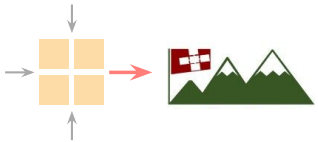
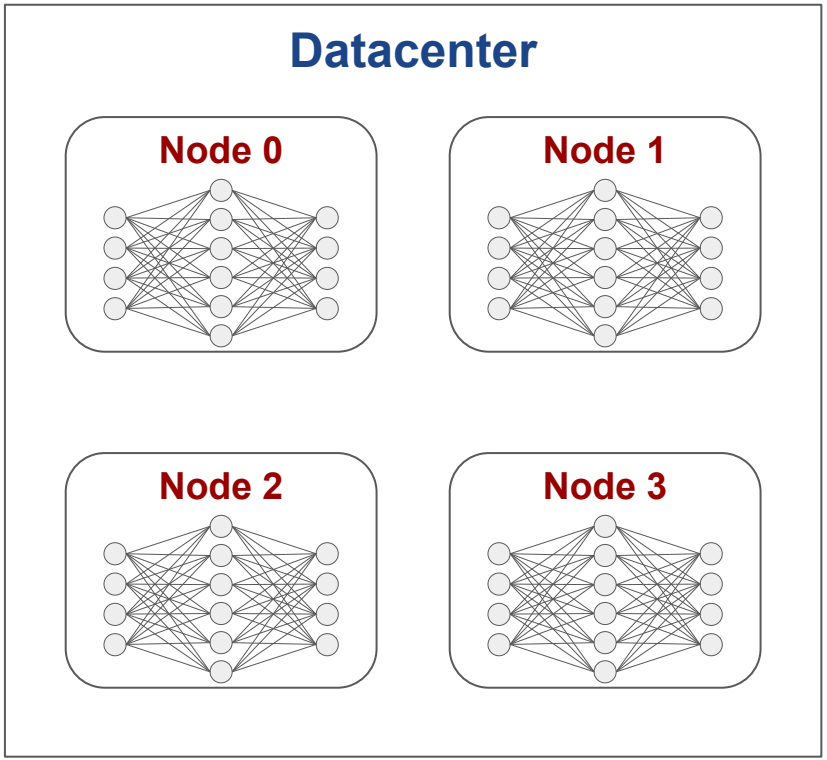
Ampel: Scheduling at the Network Cut in ML Training

Valerio Torsiello, Ayush Mishra, Sushovan Das, Lukas Röllin
Tommaso Bonato, Torsten Hoefler, Laurent Vanbever

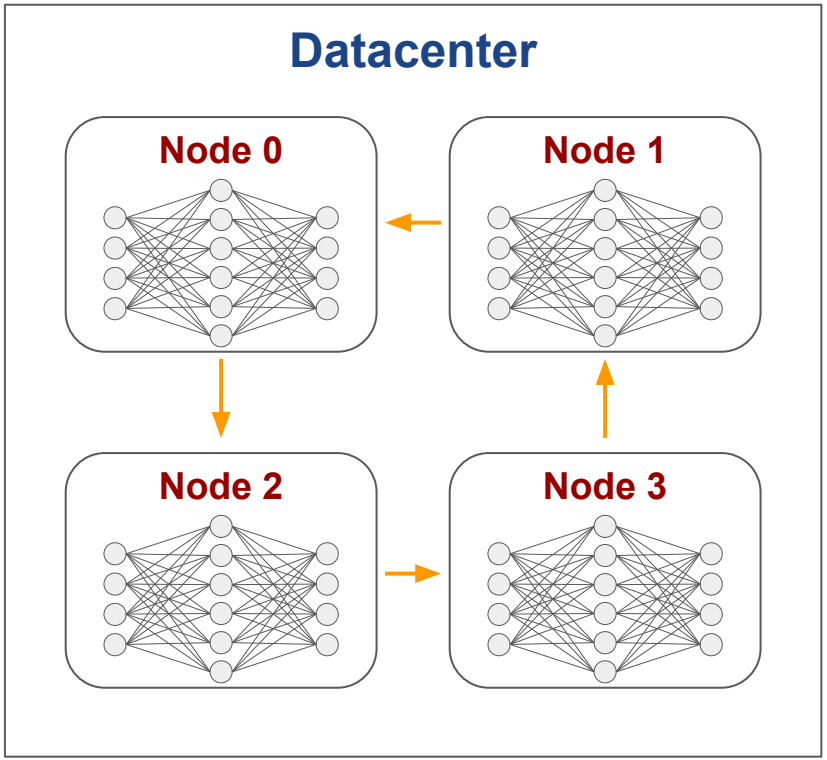
ETH zürich



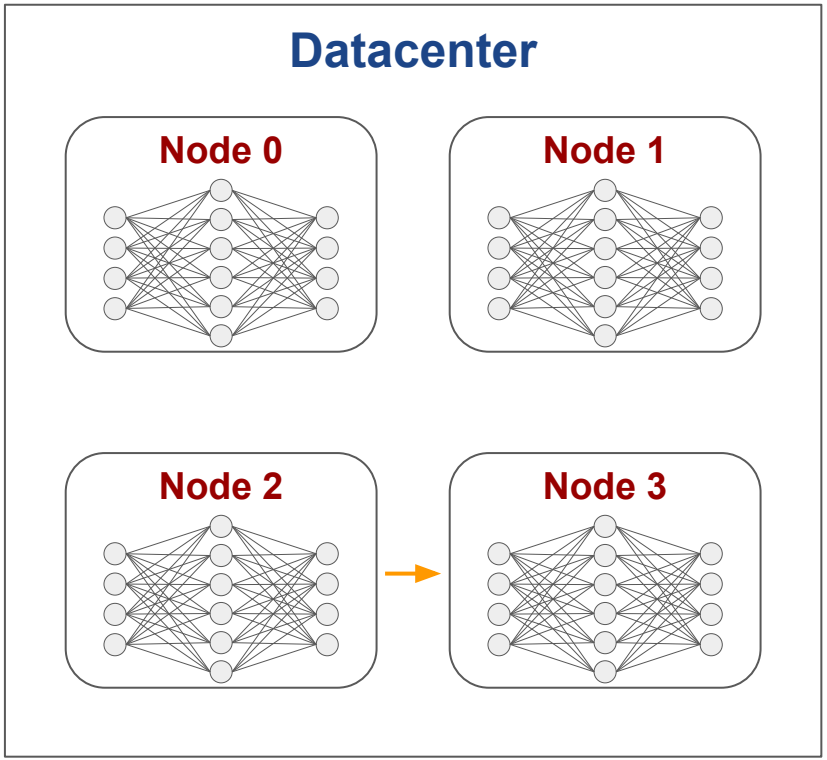
- Train models in a distributed fashion



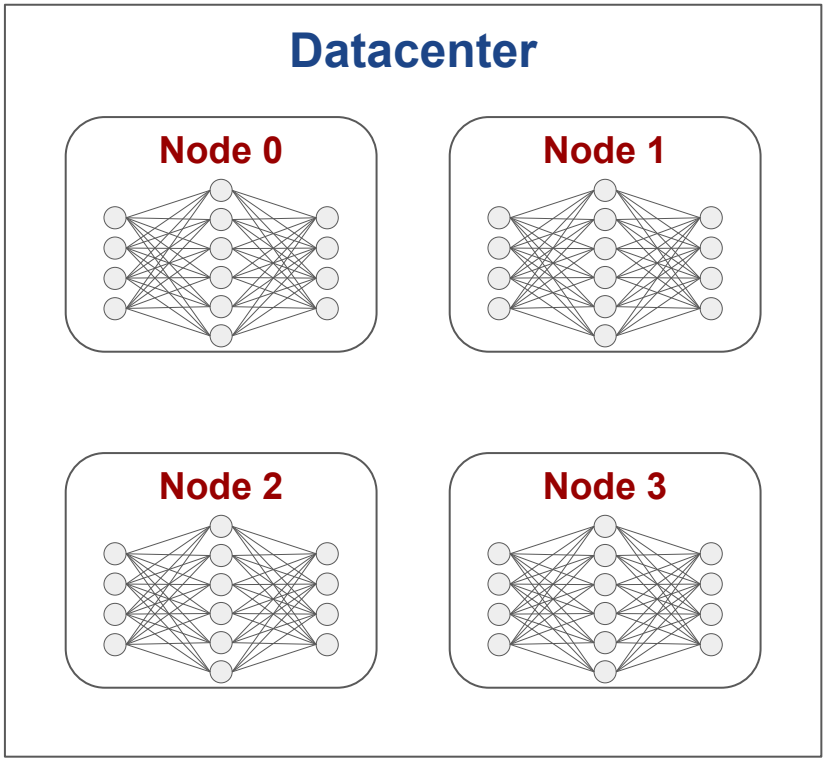
- Train models in a distributed fashion
- Regularly sync via collectives



- Train models in a distributed fashion
- Regularly sync via collectives
- Wait for the collective's flows to terminate

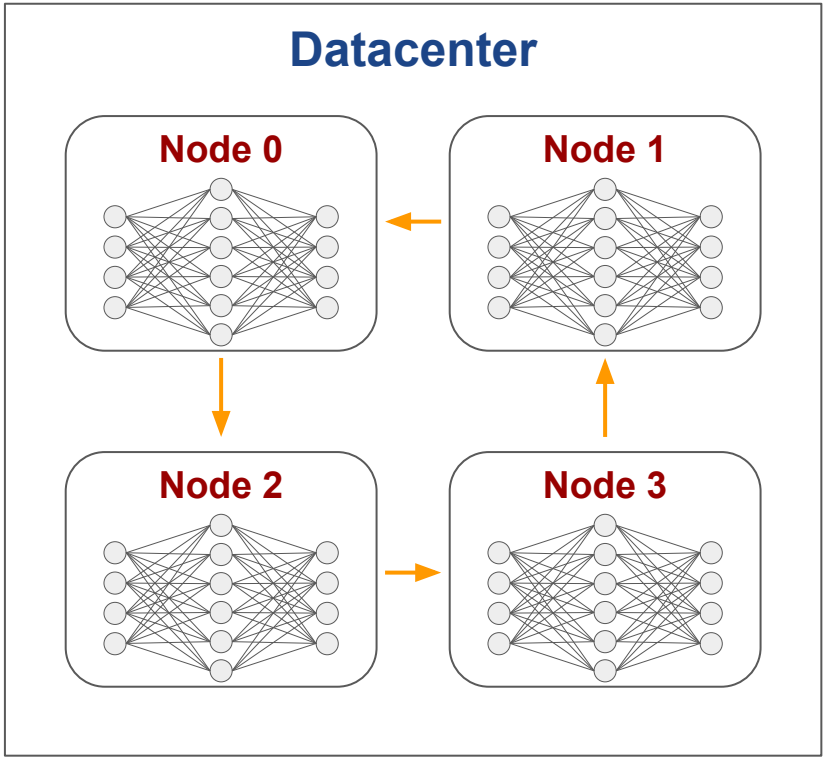


- Train models in a distributed fashion
- Regularly sync via collectives
- Wait for the collective's flows to terminate
- Resume computation



Ampel speeds up the communication phase

→ Tackles slowdowns induced by bandwidth demand exceeding capacity

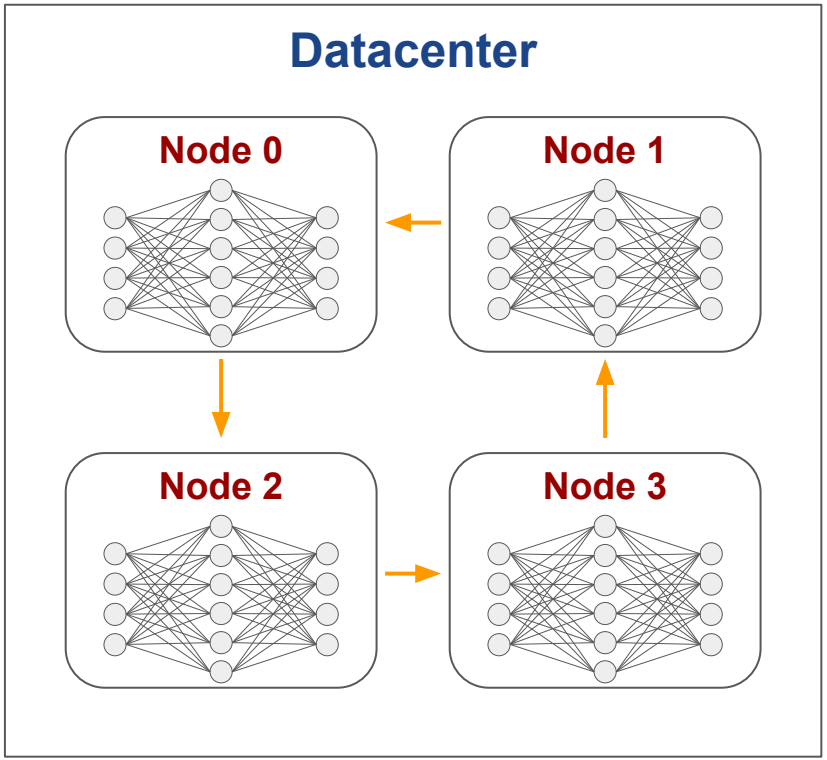


Ampel speeds up the communication phase

→ Tackles slowdowns induced by bandwidth demand exceeding capacity

Due to:

- Oversubscription[1,2]
- Failures[1,3]
- Job fragmentation[2,4]
- “Hard” collectives[5]
- etc.

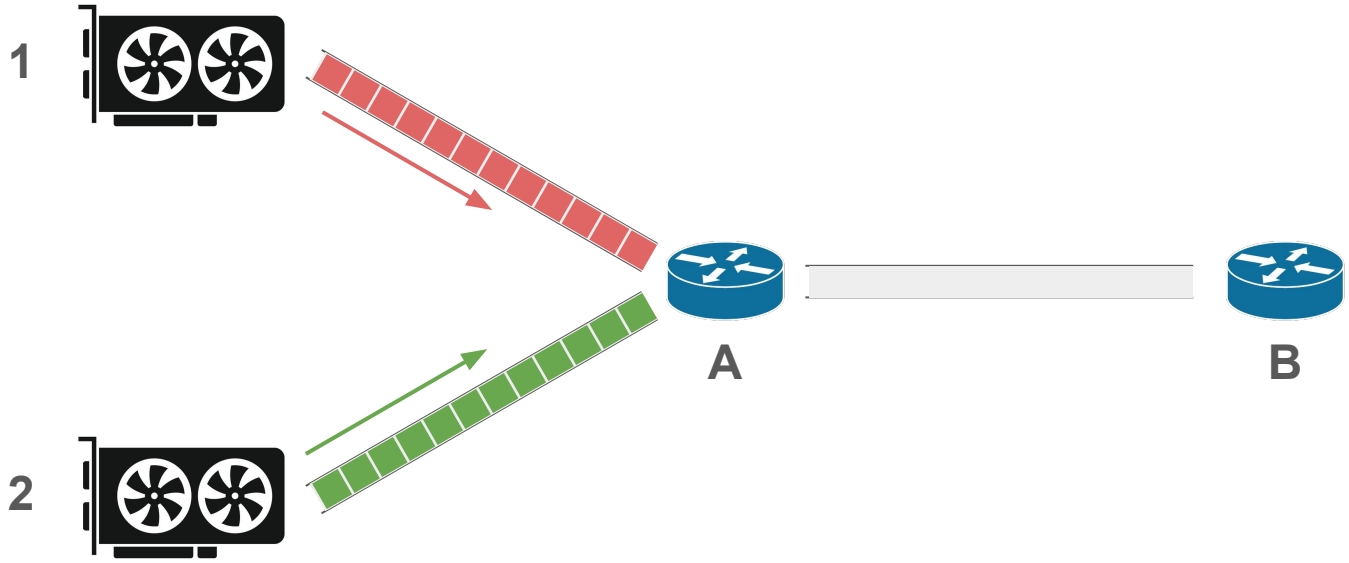


[1] Alibaba HPN: A Data Center Network for Large Language Model Training
[2] RDMA over Ethernet for Distributed AI Training at Meta Scale
[3] REPS: Recycled Entropy Packet Spraying for Adaptive Load Balancing and Failure Mitigation
[4] Astral: A Datacenter Infrastructure for Large Language Model Training at Scale
[5] MixNet: A Runtime Reconfigurable Optical-Electrical Fabric for Distributed Mixture-of-Experts Training

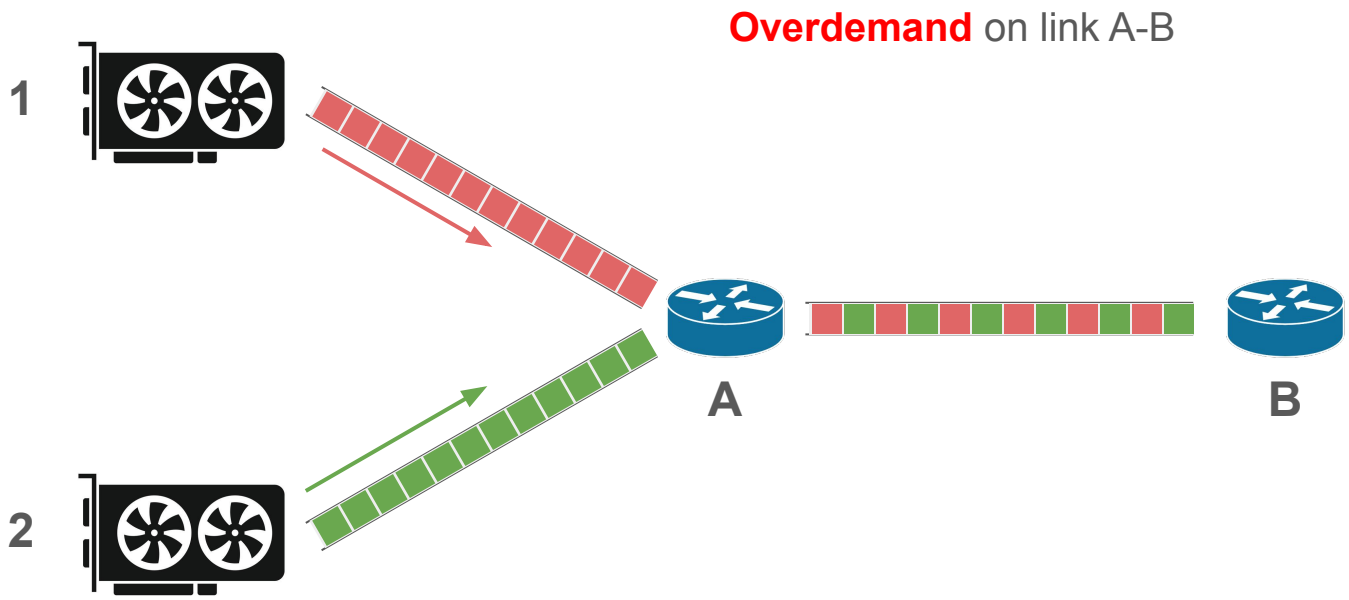
- **Background**: scheduling, packet spraying
- **Ampel**: identifying bottlenecks, coflow scheduling at the network cut
- **Evaluation**: consistently faster training iterations

How scheduling helps

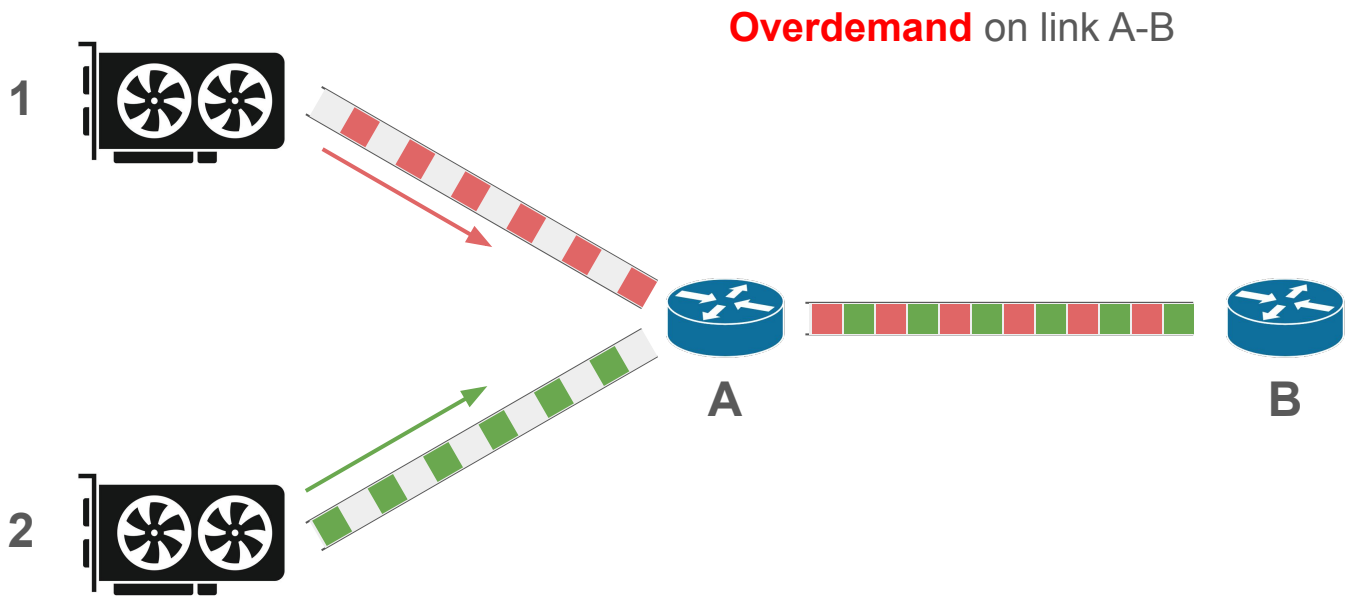
How scheduling helps



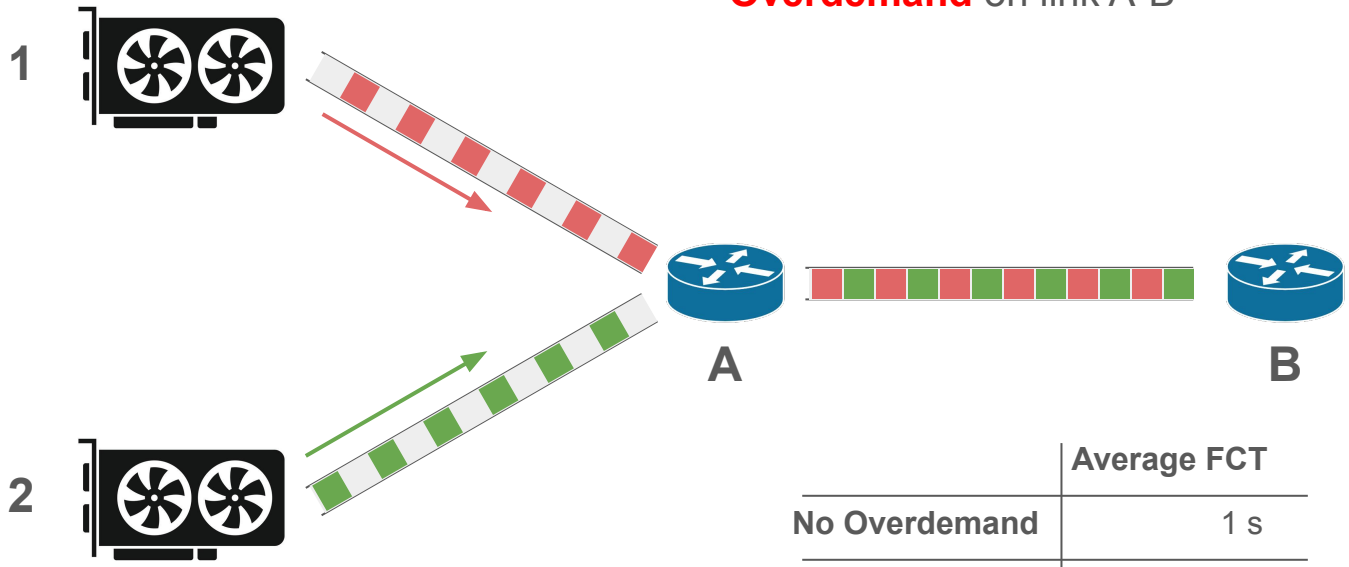
How scheduling helps



How scheduling helps



How scheduling helps



	Average FCT
No Overdemand	1 s
Overdemand	2 s

How scheduling helps

Schedule red flow

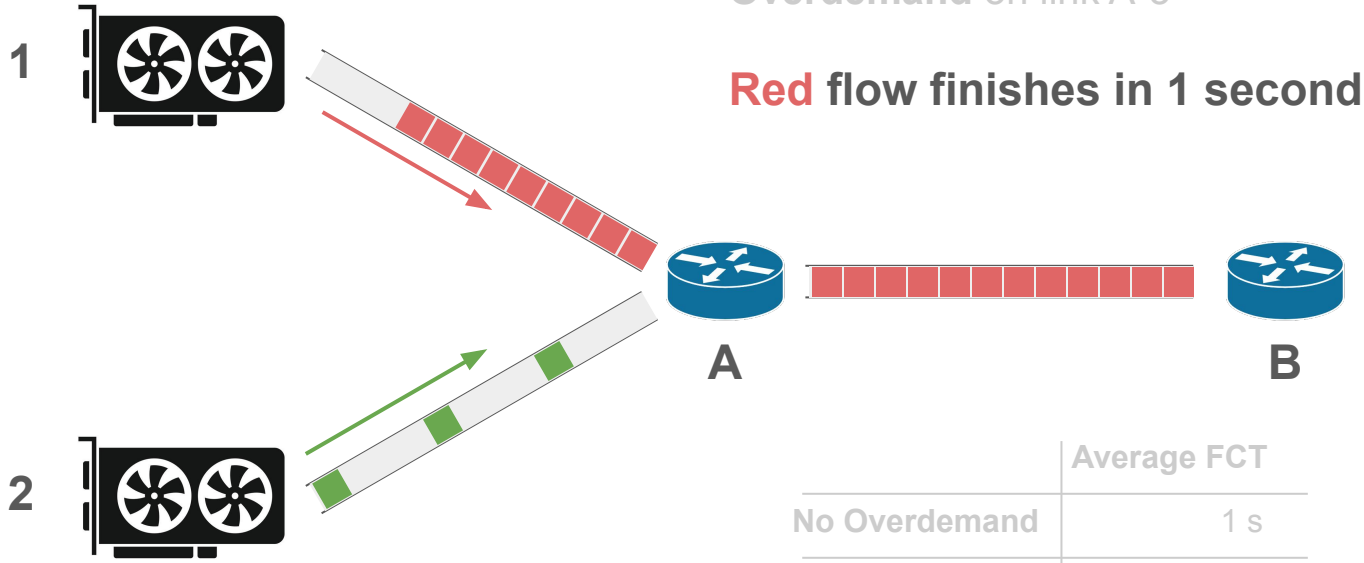
How scheduling helps

Schedule red flow

Red → High priority

Green → Low priority

How scheduling helps

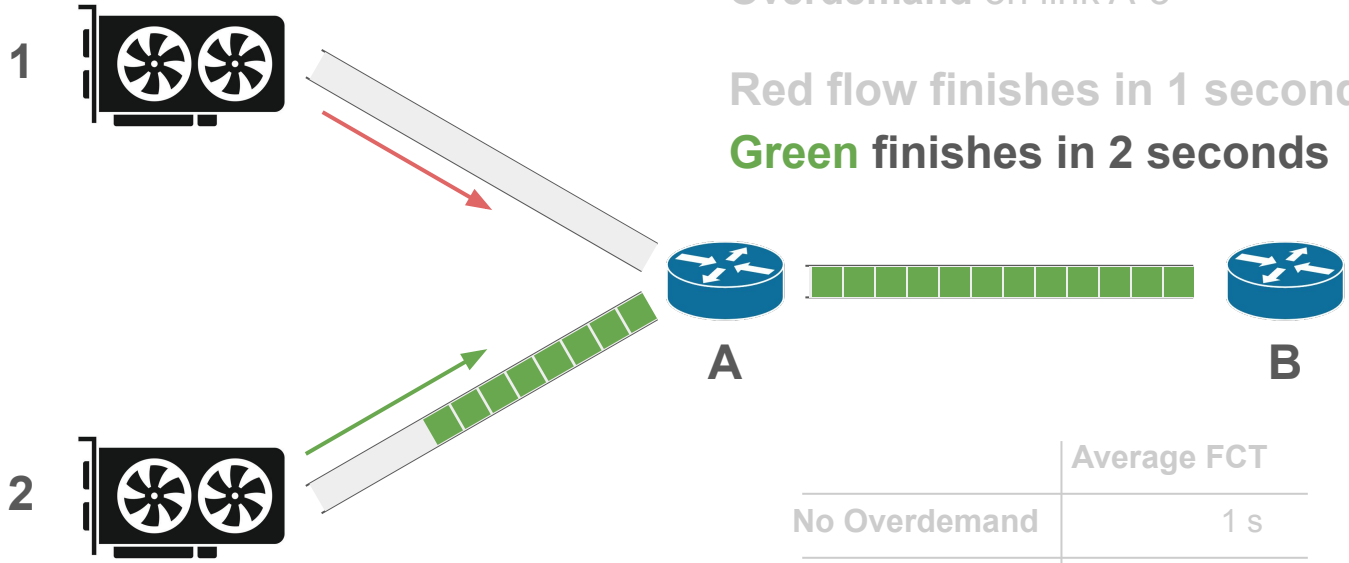


Overdemand on link A-3

Red flow finishes in 1 second

	Average FCT
No Overdemand	1 s
Overdemand	2 s

How scheduling helps



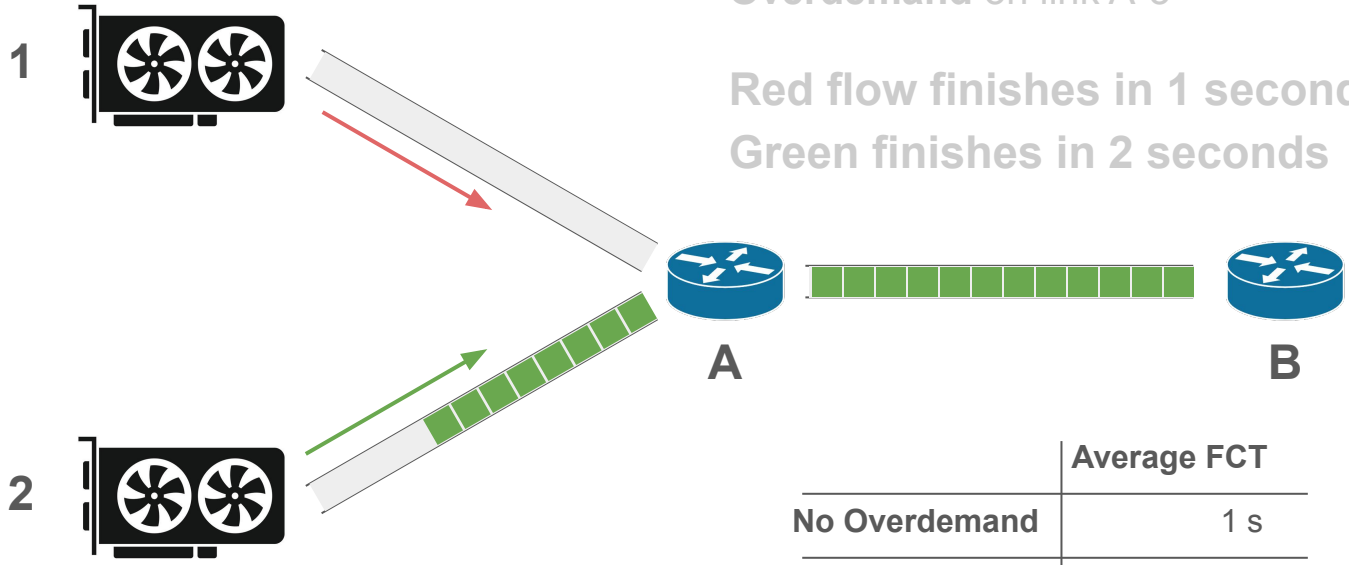
Overdemand on link A-3

Red flow finishes in 1 second

Green finishes in 2 seconds

	Average FCT
No Overdemand	1 s
Overdemand	2 s

How scheduling helps

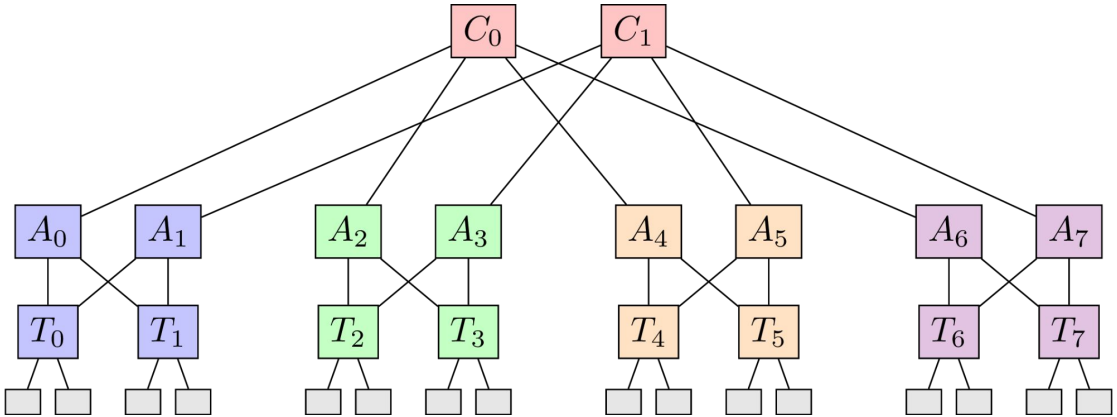


Overdemand on link A-3

Red flow finishes in 1 second
Green finishes in 2 seconds

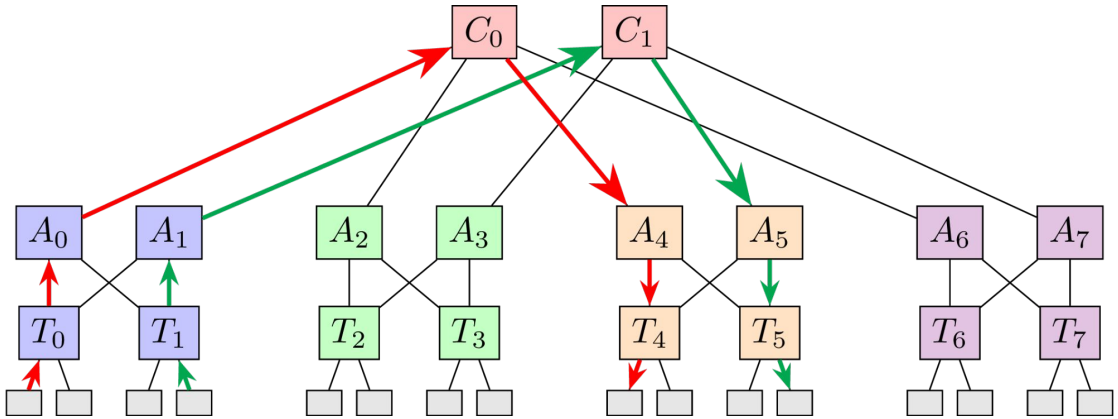
	Average FCT
No Overdemand	1 s
Overdemand	2 s
Overdemand + scheduling	**1.5 s**

Big Networks & Load-Balancing



Big Networks & Load-Balancing

Lucky hashing ☺

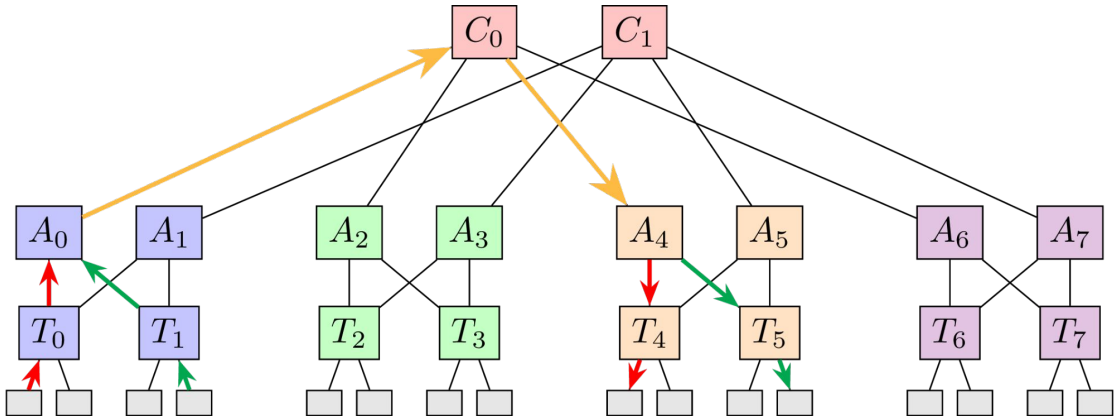


Single-path load balancing

Big Networks & Load-Balancing

Lucky hashing ☺

Not so lucky hashing ☹

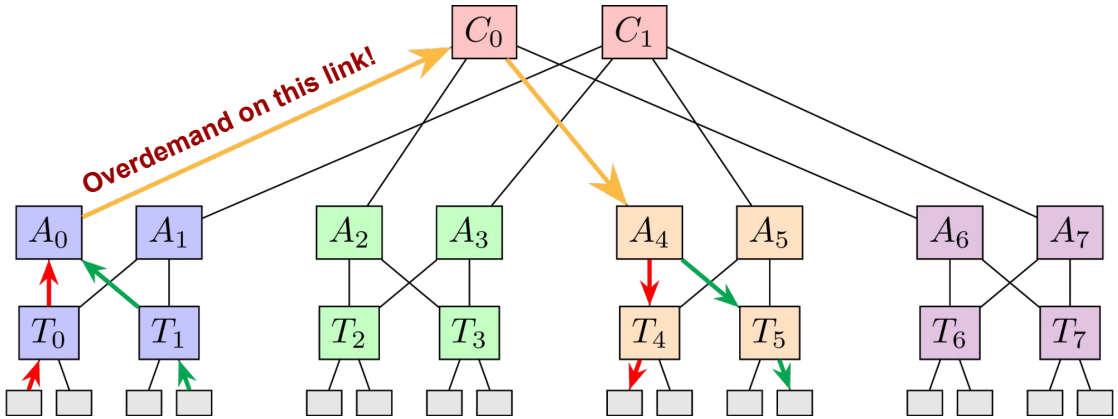


Single-path load balancing

Big Networks & Load-Balancing

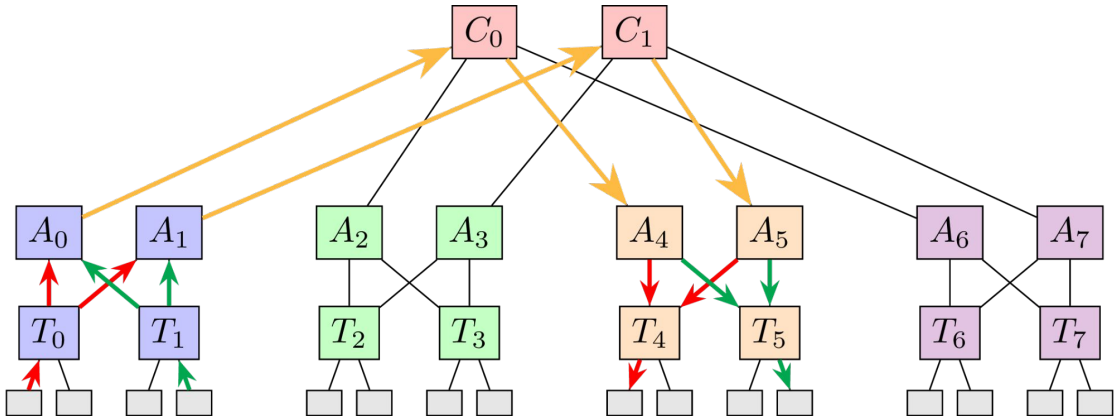
Lucky hashing ☺

Not so lucky hashing ☹



Single-path load balancing

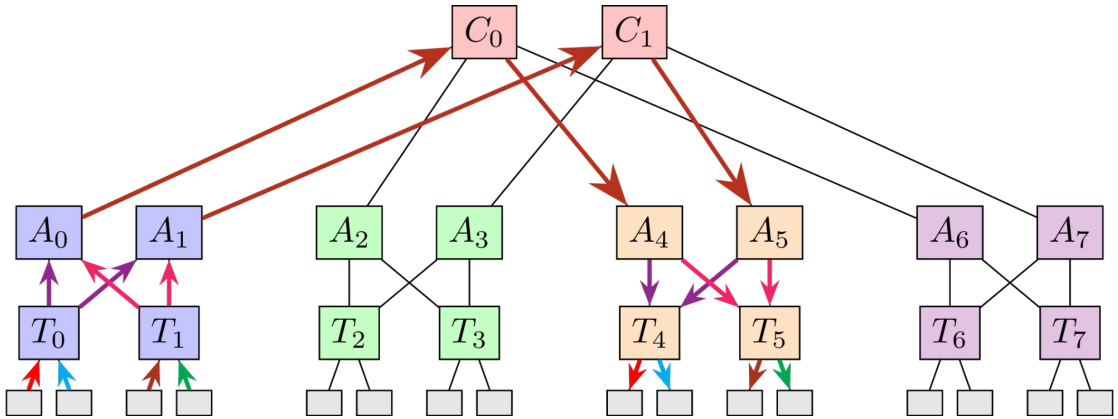
Big Networks & Load-Balancing



Packet Spraying

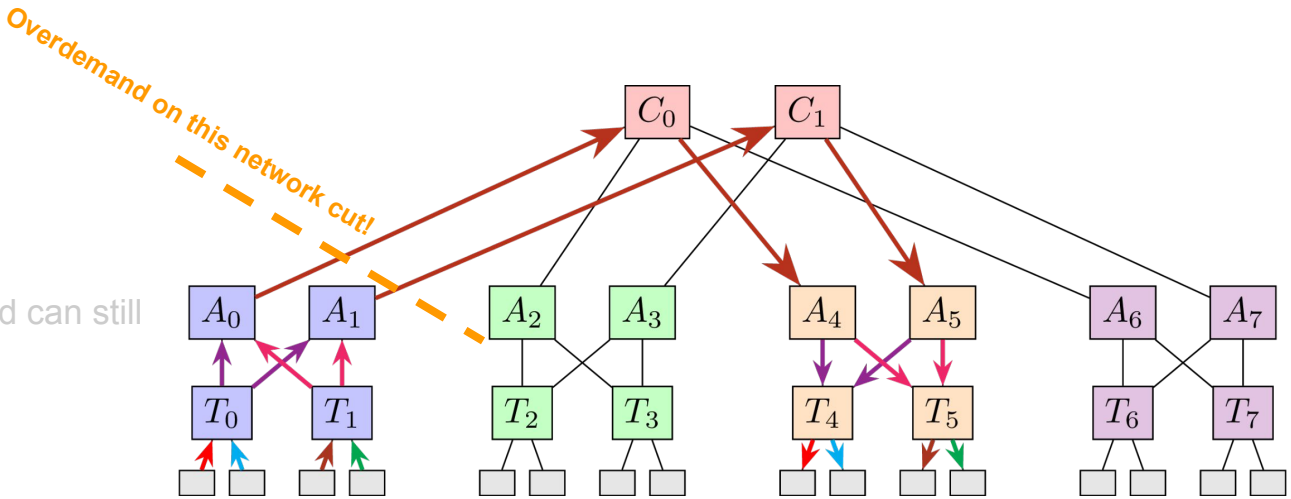
Big Networks & Load-Balancing

Overdemand can still occur!



Packet Spraying

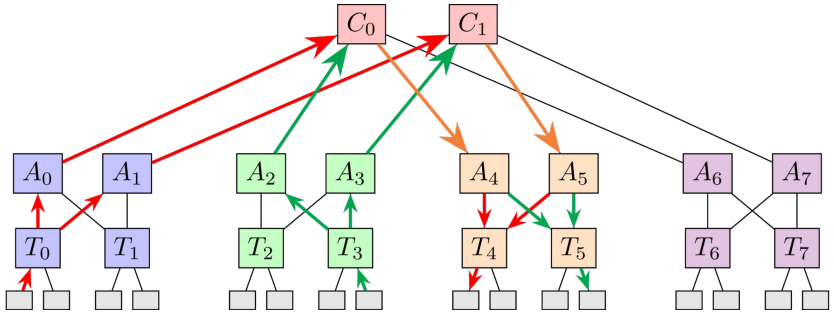
Big Networks & Load-Balancing



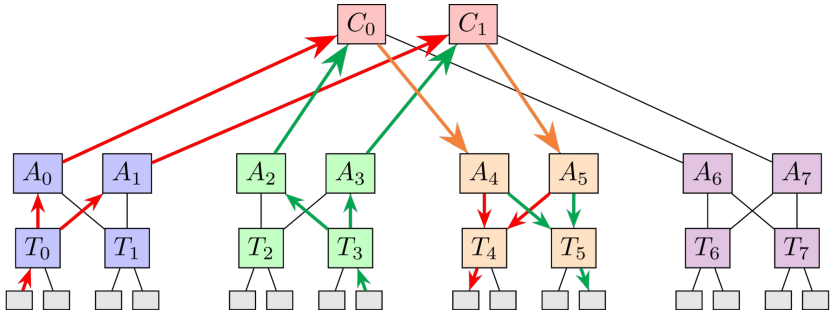
Overdemand can still occur!

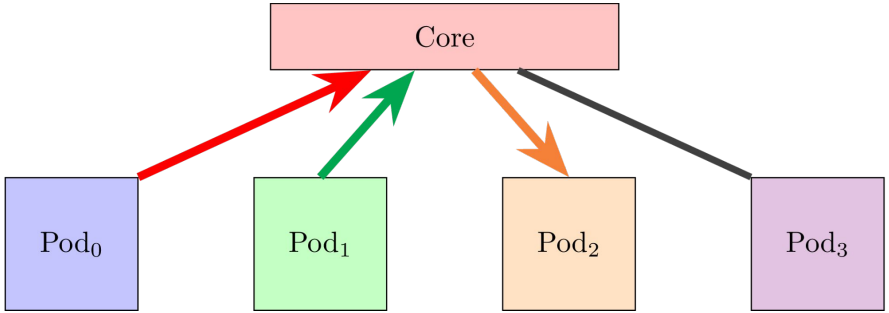
Packet Spraying

Abstracting the graph

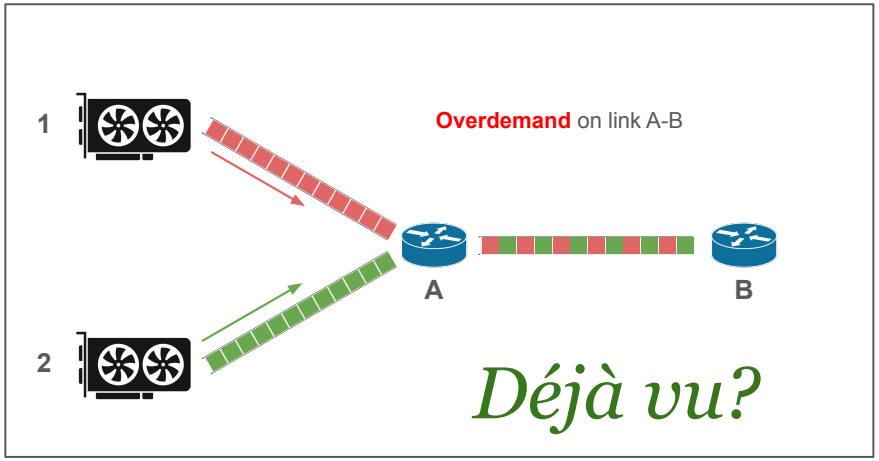


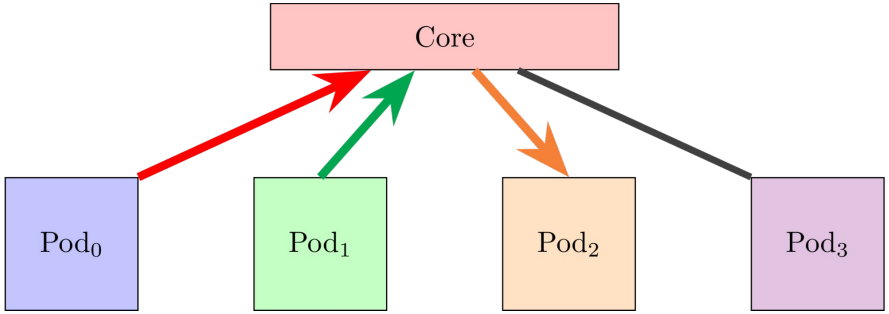
Abstracting the graph



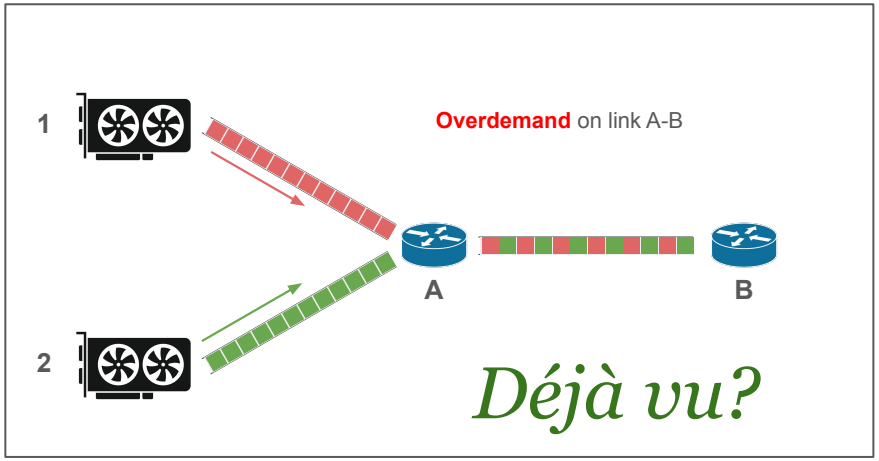


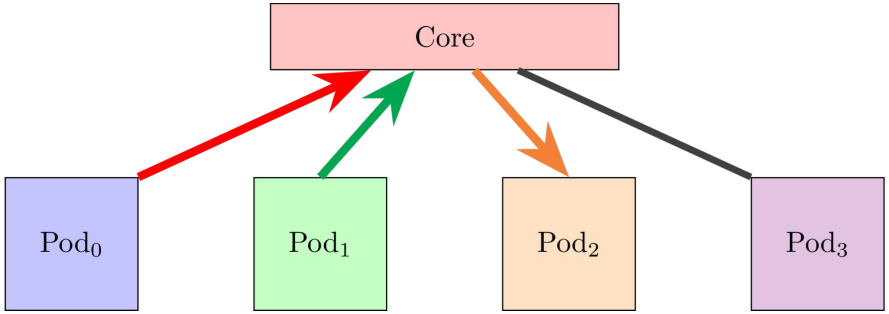
Abstracting the graph





Abstracting the graph





But there are many constantly changing flows with different sizes, dependencies, start times…

Scheduling Goals

Goal: dynamically schedule sets of flows so that they do not exceed any cut's capacity

Scheduling Goals

Goal: dynamically schedule sets of flows so that they do not exceed any cut's capacity

Problem: how do we choose these sets of flows?

Coflow Scheduling

A **coflow** is a set of flows that share the same downstream dependency

Coflow Scheduling

A **coflow** is a set of flows that share the same downstream dependency

Coflow scheduling optimises the tail completion of the flows in the coflow

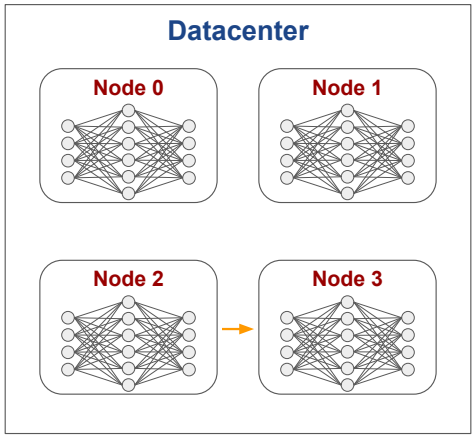
Coflow Scheduling

A **coflow** is a set of flows that share the same downstream dependency

Coflow scheduling optimises the tail completion of the flows in the coflow

Déjà vu?

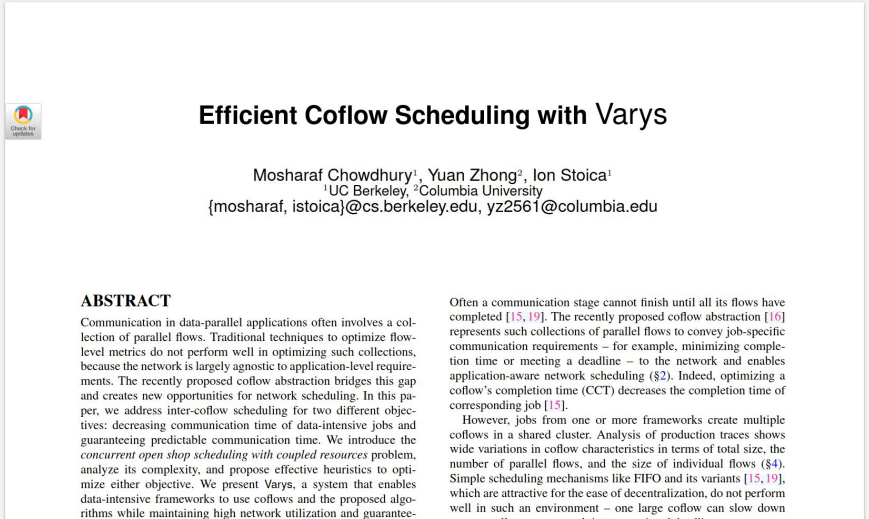
- Train models in a distributed fashion
- Regularly sync via collectives
- Wait for the collective's flows to terminate



Coflow Scheduling

A **coflow** is a set of flows that share the same downstream dependency

Coflow scheduling optimises the tail completion of the flows in the coflow



Efficient Coflow Scheduling with Varys

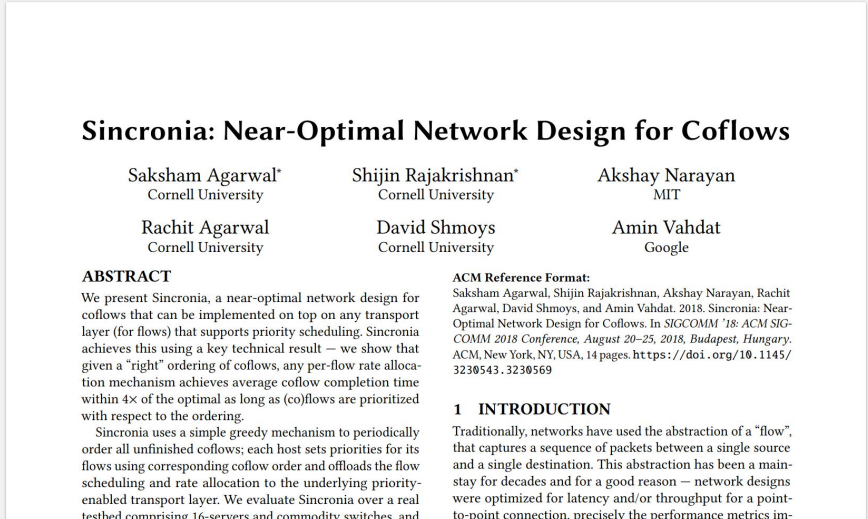
Mosharaf Chowdhury[1], Yuan Zhong[2], Ion Stoica[1]
[1]UC Berkeley, [2]Columbia University
{mosharaf, istoica}@cs.berkeley.edu, yz2561@columbia.edu

ABSTRACT

Communication in data-parallel applications often involves a collection of parallel flows. Traditional techniques to optimize flow-level metrics do not perform well in optimizing such collections, because the network is largely agnostic to application-level requirements. The recently proposed coflow abstraction bridges this gap and creates new opportunities for network scheduling. In this paper, we address inter-coflow scheduling for two different objectives: decreasing communication time of data-intensive jobs and guaranteeing predictable communication time. We introduce the *concurrent open shop scheduling with coupled resources* problem, analyze its complexity, and propose effective heuristics to optimize either objective. We present Varys, a system that enables data-intensive frameworks to use coflows and the proposed algorithms while maintaining high network utilization and guarantee-

Often a communication stage cannot finish until all its flows have completed [15, 19]. The recently proposed coflow abstraction [16] represents such collections of parallel flows to convey job-specific communication requirements – for example, minimizing completion time or meeting a deadline – to the network and enables application-aware network scheduling (§2). Indeed, optimizing a coflow's completion time (CCT) decreases the completion time of corresponding job [15].

However, jobs from one or more frameworks create multiple coflows in a shared cluster. Analysis of production traces shows wide variations in coflow characteristics in terms of total size, the number of parallel flows, and the size of individual flows (§4). Simple scheduling mechanisms like FIFO and its variants [15, 19], which are attractive for the ease of decentralization, do not perform well in such an environment – one large coflow can slow down



Sincronia: Near-Optimal Network Design for Coflows

Saksham Agarwal*
Cornell University

Shijin Rajakrishnan*
Cornell University

Akshay Narayan
MIT

Rachit Agarwal
Cornell University

David Shmoys
Cornell University

Amin Vahdat
Google

ABSTRACT

We present Sincronia, a near-optimal network design for coflows that can be implemented on top on any transport layer (for flows) that supports priority scheduling. Sincronia achieves this using a key technical result — we show that given a "right" ordering of coflows, any per-flow rate allocation mechanism achieves average coflow completion time within 4× of the optimal as long as (co)flows are prioritized with respect to the ordering.

Sincronia uses a simple greedy mechanism to periodically order all unfinished coflows; each host sets priorities for its flows using corresponding coflow order and offloads the flow scheduling and rate allocation to the underlying priority-enabled transport layer. We evaluate Sincronia over a real testbed comprising 16-servers and commodity switches, and

ACM Reference Format:
Saksham Agarwal, Shijin Rajakrishnan, Akshay Narayan, Rachit Agarwal, David Shmoys, and Amin Vahdat. 2018. Sincronia: Near-Optimal Network Design for Coflows. In *SIGCOMM '18: ACM SIGCOMM 2018 Conference, August 20–25, 2018, Budapest, Hungary.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3230543.3230569

1 INTRODUCTION

Traditionally, networks have used the abstraction of a "flow", that captures a sequence of packets between a single source and a single destination. This abstraction has been a mainstay for decades and for a good reason — network designs were optimized for latency and/or throughput for a point-to-point connection, precisely the performance metrics im-

Coflow Scheduling

A **coflow** is a set of flows that share the same downstream dependency

Coflow scheduling optimises the tail completion of the flows in the coflow

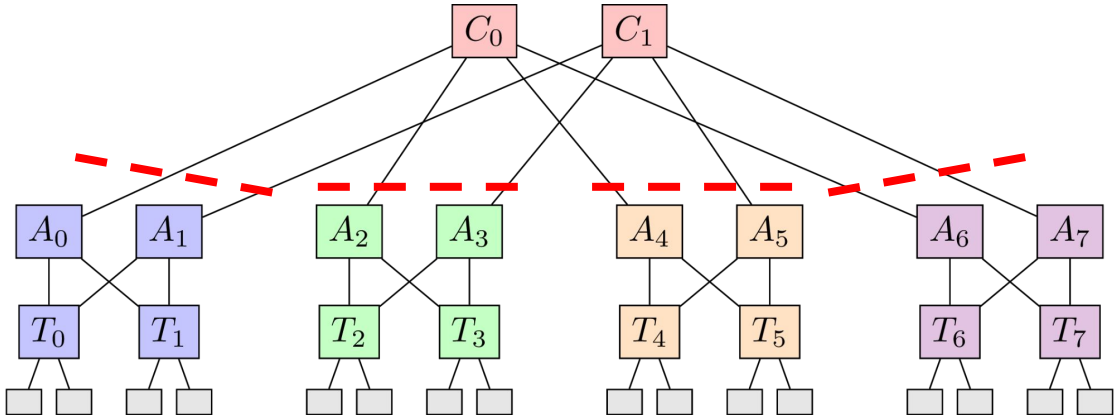
We reformulate the **Coflow Scheduling** problem as the
Network Cut Coflow Scheduling problem

Ampel

1. Identify oversubscribed network cuts and their capacity

Ampel

1. Identify oversubscribed network cuts and their capacity



Ampel

1. Identify oversubscribed network cuts and their capacity
2. Identify flows crossing these cuts grouped by the coflows they belong to

Ampel

1. Identify oversubscribed network cuts and their capacity
2. Identify flows crossing these cuts grouped by the coflows they belong to
3. Rank these coflows, e.g. via a coflow scheduler

Ampel

1. Identify oversubscribed network cuts and their capacity
2. Identify flows crossing these cuts grouped by the coflows they belong to
3. Rank these coflows, e.g. via a coflow scheduler
4. Greedily allocate bandwidth to the coflows as long as the cuts have capacity

Ampel

1. Identify oversubscribed network cuts and their capacity
2. Identify flows crossing these cuts grouped by the coflows they belong to
3. Rank these coflows, e.g. via a coflow scheduler
4. Greedily allocate bandwidth to the coflows as long as the cuts have capacity

The scheduling is enforced via network priorities:

Scheduled Coflows → High priority

Unscheduled Coflows → Low priority

Experiment setup

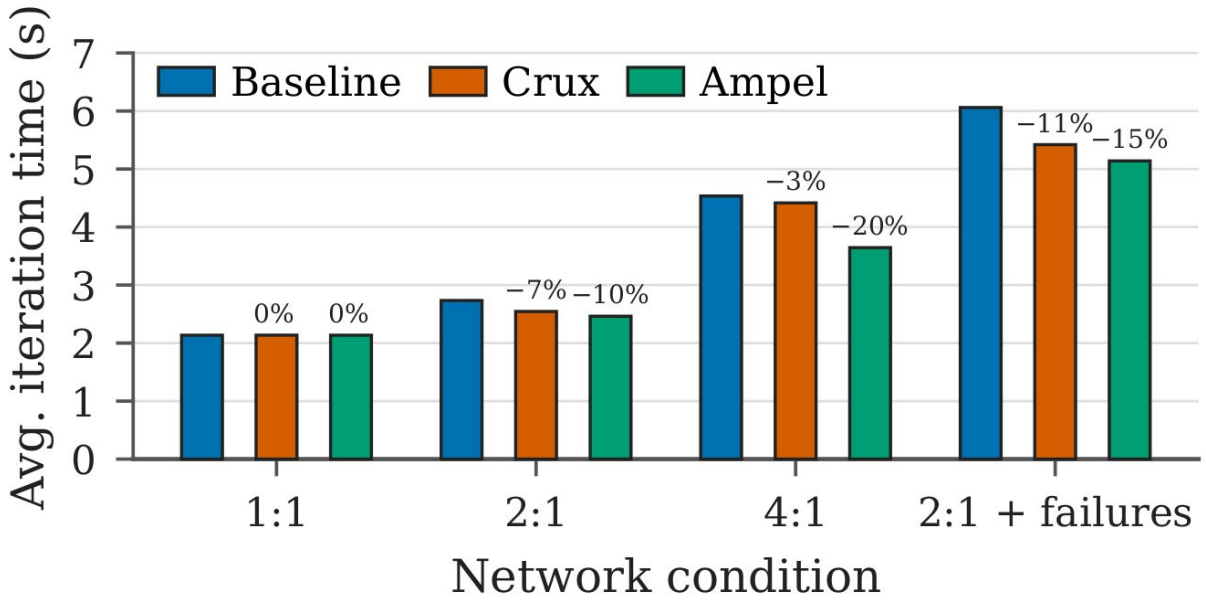
Simulations on htsim with packet spraying

8 Llama7B models using FSDP

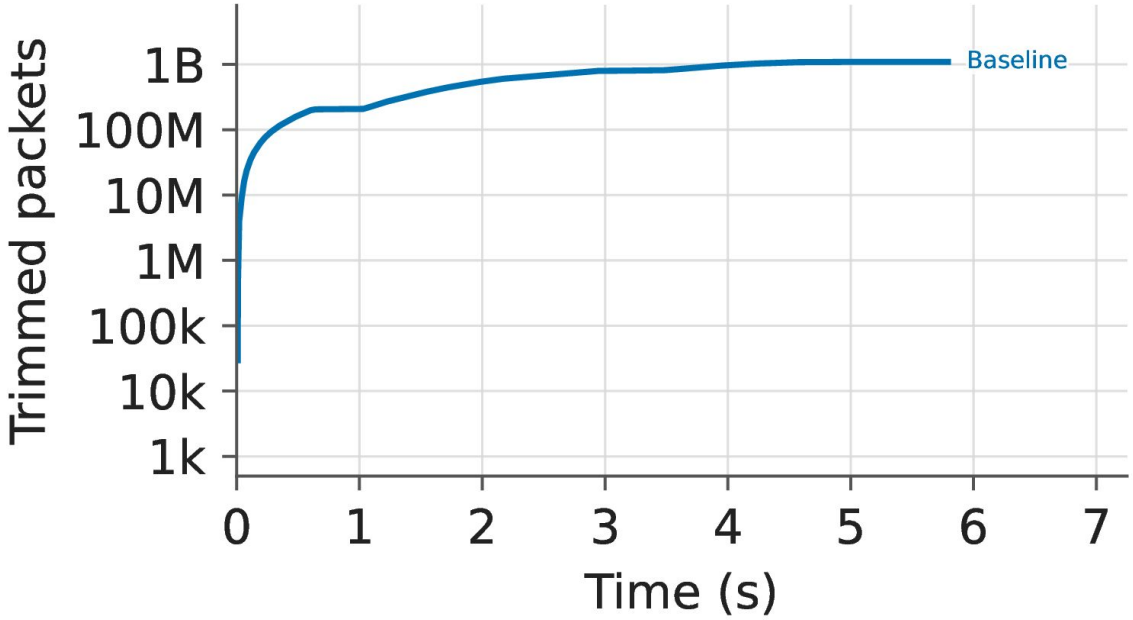
97 AllGathers and 97 ReduceScatters per iteration

128 node Fat Tree topology, varying degrees of oversubscription

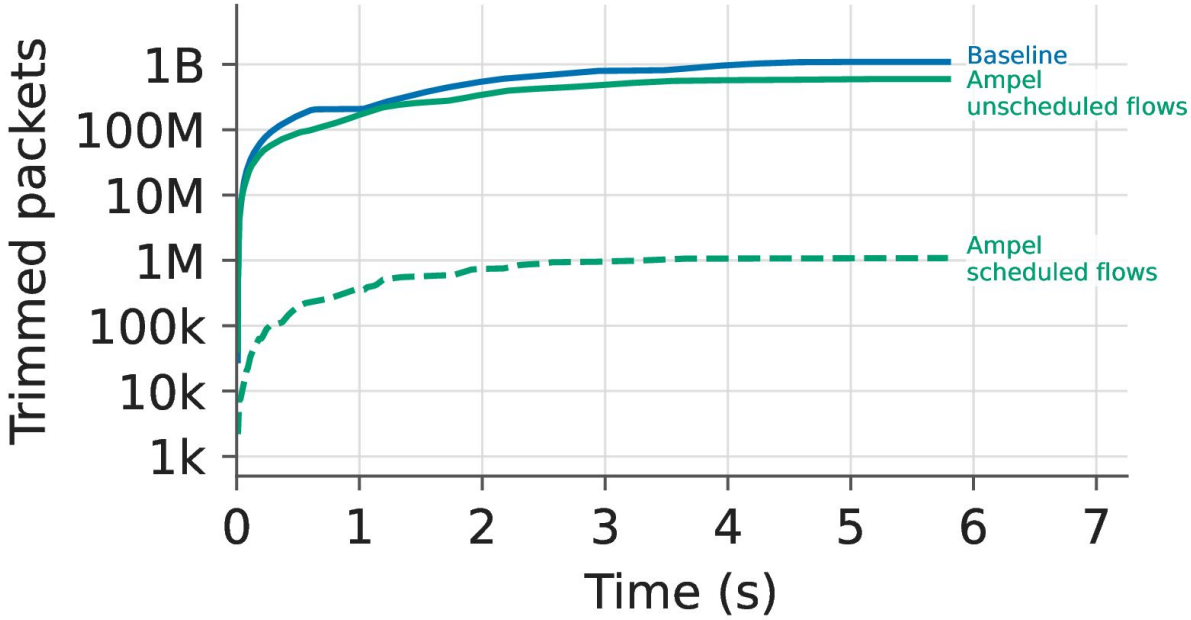
Lower training times



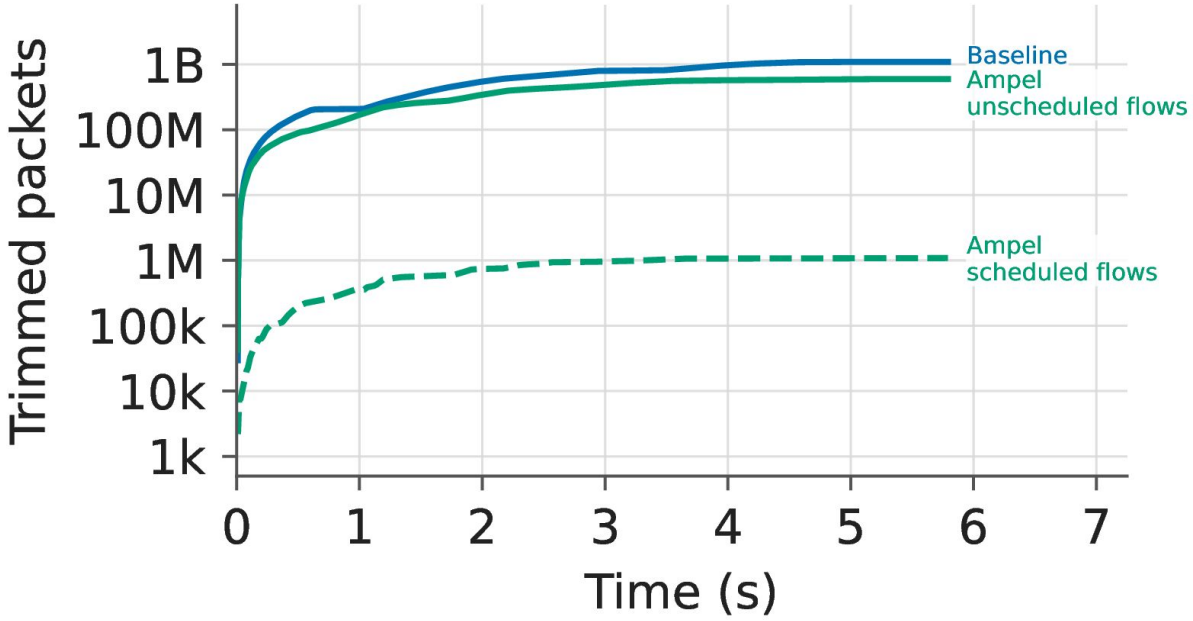
Scheduled coflows perform as if isolated

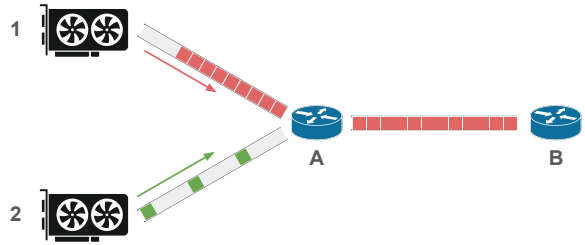


Scheduled coflows perform as if isolated



Scheduled coflows perform as if isolated



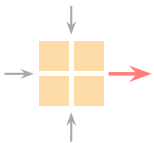


Conclusion

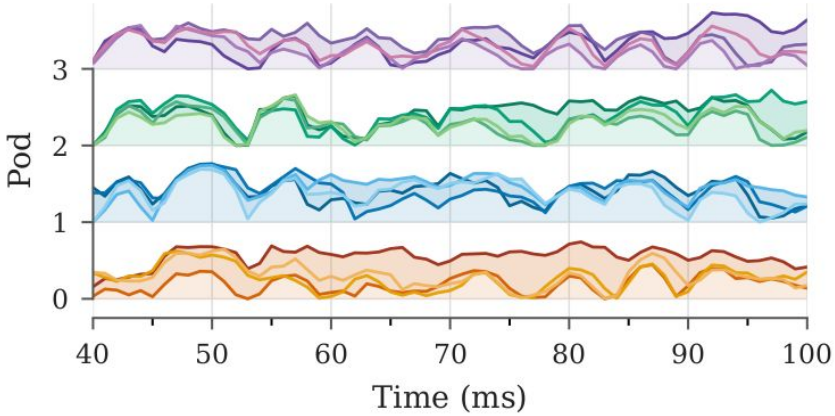
- Nature of bottlenecks is changing
- With packet spraying, overdemand occurs across cuts rather than single links
- The coflow abstraction lends itself well to ML workloads
- Ampel shows large gains in training performance over the state of the art

Future Work

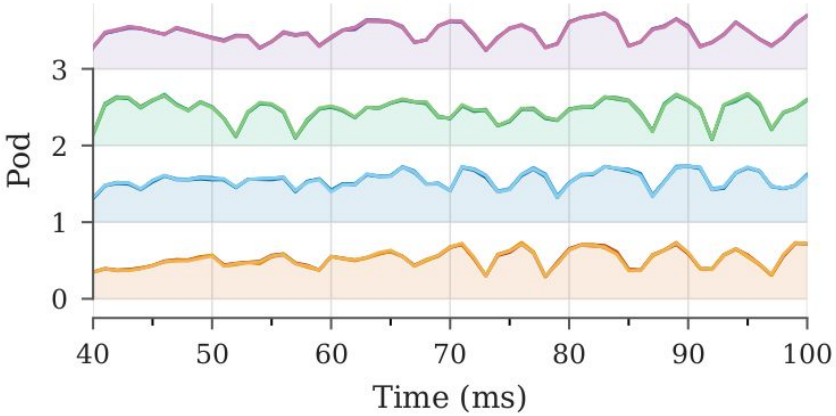
- How well does Ampel generalise to other topologies?
- How best do we group flows into coflows?



Packet Spraying's even flow spread

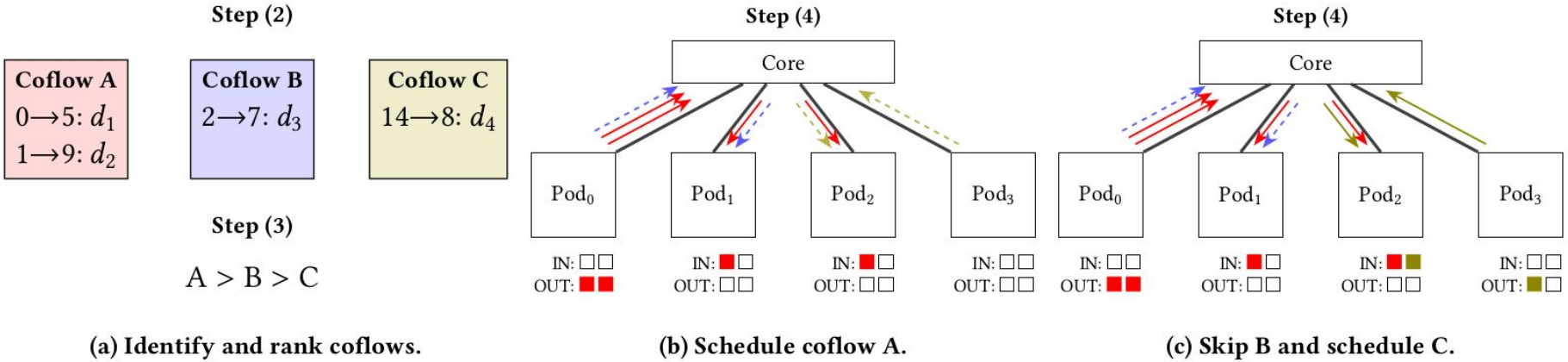


(a) Single-Path Ultra-Ethernet

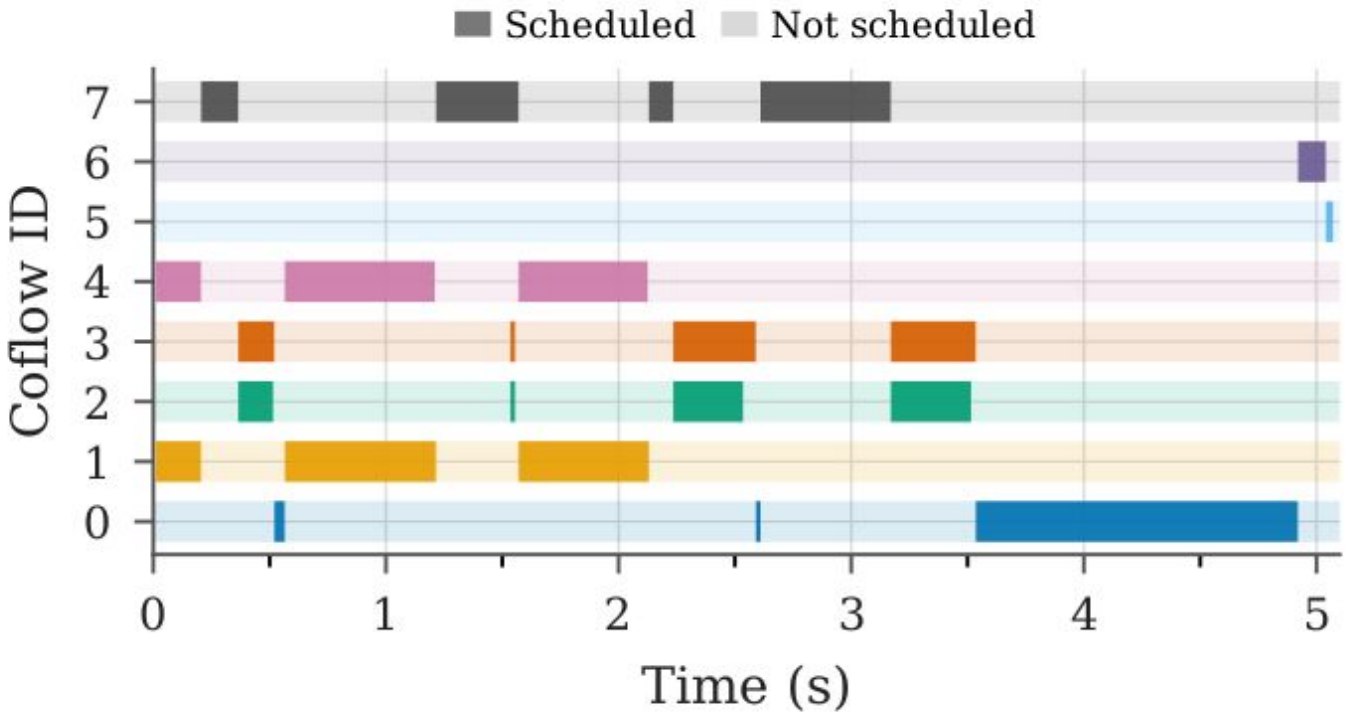


(b) Packet Spraying Ultra-Ethernet

Ampel algorithm



(a) Identify and rank coflows. (b) Schedule coflow A. (c) Skip B and schedule C.

Figure 4: Overview of Ampel's scheduling algorithm.

Dynamic priorities