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 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 victorious 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 effectiveness of RTT as a congestion signal.

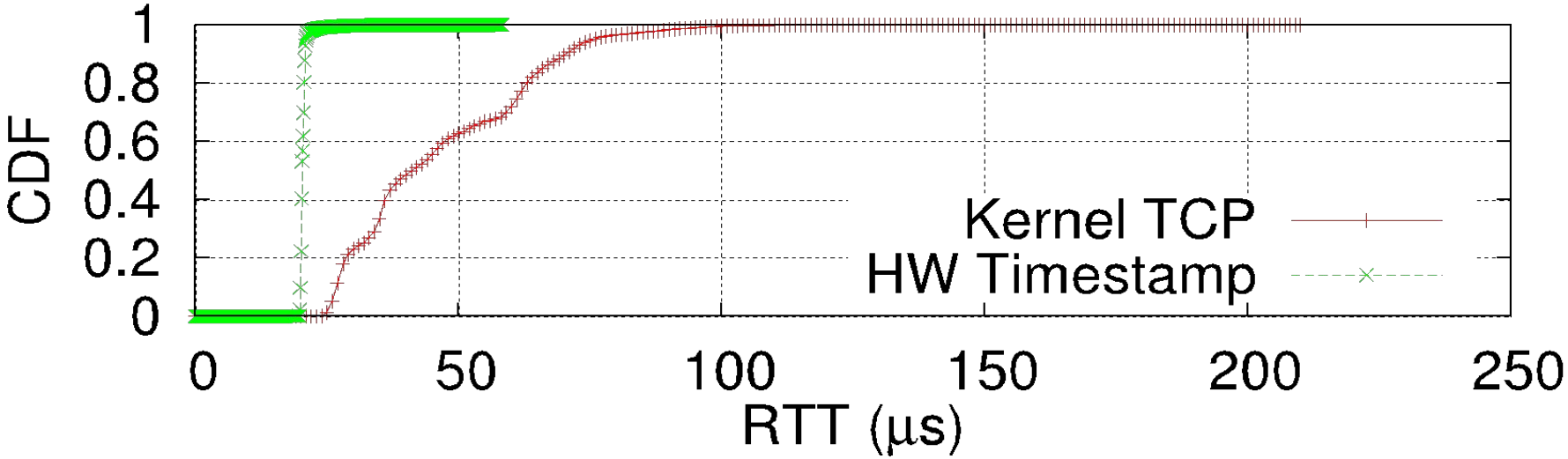
3. Develop TIMELY, an RTT-based congestion control for the datacenters.

Accurate RTT Measurement

Hardware Assisted RTT Measurement

- HW timestamps
  - mitigate noise in measurement
- HW Acknowledgements
  - low processing overhead

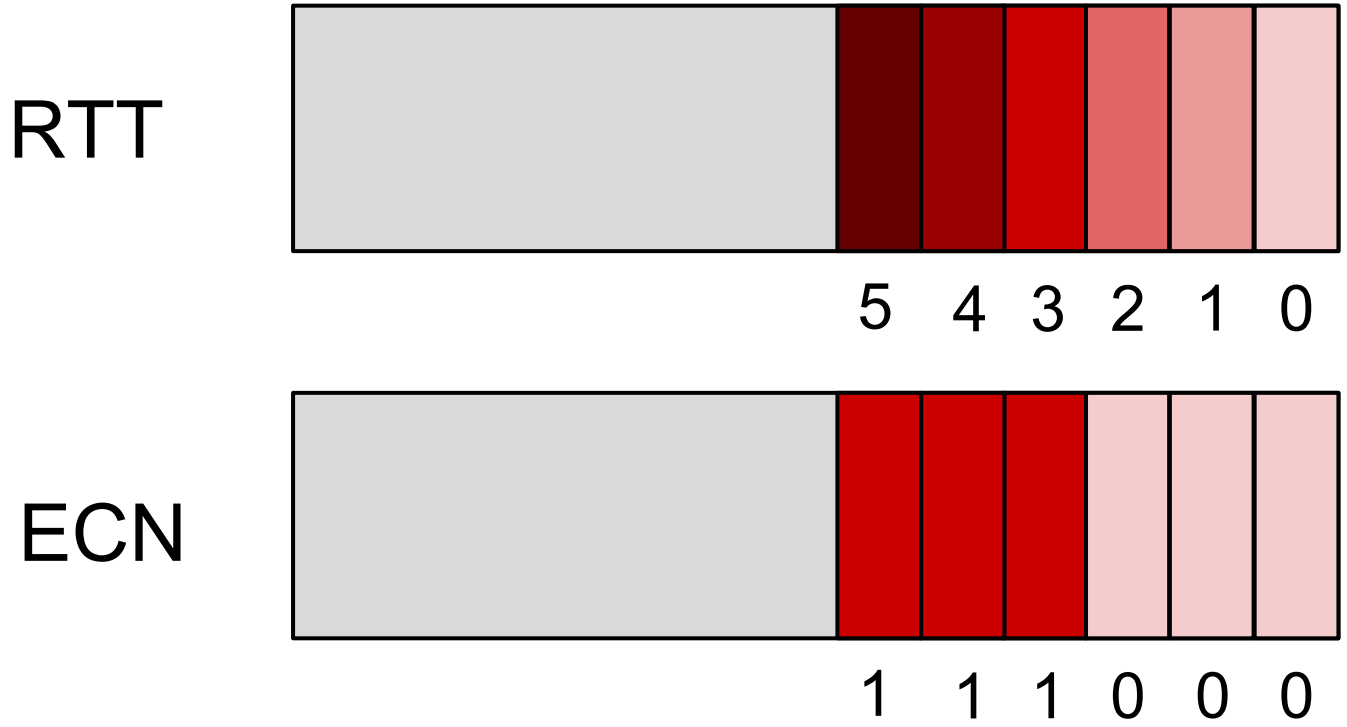
Hardware vs Software Timestamps



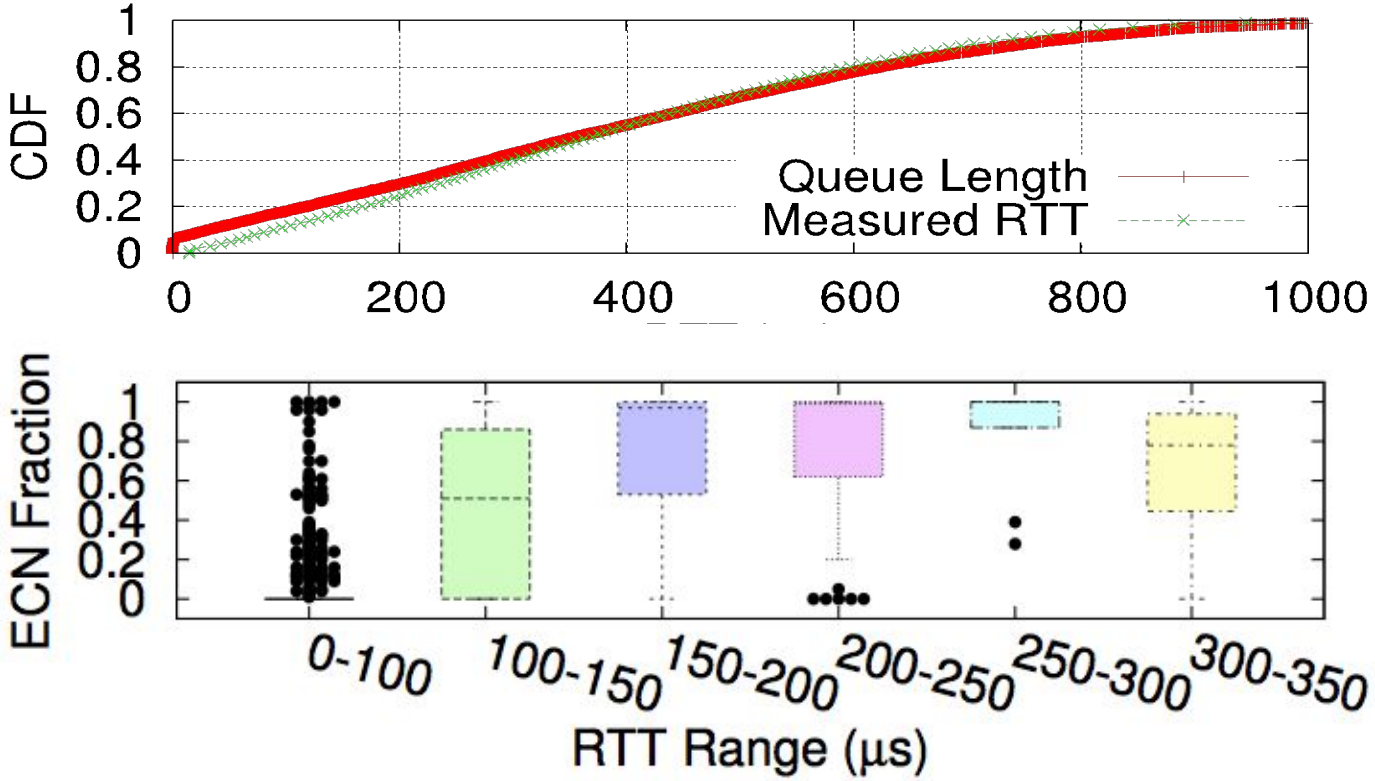
Kernel Timestamps introduce significant noise in RTT measurements compared to HW Timestamps.

RTT as a congestion signal

RTT is a multi-bit signal



RTT correlates with Queuing Delay

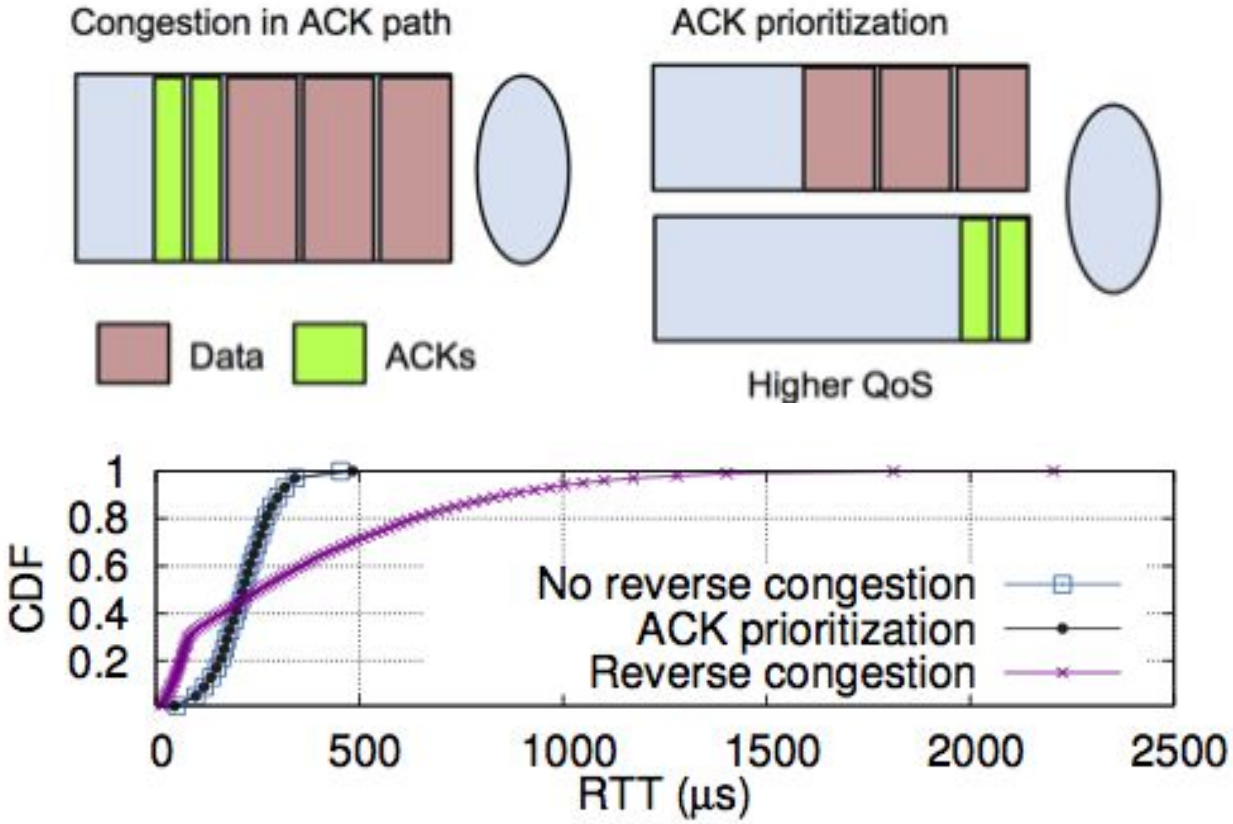


Limitations of RTT

RTT Limitations

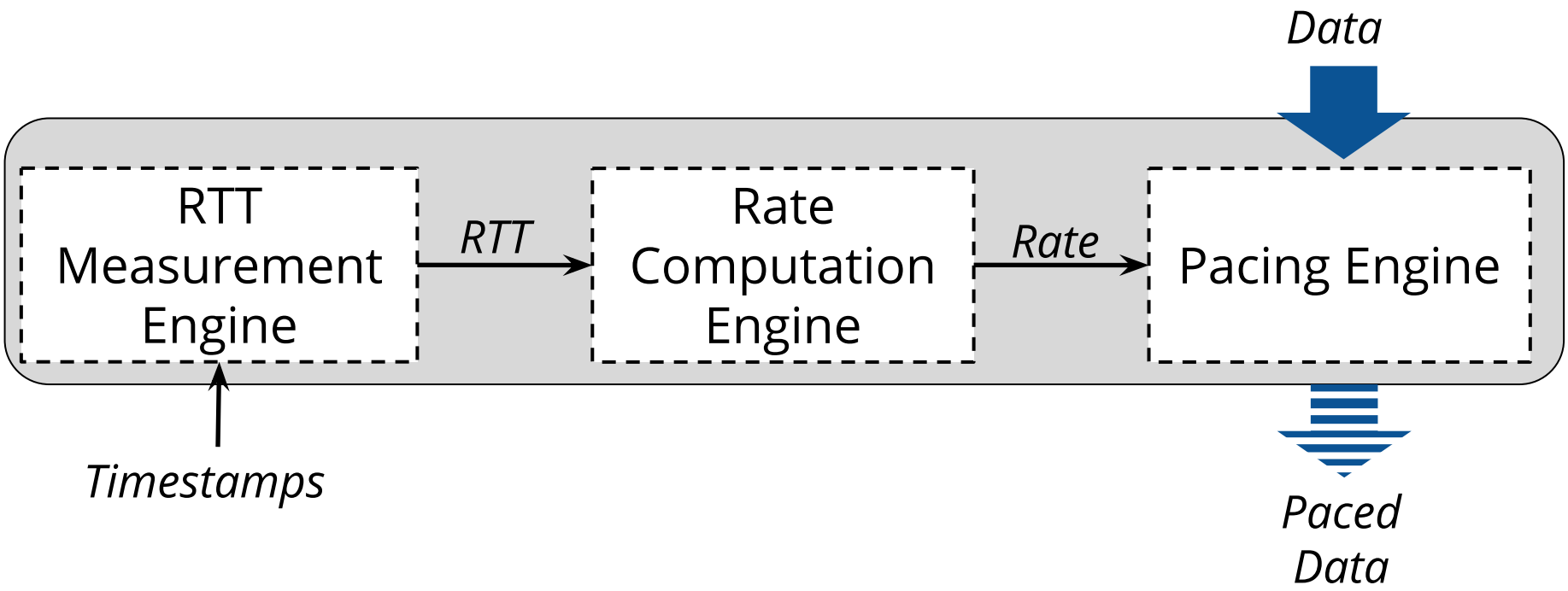
- RTT is a lumped signal
  - Confuses reverse and forward path congestion
- Changing network paths can cause disparate delays

ACK Prioritization

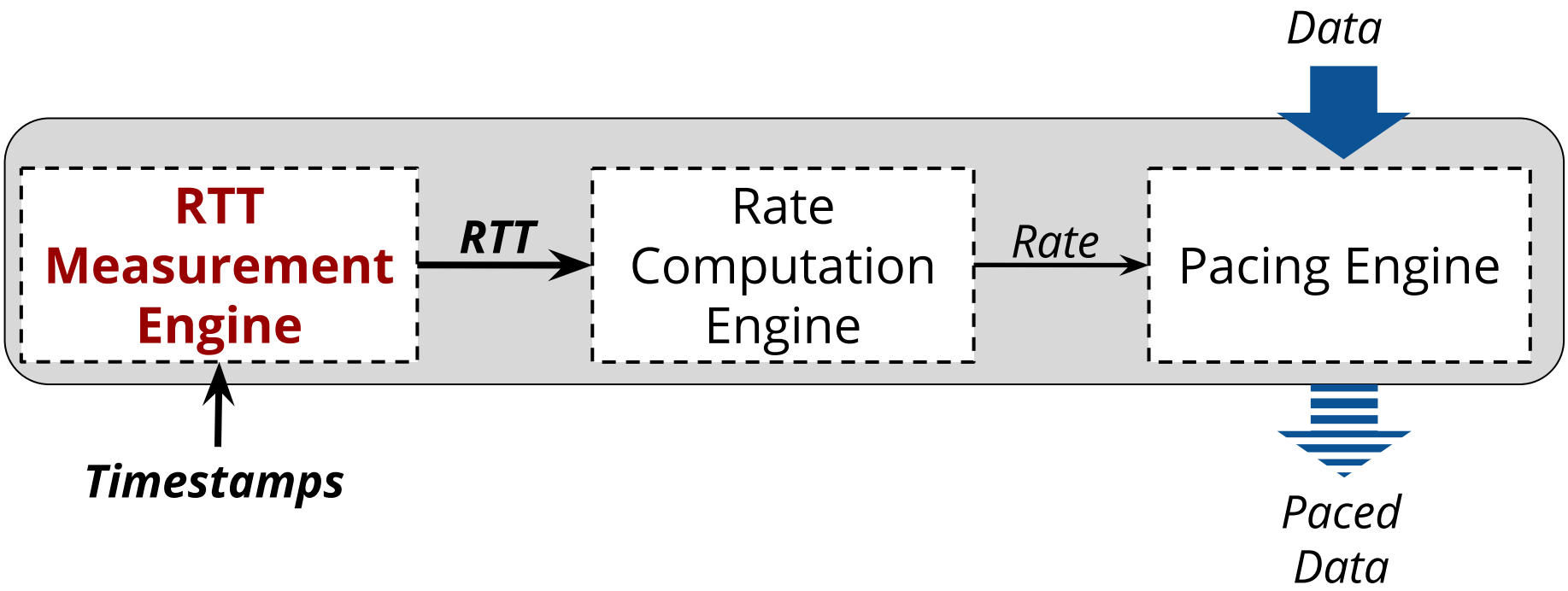


TIMELY Framework

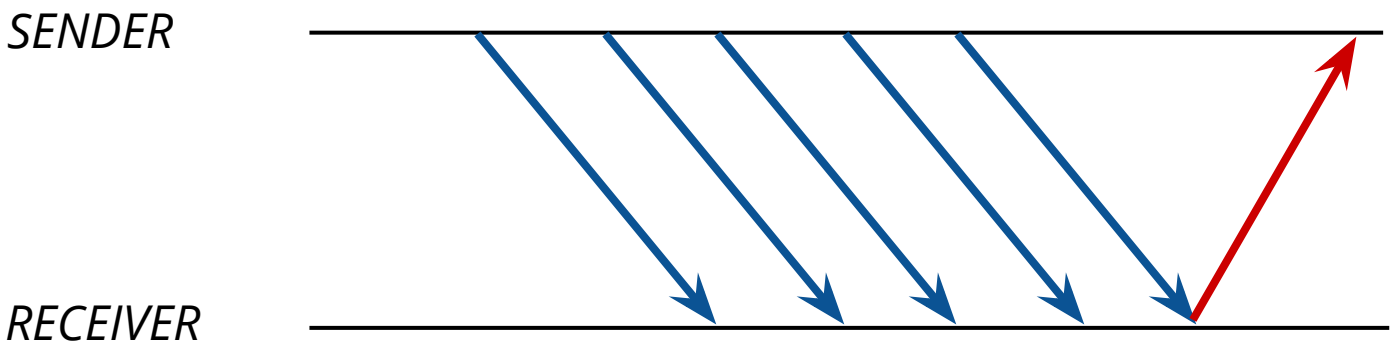
Overview


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

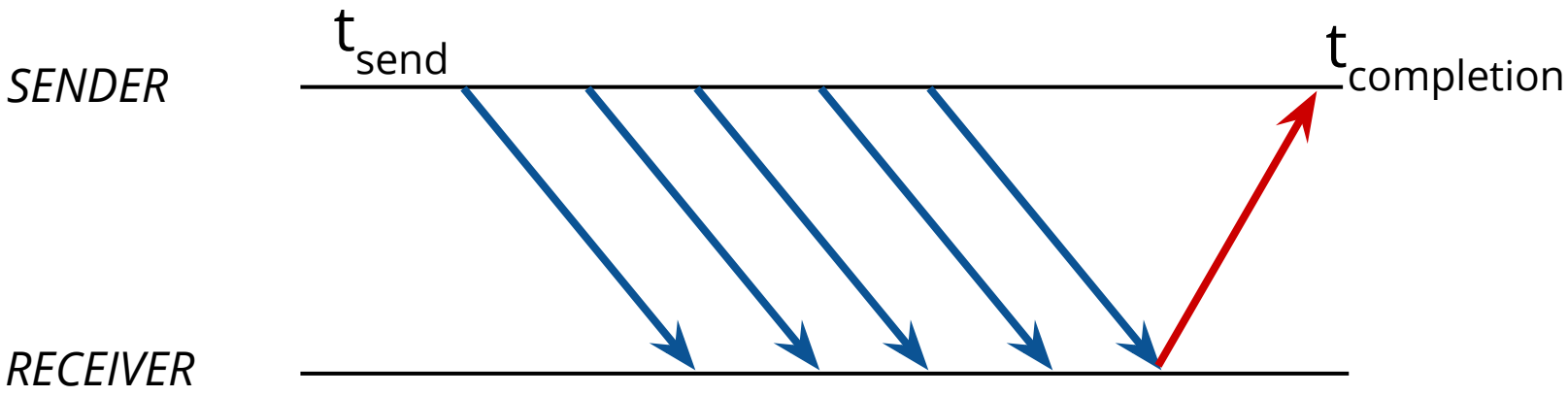
Overview


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

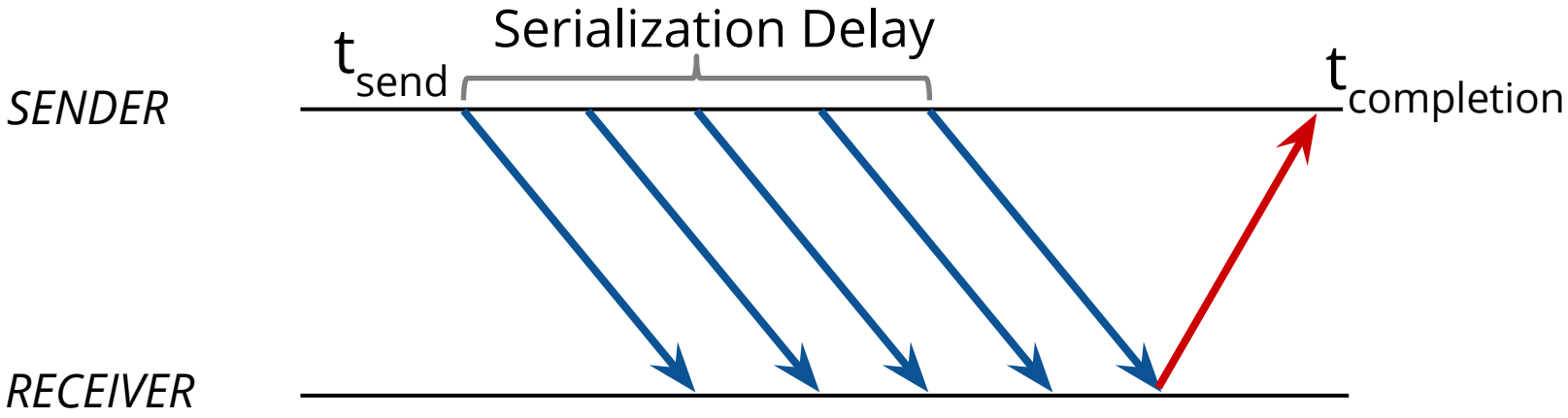
RTT Measurement Engine



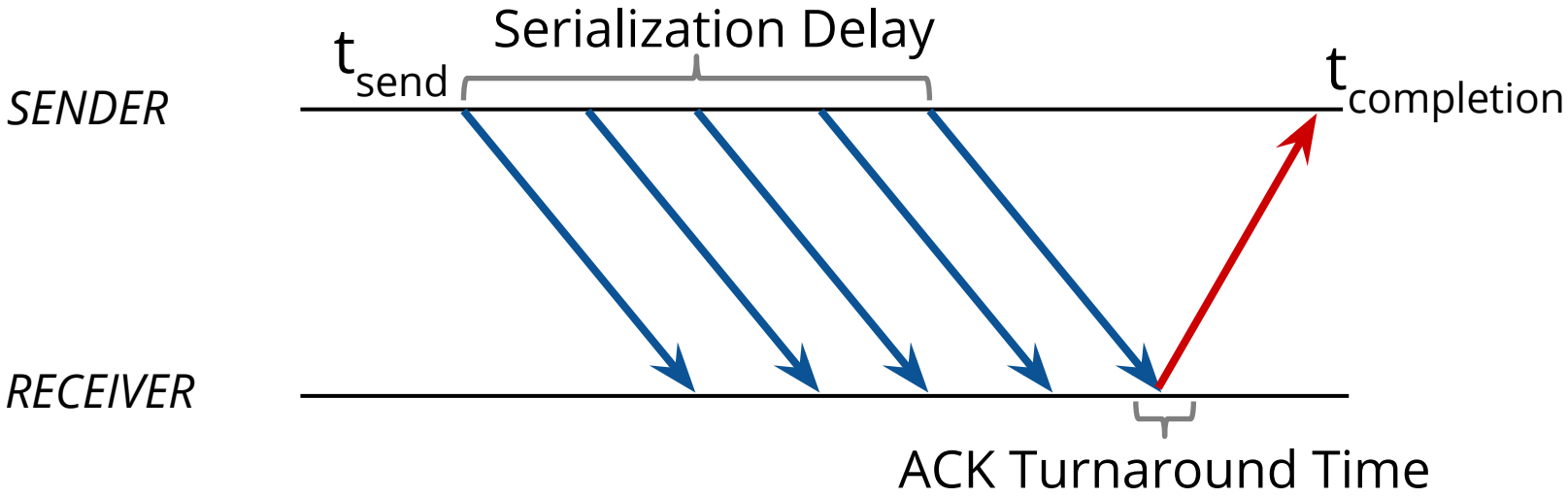
RTT Measurement Engine



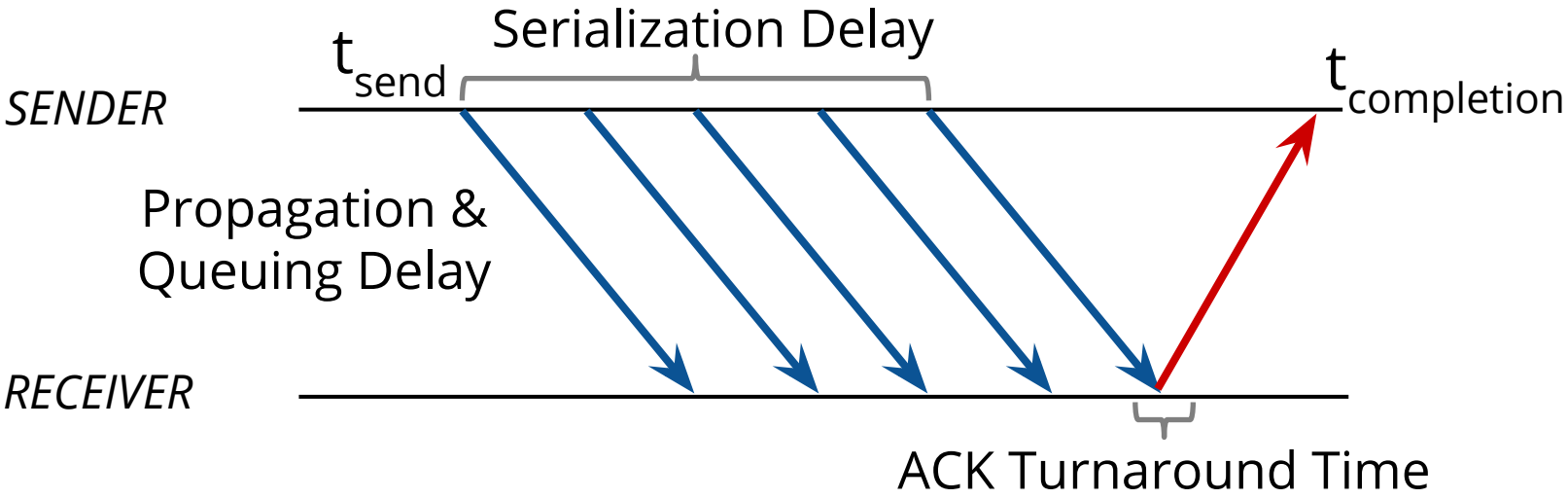
RTT Measurement Engine



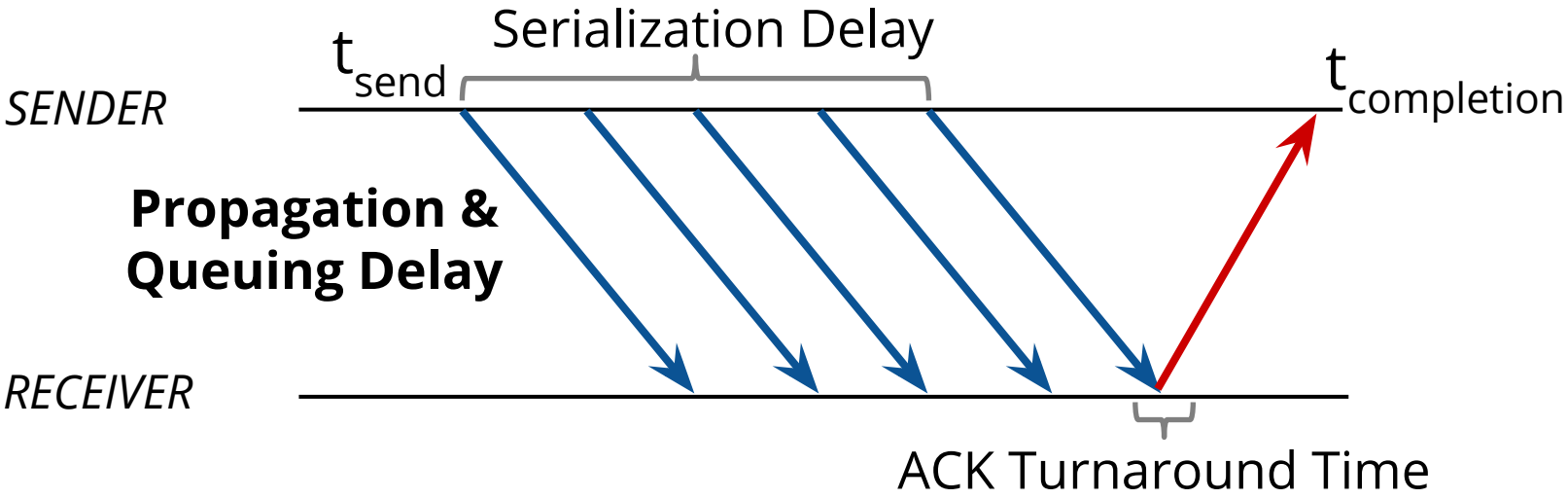
RTT Measurement Engine



RTT Measurement Engine

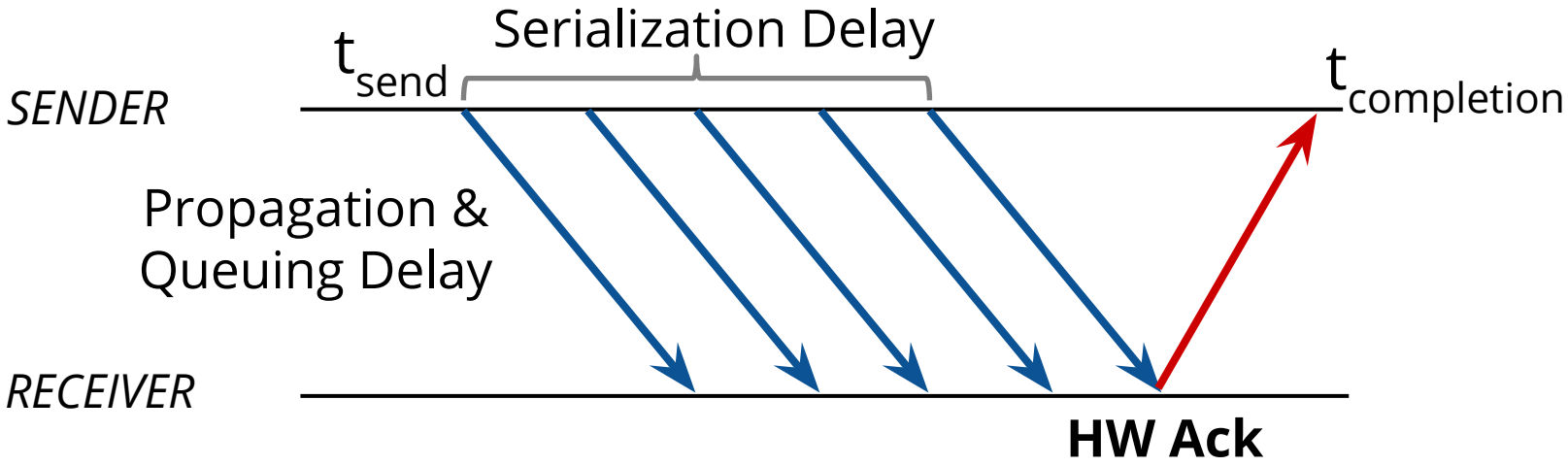


RTT Measurement Engine



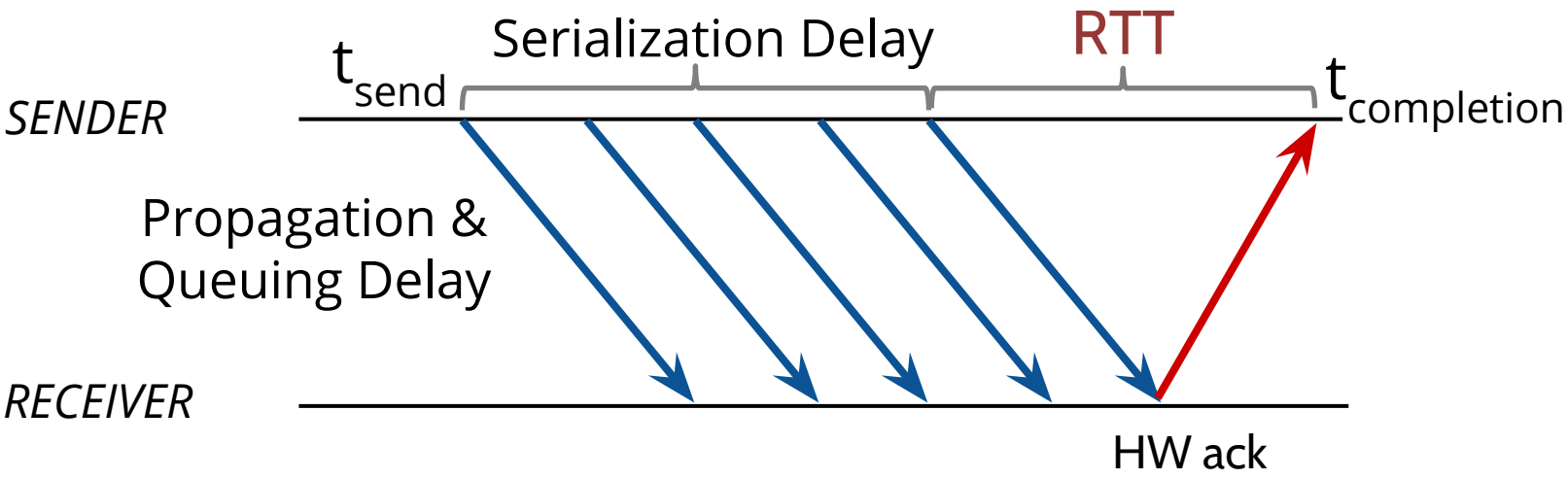
RTT = Propagation & Queuing Delay

RTT Measurement Engine



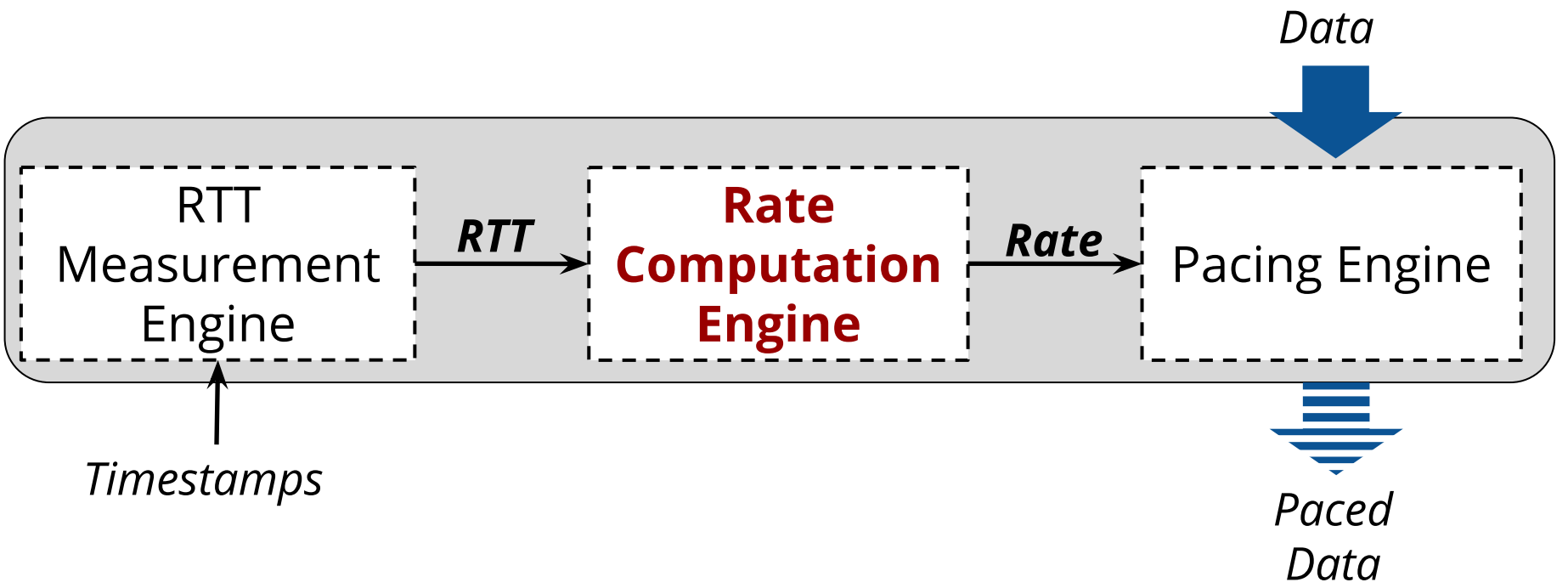
RTT = Propagation + Queuing Delay

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

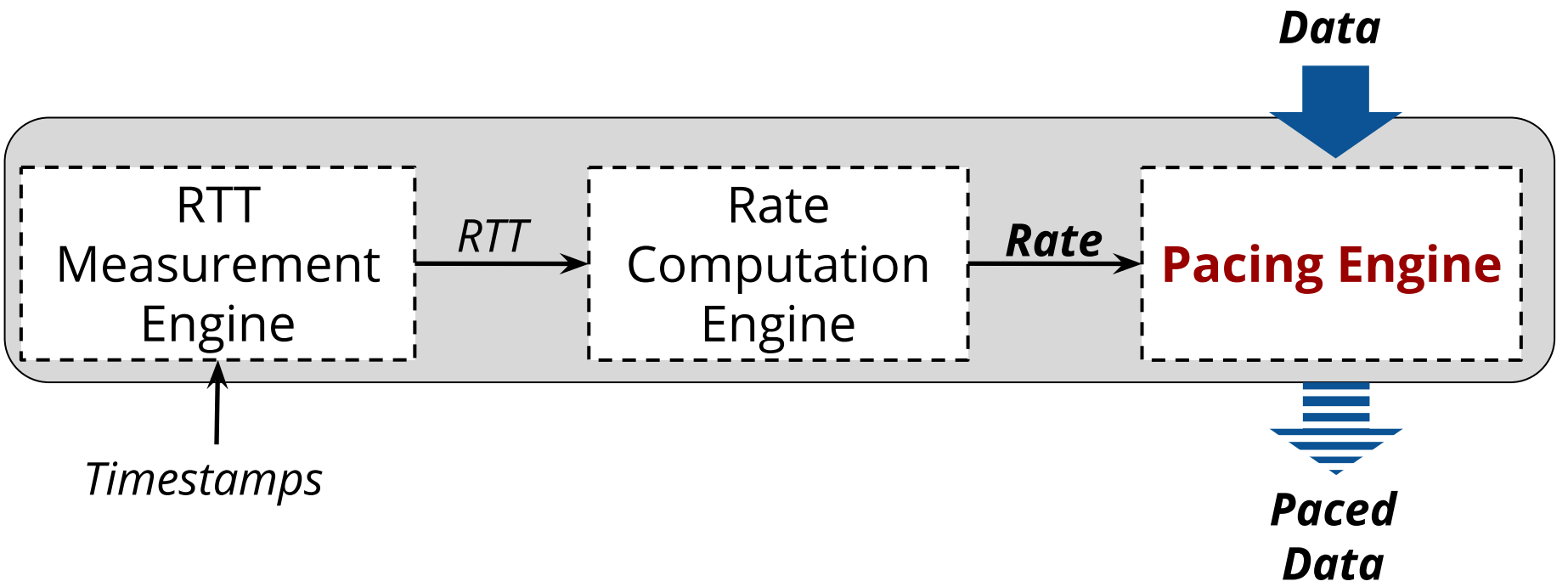
Rate Computation Engine

- 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

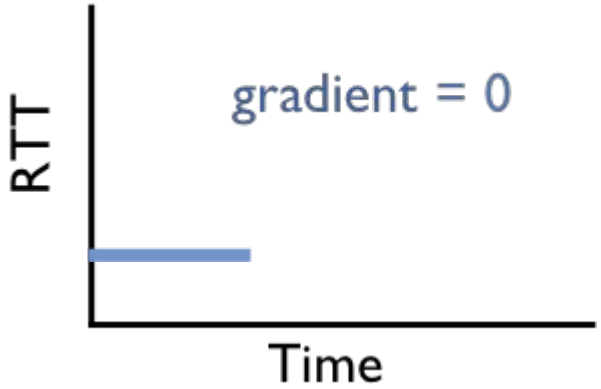
- Flow Completion Time
  - Large Flows: High Throughput
  - Short Flows: Low tail latencies
- Ride the throughput-latency curve
  - until tail latencies become unacceptable
  - low latency prioritized over throughput
- Fairness and Stability

Challenges

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

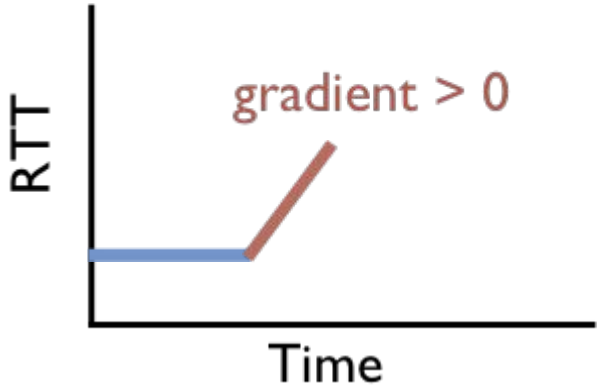
Algorithm Overview

Gradient-based Increase / Decrease



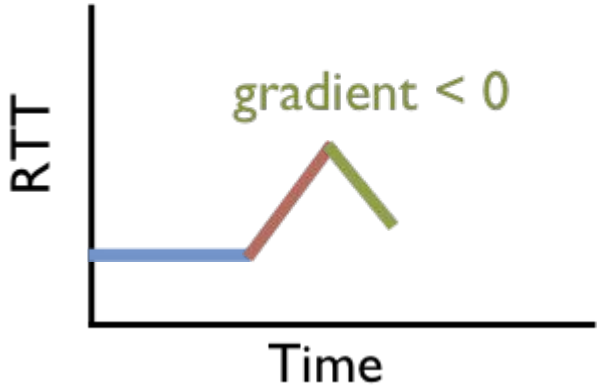
Algorithm Overview

Gradient-based Increase / Decrease



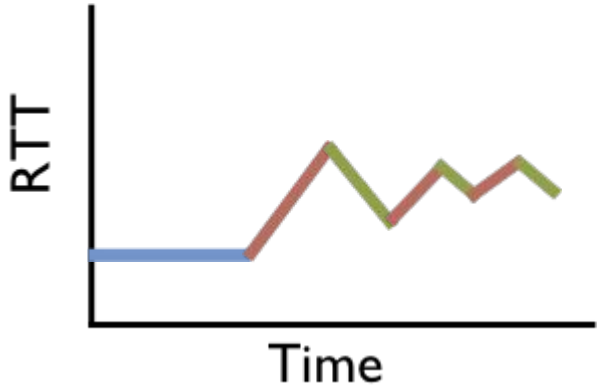
Algorithm Overview

Gradient-based Increase / Decrease

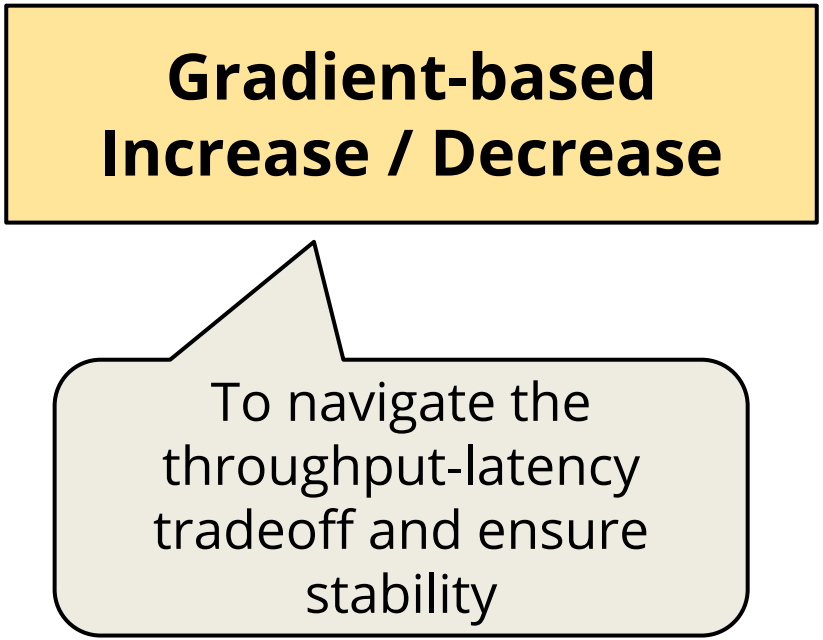


Algorithm Overview

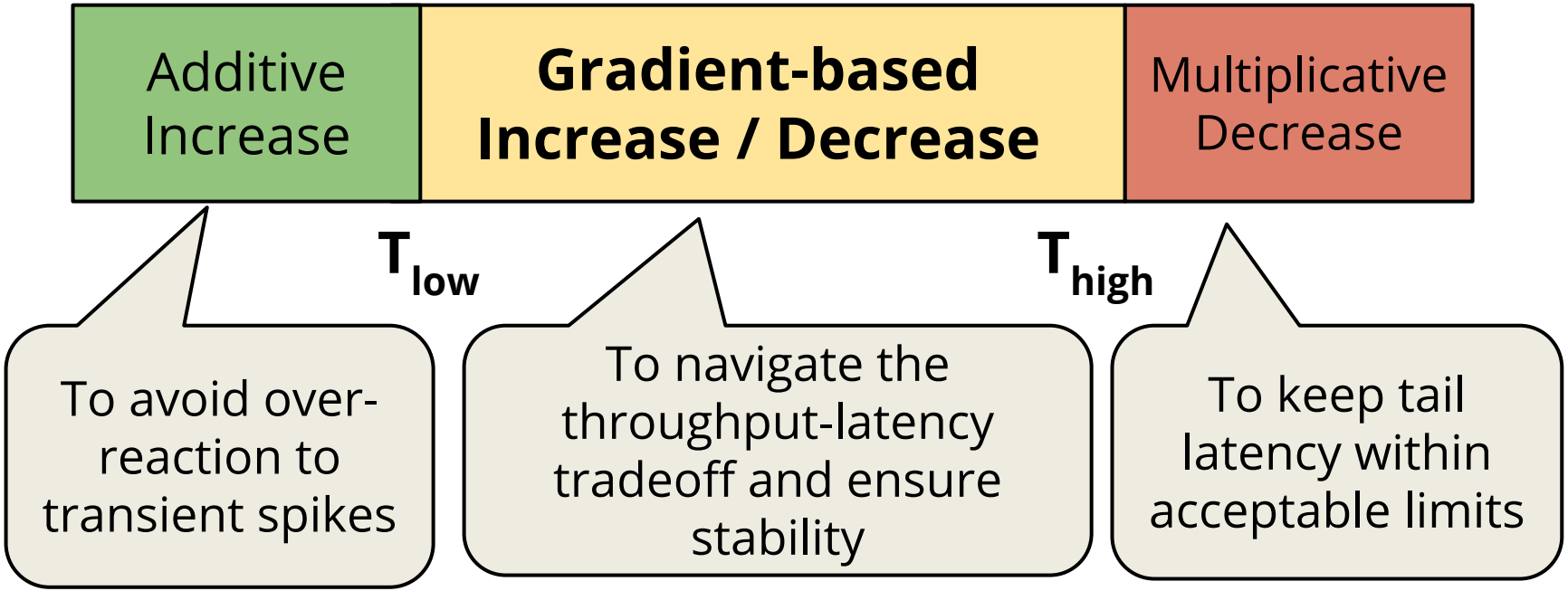
Gradient-based Increase / Decrease



Algorithm Overview



Algorithm Overview



Evaluation

Implementation Set-up

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

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

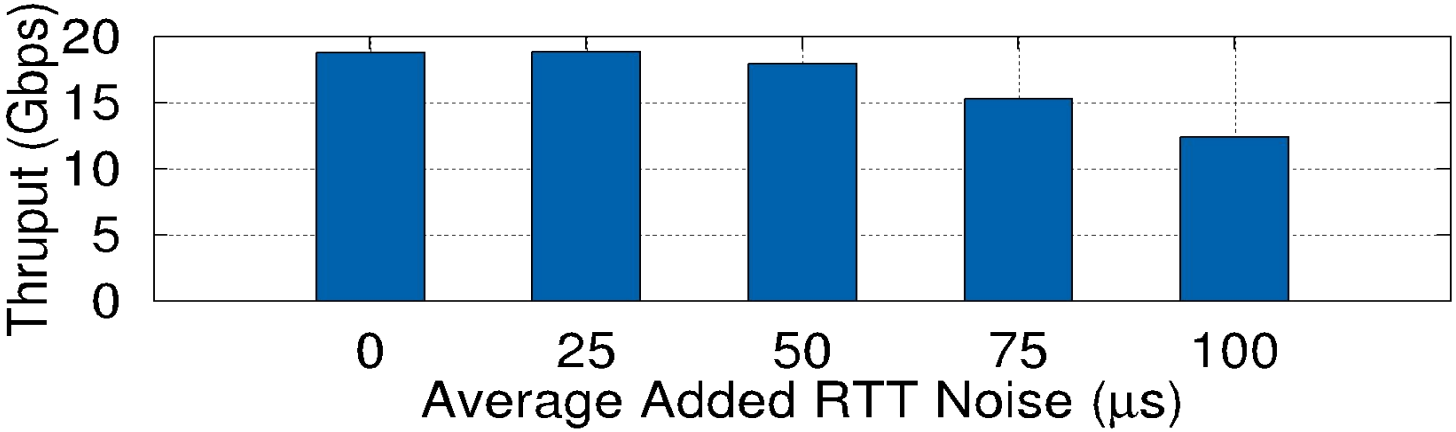
Experiments Set-up

- Small-scale experiments:
  - In-cast 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 DCTCP

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

Conclusion

- Conventional wisdom considers delay to be noisy signal for DC
  - Experience with TIMELY shows the opposite.

- TIMELY detects and react to 10s of us of delay.

- Open Question: Effectiveness of RTT as DC speeds scale up by order of magnitude; buffer sizes shrink.

Back-up

Hurdles for RTT

	Wide Area Networks	Datacenters
Coexistence	Competes poorly with loss-based schemes	N/A
Measurement	Inaccuracies due to varying paths	Too hard to measure at microsecond granularity

Hurdles for RTT

	Wide Area Networks	Datacenters
Coexistence	**Competes poorly with loss-based schemes**	N/A
Measurement	Inaccuracies due to varying paths	Too hard to measure at microsecond granularity

Hurdles for RTT

	Wide Area Networks	Datacenters
Coexistence	Competes poorly with loss-based schemes	N/A
Measurement	**Inaccuracies due to varying paths**	Too hard to measure at microsecond granularity

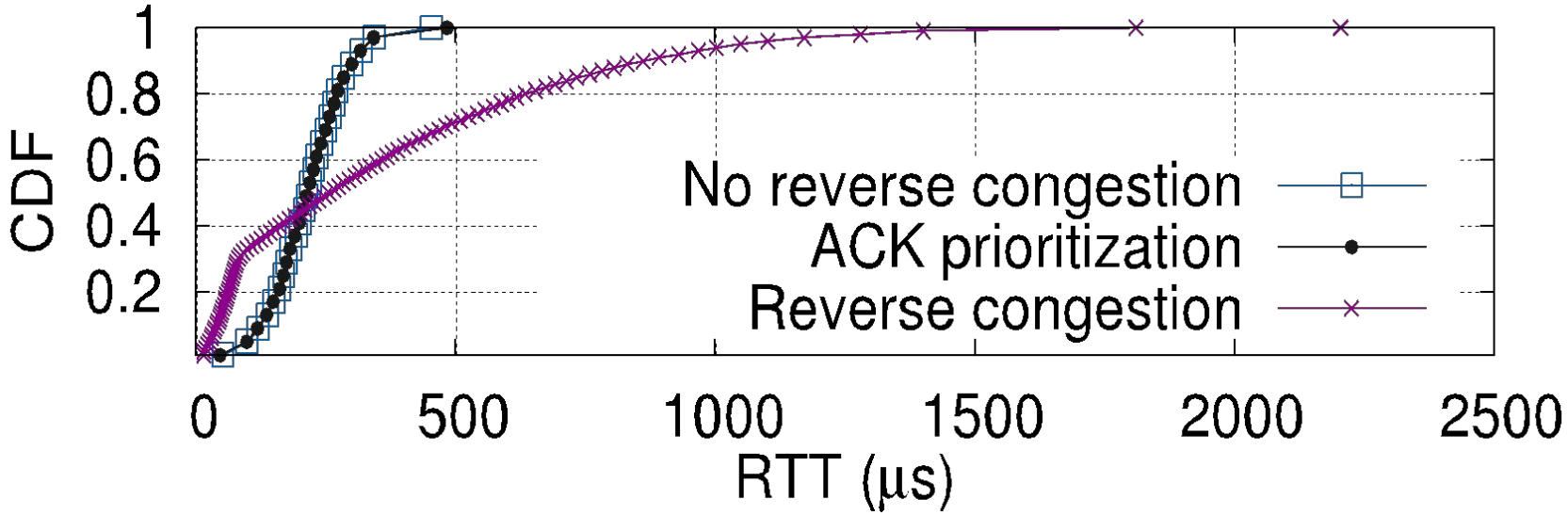
Hurdles for RTT

	Wide Area Networks	Datacenters
Coexistence	Competes poorly with loss-based schemes	**N/A**
Measurement	Inaccuracies due to varying paths	Too hard to measure at microsecond granularity

Hurdles for RTT

	Wide Area Networks	Datacenters
Coexistence	Competes poorly with loss-based schemes	N/A
Measurement	Inaccuracies due to varying paths	***Too hard to measure at microsecond granularity***

Excluding Reverse Path Congestion



RTT is a holistic signal

RTT	ECN
• Total queuing across multiple hops	• Queuing at a single hop
• Includes queuing at the endhost	• Excludes queuing at the endhost
• Total delay seen by low priority packets	• Only the occupancy of low priority queue

Pacing Engine

- Sends segments to the NIC for transmission
- Uses a single scheduler
- Segments with elapsed *send times* serviced using round robin
- Segments with future *send times* queued up

Rate-Based Protocol

- Why not set a window of outstanding bytes?
  - Segment sizes as high as 64KB
  - (16us RTT x 10Gbps) = 20KB window size
  - 20KB < 64KB : Window makes no sense

Reaction to Delay Gradient

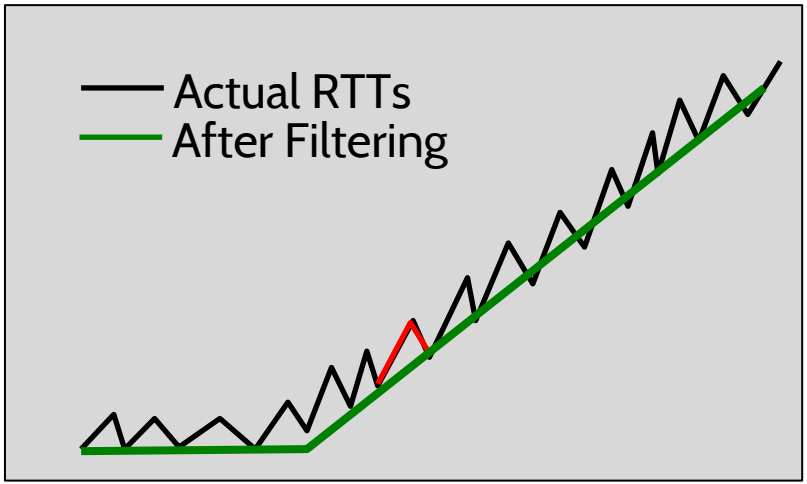
- Why not absolute delay?
  - Bursty traffic and small propagation delays
  - Absolute delay not a stable signal
- Why gradient?
  - allows prompt detection of *rising and falling queue*
  - is a direct proxy for rate-mismatch

TIMELY Algorithm : Basic

Step 1: Compute smoothed delay gradient

Difference in previous and new RTT passed through EWMA filter

Result divided by a fixed minRTT value



TIMELY Algorithm : Basic

Step 1: Compute smoothed delay gradient

Difference in previous and new RTT passed through EWMA filter

Result divided by a fixed minRTT value

Step 2: Compute New Rate

If (gradient ≤ 0): rate = rate + δ

Else: rate = rate . (1 – β . gradient)

AIMD ensures fairness across flows.

TIMELY Algorithm : For Transient Spikes

Step 1: Compute smoothed delay gradient

Difference in previous and new RTT passed through EWMA filter
Result divided by a fixed minRTT value

Step 2: Compute New Rate

If($rtt < T_{low}$) : rate + δ
return

Avoids reaction to transient spikes

If (gradient $\leq$ 0) : rate = rate + δ
Else: rate = rate . (1 – β . gradient)

TIMELY Algorithm : Adding a Safety Net

Step 1: Compute smoothed delay gradient

Difference in previous and new RTT passed through EWMA filter
Result divided by a fixed minRTT value

Step 2: Compute New Rate

If($rtt < T_{low}$) : rate + δ
return

If($rtt > T_{high}$) : rate = rate . $(1 - \beta (1 - T_{high}/rtt))$
return

Safety Net

If (gradient ≤ 0) : rate = rate + δ
Else: rate = rate . $(1 - \beta$. gradient)

TIMELY Algorithm : Hyperactive Increment

Step 1: Compute smoothed delay gradient

Difference in previous and new RTT passed through EWMA filter
Result divided by a fixed minRTT value

Step 2: Compute New Rate

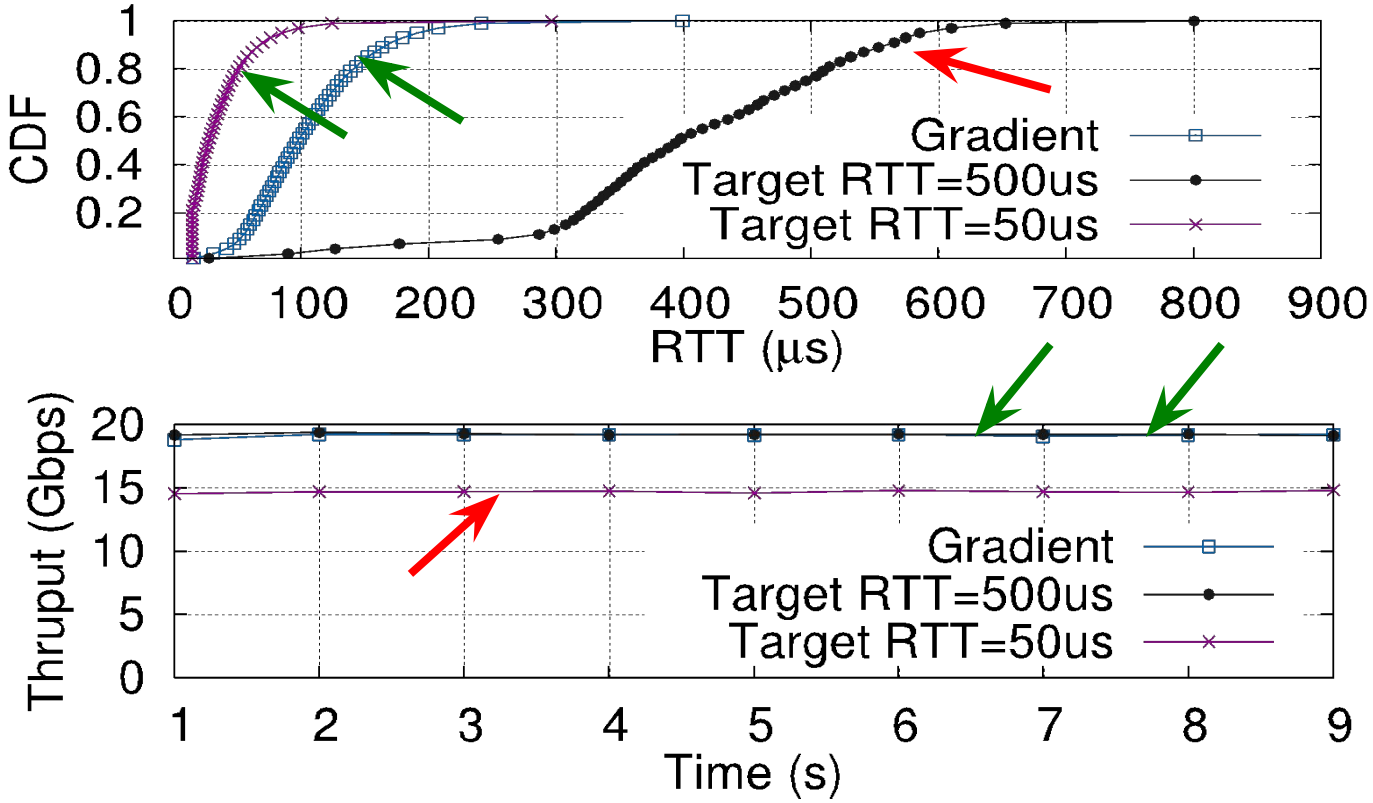
If($rtt < T_{low}$) : rate + δ
return

If($rtt > T_{high}$) : rate = rate . $(1 - \beta (1 - T_{high}/rtt))$
return

If (gradient ≤ 0) : rate = rate + N. δ
Else: rate = rate . $(1 - \beta$. gradient)

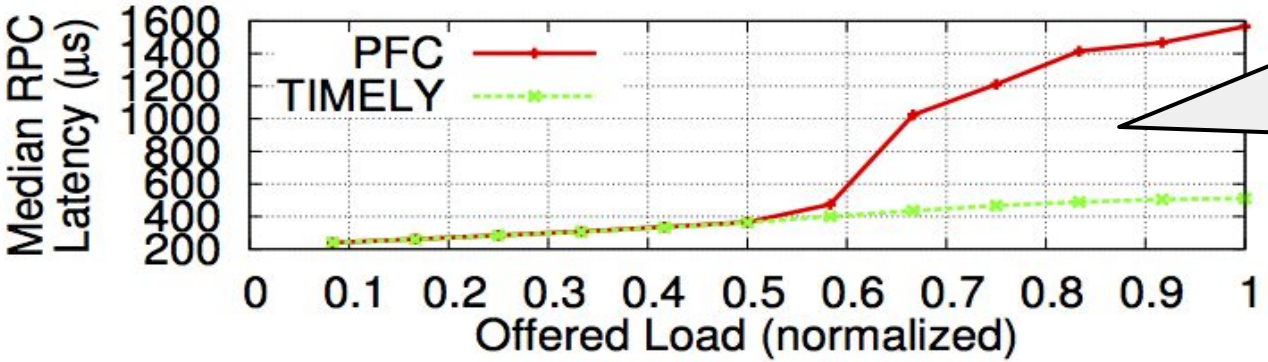
If gradient < 0 for 5 consecutive events:
N = 5
Else, N = 1

Gradient vs Absolute Delay

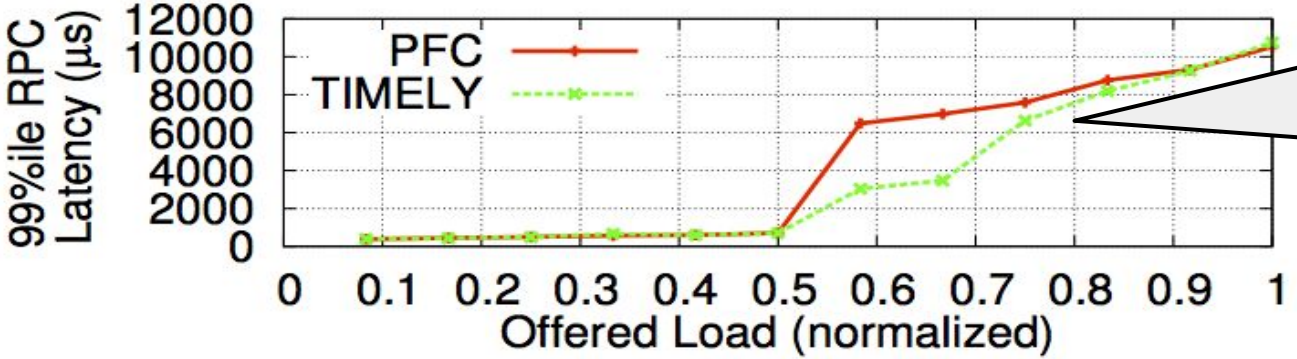


Delay Gradient achieves high throughput with low latencies

Comparison with PFC - Large Scale



Reduction in median RPC latencies increase with increasing load.



Reduction in tail RPC latencies decrease with increasing load.