Balancing (GLB) engine—which utilizes global congestion information to distribute traffic across all available paths. We address three key challenges in designing SGLB: fast routing convergence to minimize downtime in the event of link failures, scalable maintenance of congestion profiles within the constraints of limited switch hardware resources, and preventing GLB throughput suppression in scenarios where path bandwidths are asymmetric. We prototype SGLB and conduct extensive experiments to evaluate SGLB. SGLB ensures rapid routing convergence in the event of link failures, recovering in as little as 45 μ s to guarantee network robustness for long-term, stable model training. Additionally, SGLB effectively load-balances traffic across paths, avoiding those with global congestion, which accelerates All-to-All collective communication by up to 60%.

Zhou, Jian Wang, and Shan Ding. 2025. SGLB: Scalable and Robust Global Load Balancing in Commodity AI Clusters. In *ACM SIGCOMM 2025 Conference (SIGCOMM '25)*, September 8–11, 2025, Coimbra, Portugal. ACM, New York, NY, USA, 19 pages. <https://doi.org/10.1145/3718958.3750527>

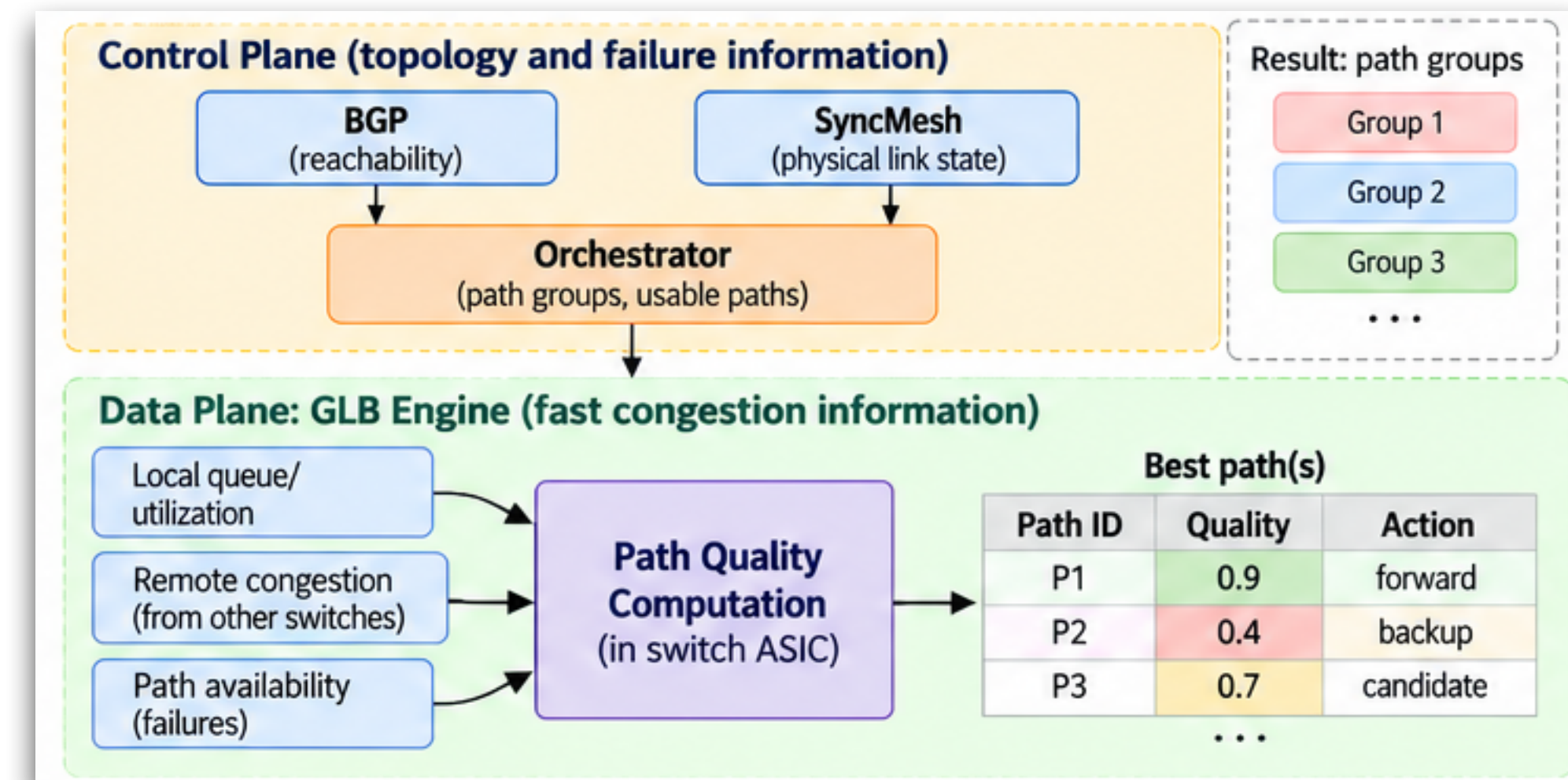
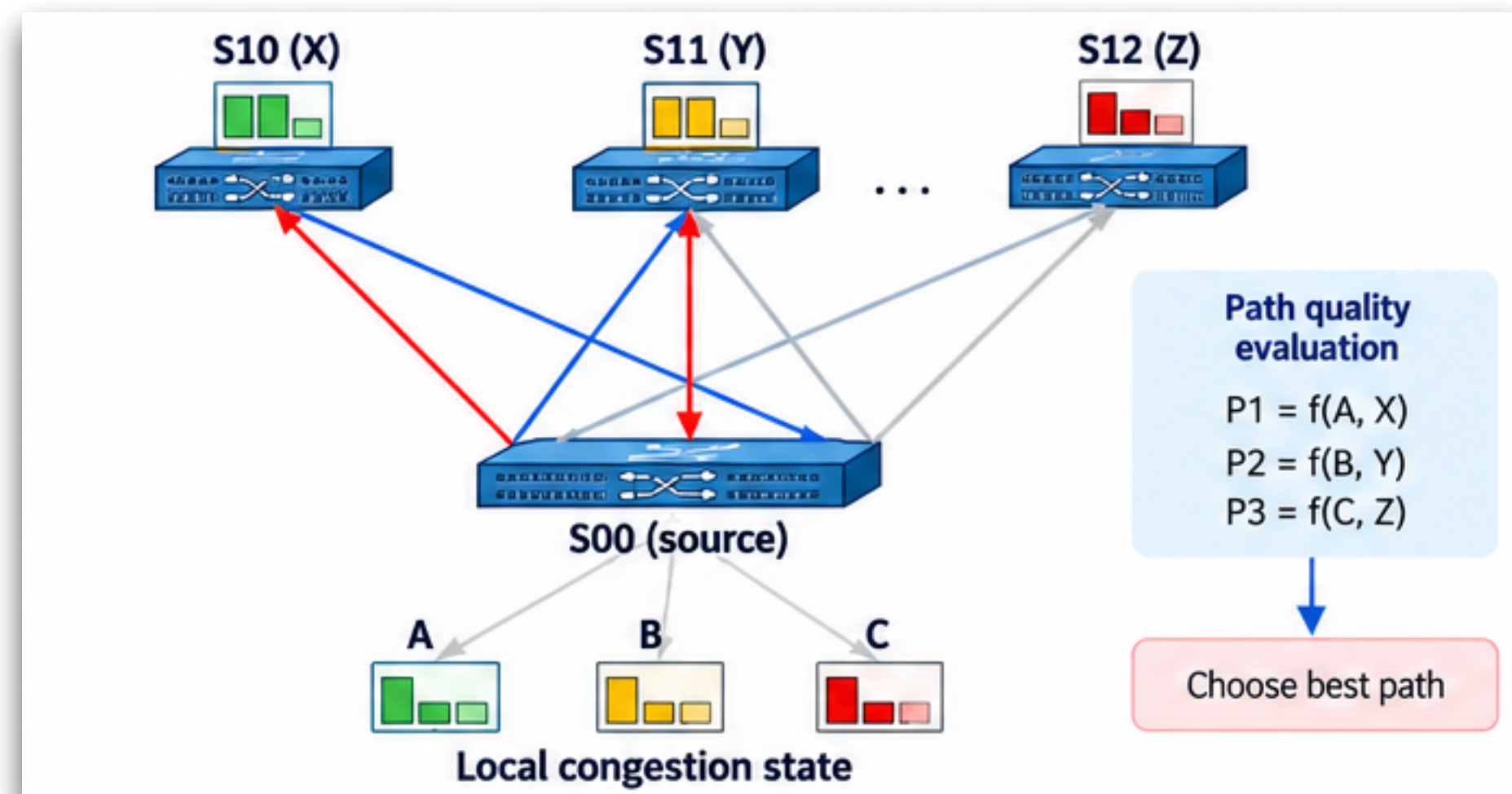
1 INTRODUCTION

To empower online services such as social media platforms and recommendation systems, Internet companies have a strong motivation to train machine learning (ML) models. Currently, high-quality model training is performed in large-scale clusters equipped with tens of thousands of GPUs and thousands of servers, which we refer to as *AI clusters*. Instead of purchasing specialized hardware to build such clusters, service providers prefer to use commodity Ethernet switches, which allows them to repurpose current data center networks and align with the operator's data center management experience (Section 2.2).

AI model training traffic exhibits burstiness and low entropy characteristics, in stark contrast to traditional cloud computing workloads. Furthermore, intensive computation demands sustained stability throughout training. Therefore, AI model training clusters must be designed for *high peak throughput* to accommodate the bursty, low-entropy traffic profile and *robustness* to ensure long-term stability (Section 2.1).

CONGA-like Design

- Commodity switches expose a GLB engine
 - Path-quality table + Local congestion registers
 - Embedded programmable processor
 - Exchange congestion information with other switches



Alibaba Stellar (Sigcomm'25)

- AI training traffic
 - RDMA, low entropy, large and regular elephant transfers

Alibaba STELLAR: A New Generation RDMA Network for Cloud AI

Jie Lu*, Jiaqi Gao*, Fei Feng*, Zhiqiang He, Menglei Zheng, Kun Liu, Jun He, Binbin Liao, Suwei Xu, Ke Sun, Yongjia Mo, Qinghua Peng, Jilie Luo, Qingxu Li, Gang Lu, Zishu Wang, Jianbo Dong, Kunling He, Sheng Cheng, Jiamin Cao, Hairong Jiao, Pengcheng Zhang, Shu Ma, Lingjun Zhu, Chao Shi, Yangming Zhang, Yiquan Chen, Wei Wang, Shuhong Zhu, Xingru Li, Qiang Wang, Jiang Liu, Chao Wang, Wei Lin, Ennan Zhai, Jiesheng Wu, Qiang Liu, Binzhang Fu, Dennis Cai
Alibaba Cloud

ABSTRACT

The rapid adoption of Large Language Models (LLMs) in cloud environments has intensified the demand for high-performance AI training and inference, where Remote Direct Memory Access (RDMA) plays a critical role. However, existing RDMA virtualization solutions, such as Single-Root Input/Output Virtualization (SR-IOV), face significant limitations in scalability, performance, and stability. These issues include lengthy container initialization times, hardware resource constraints, and inefficient traffic steering. To address these challenges, we propose STELLAR, a new generation RDMA network for cloud AI. STELLAR introduces three key innovations: Para-Virtualized Direct Memory Access (PVDMA) for on-demand memory pinning, extended Memory Translation Table (eMTT) for optimized GPU Direct RDMA (GDR) performance, and RDMA Packet Spray for efficient multi-path utilization. Deployed in our large-scale AI clusters, STELLAR spins up virtual devices in seconds, reduces container initialization time by 15 times, and improves LLM training speed by up to 14%. Our evaluations demonstrate that STELLAR significantly outperforms existing solutions, offering a scalable, stable, and high-performance RDMA network for cloud AI.

CCS CONCEPTS

• **Networks** → **Network architectures; Data center networks; Network components;**

Wang, Jiang Liu, Chao Wang, Wei Lin, Ennan Zhai, Jiesheng Wu, Qiang Liu, Binzhang Fu, Dennis Cai. 2025. Alibaba STELLAR: A New Generation RDMA Network for Cloud AI. In *ACM SIGCOMM 2025 Conference (SIGCOMM '25)*, September 8–11, 2025, Coimbra, Portugal. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3718958.3750539>

1 INTRODUCTION

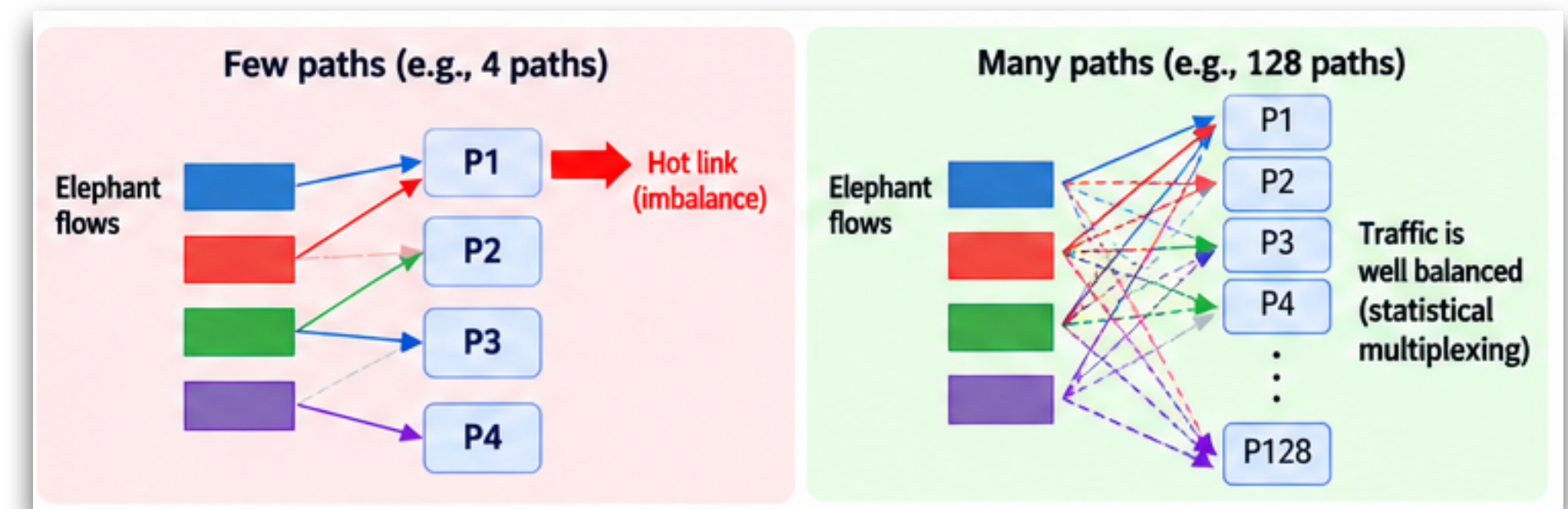
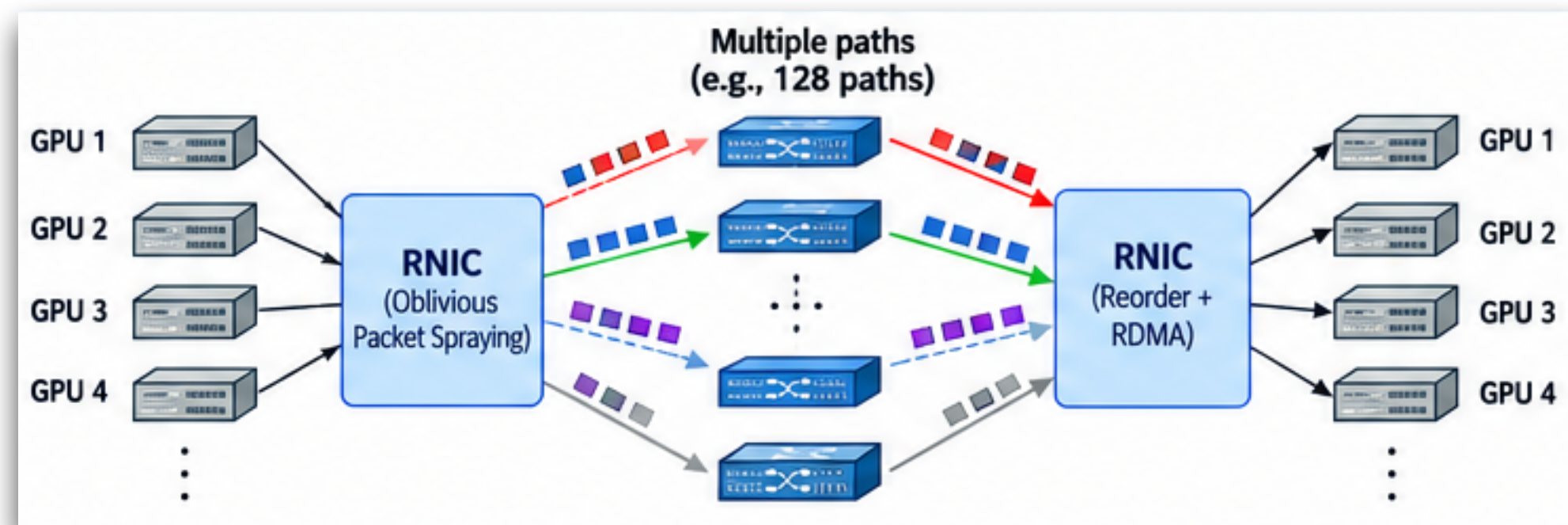
Recent years have seen significant advancements in Artificial Intelligence (AI), driving rapid growth in demand for cloud-based AI training and inference, particularly for Large Language Models (LLMs). The performance of these models heavily relies on Remote Direct Memory Access (RDMA) [4] to accelerate communication between computational devices like Graphics Processing Units (GPUs). Consequently, the performance of RDMA virtualization in the cloud is on the critical path for effective LLM training and inference.

As one of the largest public cloud providers, Alibaba has been deploying large-scale RDMA networks and serving massive AI applications since 2019. However, we have observed that the current RDMA solution has inherent limitations in providing high-performance, stable, and scalable cloud-based AI services:

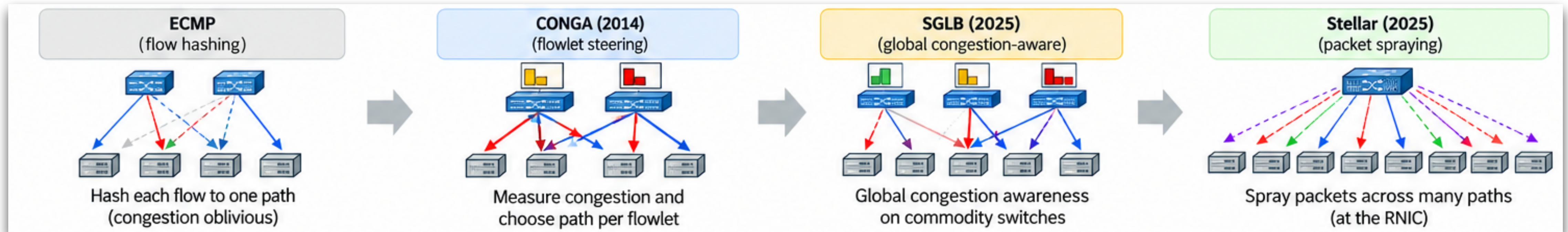
First, at the **host-level**, the current RDMA virtualization solution lacks scalability and is not lightweight. The number of Virtual Functions (VFs) an RDMA Network Interface Card (RNIC) can create is static; it can only be toggled between zero and a fixed maximum.

Flowlet Switching is Ineffective!

- 128-path oblivious packet spraying (OPS)
 - No need to continuously identify a single “best” path
 - Receiver handles out-of-order packets
 - Short RTO provides fast recovery from packet loss



How Has Load Balancing Evolved?



How Has Load Balancing Evolved?

	ECMP	CONGA	SGLB	Stellar
Era	~2000 (RFC)	2014	2025	2025
Target	General App	General App	AI Cluster	Cloud AI
Granularity	Flow	Flowlet	Dynamic	Packet
Congestion Aware	No	Global	Global	No
Network State	Minimal	Leaf feedback	Distributed global state	Minimal for path selection
Hardware	Commodity	Specialized Support	Commodity	FPGA RNIC
Key Idea	Hashing	Measure & Steer	Measure & Steer	Spray

Summary

- Today
 - Flow scheduling in data center networks (II)
- Next two lectures
 - Load balancers in the data centers
 - Maglev (NSDI'16)
 - Duet (SIGCOMM'14)