OSCAR: *O(1)*-Step Convergence And Readily-deployable Congestion Control

Zhaochen Zhang, Feiyang Xue, Rui Ning, Keqiang He, Gianni Antichi,
Jiaqi Gao, Zhimeng Yin, Kexin Liu, Rui Li, Zhengqi Cui, Zhehao Lin,
Peirui Cao, Guihai Chen, Chen Tian





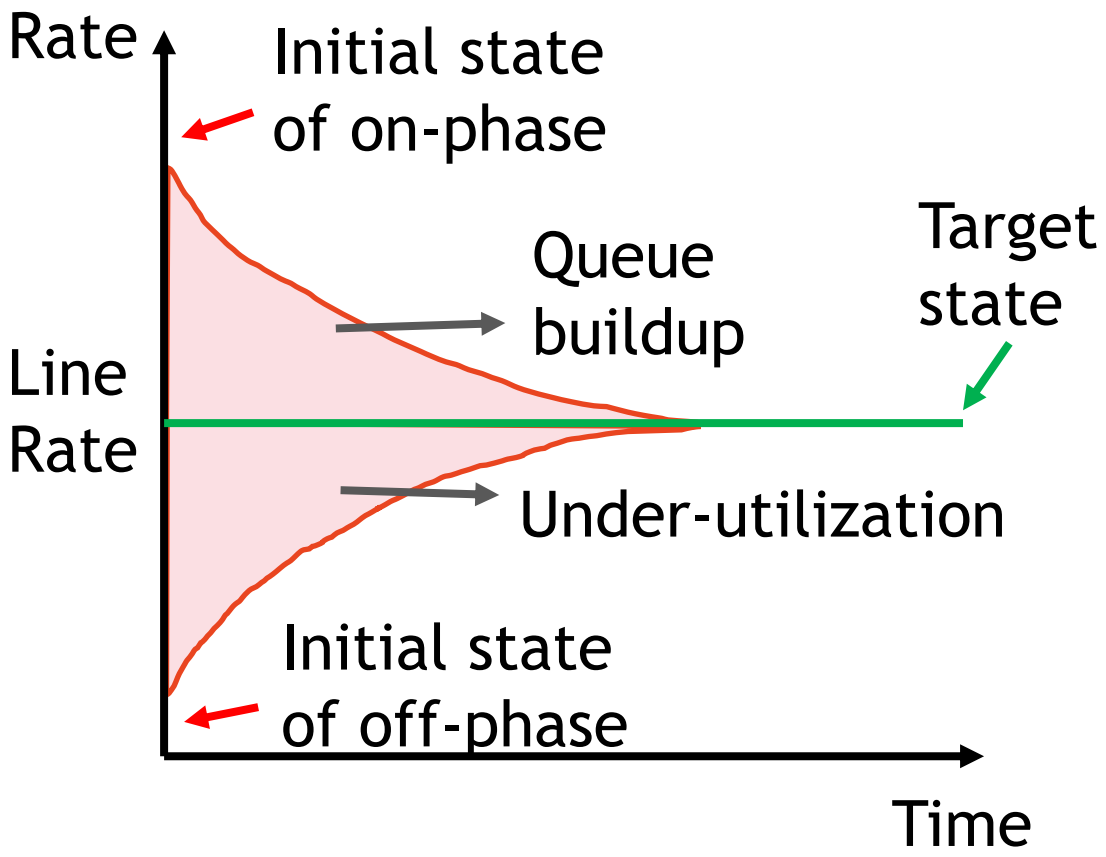
CC's Convergence Speed Matters

Trend1:
Workloads like AI training and inference exhibit frequent on-off traffic patterns.

CC's Convergence Speed Matters

Trend1:
Workloads like AI training and inference exhibit frequent on-off traffic patterns.

Frequent convergence processes.



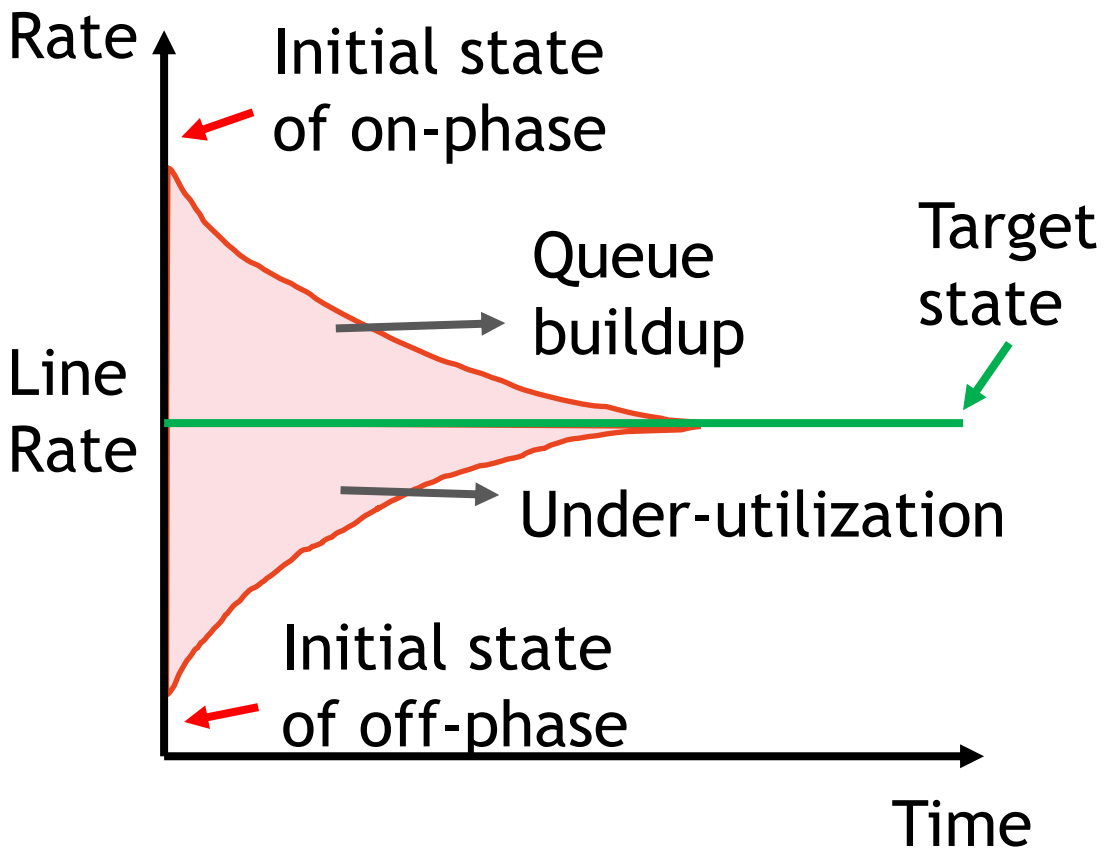
CC's Convergence Speed Matters

Trend1:
Workloads like AI training and inference exhibit frequent on-off traffic patterns.

Frequent convergence processes.

Trend2:
Bandwidth keeps climbing — from 100 Gbps to 800 Gbps.

Larger state gap to converge



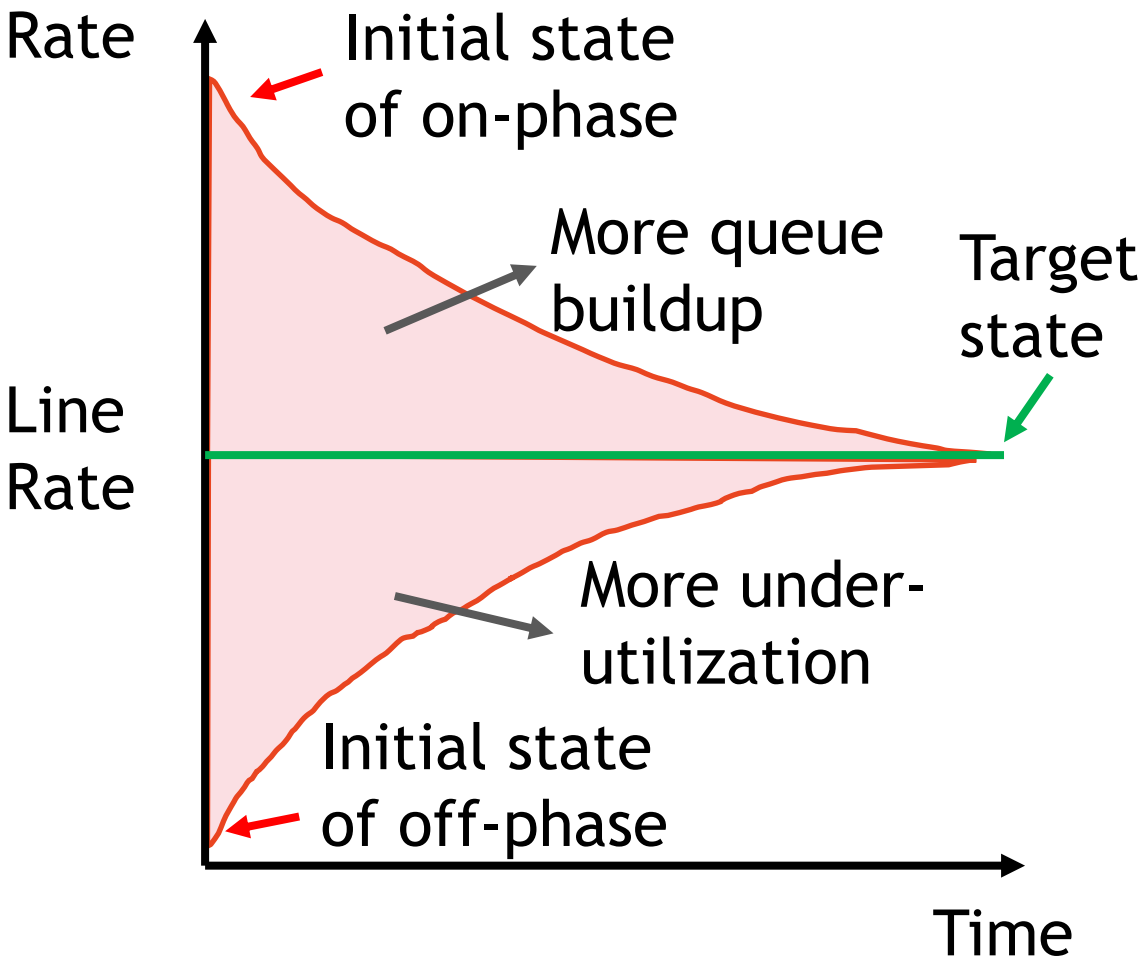
CC's Convergence Speed Matters

Trend1:
Workloads like AI training and inference exhibit frequent on-off traffic patterns.

Frequent convergence processes.

Trend2:
Bandwidth keeps climbing — from 100 Gbps to 800 Gbps.

Larger state gap to converge

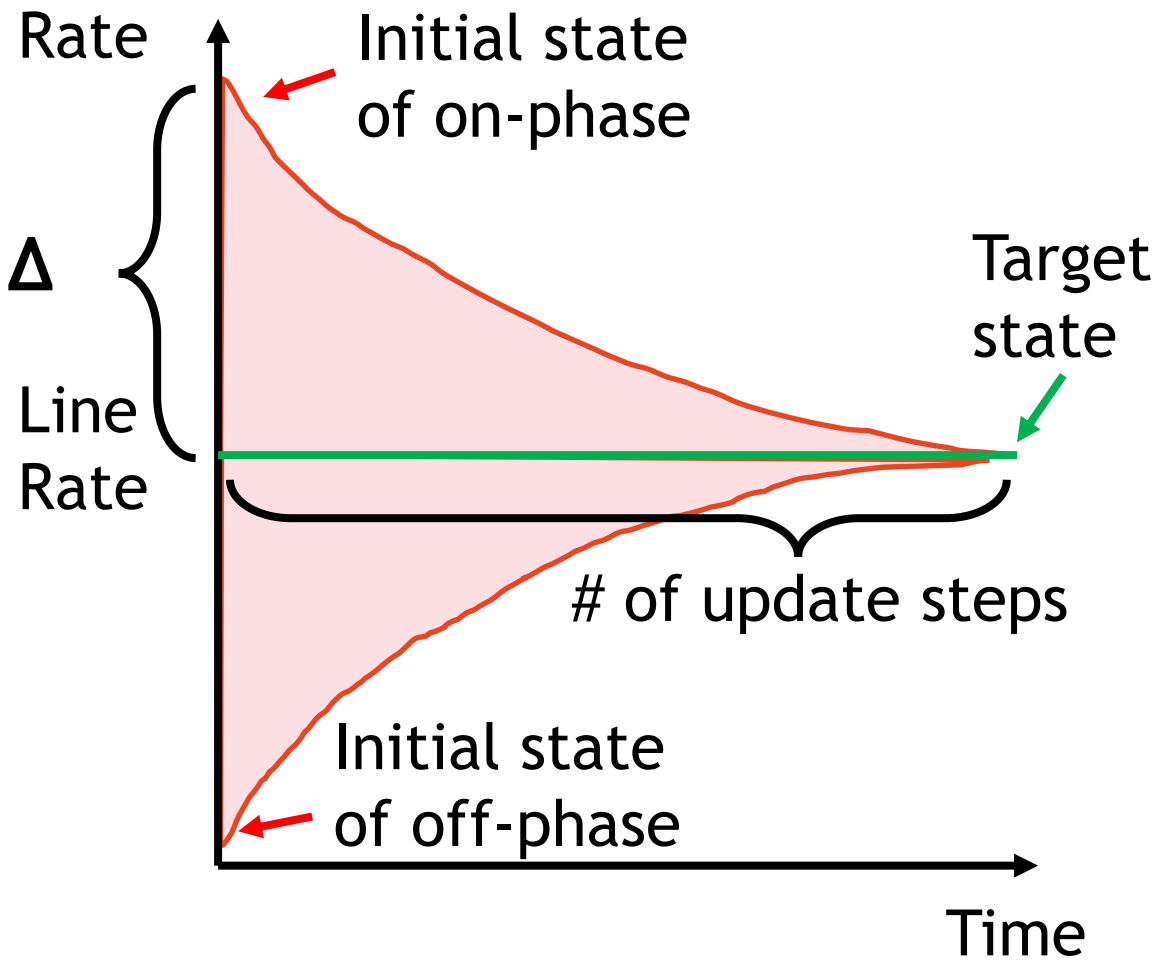


CC's Convergence Speed Matters

We borrow big-O notation to describe CC's convergence speed

Common Algorithms	Congestion Control
Input size n	State gap Δ
Time cost	# of update steps
O(n),O(log n),O(1)	O(Δ),O(log Δ),O(1)

O(1) = converges in constant # of steps, regardless of how big the gap is.

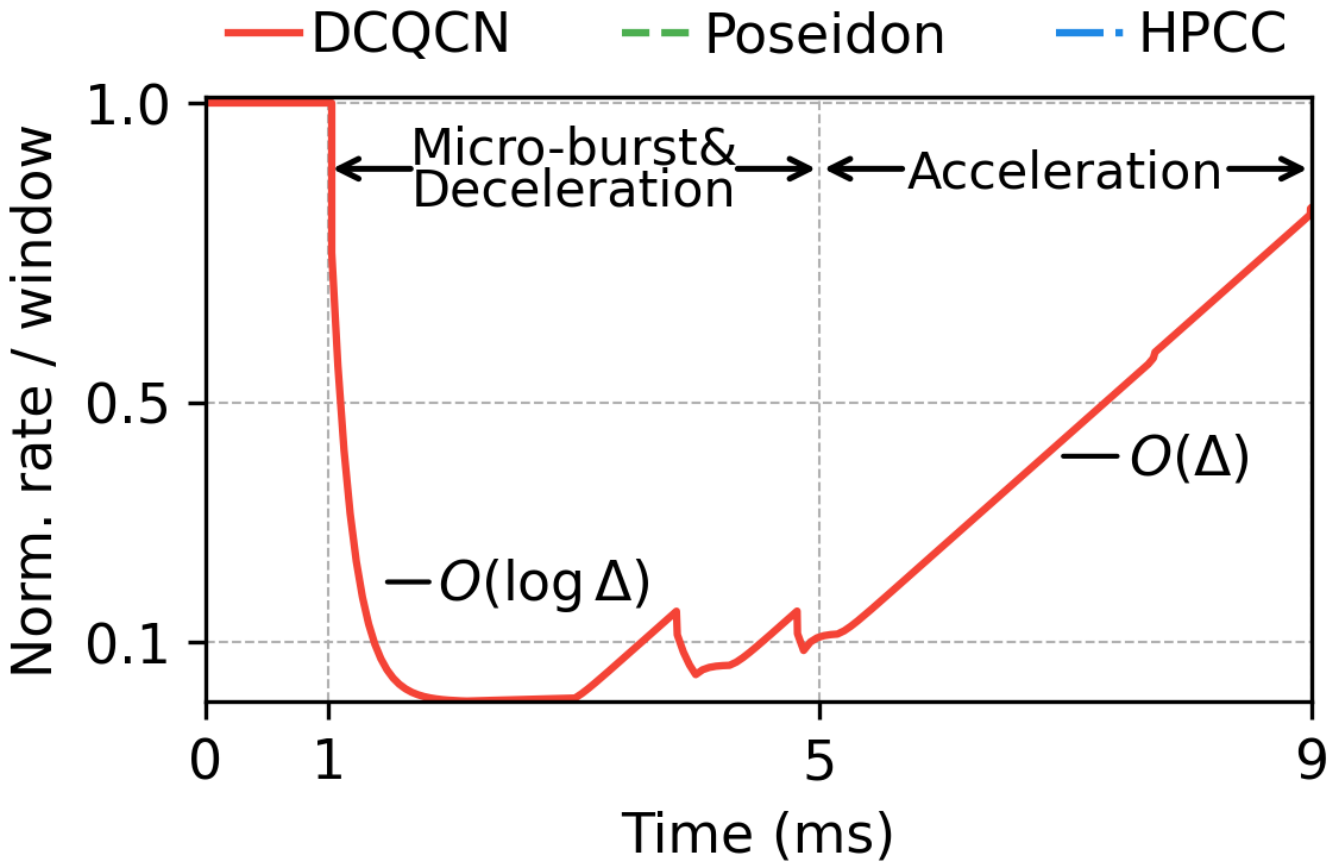


CCs Have Different Convergence Speeds

A micro-benchmark shows convergence speeds ➡

Setup: 1 long flow + 9 short flows form a microburst (1ms–5ms).

Y-axis: the long flow's rate/window, normalized to link capacity.

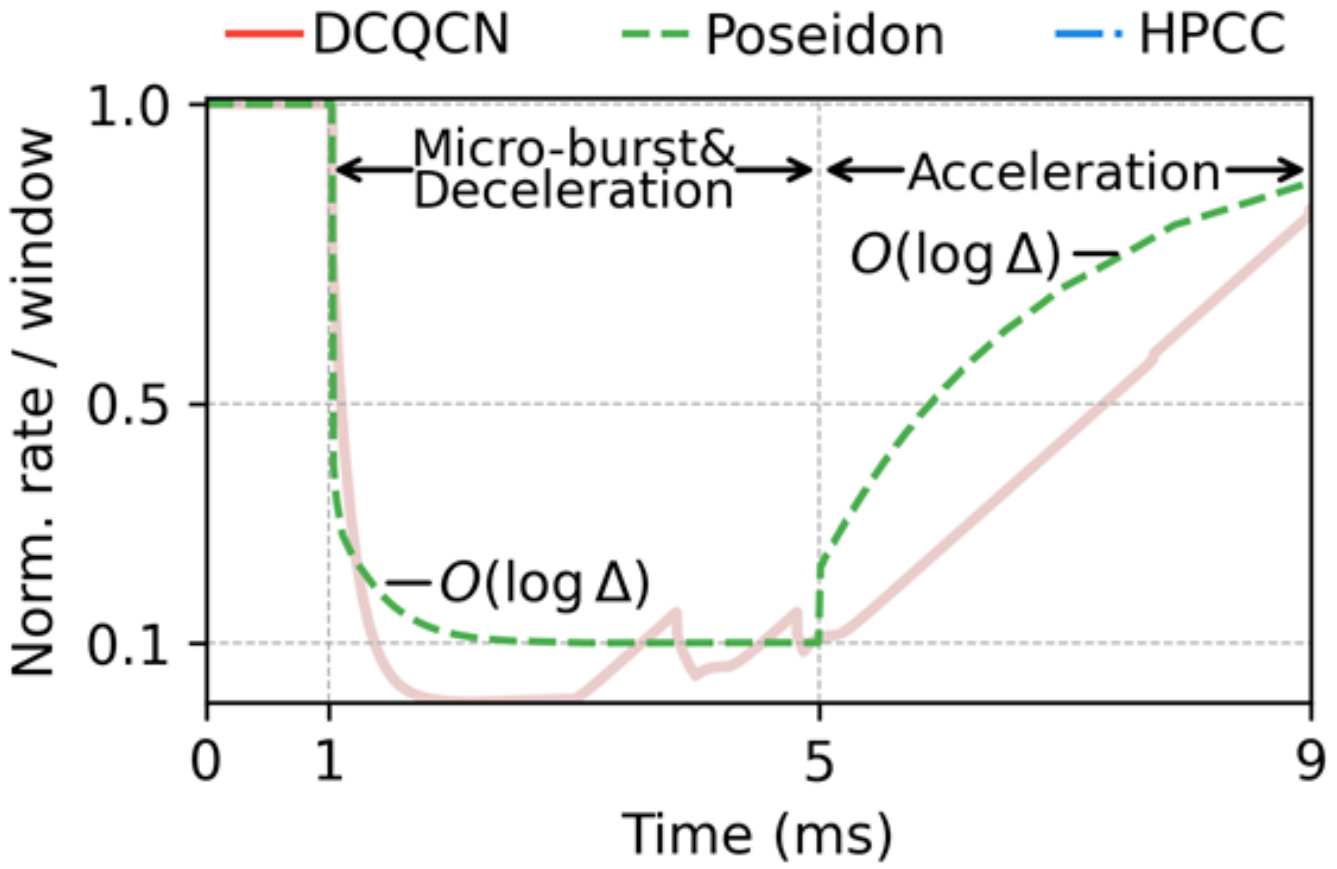


CCs Have Different Convergence Speeds

A micro-benchmark shows convergence speeds ➡

Setup: 1 long flow + 9 short flows form a microburst (1ms–5ms).

Y-axis: the long flow's rate/window, normalized to link capacity.

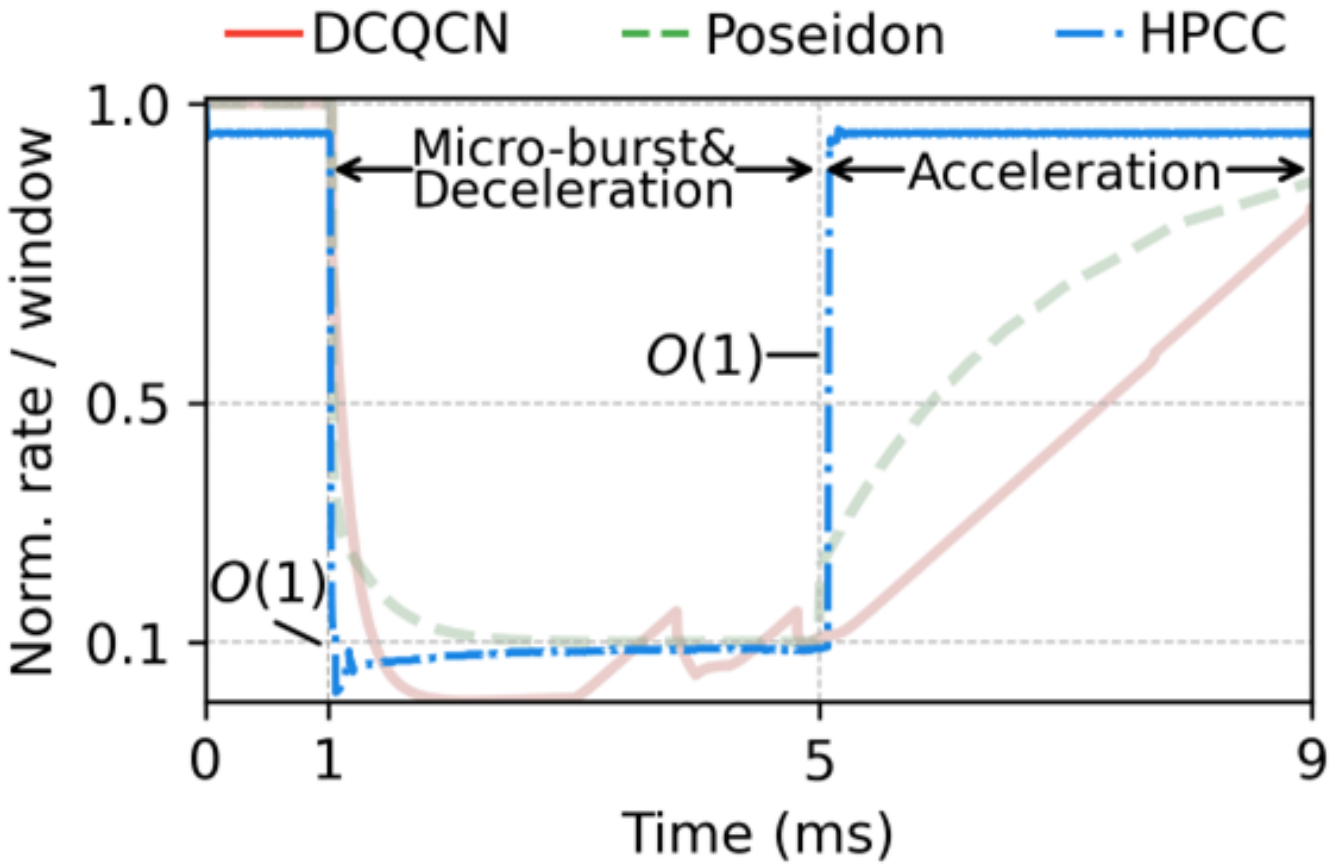


CCs Have Different Convergence Speeds

A micro-benchmark shows convergence speeds ➡

Setup: 1 long flow + 9 short flows form a microburst (1ms–5ms).

Y-axis: the long flow's rate/window, normalized to link capacity.



CCs Have Different Convergence Speeds

Category		Representative CCs	Acceleration	Deceleration
Sender-driven	AIMD-based	DCQCN, TIMELY, Swift	O(Δ)	O(log Δ)
	Short-INT-based	Poseidon	O(log Δ)	O(log Δ)
	Precise-INT-based	HPCC, PowerTCP	O(1)	O(1)
Receiver-driven		NDP, Homa	O(1)	O(1)
Switch-driven		XCP, Bolt, Harmony	O(1)	O(1)

CCs Have Different Convergence Speeds

Category		Representative CCs	Acceleration	Deceleration
Sender-driven	AIMD-based	DCQCN, TIMELY, Swift	O(Δ)	O(log Δ)
	Short-INT-based	Poseidon	O(log Δ)	O(log Δ)
	Precise-INT-based	HPCC, PowerTCP	O(1)	O(1)
Receiver-driven		NDP, Homa	O(1)	O(1)
Switch-driven		XCP, Bolt, Harmony	O(1)	O(1)

Three types of CCs can achieve O(1)-step convergence!

Current O(1)-step CCs Are Not Deployment-Friendly

Category		Representative CCs	Acceleration	Deceleration
Sender-driven	AIMD-based	DCQCN, TIMELY, Swift	O(Δ)	O(log Δ)
	Short-INT-based	Poseidon	O(log Δ)	O(log Δ)
	Precise-INT-based	HPCC, PowerTCP	O(1)	O(1)
Receiver-driven		NDP, Homa	O(1)	O(1)
Switch-driven		XCP, Bolt, Harmony	O(1)	O(1)

Lengthy INT headers are hard to deploy on high-speed hardware[1]!

[1] (NSDI23) Poseidon: Efficient, robust, and practical datacenter CC via deployable INT.

Current O(1)-step CCs Are Not Deployment-Friendly

Category		Representative CCs	Acceleration	Deceleration
Sender-driven	AIMD-based	DCQCN, TIMELY, Swift	O(Δ)	O(log Δ)
	Short-INT-based	Poseidon	O(log Δ)	O(log Δ)
	Precise-INT-based	HPCC, PowerTCP	O(1)	O(1)
Receiver-driven		NDP, Homa	O(1)	O(1)
Switch-driven		XCP, Bolt, Harmony	O(1)	O(1)

Packet spraying is poorly supported by legacy RDMA NICs[1].
Topology assumptions do not always hold[2].

[1] (SIGCOMM25) Falcon [2](SIGCOMM24) Alibaba HPN

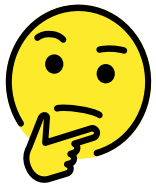
Current O(1)-step CCs Are Not Deployment-Friendly

Category		Representative CCs	Acceleration	Deceleration
Sender-driven	AIMD-based	DCQCN, TIMELY, Swift	O(Δ)	O(log Δ)
	Short-INT-based	Poseidon	O(log Δ)	O(log Δ)
	Precise-INT-based	HPCC, PowerTCP	O(1)	O(1)
Receiver-driven		NDP, Homa	O(1)	O(1)
Switch-driven		XCP, Bolt, Harmony	O(1)	O(1)

Commodity switches can not support such computational capabilities.

Current O(1)-step CCs Are Not Deployment-Friendly

Category	Representative CCs	Acceleration	Deceleration
Switch-driven	XCP, Bolt, Harmony	O(1)	O(1)



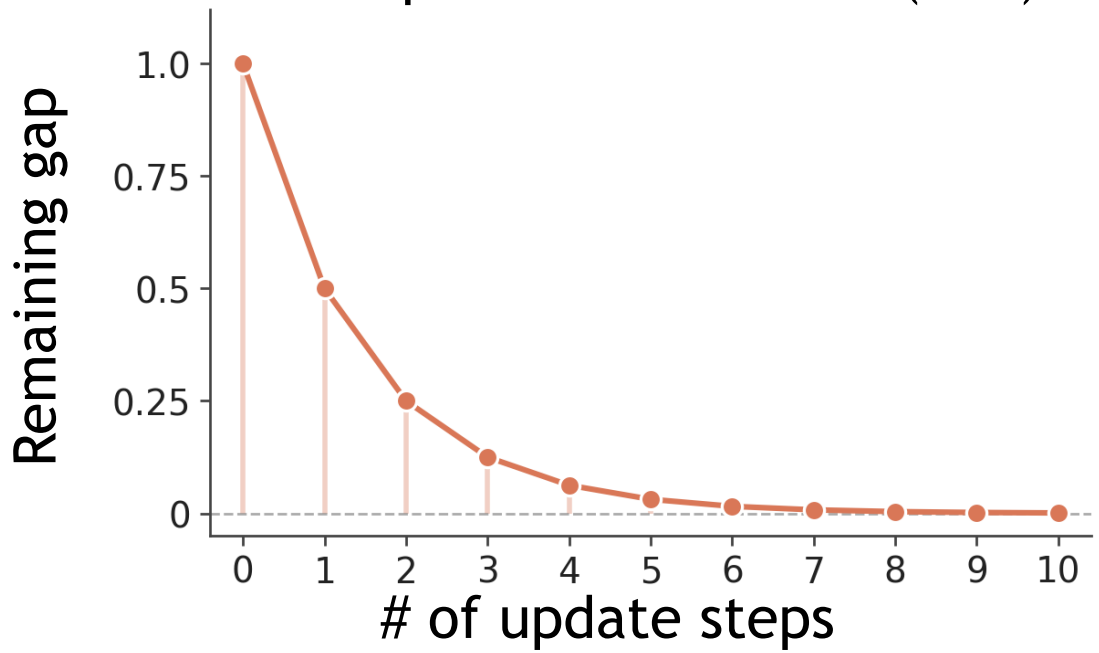
Can a readily deployable CC achieve O(1)-step convergence?

Commodity switches can not support such computational capabilities.

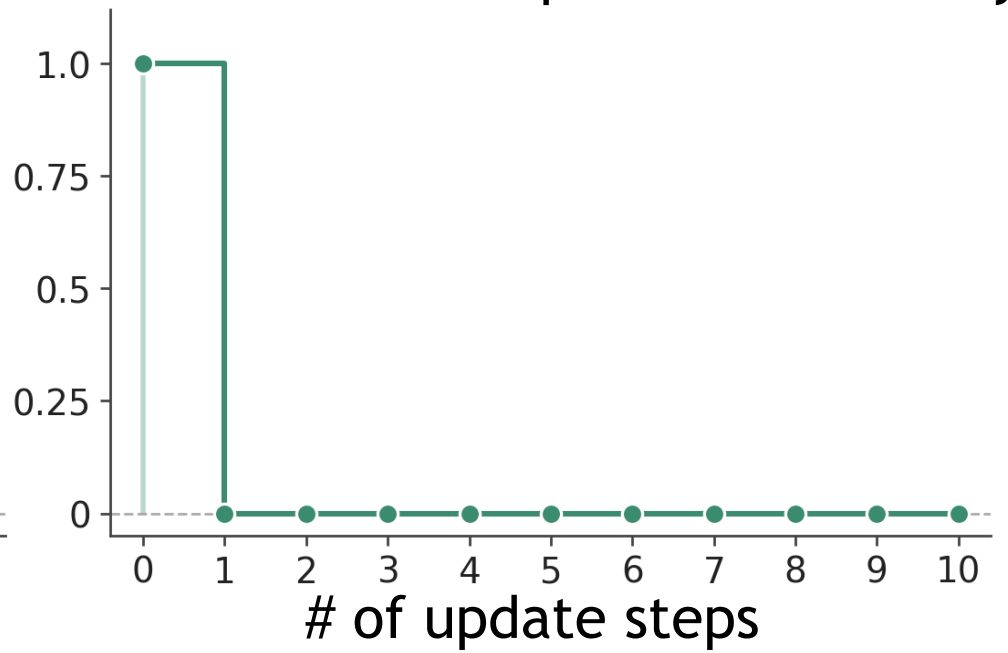
O(1)-step Boils Down to One-step Convergence

Convergence in O(1) steps needs at least one of these steps can eliminate the entire remaining gap (one-step convergence).

If each step closes a fraction (50%).



If at least one step closes Δ entirely.



Principle 1: MIMD over Precise Signals is Essential

AI

A single flow's update

$$r_f(t) = r_f(t-1) + \alpha$$

MI

A single flow's update

$$r_f(t) = r_f(t-1) \cdot \beta$$

Principle 1: MIMD over Precise Signals is Essential

AI

A single flow's update

$$r_f(t) = r_f(t-1) + \alpha$$

Agg. rate of n flows

$$\sum r_f(t) = \sum r_f(t-1) + n \cdot \alpha$$

MI

A single flow's update

$$r_f(t) = r_f(t-1) \cdot \beta$$

Principle 1: MIMD over Precise Signals is Essential

AI

A single flow's update

$$r_f(t) = r_f(t-1) + \alpha$$

Agg. rate of n flows

$$\sum r_f(t) = \sum r_f(t-1) + n \cdot \alpha$$

To eliminate the gap Δ

$$\alpha = \Delta / n$$

MI

A single flow's update

$$r_f(t) = r_f(t-1) \cdot \beta$$

Principle 1: MIMD over Precise Signals is Essential

AI

A single flow's update

$$r_f(t) = r_f(t-1) + \alpha$$

Agg. rate of n flows

$$\sum r_f(t) = \sum r_f(t-1) + n \cdot \alpha$$

To eliminate the gap Δ

$$\alpha = \Delta / n$$ ✗

No current congestion signal can measure n!

MI

A single flow's update

$$r_f(t) = r_f(t-1) \cdot \beta$$

Principle 1: MIMD over Precise Signals is Essential

AI

A single flow's update

$$r_f(t) = r_f(t-1) + \alpha$$

Agg. rate of n flows

$$\sum r_f(t) = \sum r_f(t-1) + n \cdot \alpha$$

To eliminate the gap Δ

$$\alpha = \Delta / n \times$$

No current congestion signal can measure n!

MI

A single flow's update

$$r_f(t) = r_f(t-1) \cdot \beta$$

Agg. rate of n flows

$$\sum r_f(t) = \sum r_f(t-1) \cdot \beta$$

Principle 1: MIMD over Precise Signals is Essential

AI

A single flow's update

$$r_f(t) = r_f(t-1) + \alpha$$

Agg. rate of n flows

$$\sum r_f(t) = \sum r_f(t-1) + n \cdot \alpha$$

To eliminate the gap Δ

$$\alpha = \Delta / n \times$$

No current congestion signal can measure n!

MI

A single flow's update

$$r_f(t) = r_f(t-1) \cdot \beta$$

Agg. rate of n flows

$$\sum r_f(t) = \sum r_f(t-1) \cdot \beta$$

To eliminate the gap Δ

$$\beta = T / \sum r_f(t-1)$$

Given $\Delta = T - \sum r_f(t-1)$

$$\boldsymbol{\beta = T / (T - \Delta)}$$

Principle 1: MIMD over Precise Signals is Essential

AI

A single flow's update

$$r_f(t) = r_f(t-1) + \alpha$$

Agg. rate of n flows

$$\sum r_f(t) = \sum r_f(t-1) + n \cdot \alpha$$

To eliminate the gap Δ

$$\alpha = \Delta / n \times$$

No current congestion signal can measure n!

MI

A single flow's update

$$r_f(t) = r_f(t-1) \cdot \beta$$

Agg. rate of n flows

$$\sum r_f(t) = \sum r_f(t-1) \cdot \beta$$

To eliminate the gap Δ

$$\beta = T / \sum r_f(t-1)$$

Given $\Delta = T - \sum r_f(t-1)$

$$\boldsymbol{\beta = T / (T - \Delta)} \checkmark$$

T is known previously
Δ can be measured from signals

Principle 2: Delay & Delay Gradient as Precise Signals

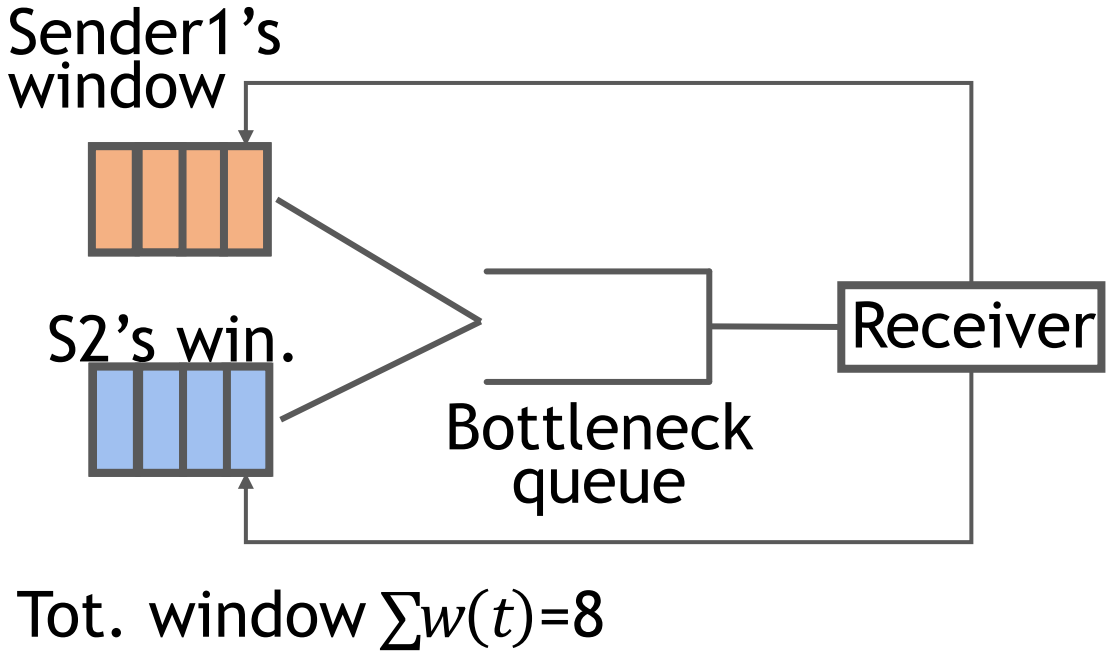
Analytical Formulation

$$d(t) = RTT_{base} + d_q(t) = RTT_{base} + \frac{q(t)}{\mu}$$

$$= RTT_{base} + \frac{\sum i(t) - BDP_{base}}{\mu} = \frac{\sum i(t)}{\mu}$$

$$\leq \frac{\sum w(t)}{\mu}$$

Visual Illustration



Principle 2: Delay & Delay Gradient as Precise Signals

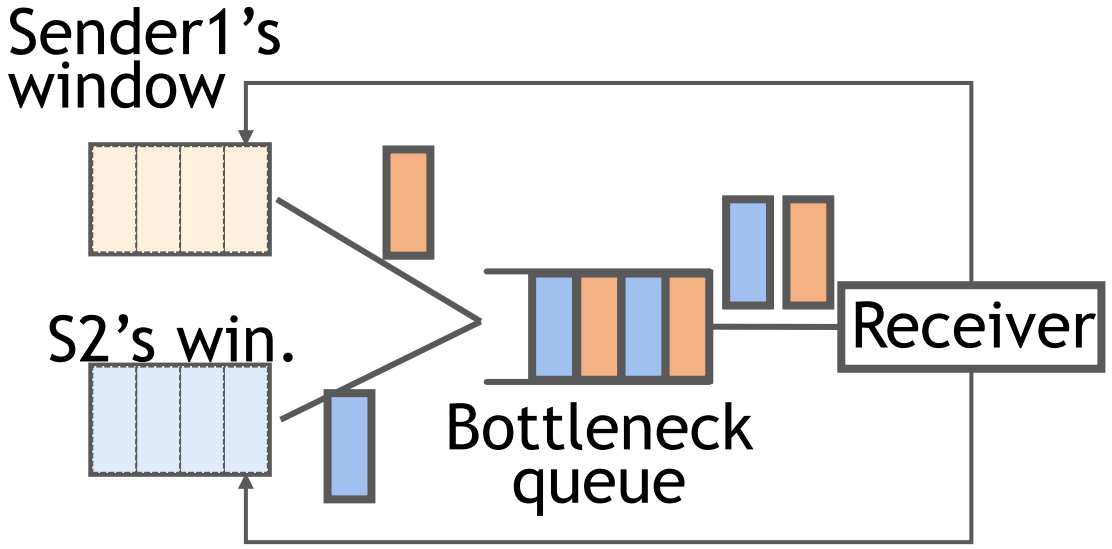
Analytical Formulation

$$d(t) = RTT_{base} + d_q(t) = RTT_{base} + \frac{q(t)}{\mu}$$

$$= RTT_{base} + \frac{\sum i(t) - BDP_{base}}{\mu} = \frac{\sum i(t)}{\mu}$$

$$\leq \frac{\sum w(t)}{\mu}$$

Visual Illustration



Principle 2: Delay & Delay Gradient as Precise Signals

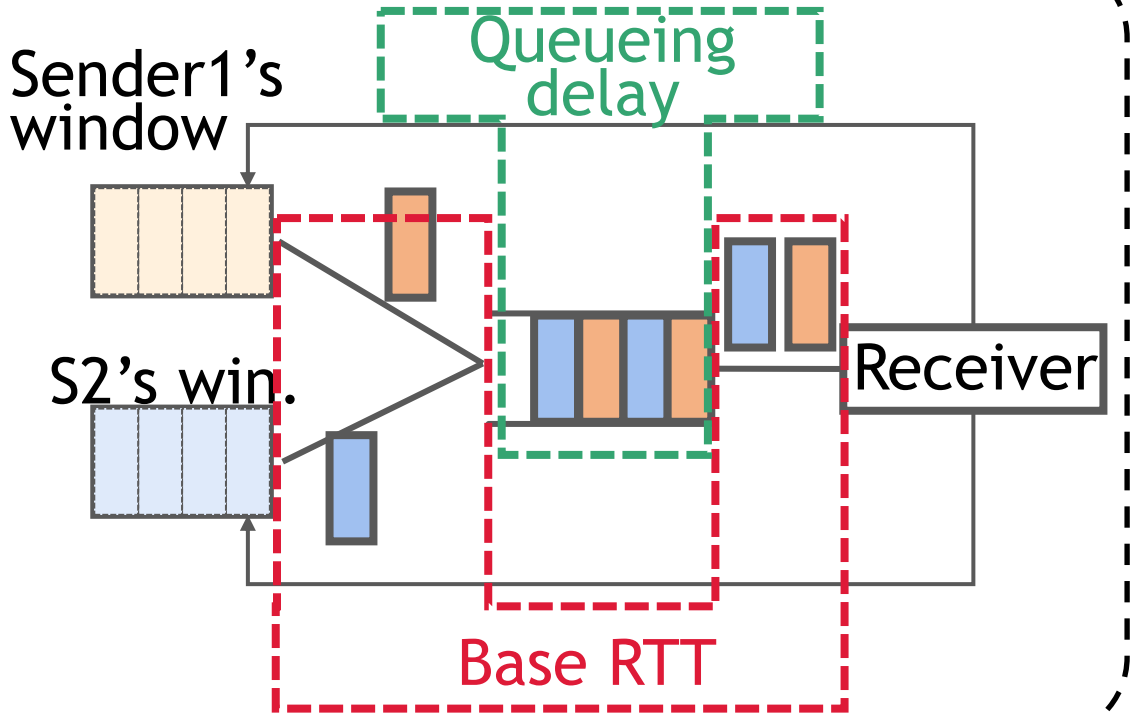
Analytical Formulation

$$d(t) = RTT_{base} + d_q(t) = RTT_{base} + \frac{q(t)}{\mu}$$

$$= RTT_{base} + \frac{\sum i(t) - BDP_{base}}{\mu} = \frac{\sum i(t)}{\mu}$$

$$\leq \frac{\sum w(t)}{\mu}$$

Visual Illustration



Principle 2: Delay & Delay Gradient as Precise Signals

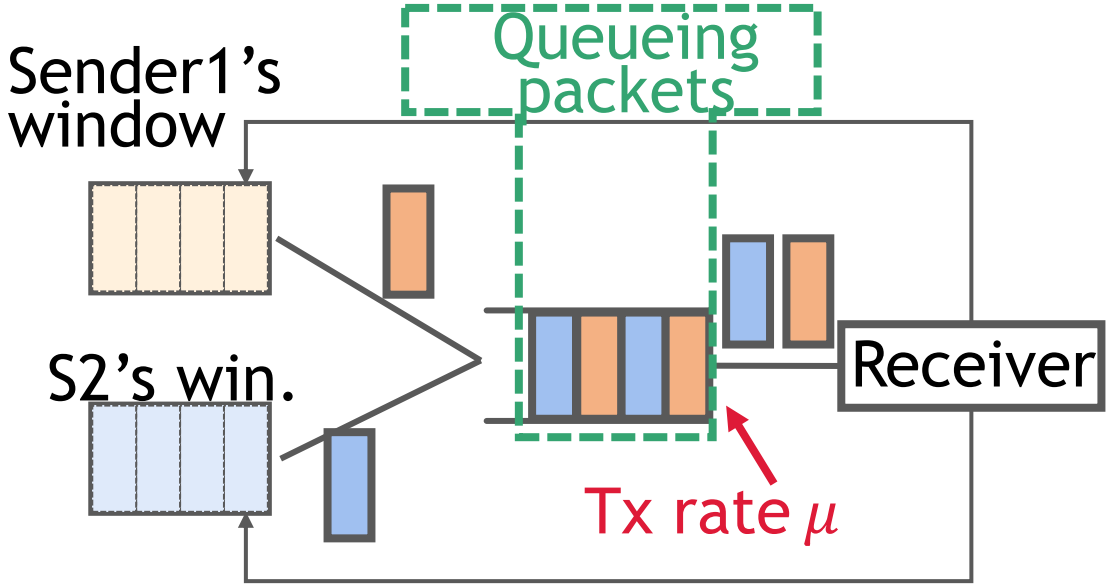
Analytical Formulation

$$d(t) = RTT_{base} + d_q(t) = RTT_{base} + \frac{q(t)}{\mu}$$

$$= RTT_{base} + \frac{\sum i(t) - BDP_{base}}{\mu} = \frac{\sum i(t)}{\mu}$$

$$\leq \frac{\sum w(t)}{\mu}$$

Visual Illustration



Principle 2: Delay & Delay Gradient as Precise Signals

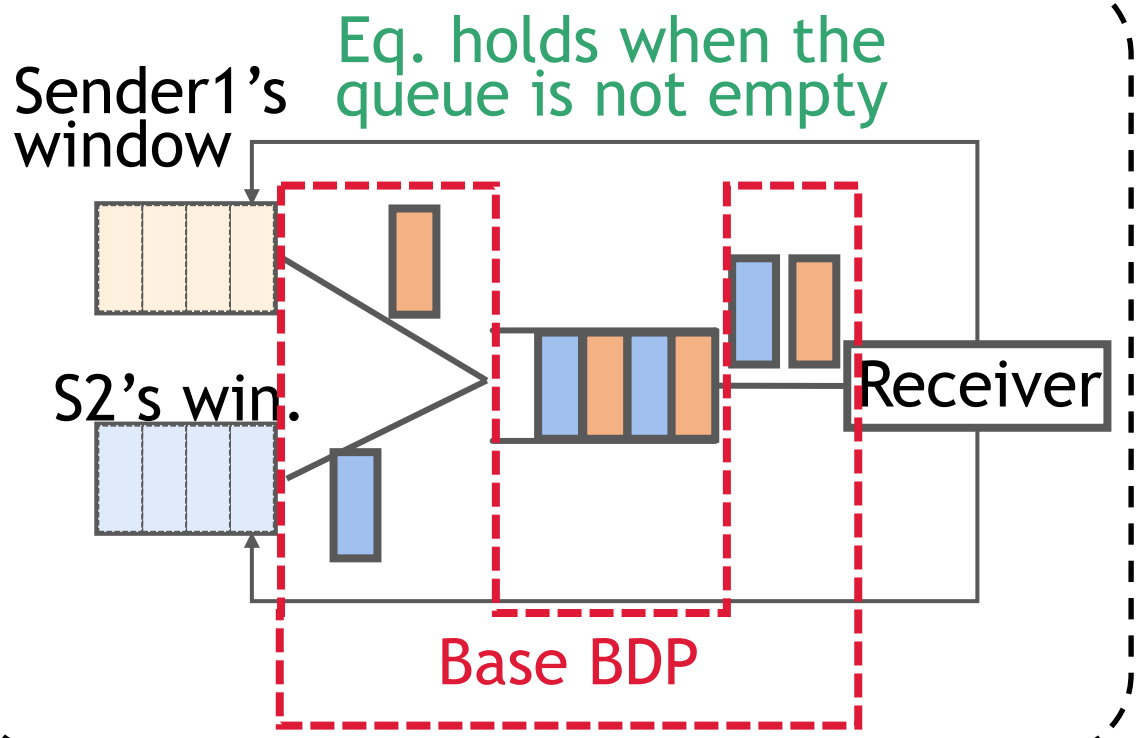
Analytical Formulation

$$d(t) = RTT_{base} + d_q(t) = RTT_{base} + \frac{q(t)}{\mu}$$

$$= RTT_{base} + \frac{\sum i(t) - BDP_{base}}{\mu} = \frac{\sum i(t)}{\mu}$$

$$\leq \frac{\sum w(t)}{\mu}$$

Visual Illustration



Principle 2: Delay & Delay Gradient as Precise Signals

Analytical Formulation

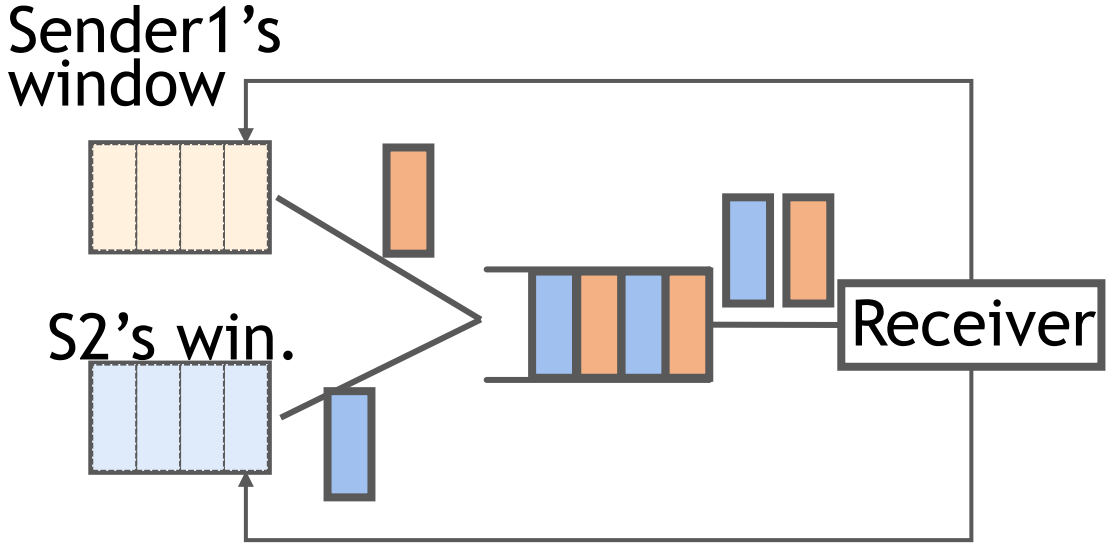
$$d(t) = RTT_{base} + d_q(t) = RTT_{base} + \frac{q(t)}{\mu}$$

$$= RTT_{base} + \frac{\sum i(t) - BDP_{base}}{\mu} = \frac{\sum i(t)}{\mu}$$

$$\leq \frac{\sum w(t)}{\mu}$$

Equality holds when window-bounded

Visual Illustration



Tot. window $\sum w(t)$=8, inflight $\sum i(t)$=8

Principle 2: Delay & Delay Gradient as Precise Signals

Analytical Formulation

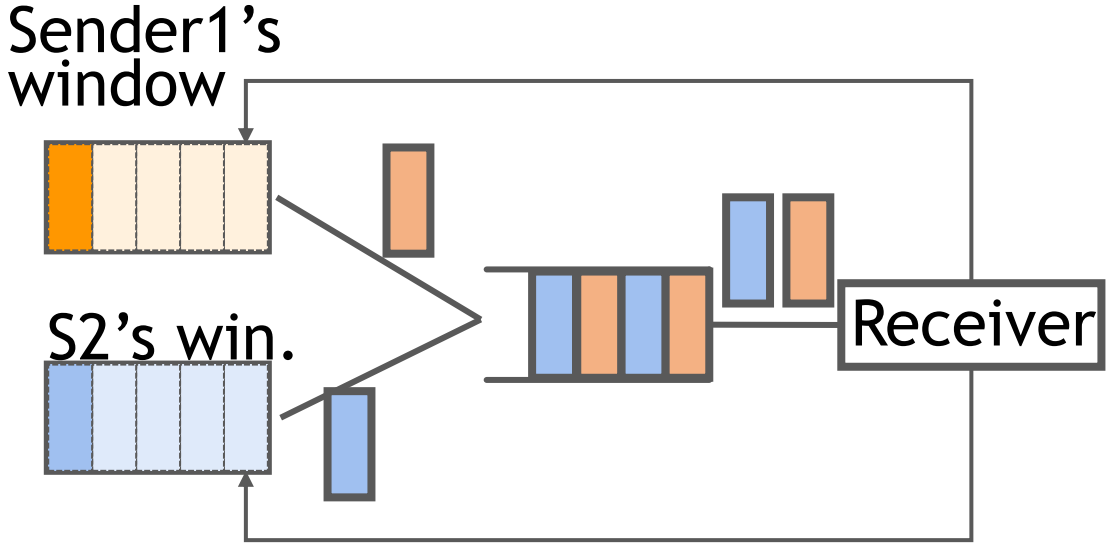
$$d(t) = RTT_{base} + d_q(t) = RTT_{base} + \frac{q(t)}{\mu}$$

$$= RTT_{base} + \frac{\sum i(t) - BDP_{base}}{\mu} = \frac{\sum i(t)}{\mu}$$

$$\leq \frac{\sum w(t)}{\mu}$$

Equality holds when window-bounded

Visual Illustration



Tot. window $\sum w(t)$=10, inflight $\sum i(t)$=8

Principle 2: Delay & Delay Gradient as Precise Signals

Precise congestion state can be extracted from delay!

Analytical Formulation

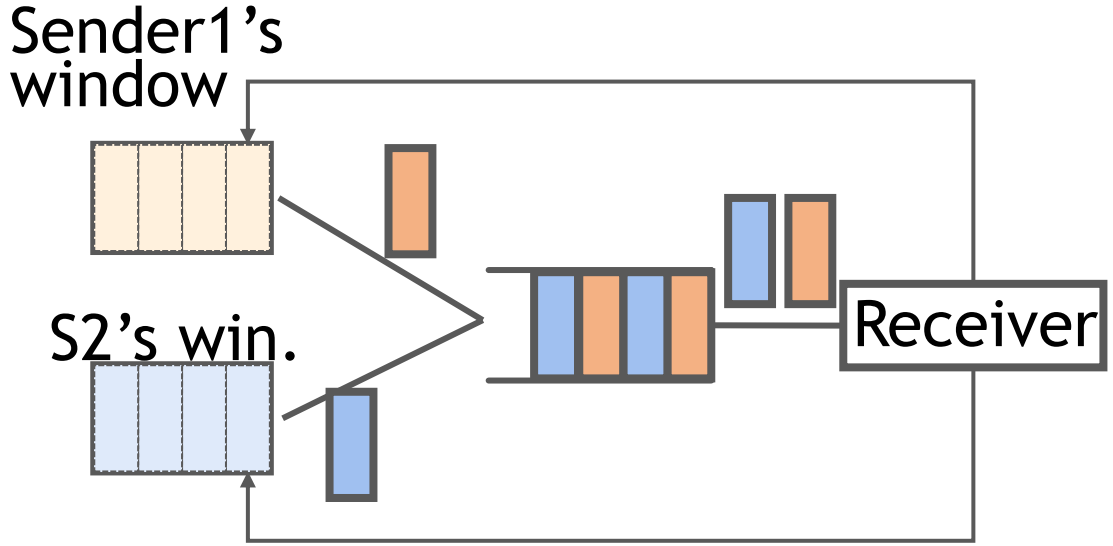
$$d(t) = RTT_{base} + d_q(t) = RTT_{base} + \frac{q(t)}{\mu}$$

$$= RTT_{base} + \frac{\sum i(t) - BDP_{base}}{\mu} = \frac{\sum i(t)}{\mu}$$

$$\leq \frac{\boxed{\sum w(t)}}{\mu}$$

The total window size of flows passing through the bottleneck

Visual Illustration



Principle 2: Delay & Delay Gradient as Precise Signals

Similarly, an analogous relationship holds for the delay gradient

(Please see our paper for details on the delay gradient)

Delay — Window

$$d(t) \leq \frac{\sum w(t)}{\mu}$$

Equality holding when the queue is neither empty nor overflowing, and flows are **bounded by window.**

Delay gradient — Rate

$$g(t) \leq \frac{\sum r(t)}{\mu} - 1$$

Equality holding when the queue is neither empty nor overflowing, and flows are **bounded by rate.**

Principle 2: Delay & Delay Gradient as Precise Signals

Similarly, an analogous relationship holds for the delay gradient

(Please see our paper for details on the delay gradient)

Delay — Window

$$d(t) \leq \frac{\sum w(t)}{\mu}$$

Equality holding when the queue is neither empty nor overflowing, and flows are **bounded by window.**

Delay gradient — Rate

$$g(t) \leq \frac{\sum r(t)}{\mu} - 1$$

Equality holding when the queue is neither empty nor overflowing, and flows are **bounded by rate.**

The two signals are precise under mutually exclusive conditions

Principle 2: Delay & Delay Gradient as Precise Signals

Similarly, an analogous relationship holds for the delay gradient

(Please see our paper for details on the delay gradient)

Delay — Window

$$d(t) \leq \frac{\sum w(t)}{\mu}$$

Equality holding when the queue is neither empty nor overflowing, and flows are **bounded by window.**

Delay gradient — Rate

$$g(t) \leq \frac{\sum r(t)}{\mu} - 1$$

Equality holding when the queue is neither empty nor overflowing, and flows are **bounded by rate.**

The two signals are precise under mutually exclusive conditions

Therefore, they should be used complementarily

OSCAR: O(1)-Step Convergence And Readily-deployable

OSCAR has a simple control law

$$
\begin{aligned}
w(t) &= \frac{D_{target}}{d(t-1)} \cdot w(t-1) \\
r(t) &= \frac{1}{1+g(t-1)} \cdot r(t-1)
\end{aligned}
$$

OSCAR: O(1)-Step Convergence And Readily-deployable

OSCAR has a simple control law, and four novel technical challenges.

How to calculate congestion signals precisely?

How to select the reference state for updates?

$$w(t) = \frac{D_{target}}{d(t-1)} \cdot w(t-1)$$

$$r(t) = \frac{1}{1+g(t-1)} \cdot r(t-1)$$

How to avoid conflicts between independent control loops?

How to make MIMD fair?

OSCAR Design 1: Batched Least Square

Naive: divide & EWMA

1. Record $sendTs_{0\ldots n}$ and $rtt_{0\ldots n}$
2. Divide: $g_i = \frac{rtt_i - rtt_{i-1}}{sendTs_i - sendTs_{i-1}}$
3. EWMA: $g = \alpha \cdot g + (1-\alpha) \cdot g_i$

Problem: Noise amplification by small denominator

$$g_i = \frac{rtt_i - rtt_{i-1}}{sendTs_i - sendTs_{i-1}}$$

The interval between packets is at ~**10ns** level in high-speed network.

OSCAR Design 1: Batched Least Square

Naive: divide & EWMA

1. Record $sendTs_{0\ldots n}$ and $rtt_{0\ldots n}$
2. Divide: $g_i = \frac{rtt_i - rtt_{i-1}}{sendTs_i - sendTs_{i-1}}$
3. EWMA: $g = \alpha \cdot g + (1-\alpha) \cdot g_i$

Problem: Noise amplification by small denominator

$$g_i = \frac{rtt_i - rtt_{i-1}}{sendTs_i - sendTs_{i-1}}$$

The interval between packets is at **~10ns** level in high-speed network.

Batched Least Squares (BLS)

Denoting $sendTs_i$ as x_i and rtt_i as y_i, estimate g over batch period τ.

$$g = \frac{n \cdot \sum(x_i \cdot y_i) - \sum x_i \cdot \sum y_i}{n \cdot \sum(x_i^2) - (\sum x_i)^2}$$

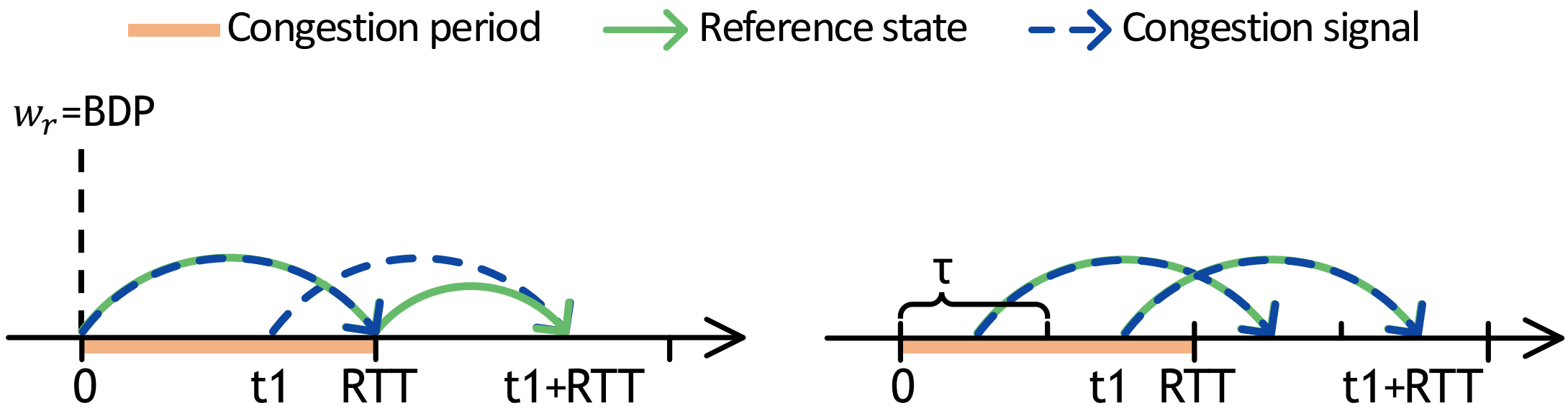
Constant time and space complexity

It only needs to store five cumulative values: $n, \sum x_i, \sum y_i, \sum x_i^2, \sum x_i y_i$.

τ set to half of base RTT. Timestamp warparound solved gracefully. Please see paper for details.

OSCAR Design 2: Reference State Selection

HPCC: reacts to past signal using an unsynchronized state

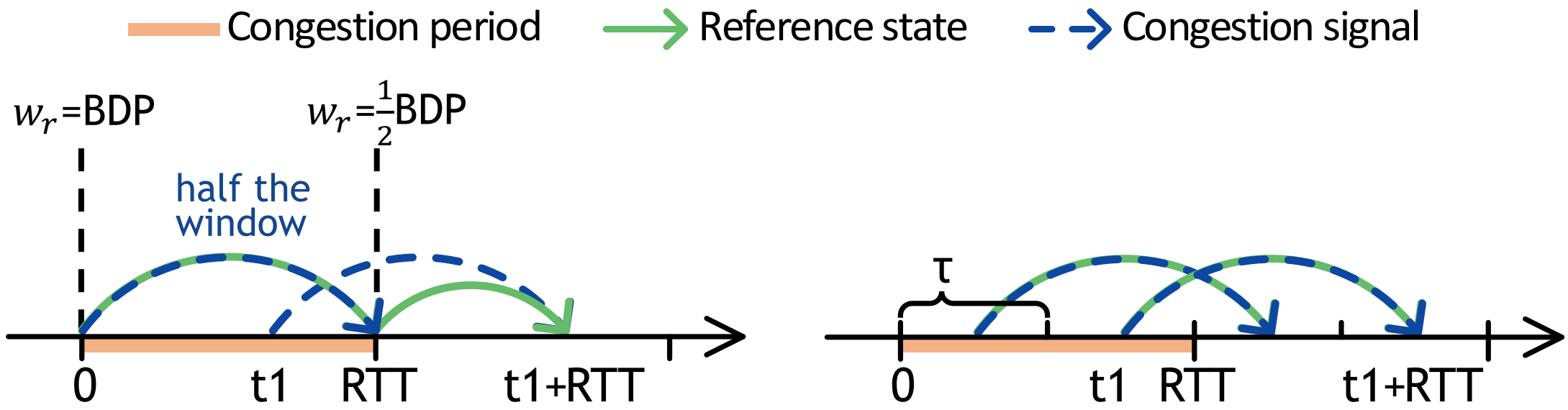


HPCC's reference state selection

OSCAR's reference state selection

OSCAR Design 2: Reference State Selection

HPCC: reacts to past signal using an unsynchronized state

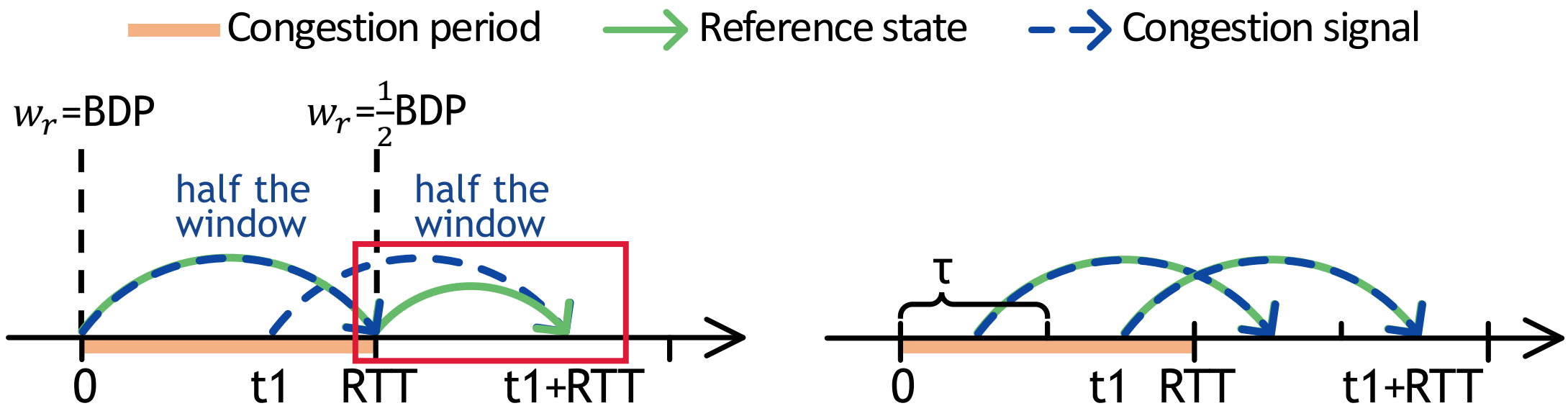


HPCC's reference state selection

OSCAR's reference state selection

OSCAR Design 2: Reference State Selection

HPCC: reacts to past signal using an unsynchronized state

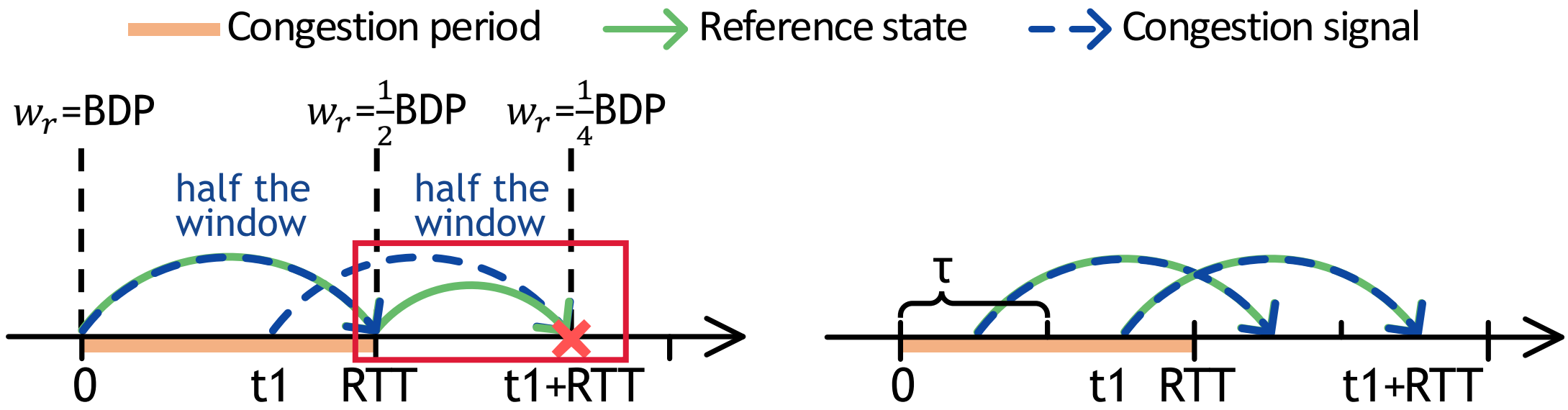


HPCC's reference state selection

OSCAR's reference state selection

OSCAR Design 2: Reference State Selection

HPCC: reacts to past signal using an unsynchronized state



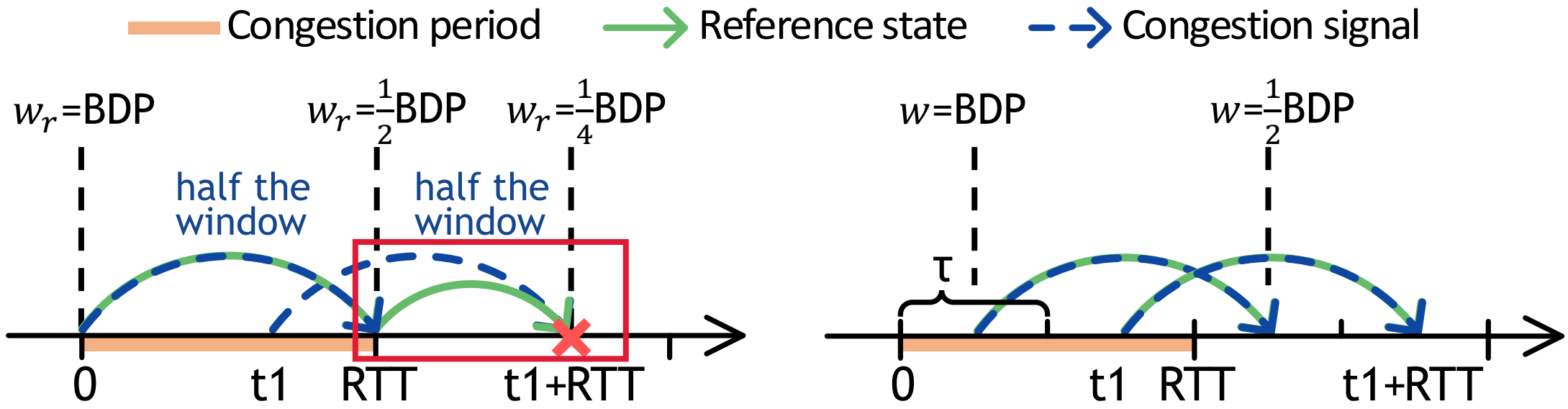
HPCC's reference state selection

OSCAR's reference state selection

OSCAR Design 2: Reference State Selection

HPCC: reacts to past signal using an unsynchronized state

OSCAR: reacts to signal using the synchronized state that produced it



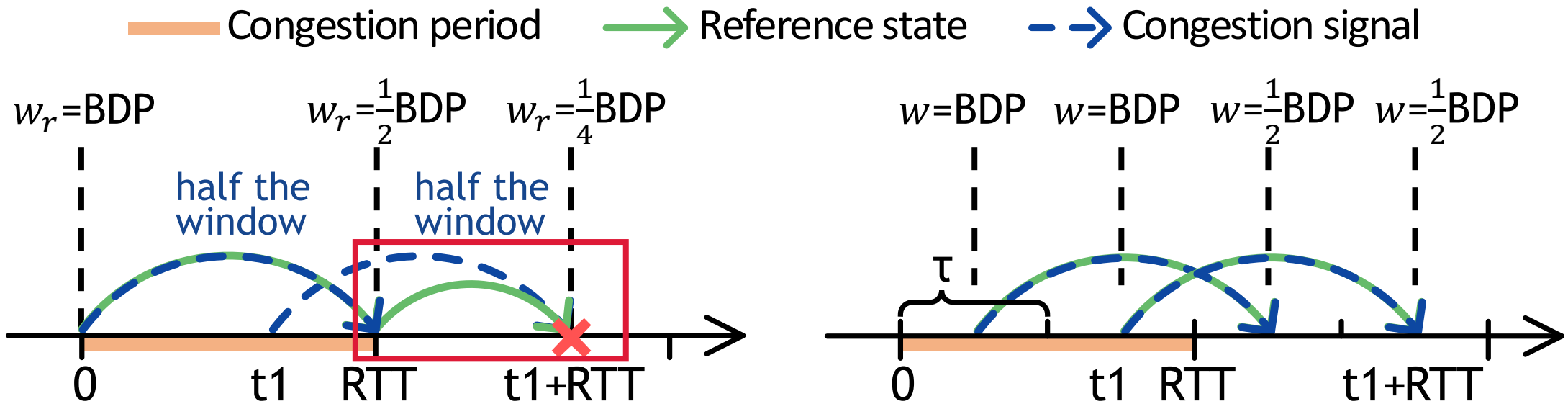
HPCC's reference state selection

OSCAR's reference state selection

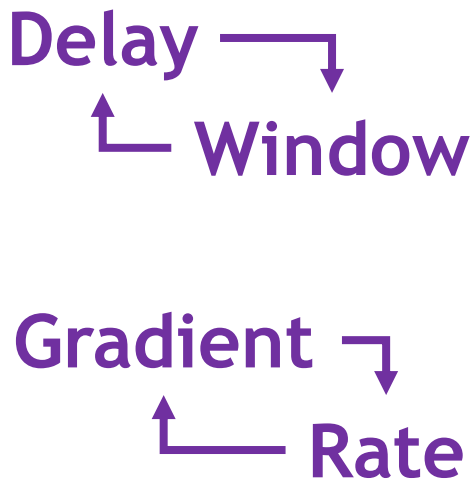
OSCAR Design 2: Reference State Selection

HPCC: reacts to past signal using an unsynchronized state

OSCAR: reacts to signal using the synchronized state that produced it



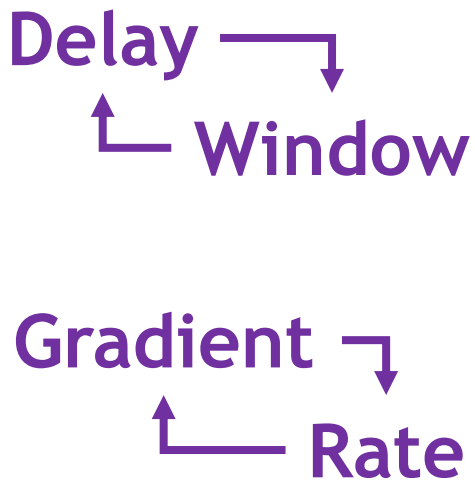
OSCAR Design 3: Coordinating Two Loops



OSCAR Design 3: Coordinating Two Loops

Theorem: Convergence is only possible when the window and rate are at the same ratio relative to their respective targets. Call that ratio *u*.

(Please see Theorem 4.1 in paper for details.)



OSCAR Design 3: Coordinating Two Loops

Theorem: Convergence is only possible when the window and rate are at the same ratio relative to their respective targets. Call that ratio *u*.

(Please see Theorem 4.1 in paper for details.)

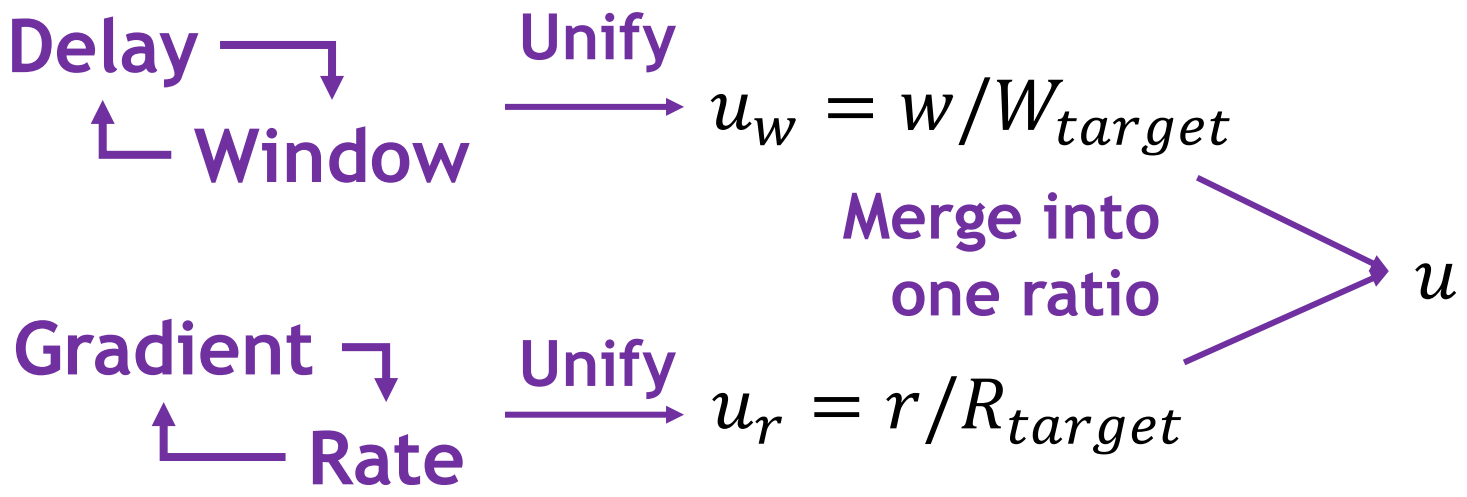
Delay ↔ Window —Unify→ $u_w = w/W_{target}$

Gradient ↔ Rate —Unify→ $u_r = r/R_{target}$

OSCAR Design 3: Coordinating Two Loops

Theorem: Convergence is only possible when the window and rate are at the same ratio relative to their respective targets. Call that ratio *u*.

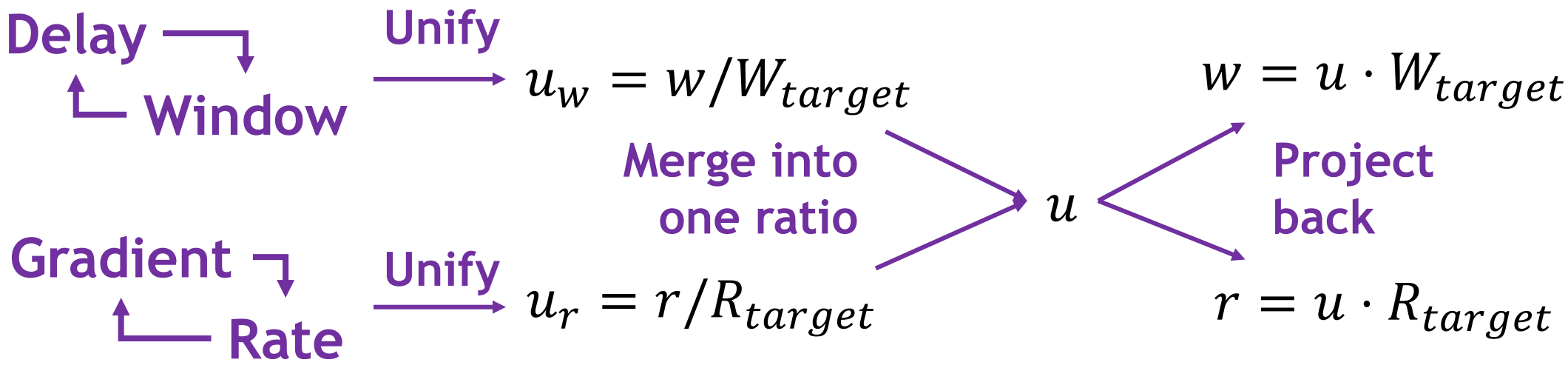
(Please see Theorem 4.1 in paper for details.)



OSCAR Design 3: Coordinating Two Loops

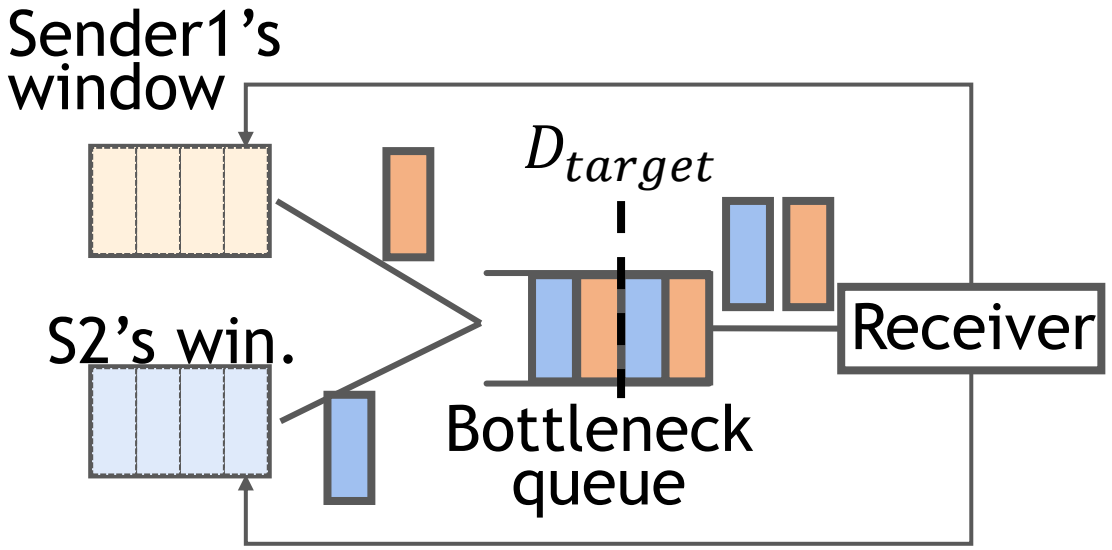
Theorem: Convergence is only possible when the window and rate are at the same ratio relative to their respective targets. Call that ratio *u*.

(Please see Theorem 4.1 in paper for details.)



OSCAR Design 3: Coordinating Two Loops

It is impossible to always satisfy the target of both loops simultaneously.



Delay — Window

Flows should decelerate their windows as delay > D_{target}

Delay gradient — Rate

Flows should keep their rate as delay gradient = 0

OSCAR Design 3: Coordinating Two Loops

OSCAR selects the ratio based on both direction and magnitude.

Pseudocode

OSCAR Design 3: Coordinating Two Loops

OSCAR selects the ratio based on both direction and magnitude.

Direction:

OSCAR prioritizes the delay loop's target (queue stability) as gradient loop's target allows arbitrary queue length.

Pseudocode

Accelerating if delay < D_{target}:

Decelerating else:

OSCAR Design 3: Coordinating Two Loops

OSCAR selects the ratio based on both direction and magnitude.

Direction:

OSCAR prioritizes the delay loop's target (queue stability) as gradient loop's target allows arbitrary queue length.

Magnitude:

Inaccurate signals only shrink magnitude. Therefore OSCAR adopts the larger result of the two loops when accelerating.

Pseudocode

Accelerating if delay < D_{target}:

Take the larger $u = max(u_w, u_r)$

Decelerating else:

Take the smaller $u = min(u_w, u_r)$

OSCAR Design 3: Coordinating Two Loops

OSCAR selects the ratio based on both direction and magnitude.

Direction:

OSCAR prioritizes the delay loop's target (queue stability) as gradient loop's target allows arbitrary queue length.

Magnitude:

Inaccurate signals only shrink magnitude. Therefore OSCAR adopts the larger result of the two loops when accelerating.

Pseudocode

Accelerating if delay < D_{target}:

Take the larger $u = max(u_w, u_r)$

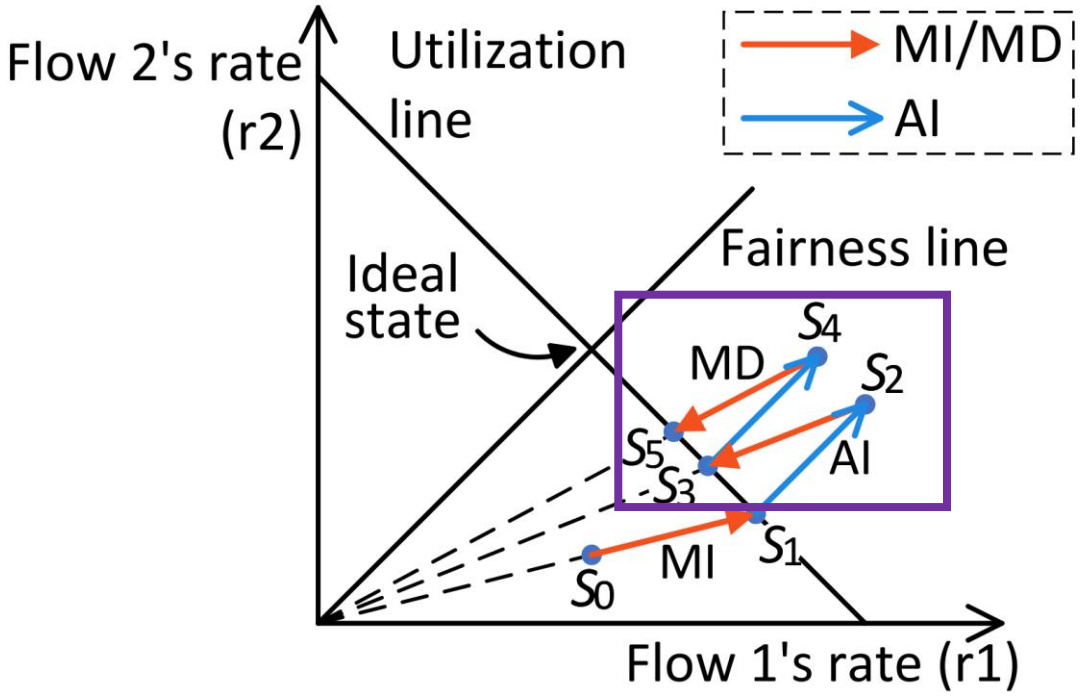
Decelerating else:

Take the smaller $u = min(u_w, u_r)$

(Please see paper for hyper increase design to avoid under-utilization and the full pseudocode.)

OSCAR Design 4: Ensuring Fairness (optional)

OSCAR adds a small AI term after each MIMD operation, which forms an AIMD cycle around the target and provably converges to fairness.



(Please see paper for details.)

Implementation and Overhead Evaluation

We implement OSCAR and other CCs for comparison on Linux 5.4.0 with DPDK 25.03 and GCC 8.4 at the highest optimization level. Servers are connected by 40Gbps Links with 13.5 μs base RTT.

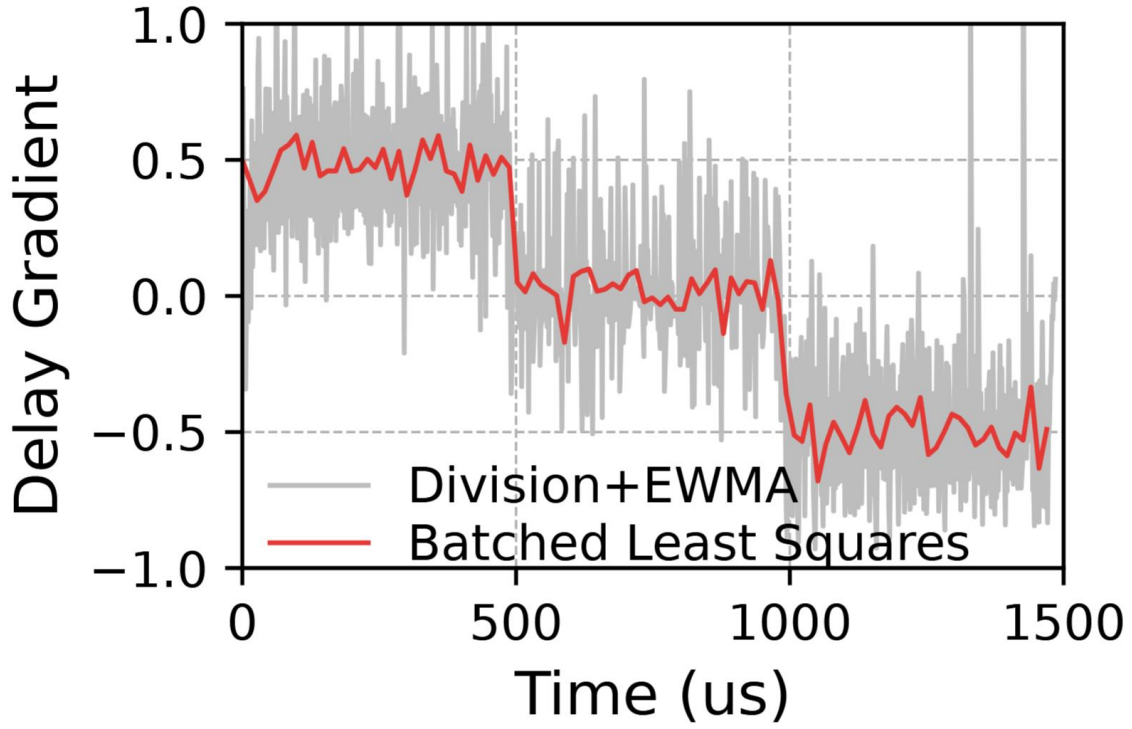
Algorithm	OSCAR	DCQCN	Swift	HPCC	PowerTCP
Overhead (CPU cycles)	52	46	104	92	159

OSCAR's calculation overhead is lower than that of most data center CCs, particularly precise-INT-based CC.

(Please see paper for details of overhead analysis and discussion on RNIC implementation.)

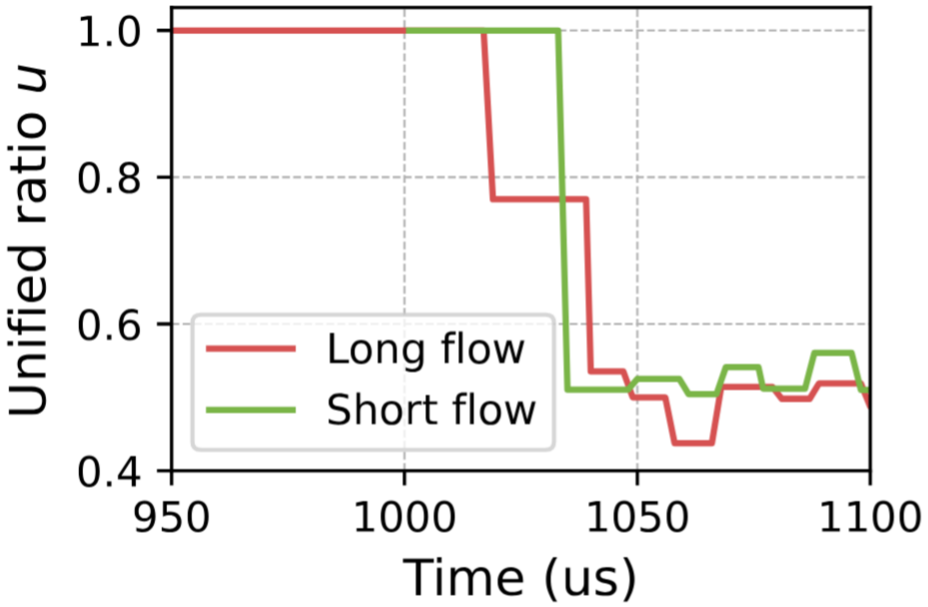
Testbed Evaluation

OSCAR's delay gradient calculation is more precise than Division+EWMA, achieving a Mean Squared Error 13.4× smaller (0.0058 vs. 0.079).

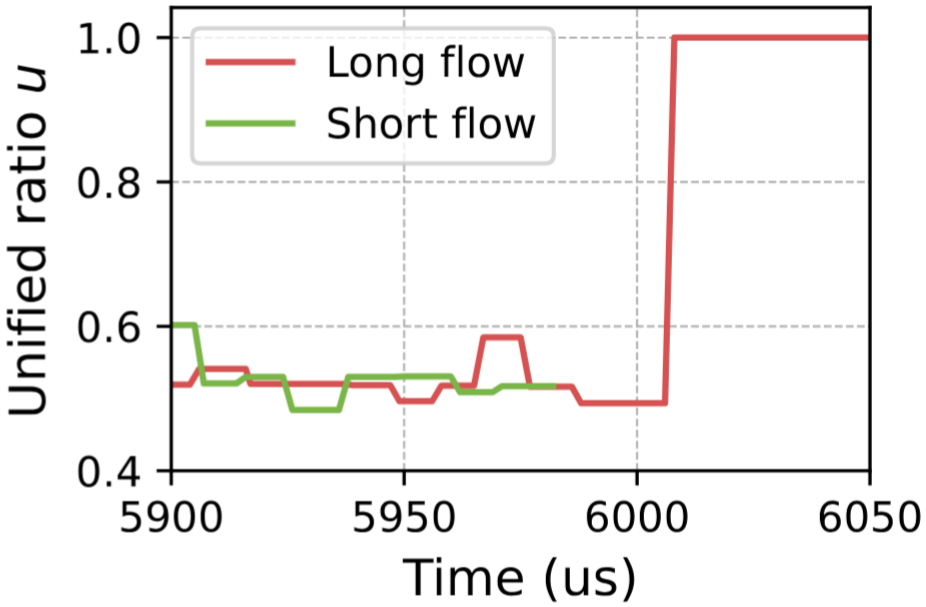


Testbed Evaluation

Microscopic view shows OSCAR can converge in O(1) steps in the testbed.



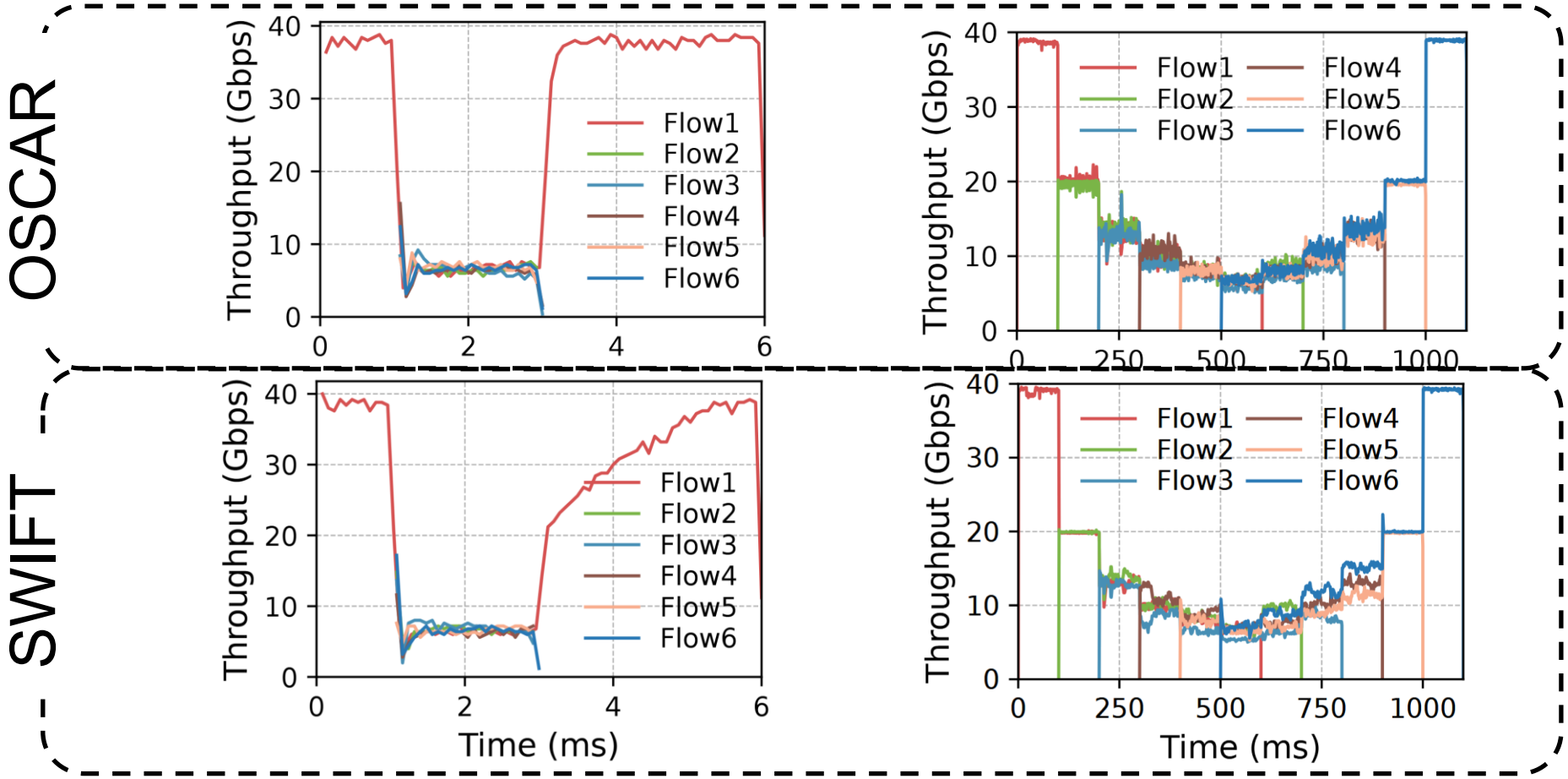
(a) Congestion starts.



(b) Congestion ends.

Testbed Evaluation

OSCAR can converge to the target state rapidly and fairly.



Simulation Evaluation - Standard ECMP Routing

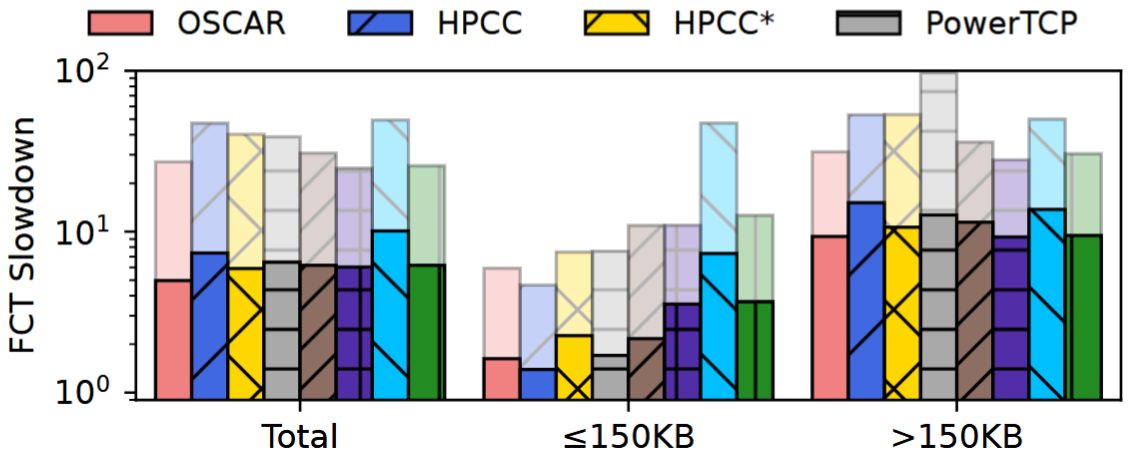
Simulation runs on a 320-host fat-tree in ns-3. FCT slowdown is shown.

Cache Follower

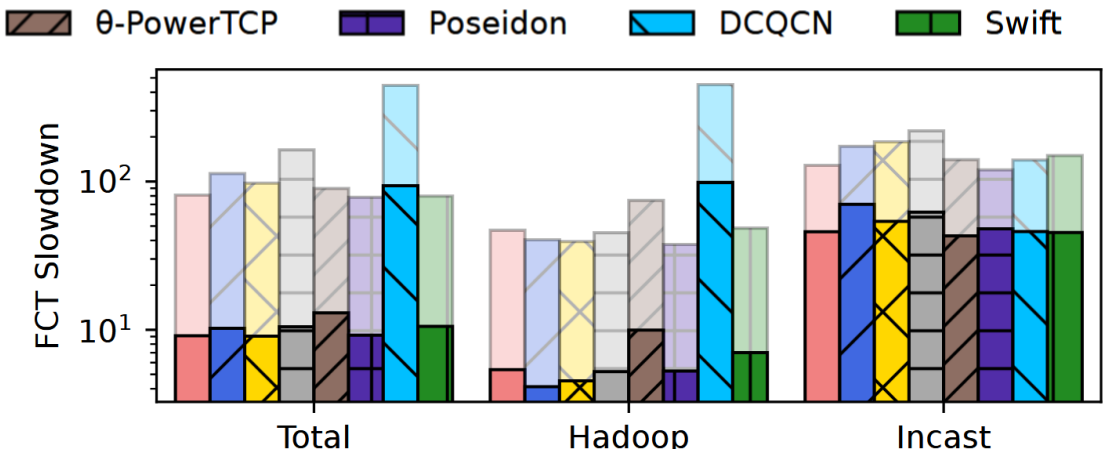
- vs HPCC: small flow +7.6%, large flow -62%
  avg −48%, tail −74%
- vs HPCC* (same target): avg −19%, tail −48%

Hadoop + Incast

- vs HPCC: Hadoop +23%, Incast -53%
  avg −12%, tail −40%
- vs HPCC*: avg matched, tail −21%



CacheFollower at 80% load

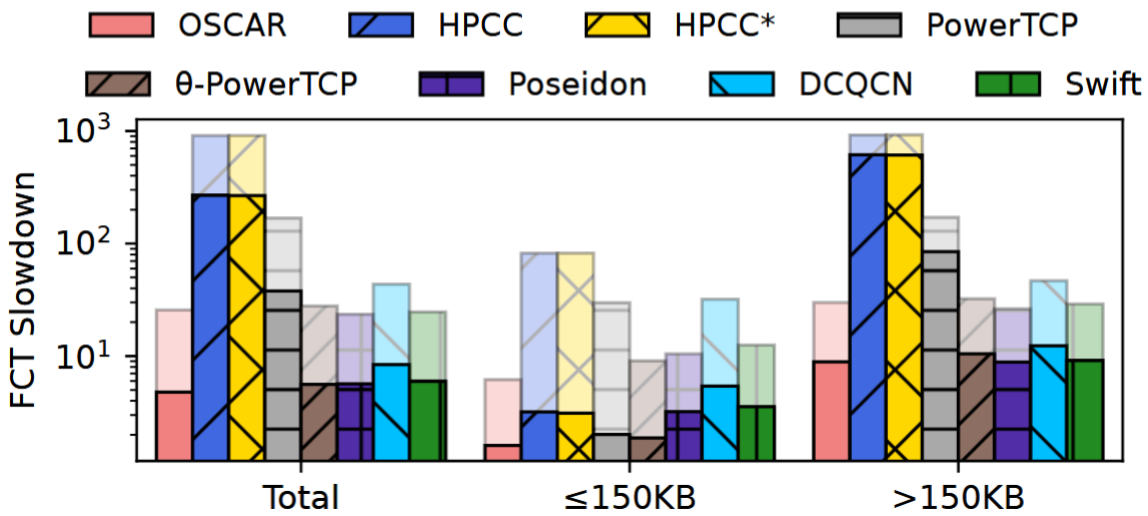


Hadoop (50% load) + Incast (20% load)

Simulation Evaluation - Per-packet Load Balancing

Simulation runs on a 320-host fat-tree in ns-3. FCT slowdown is shown.

Precise-INT-based CCs break as they needs consecutive INTs from the same port.
OSCAR vs others: avg −17% to −76%



CacheFollower at 80% load w/ packet spraying

Conclusion

OSCAR is the first readily-deployable CC achieving O(1)-step convergence.

Contributions:

- Borrowing **big-O notation** as a new lens to quantify CC convergence speed.
- Pinpointing how and when **delay & delay gradient can reach INT-level precision**.
- **Novel techniques** to realize OSCAR. We believe:
  - BLS can serve as a readily-deployable alternative to INT-based monitoring tools.
  - Synchronized reference state selection could be a drop-in improvement for existing CCs.

Thanks !

Find our open-source code with a one-click script to reproduce our results at:
https://github.com/NASA-NJU/OSCAR