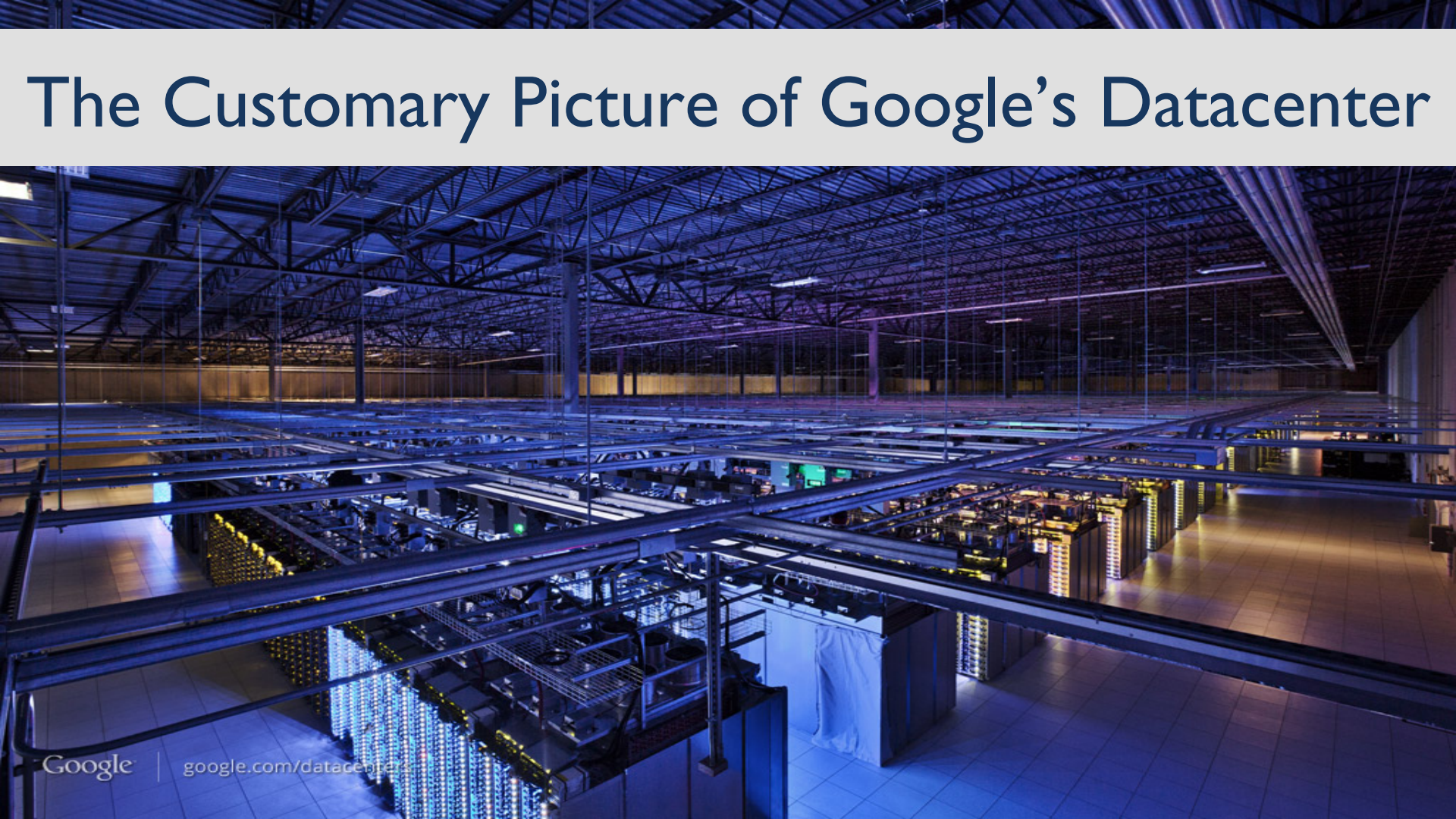
Google

TIMELY: RTT-based Congestion Control for the Datacenter

Radhika Mittal*(UC Berkeley), Vinh The Lam, Nandita Dukkipati,
Emily Blem, Hassan Wassel, Monia Ghobadi*(Microsoft),
Amin Vahdat, Yaogong Wang, David Wetherall, David Zats

* *Work done while at Google*

The Customary Picture of Google's Datacenter



The Story of RTT

Once upon a time,
there was a congestion signal,
called RTT.

It had all the qualities to rule the congestion control in Datacenters.

Qualities of RTT

- Fine-grained and informative
- Quick response time
- No switch support needed
- End-to-end metric

Applicability in Datacenters

- RTT-based schemes discarded for WANs.
  - Compete poorly with loss-based schemes.

- **This is not a concern for the Datacenters.**

Stringent Performance Requirements

- Tightly coupled computing tasks
- Require both **high throughput** and **low latencies**
- Packet losses are too costly

However, it was too hard to measure the RTTs accurately.

While RTT was banished from WANs,

it was never even
considered for Datacenters!!

And ECN emerged as a hero instead.

ECN

DCTCP (2010)
D^2TCP (2012)
HULL (2012)
TCP-Bolt (2014)
DCQCN (2015)

This is the tale of how we helped RTT become a powerful congestion signal in modern datacenters.

Contributions

1. Show that accurate RTT measurements are possible.
2. Demonstrate the goodness of RTT as a congestion signal.
3. Develop TIMELY: RTT-based congestion control for the datacenters.

Accurate RTT Measurement

Hardware Assisted RTT Measurement

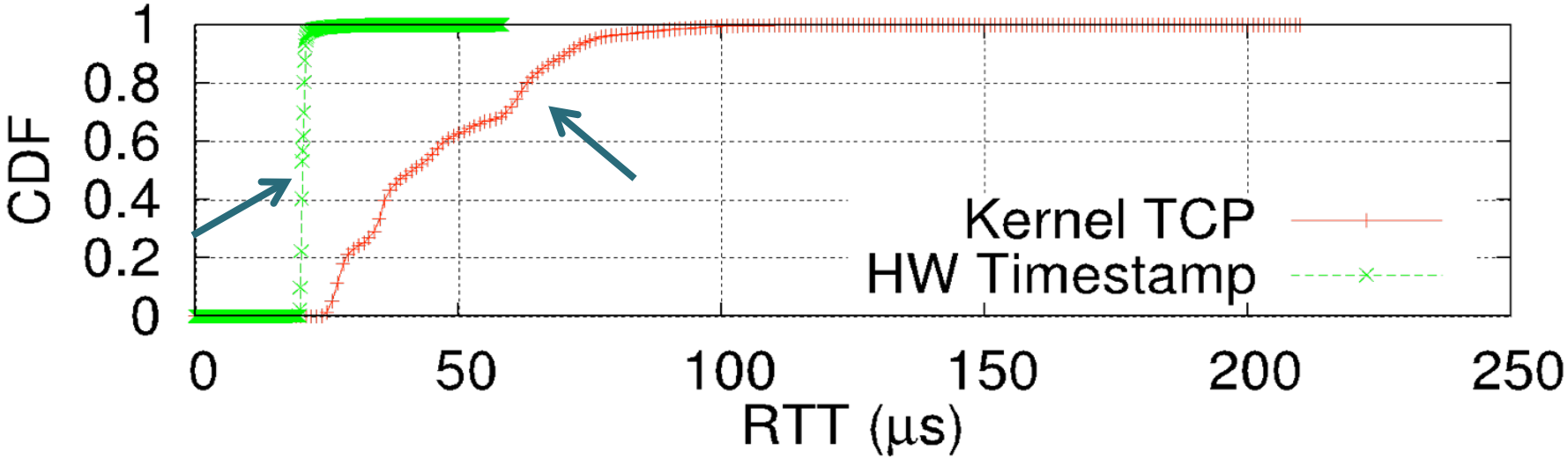
Hardware Timestamps

– mitigate noise in measurements

Hardware Acknowledgements

– avoid processing overhead

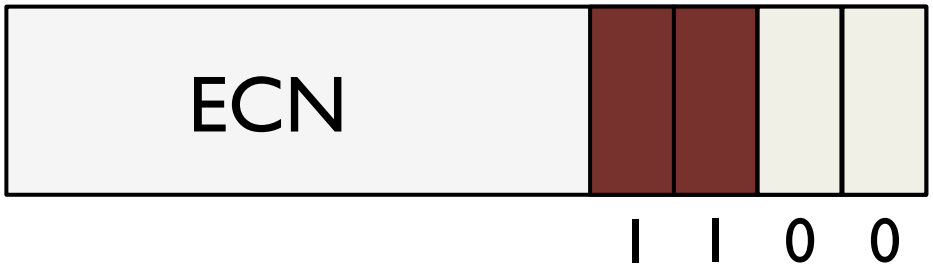
Hardware vs Software Timestamps



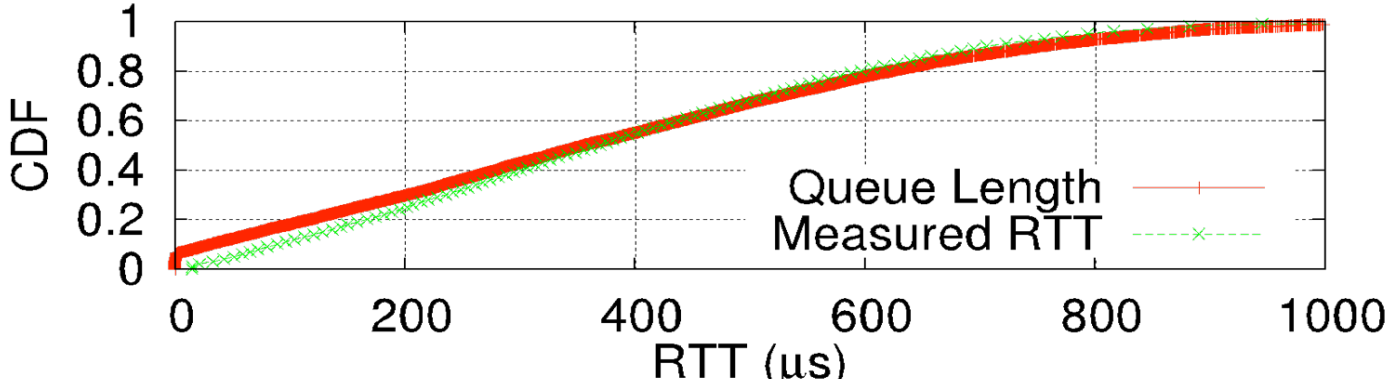
Kernel Timestamps introduce significant noise in RTT measurements compared to HW Timestamps.

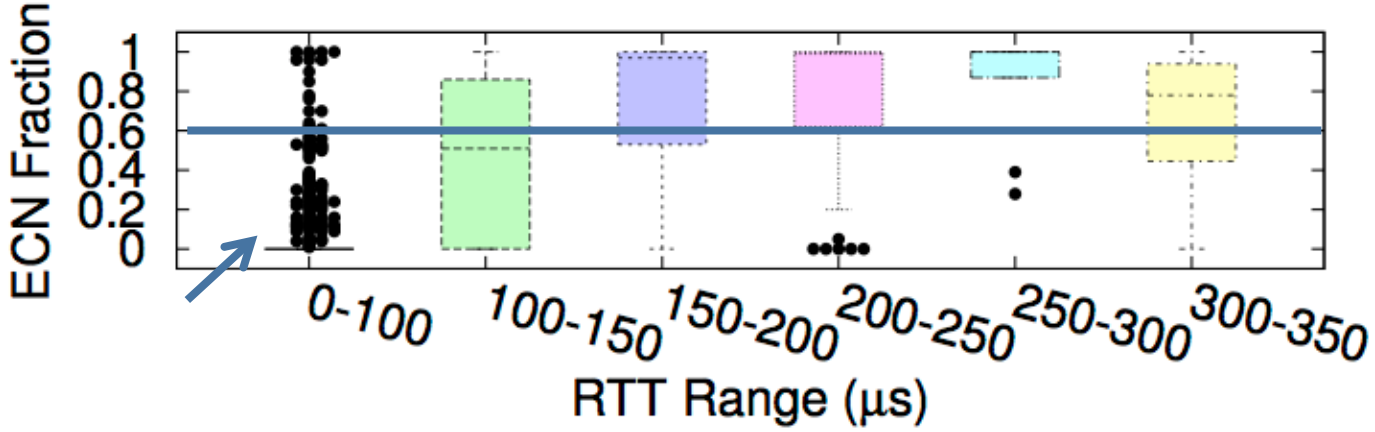
RTT as a congestion signal

RTT is a multi-bit signal



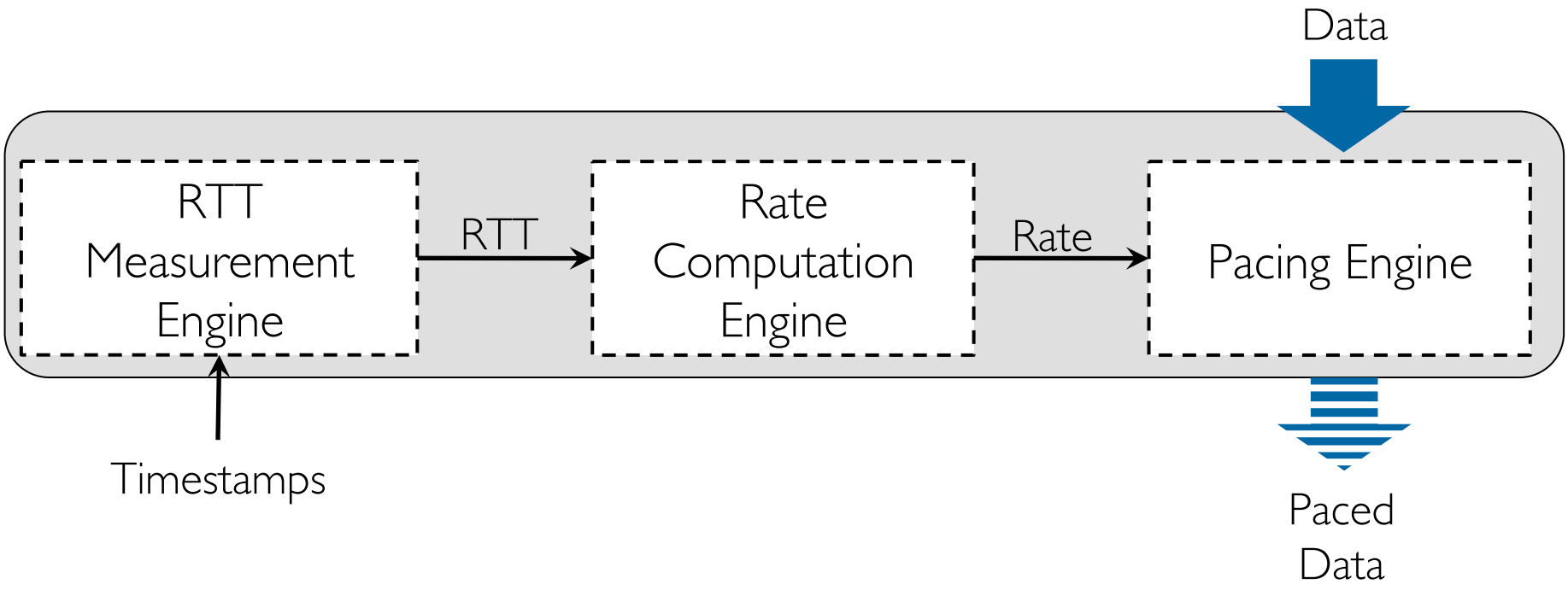
RTT correlates with queuing delay



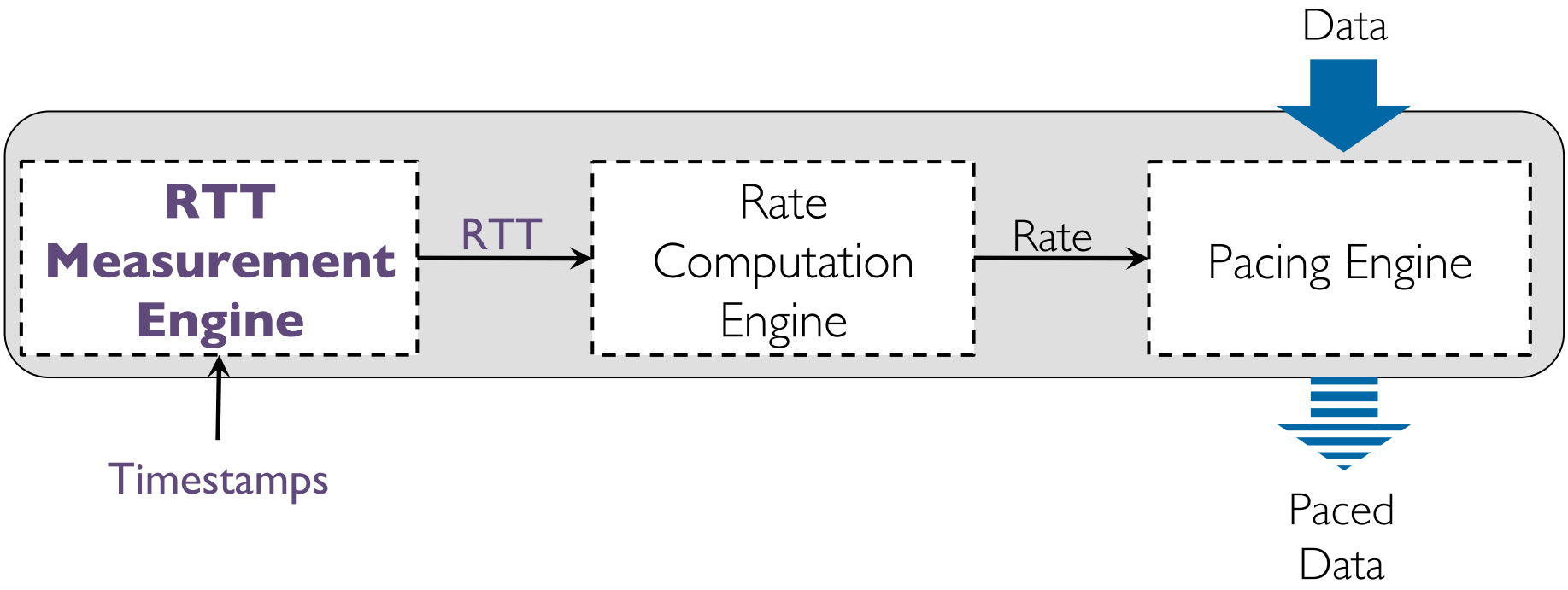


TIMELY Framework

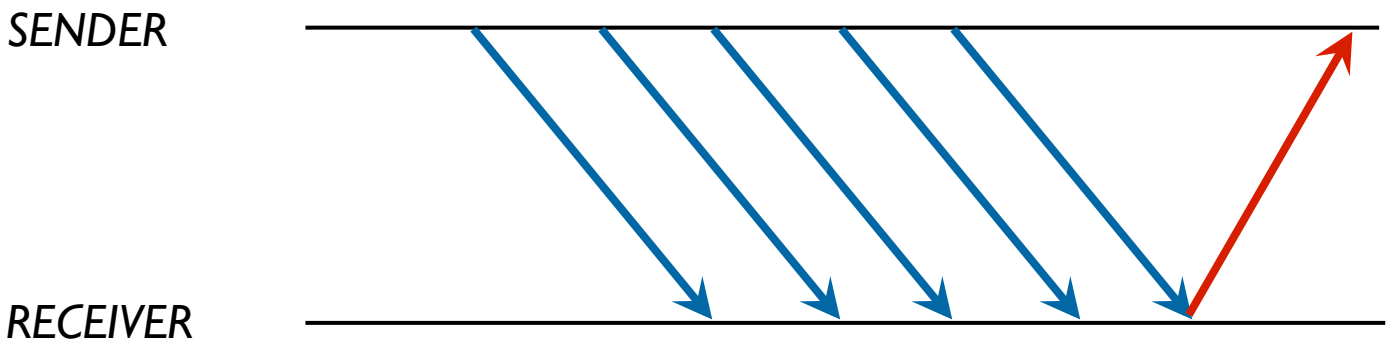
Overview


Data
RTT Measurement Engine
RTT
Rate Computation Engine
Rate
Pacing Engine
Timestamps
Paced Data

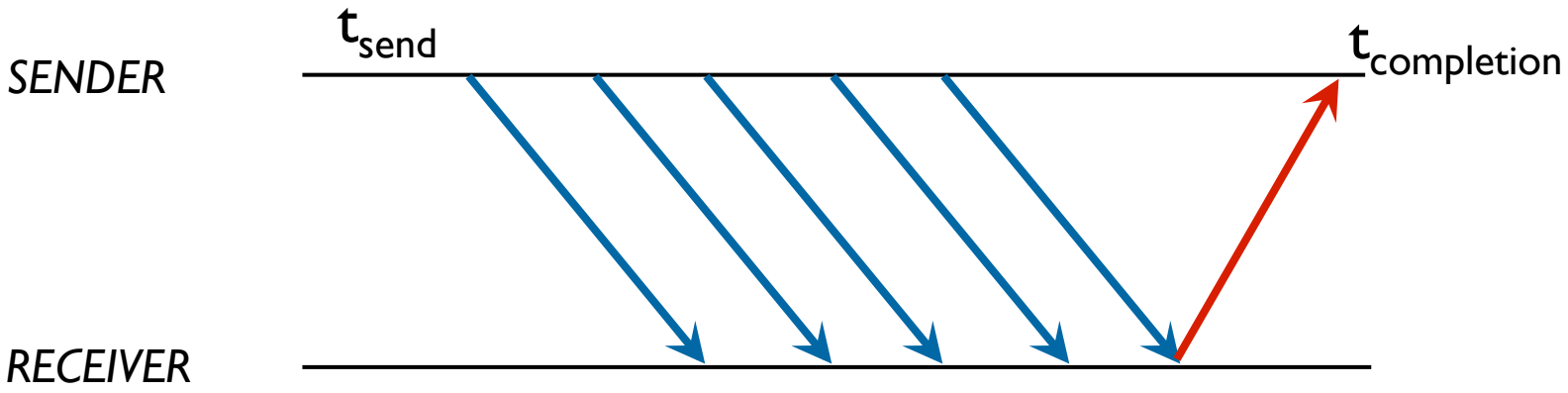
Overview


Data
RTT Measurement Engine
RTT
Rate Computation Engine
Rate
Pacing Engine
Timestamps
Paced Data

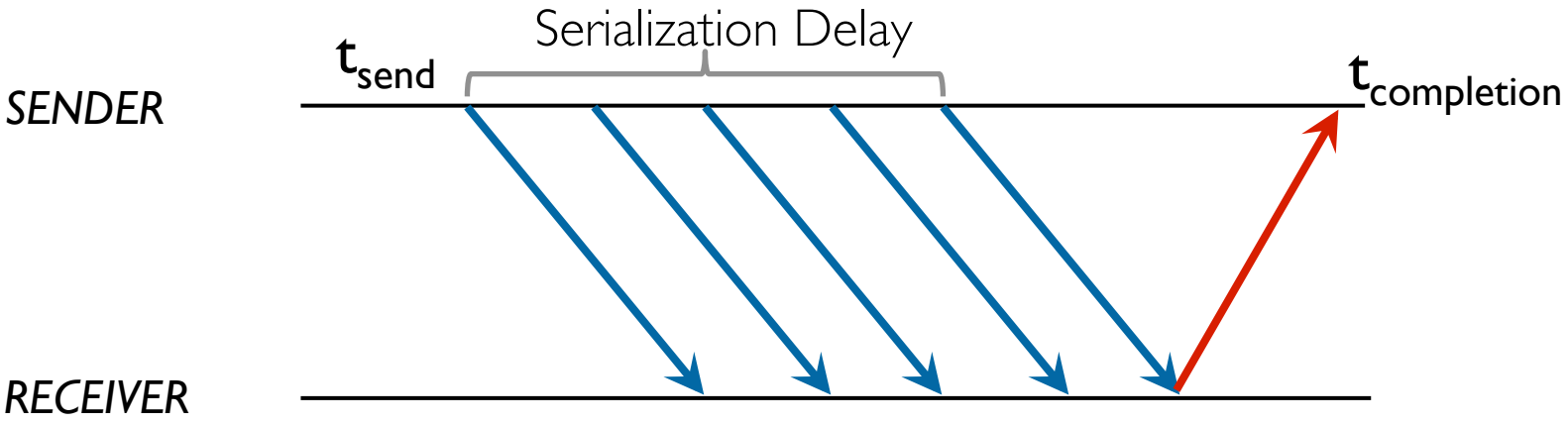
RTT Measurement Engine



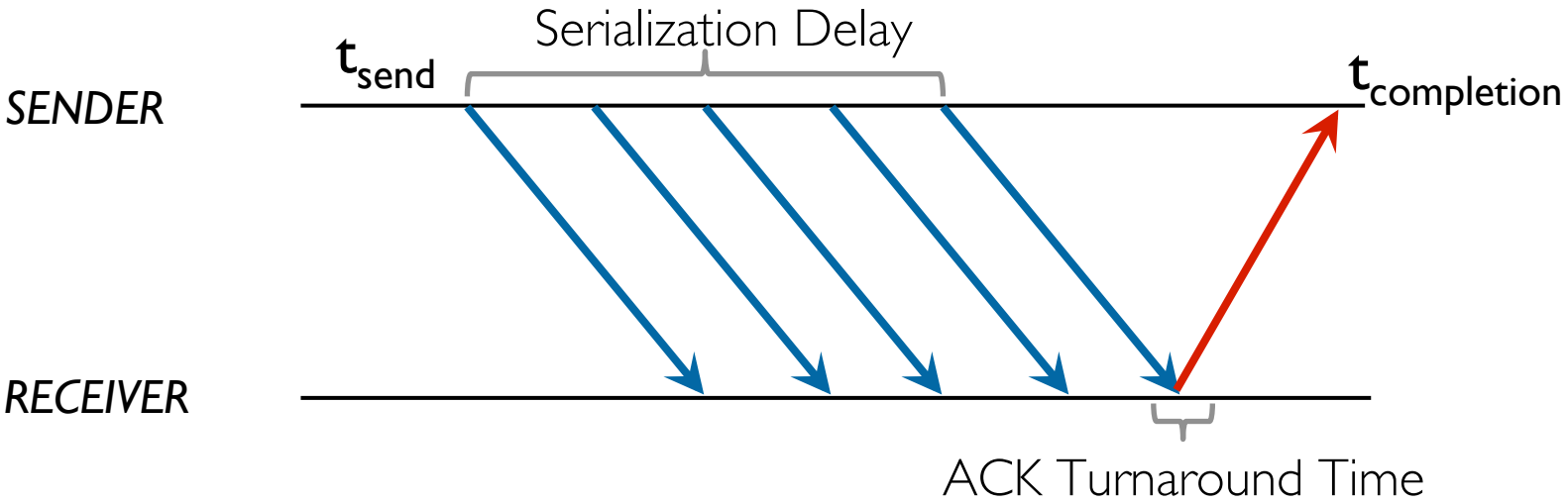
RTT Measurement Engine



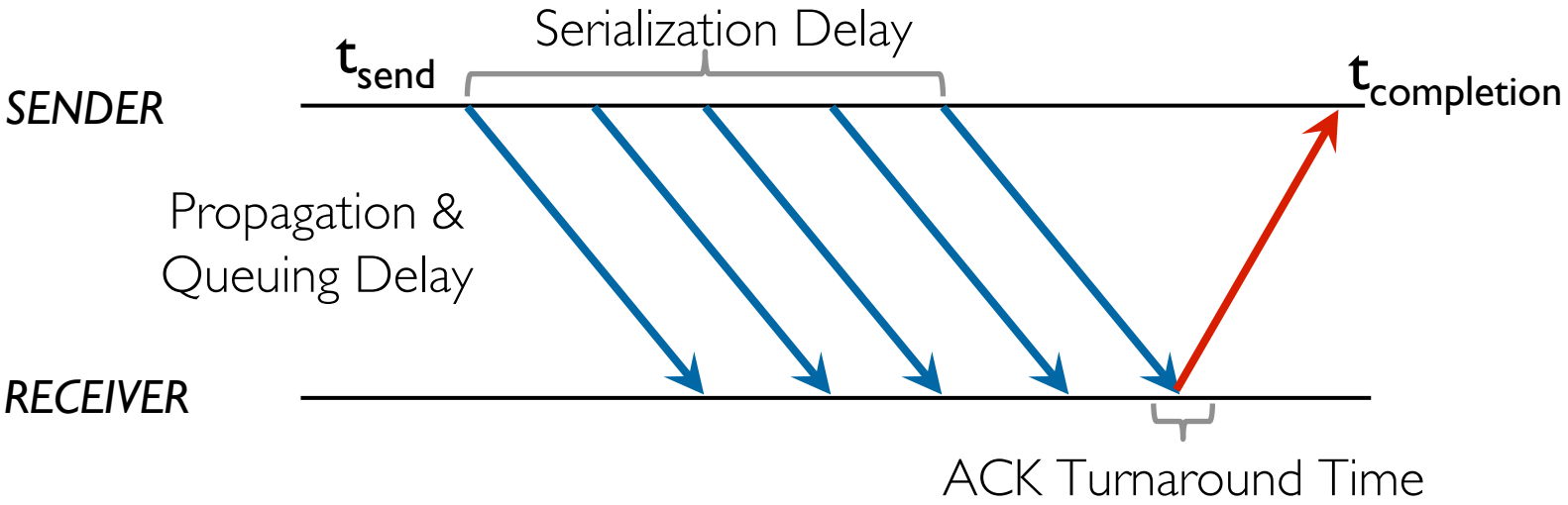
RTT Measurement Engine



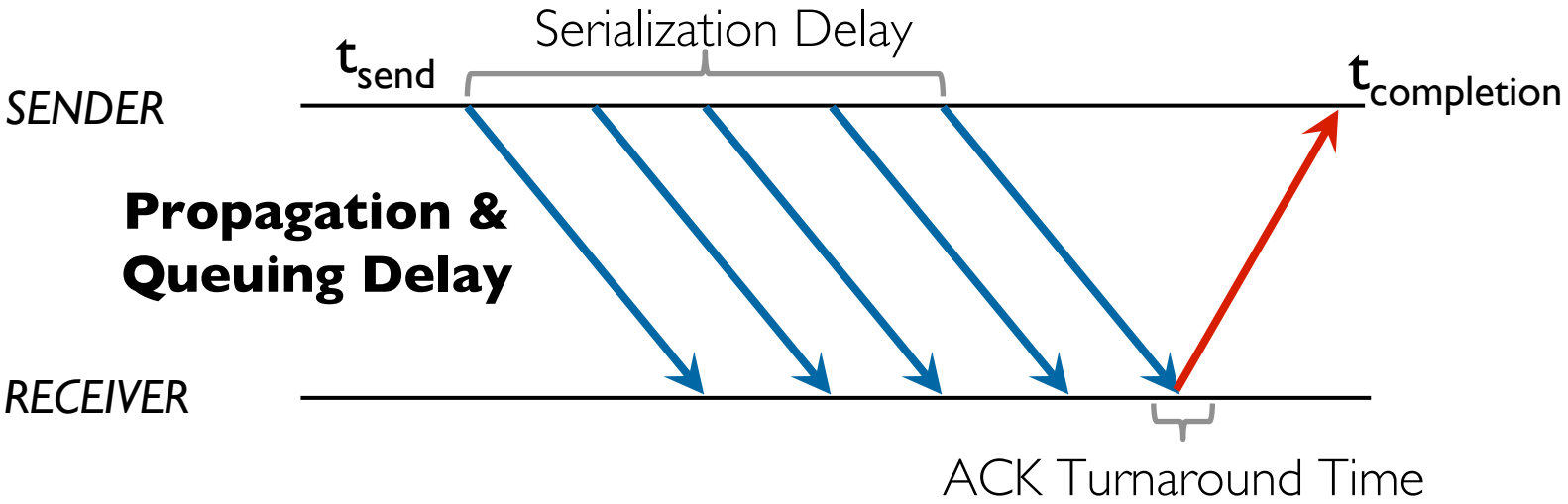
RTT Measurement Engine



RTT Measurement Engine

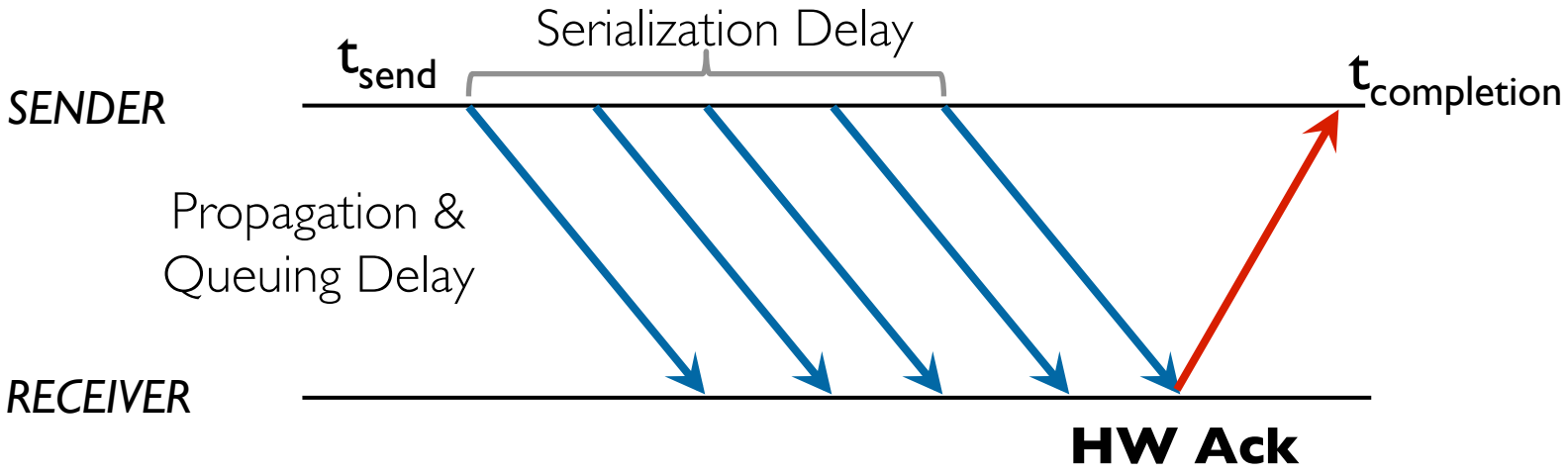


RTT Measurement Engine



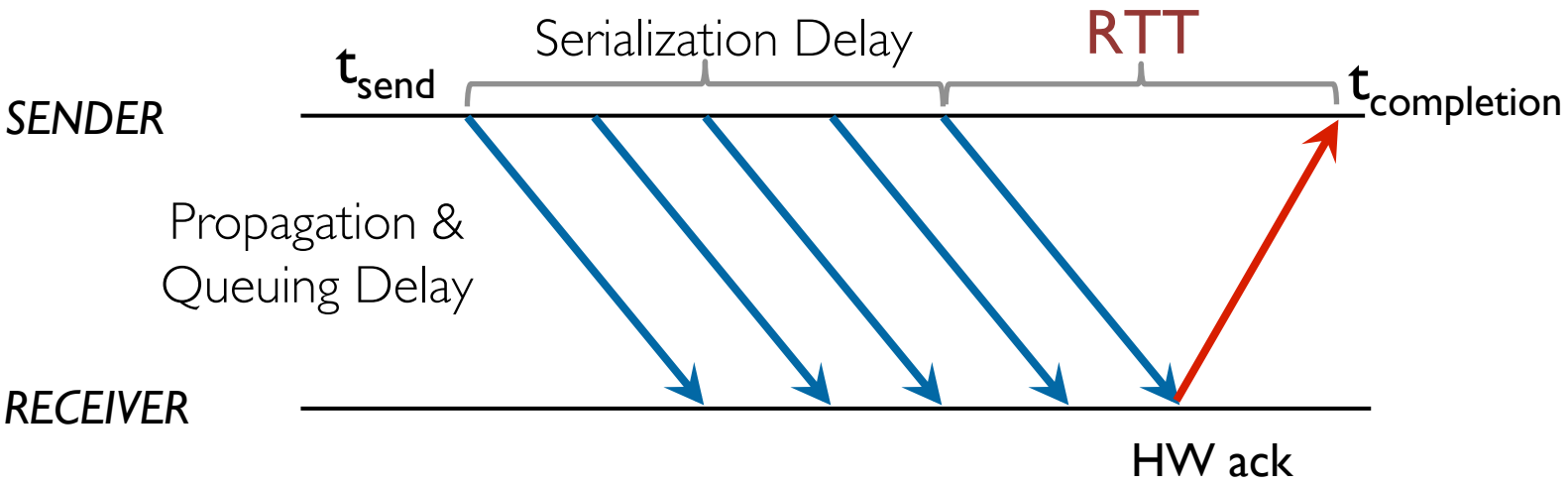
RTT = Propagation & Queuing Delay

RTT Measurement Engine



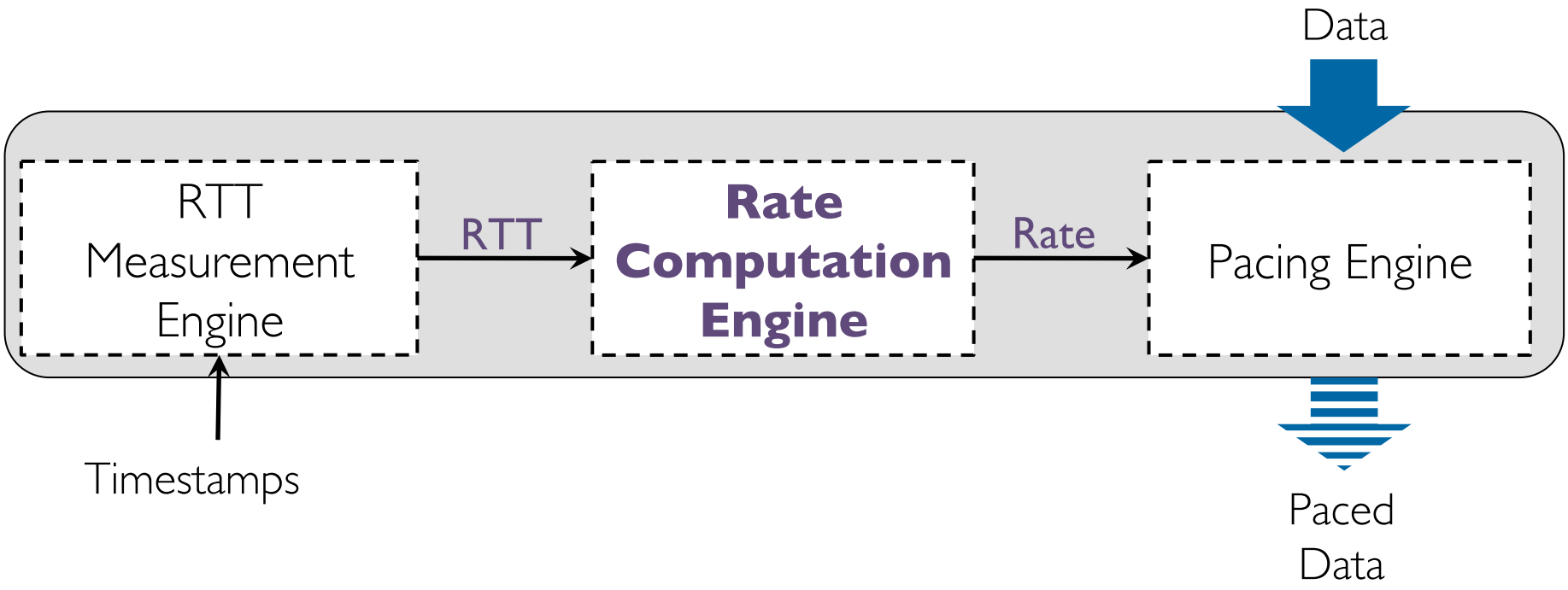
RTT = Propagation + Queuing Delay

RTT Measurement Engine



$$RTT = t_{completion} - t_{send} - \text{Serialization Delay}$$

Overview


Data
RTT Measurement Engine
RTT
Rate Computation Engine
Rate
Pacing Engine
Timestamps
Paced Data

Rate Computation Engine

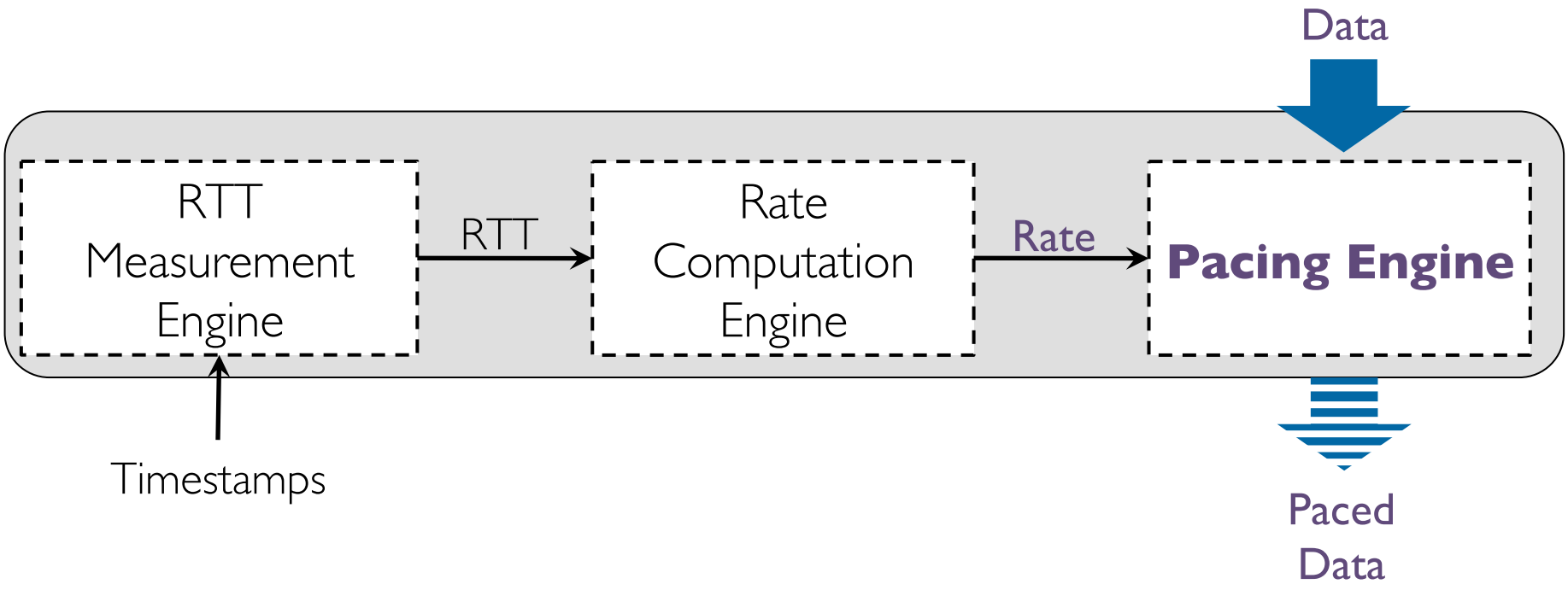
On each segment completion event

- Input: RTT sample
- Runs the rate update algorithm
- Output: updated rate

Why do we compute the rate as opposed to a window?

- Segment sizes as high as 64KB
- (32us RTT x 10Gbps) = 40KB window size
- 40KB < 64KB : Window makes no sense

Overview


Data
RTT Measurement Engine
RTT
Rate Computation Engine
Rate
Pacing Engine
Timestamps
Paced Data

Pacing Engine

Computes the send time of a segment using

- segment size
- computed flow rate
- time of last transmission

TIMELY Algorithm

Goals

- Flow Completion Time
  - Large Flows: High Throughput
  - Short Flows: Low tail latencies
- Ride the throughput-latency curve
  - until tail latencies become unacceptable
  - low latency prioritized over throughput
- Fairness and Stability

Challenges

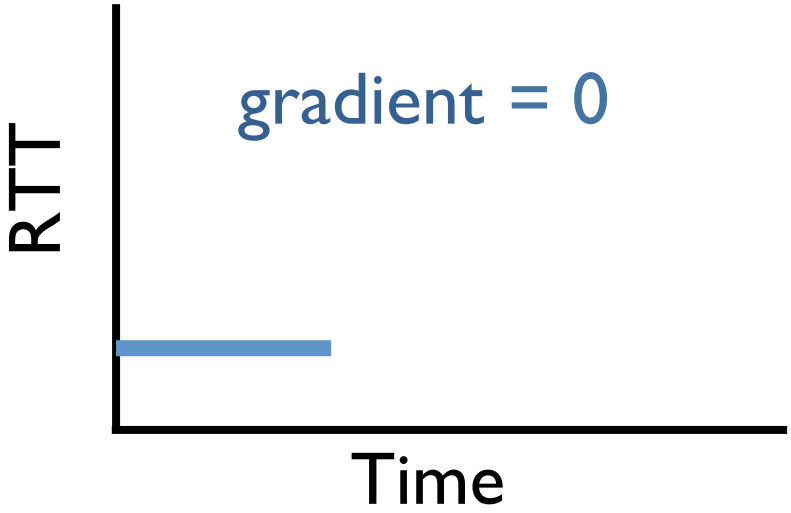
- Bursty traffic
- Coarse-grained feedback
- Existing delay-based schemes cannot be used

Algorithm Overview

Gradient-based
Increase / Decrease

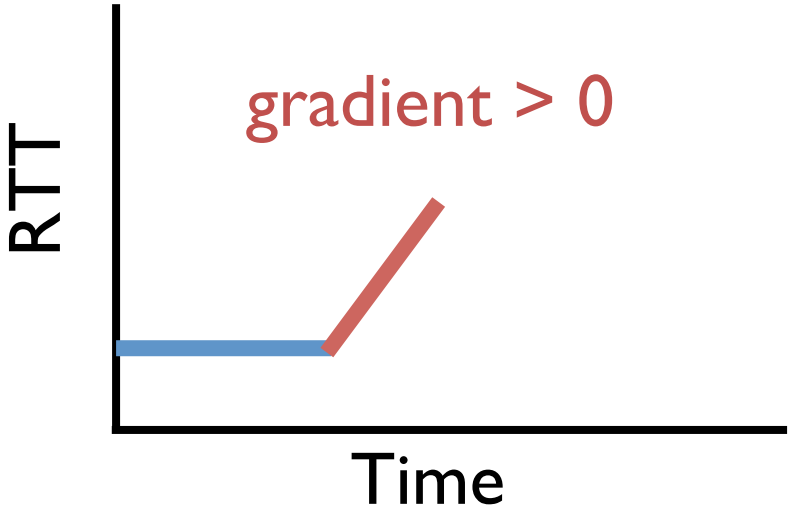
Algorithm Overview

Gradient-based Increase / Decrease



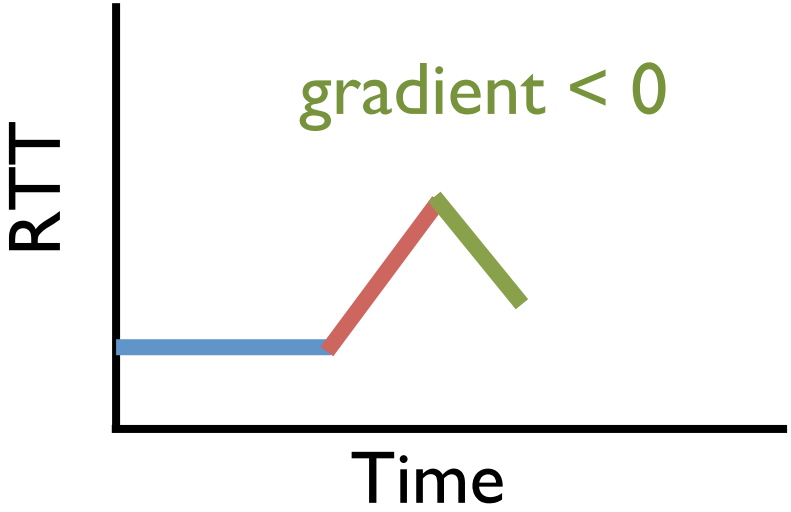
Algorithm Overview

Gradient-based Increase / Decrease



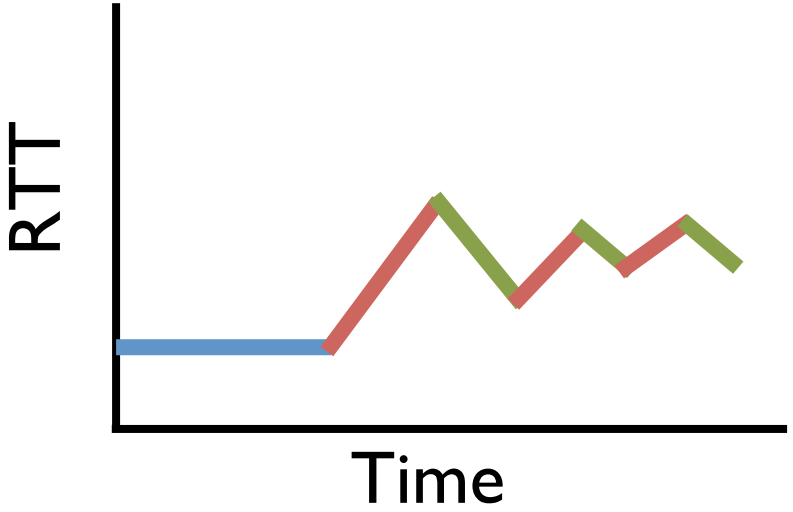
Algorithm Overview

Gradient-based Increase / Decrease

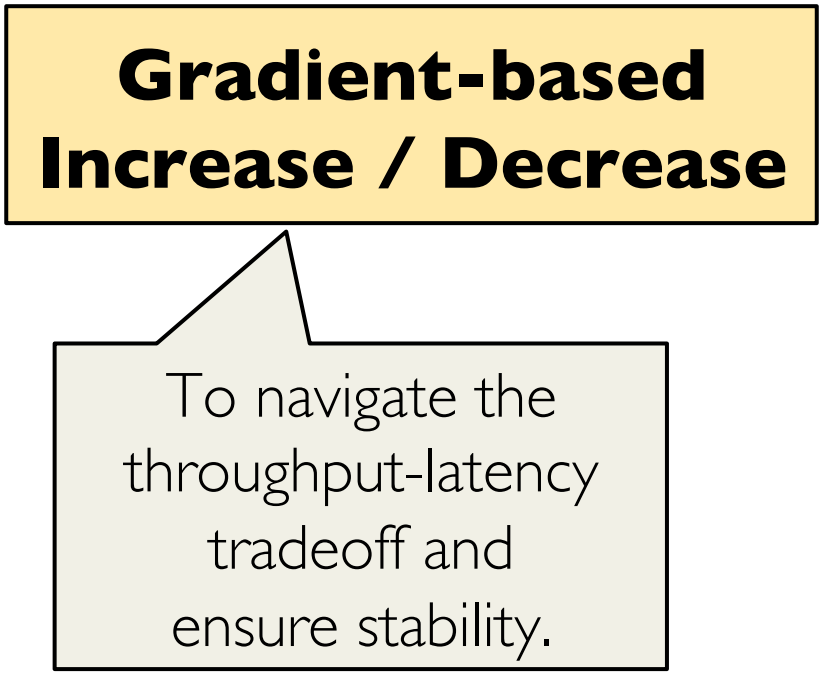


Algorithm Overview

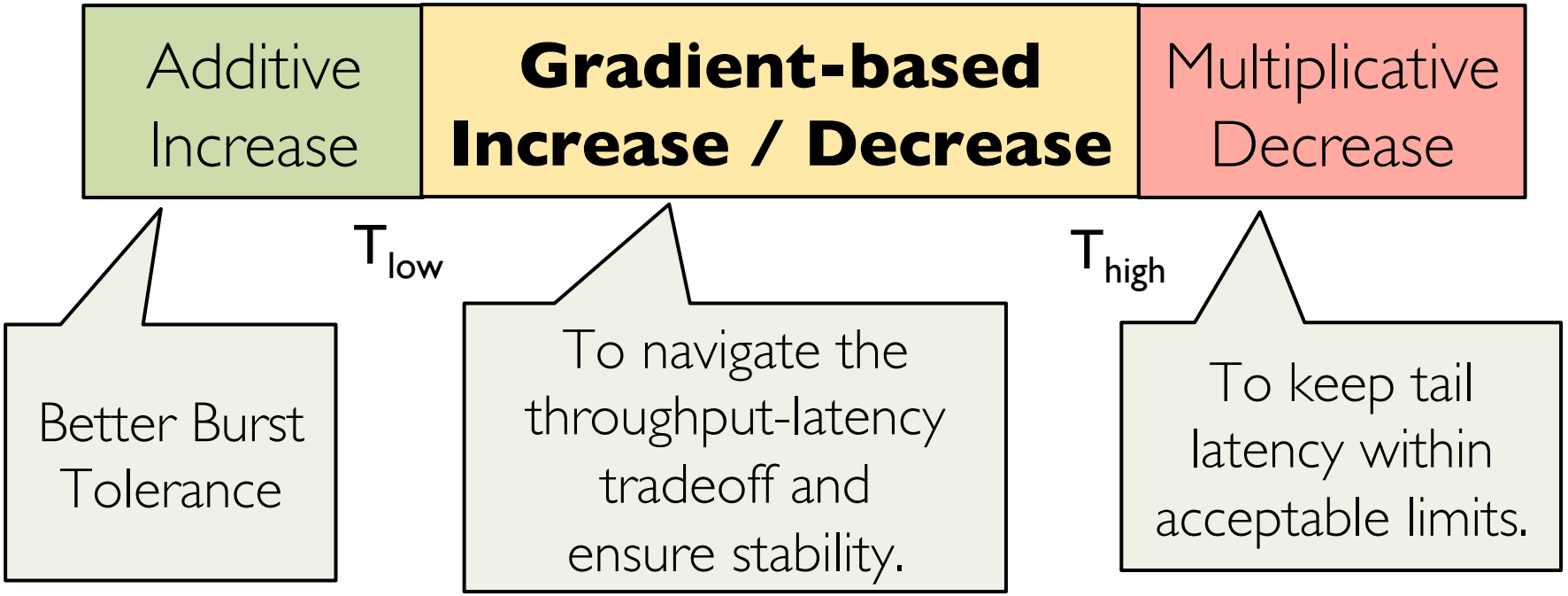
Gradient-based Increase / Decrease



Algorithm Overview



Algorithm Overview



Evaluation

Implementation Set-up

- TIMELY is implemented in the context of RDMA.
  - RDMA write and read primitives used to invoke NIC services.

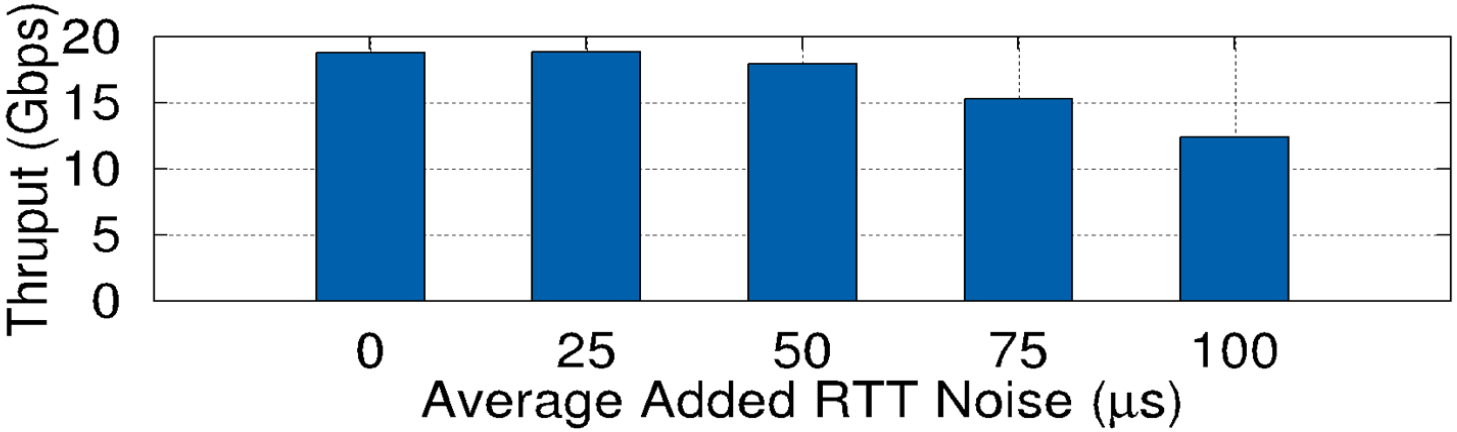
- Priority Flow Control is enabled in the network fabric.
  - RDMA transport in the NIC is sensitive to packet drops.
  - PFC sends out pause frames to ensure lossless network.

Experimental Set-up

- Small-scale experiments:
  - Incast traffic pattern with 10 clients and a server sharing the same rack.
- Large scale experiments:
  - A few hundreds of machine in a classic Clos-network

- Impact of RTT noise
- Comparison with PFC
- Comparison with DCTCP
- More results in the paper

Impact of RTT Noise



Throughput degrades with increasing noise in RTT.
Precise RTT measurement is crucial.

Comparison with PFC - Small Scale

	TIMELY	PFC
Throughput (Gbps)	19.4	19.5
Avg RTT (us)	61	658
99%ile RTT (us)	116	1036

Comparison with PFC - Small Scale

	TIMELY	PFC
Throughput (Gbps)	19.4	19.5
Avg RTT (us)	61	658
99%ile RTT (us)	116	1036

Comparison with PFC - Small Scale

	TIMELY	PFC
Throughput (Gbps)	19.4	19.5
Avg RTT (us)	61	658
99%ile RTT (us)	116	1036

Comparison with DCTCP

	TIMELY	DCTCP
Throughput (Gbps)	19.4	19.5
Avg RTT (us)	61	598
99%ile RTT (us)	116	1490

Comparison with DCTCP

	TIMELY	DCTCP
Throughput (Gbps)	19.4	19.5
Avg RTT (us)	61	598
99%ile RTT (us)	116	1490

Comparison with DCTCP

	TIMELY	DCTCP
Throughput (Gbps)	19.4	19.5
Avg RTT (us)	61	598
99%ile RTT (us)	116	1490

Summary

- Show that RTT signals measured with NIC HW strongly correlate with network queuing.
- Use RTT as a congestion signal to build TIMELY.
- Evaluate TIMELY in an RDMA framework.
  - low tail latencies with near-optimum throughput