SIRD
A Sender-Informed, Receiver-Driven Datacenter Transport

Konstantinos Prasopoulos, Ryan Kosta*, Edouard Bugnion, Marios Kogias†

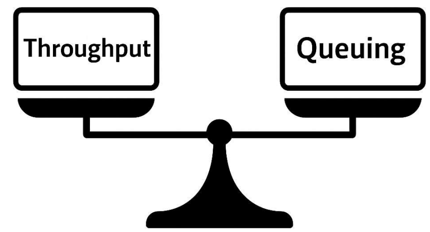
EPFL, *UCSD, †Imperial College London

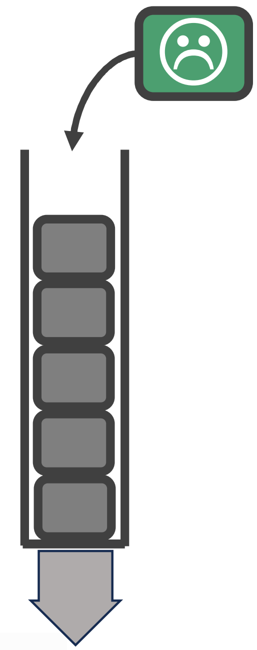
NSDI 2025 – Philadelphia, PA

Datacenter Congestion Control

"Safe to send packet to server X?"

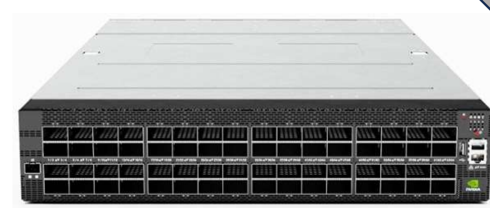
- Throughput-intensive - eg. Disaggregated Storage
  - Use the bandwidth
- Latency-sensitive - eg. Memcached
  - Limit network queueing
  - 10x from queuing & >100x from loss.





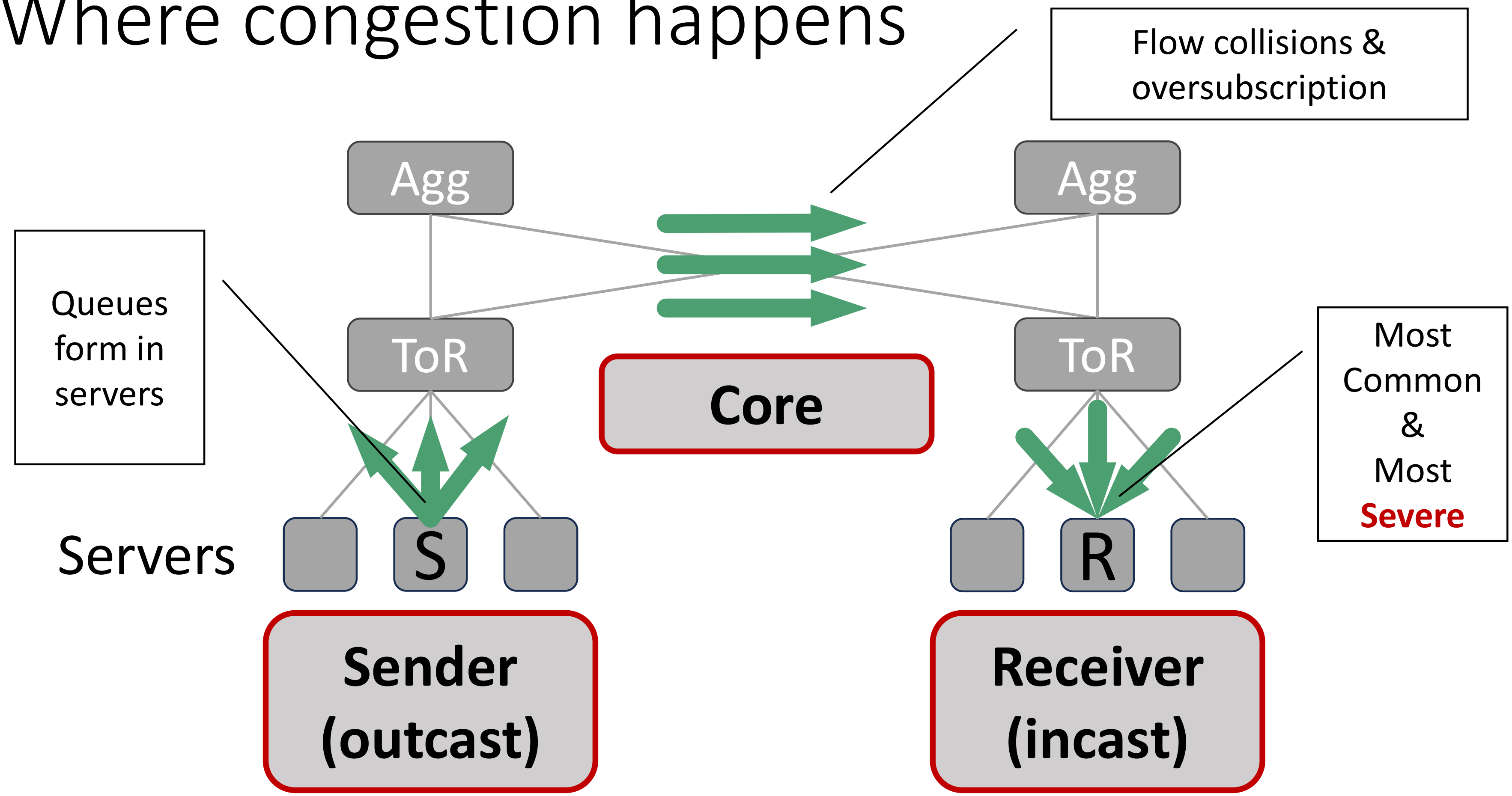
Datacenter Environment

- 800Gbps links
- Switch buffers not keeping up -> 1.6ms to fill SN5600



Affects application performance & environment evolves

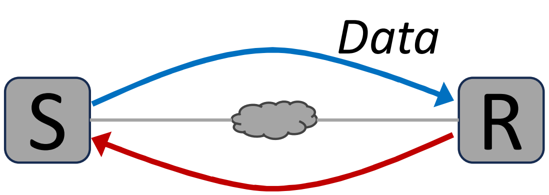
Where congestion happens



15+ years of research on DC CC

Sender Driven (SD) - Reactive

(DCTCP, HPCC, Swift…)



Feedback (Delay, ECN): "Sent Too Much"

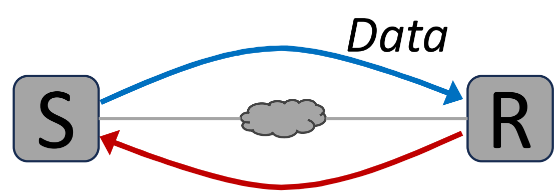
- Richer signal (INT) => switch dependency

\+ Naturally handle congestion everywhere

\- Slow to converge => more queuing

Receiver Driven (RD) - Proactive

(pHost, NDP, Homa, dcPIM…)



Credit: "I allow you to Send"

- Often require switch support (priorities)

\+ Ideal for incast – Can avoid congestion

\- Crediting based on receiver's view *alone* leads to low throughput *or* high queuing

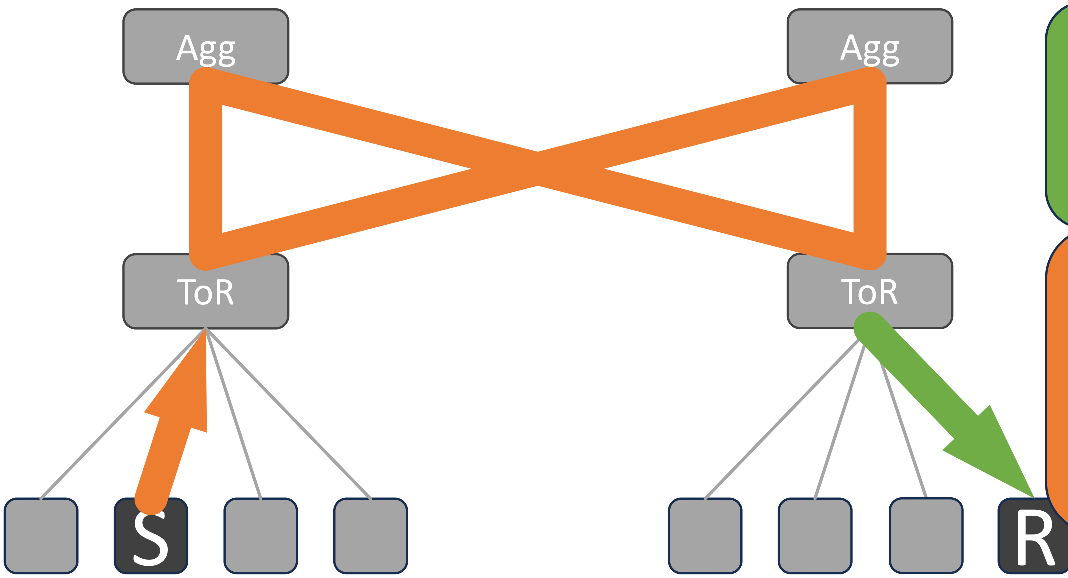
Sacrifice one of (throughput, queuing, generality, op. complexity)

Objective

A congestion control protocol that:

- Deals with incast via RD admission control
- Addresses congestion everywhere
- Achieves high utilization with minimal queuing
- Minimizes dependence on switch features

SIRD contribution

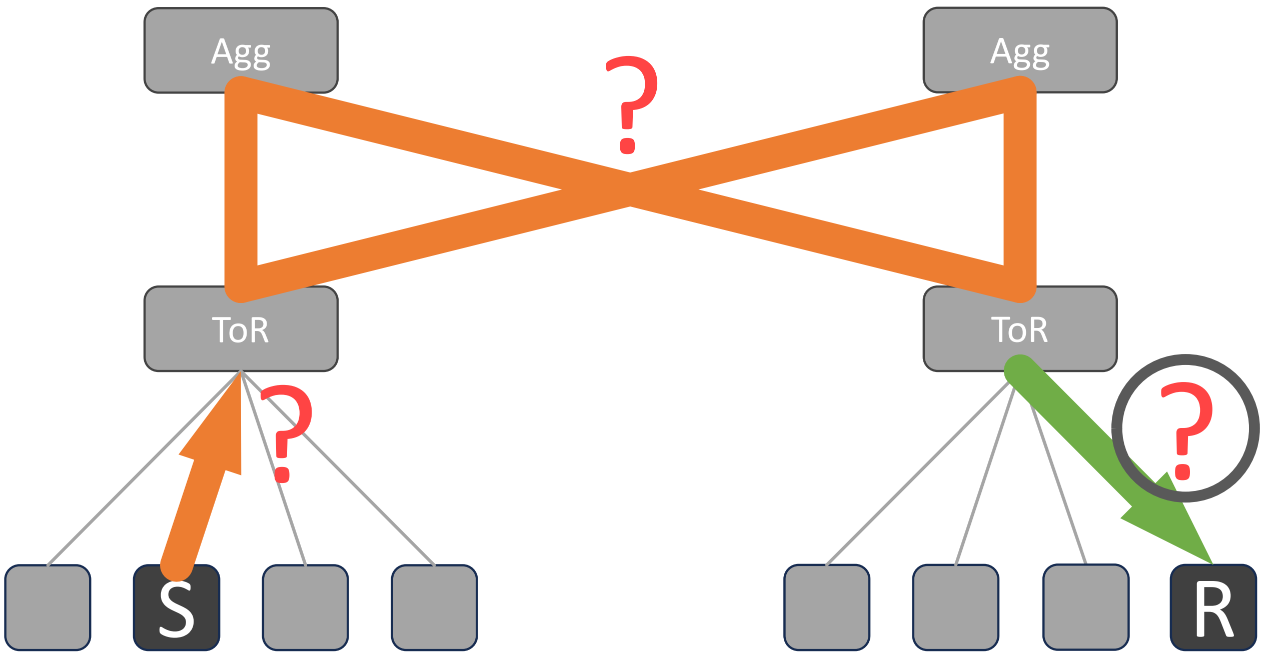


Congestion on Single-Owner Links:
Manage Proactively
=> RD credit admission

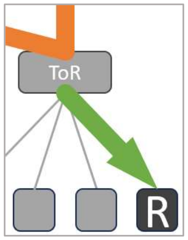
Congestion on Shared Links:
Manage Reactively
Feedback from Senders & Core
=> Credit allocation f(congestion)

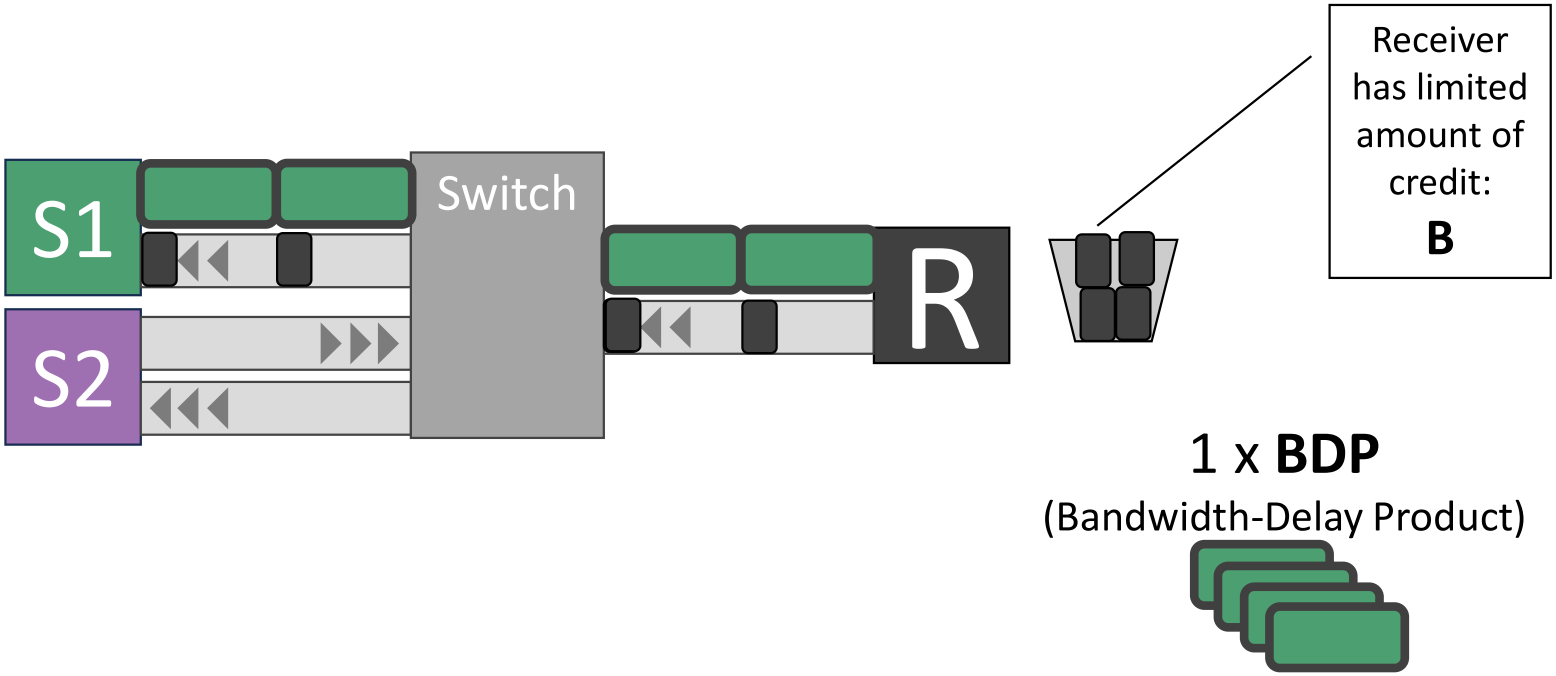
Approach enables precise credit allocation
=> High throughput with minimal queuing

Let's build SIRD: Receiver Congestion

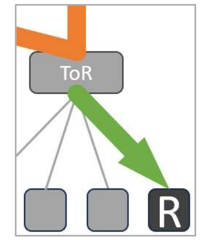


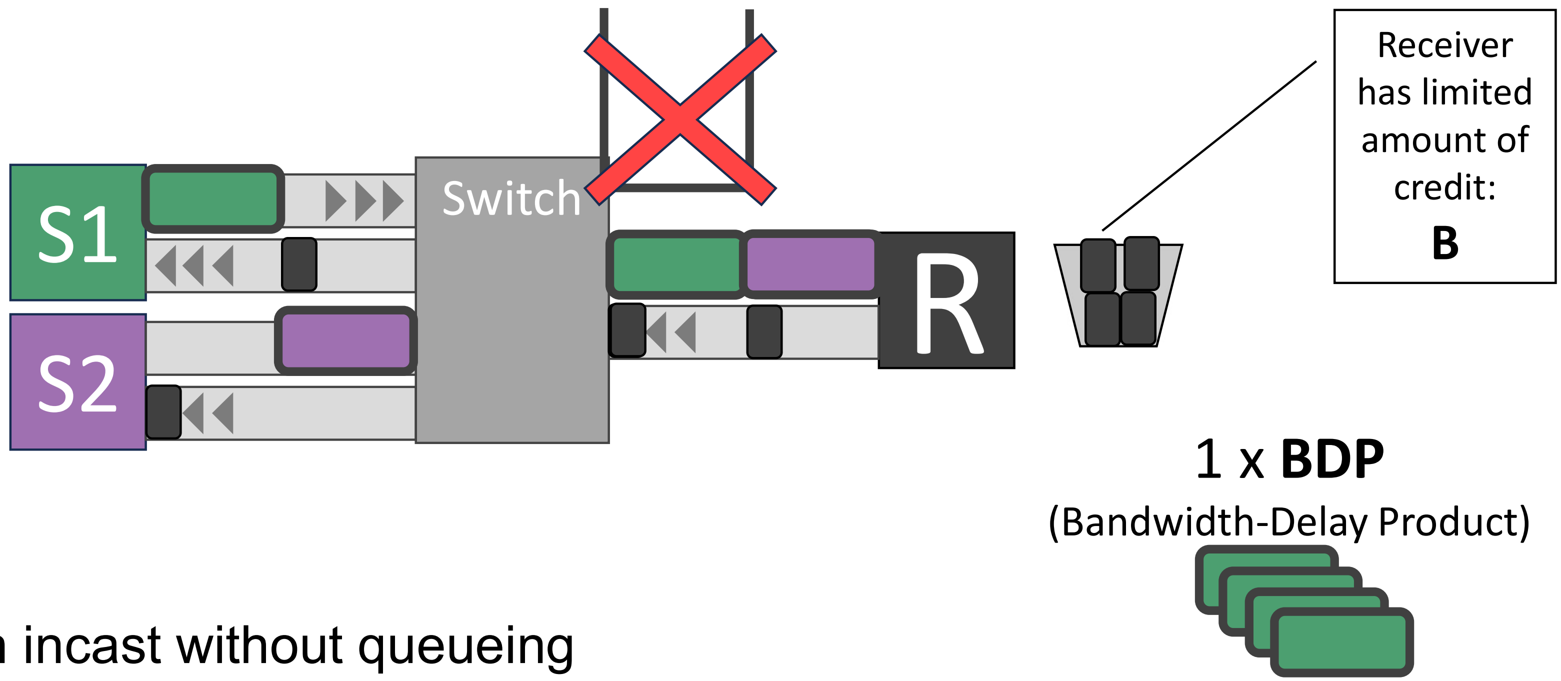
SIRD: Handling Receiver Congestion





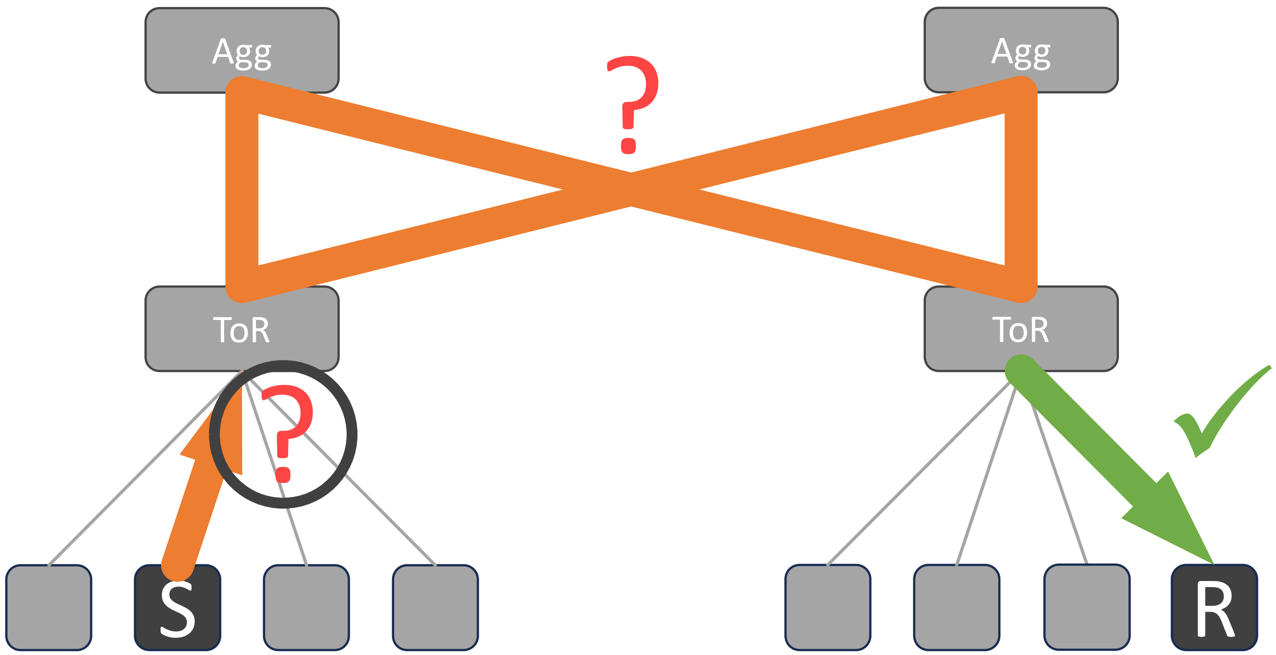
SIRD: Handling Receiver Congestion



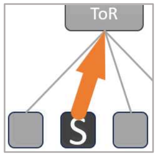


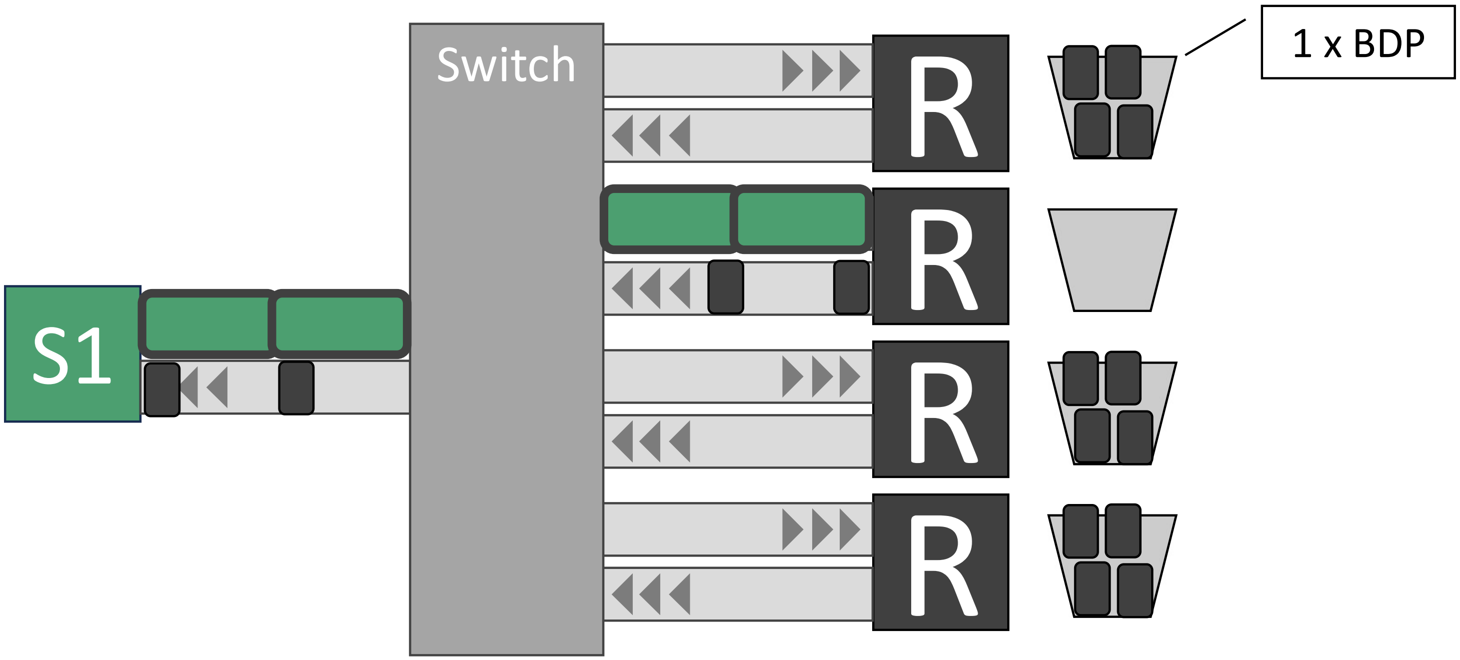
- Can deal with incast without queueing
- **B** caps the total number of packets in the network

Let’s build SIRD: Sender Congestion

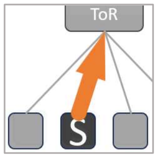


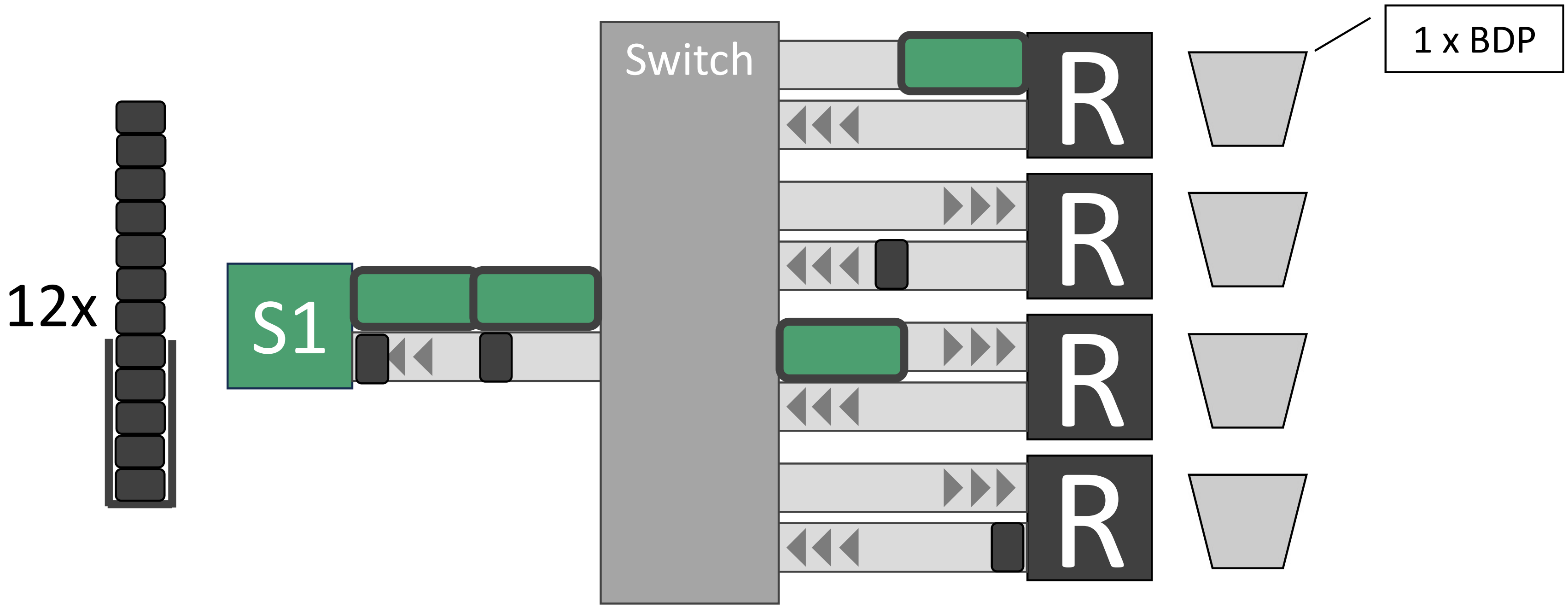
Challenge with Sender Congestion



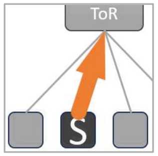


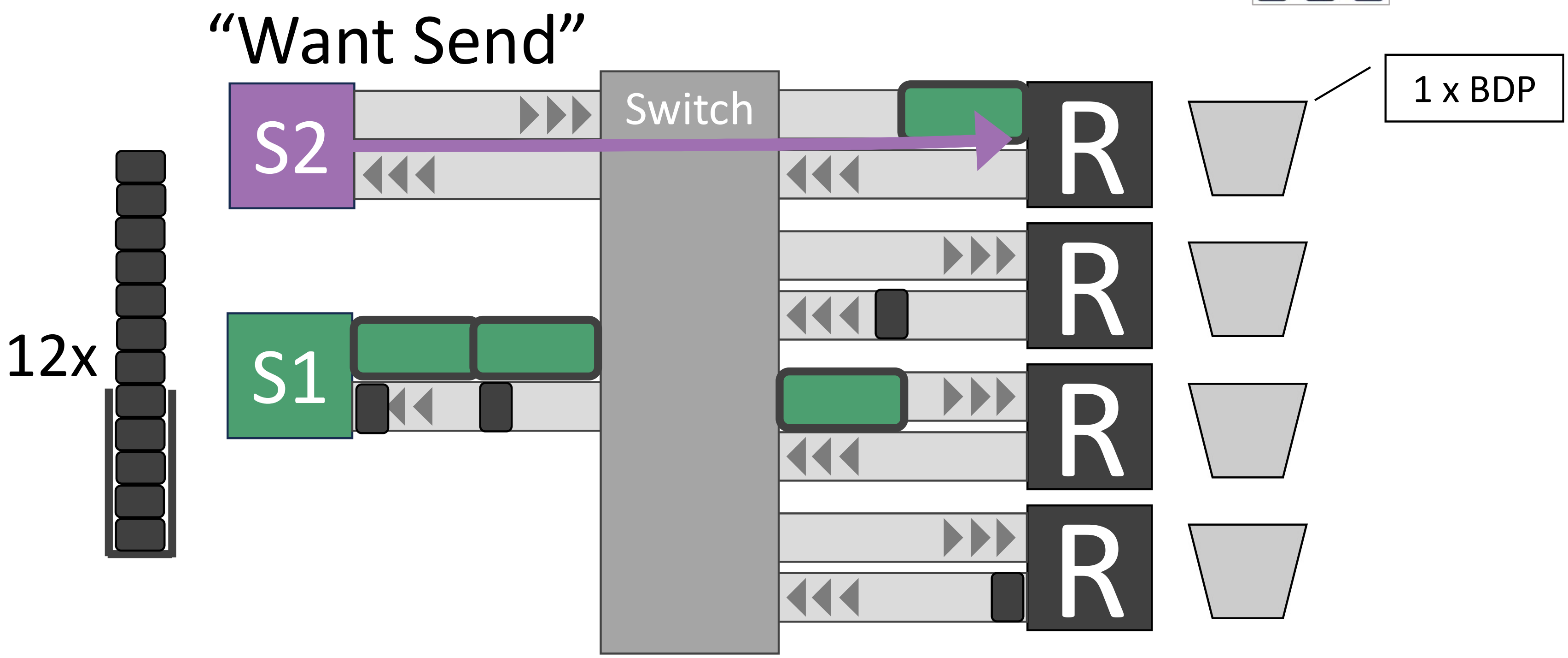
Challenge with Sender Congestion





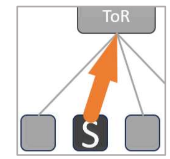
Challenge with Sender Congestion



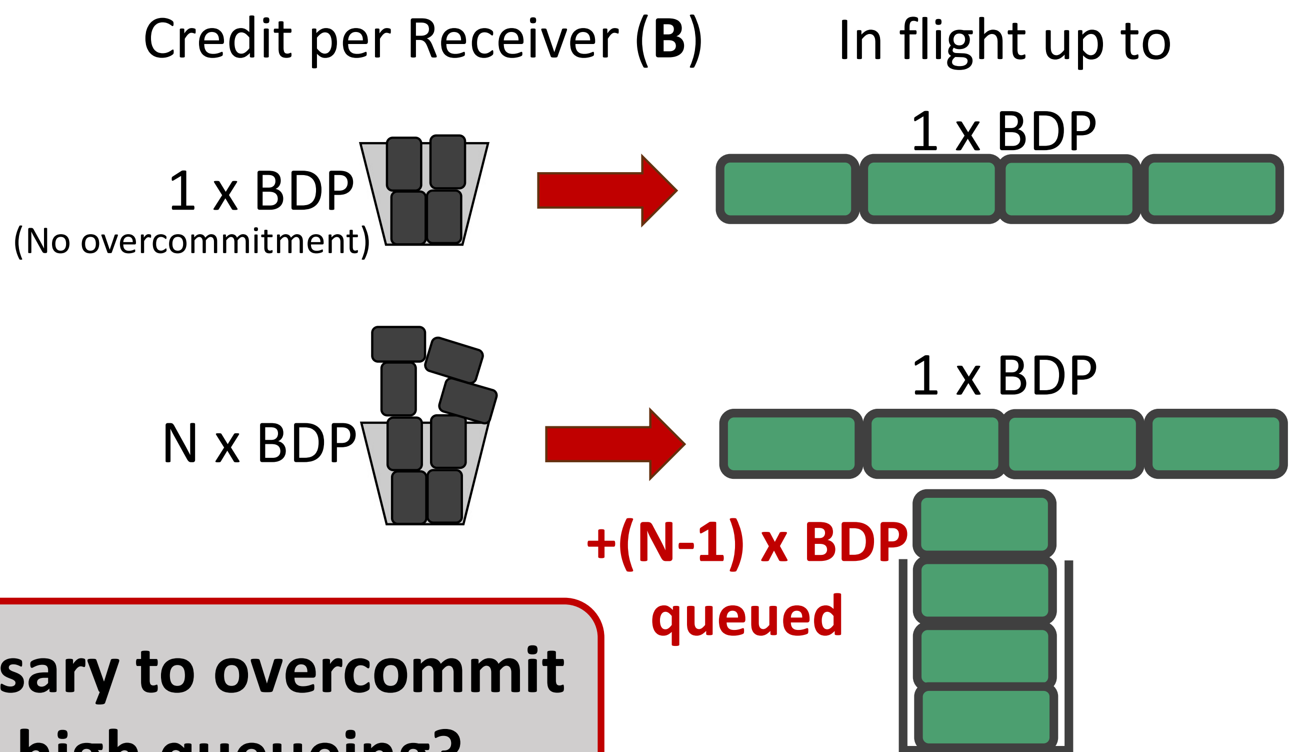


Problem: Receivers don't know how much credit to allocate to Congested Sender (S1) -> over-allocate = throughput loss

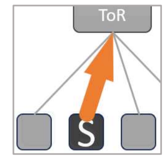
Approaches for handling congested senders



- Explicit Matching (dcPIM)
- Hop-by-hop credit management (ExpressPass)
- Controlled Overcommitment (**Homa**)
  - >> 1xBDP



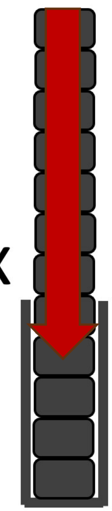
Is it necessary to overcommit & face high queueing?

Reduce Credit Accumulation 

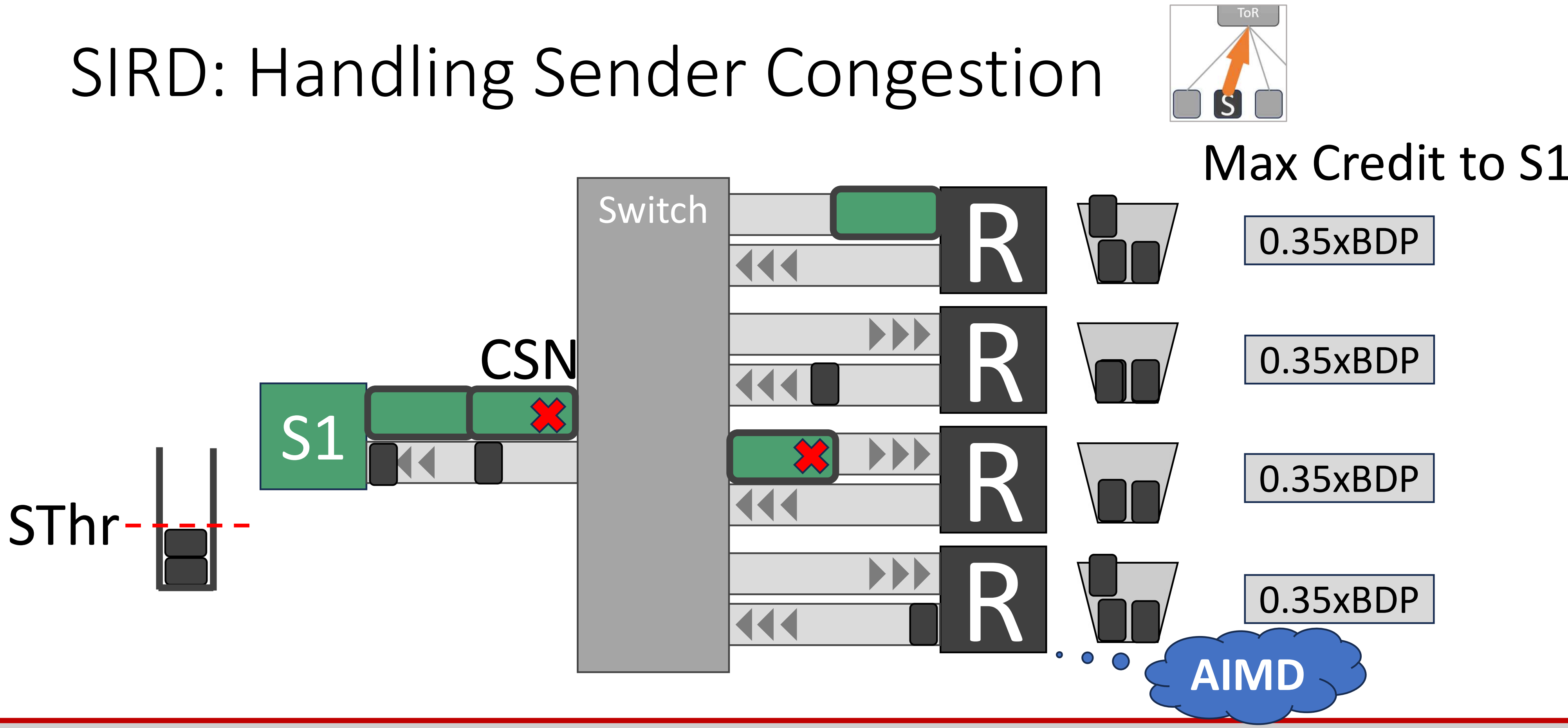
No need for overcommitment if credit is not stuck at senders

- SIRD reactively limits credit accumulation
  - If accum_credit > threshold, sender “informs” receivers
  - Receivers reduce allocation

12x



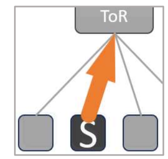
SIRD: Handling Sender Congestion

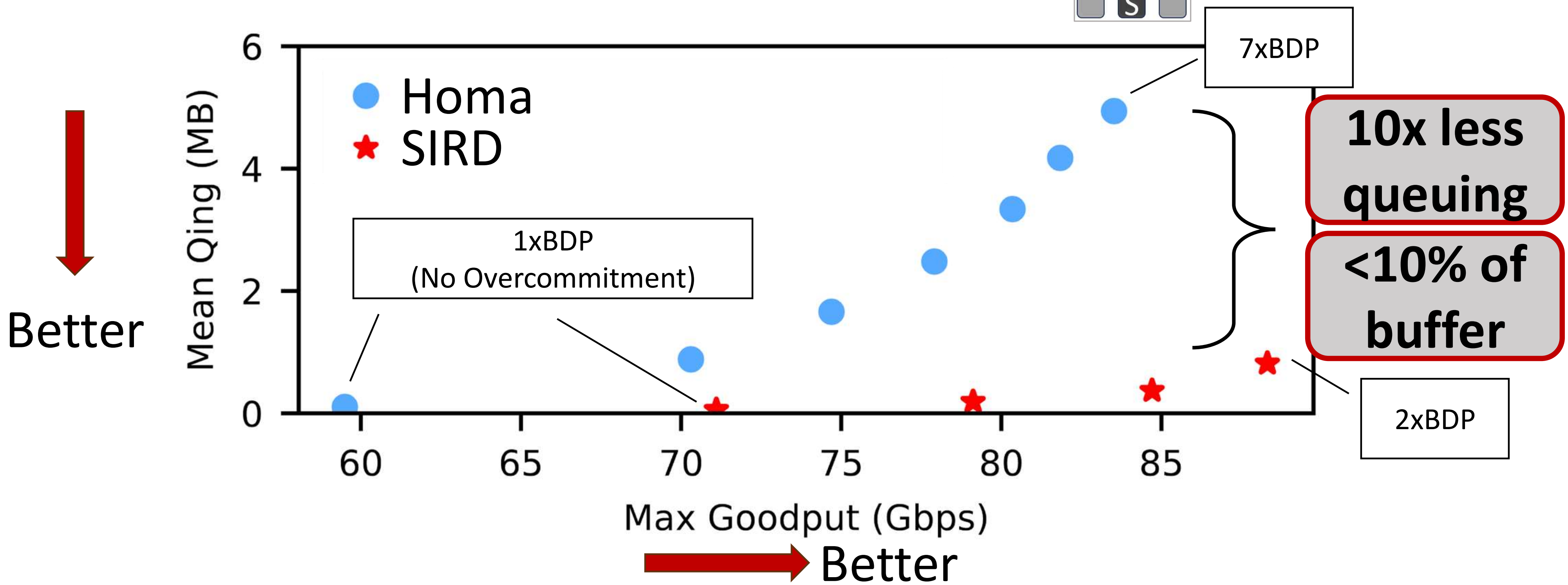


Reactively adjust to bottleneck availability

➔ Can get more throughput with less credit

Benefits of Sender-Informed RD

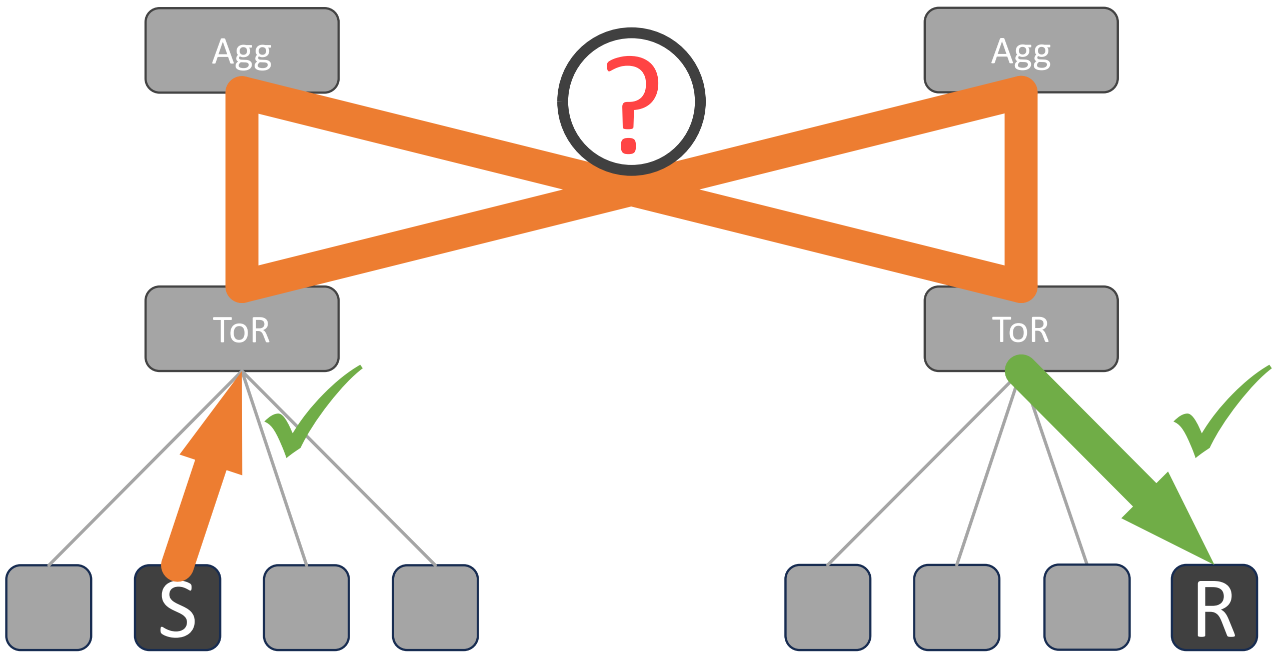




Simulated 100Gbps 144-host cluster running the Websearch workload at maximum load (worst case).

Note: SIRD benefits from modest overcommitment
Some credit accumulates by design (SThr>0)

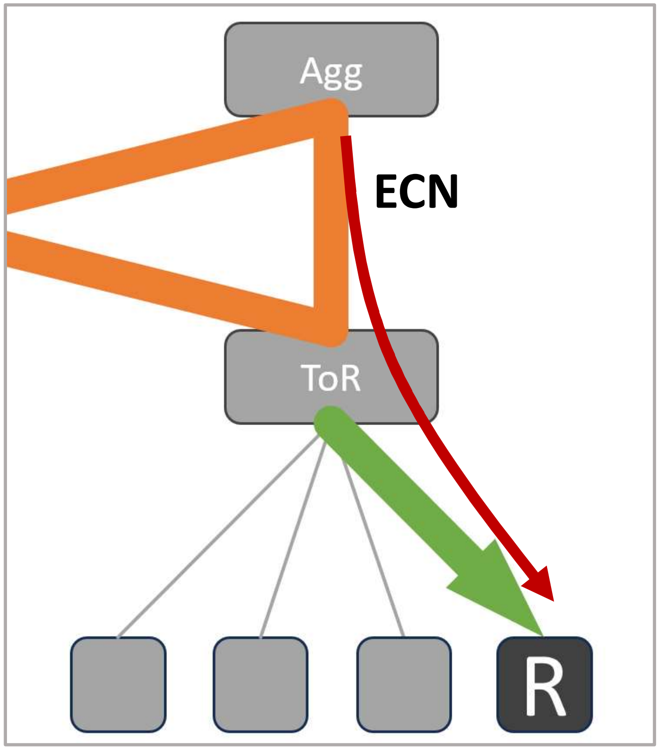
Let’s build SIRD: Core Congestion



SIRD: Handling Core Congestion

Core links are also shared links

- Extend control loop
  - Core congestion feedback: ECN
- Receiver adapts credit allocation f(sender and core feedback)
- $MaxCreditSender_i$ = min($sender_i$, core)



SIRD Design Recap

- Each receiver operates with modest amount of credit (B = 1.5 x BDP)
- Each sender gets up to 1 x BDP of that
- Adjust max credit per-sender f(CSN, ECN)
  - => High throughput with minimal credit
  - => minimal queuing

In-depth design questions:

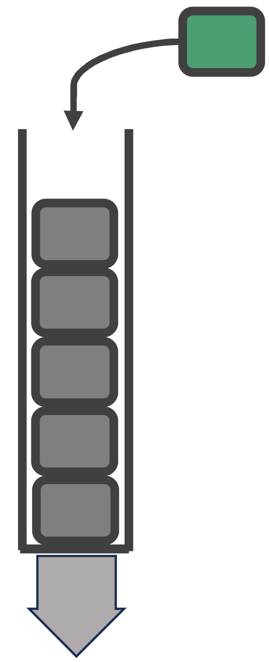
- How to reduce message latency
- How much to overcommit
- How to configure Sender threshold (SThr)
- How to start message transmission
- How to use switch priorities if available

Reducing Message Latency

1. Small messages (<BDP) sent outright (configurable)

2. Small message latency caused by network queuing

- SIRD causes minimal queuing
- => Less need for switch priorities to bypass queues
- => Reduces operational complexity
- SIRD can use priorities if available (optional)



Evaluation

Simulation & 100Gbps testbed

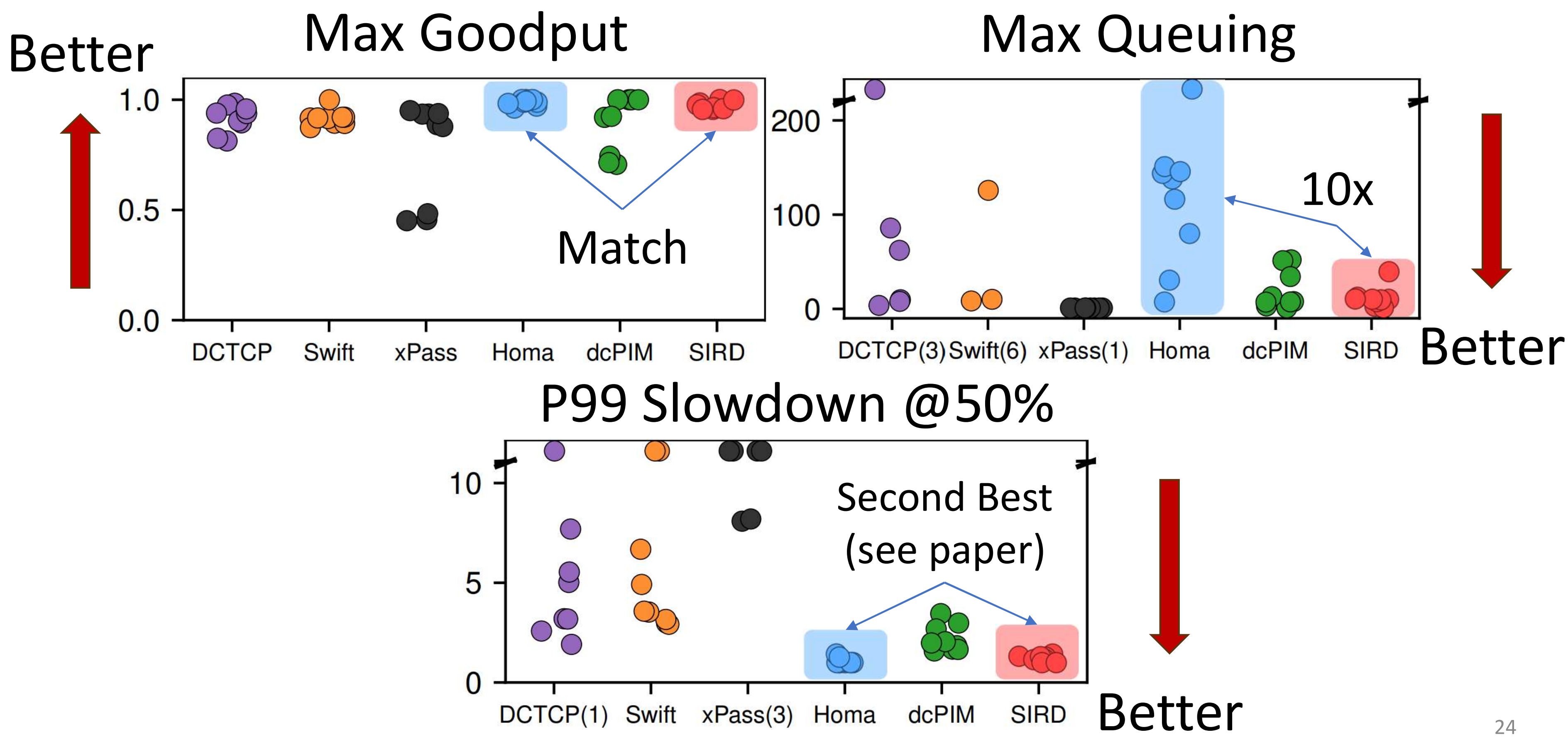
Questions answered

- How does SIRD compare overall to baselines?
- What is the contribution of sender feedback to performance?
- What is the throughput-queuing curve as a function of load?
- How does SIRD compare in terms of message latency?
- How sensitive is SIRD to a) priorities b) unscheduled transmissions?
- How well does sender feedback work in a real stack?

Simulation Evaluation Setup

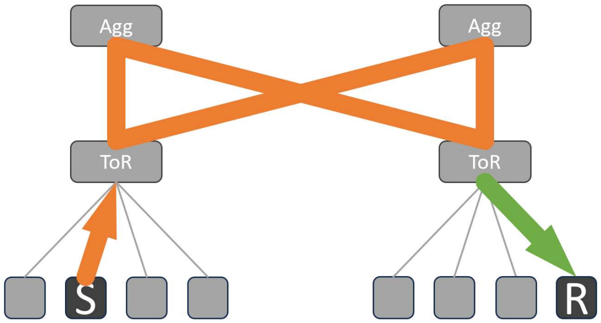
- 144 x 100Gbps hosts over 9 ToRs and 4 Spines.
- 3 Workloads
  (Google Search, Hadoop, WebSearch)
- x3 Traffic matrices
  (leaf-bottleneck, core-bottleneck, incast)
- 2 switch priority lanes for SIRD

Normalized Comparison



Summary

- **Single-owner links**: Manage **Proactively**
- **Shared links**: Manage **Reactively** (coordinate receivers)
  - => Handles congestion everywhere
- Enables: high throughput with little credit
  - => **little** in-network **queueing**
  - => **low** network **latency** for short messages
- SIRD implementation: github.com/epfl-dcsl/SIRD-Caladan-Impl
- SIRD simulator: github.com/epfl-dcsl/SIRD-Simulator



Thank you!