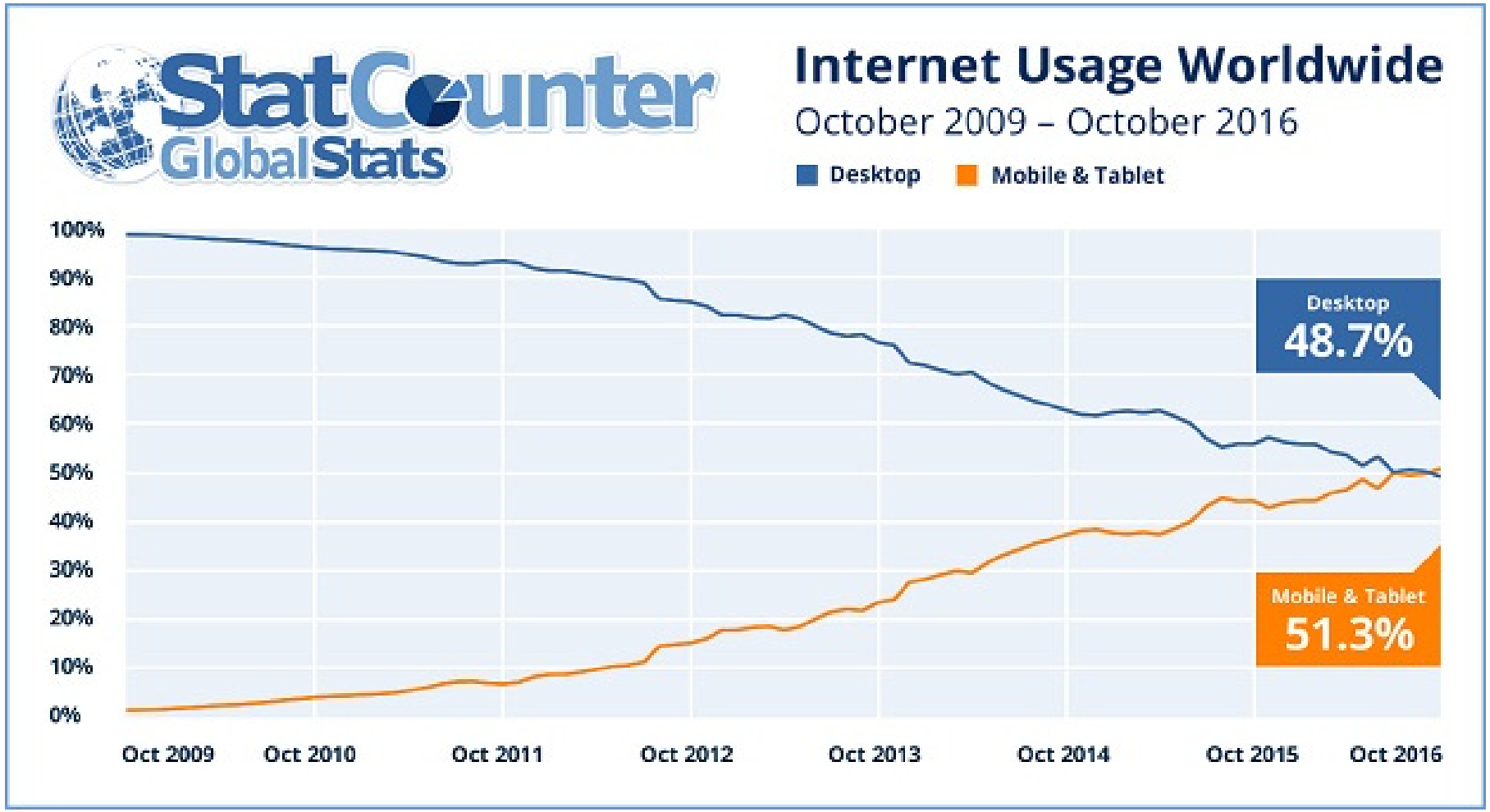
NUS
National University
of Singapore

TCP Congestion Control
Beyond Bandwidth-Delay Product
for Mobile Cellular Networks

Wai Kay Leong, Zixiao Wang, Ben Leong

The 13th International Conference on emerging Networking EXperiments and Technologies

Mobile Internet usage exceeds Desktop



What's different about cellular?

Negligible random packet losses

- Hybrid ARQ scheme
- As compared to 802.11 Wi-Fi

Large buffers

- In the Megabytes

Asymmetric uplink/downlink

- ACK delay

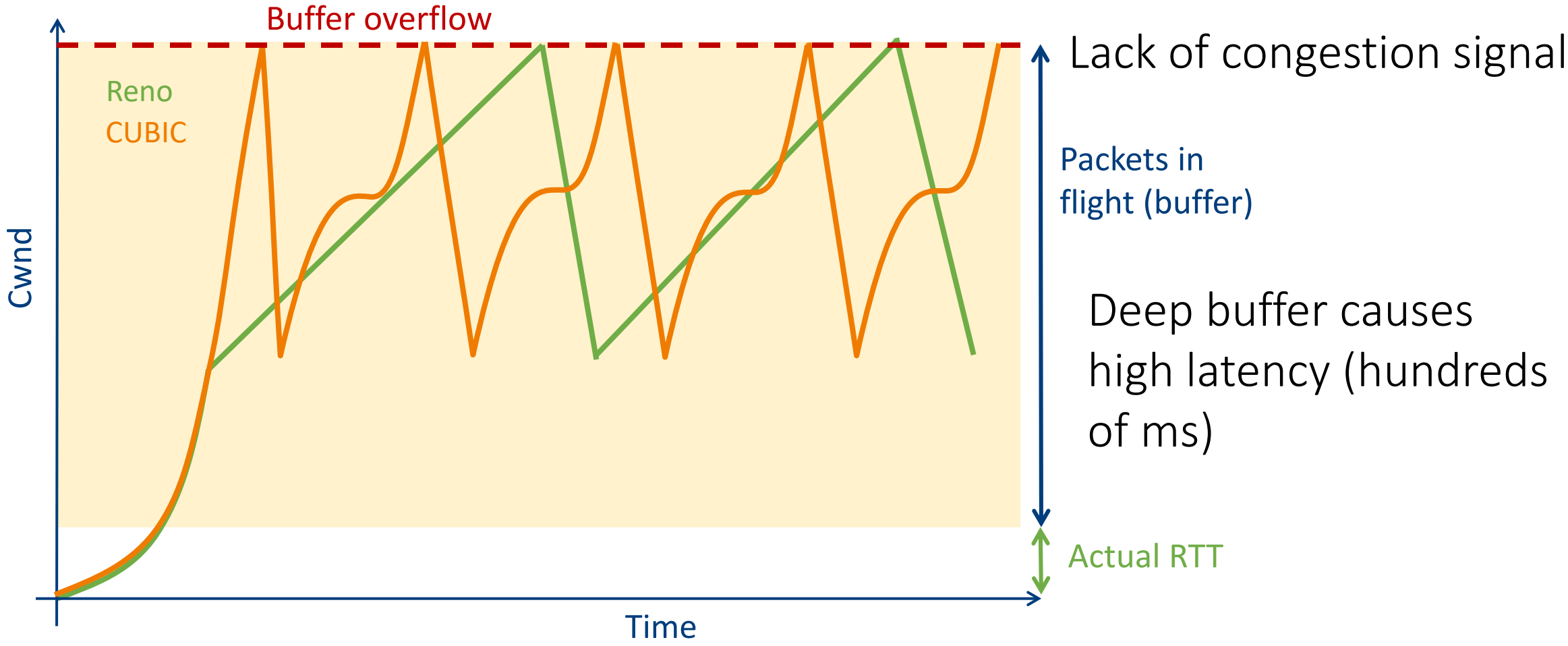
Fair scheduling at “base station”

- No contention with other users

Why Traditional TCP does not work

IN MOBILE/CELLULAR NETWORKS

1. Large/Deep buffers



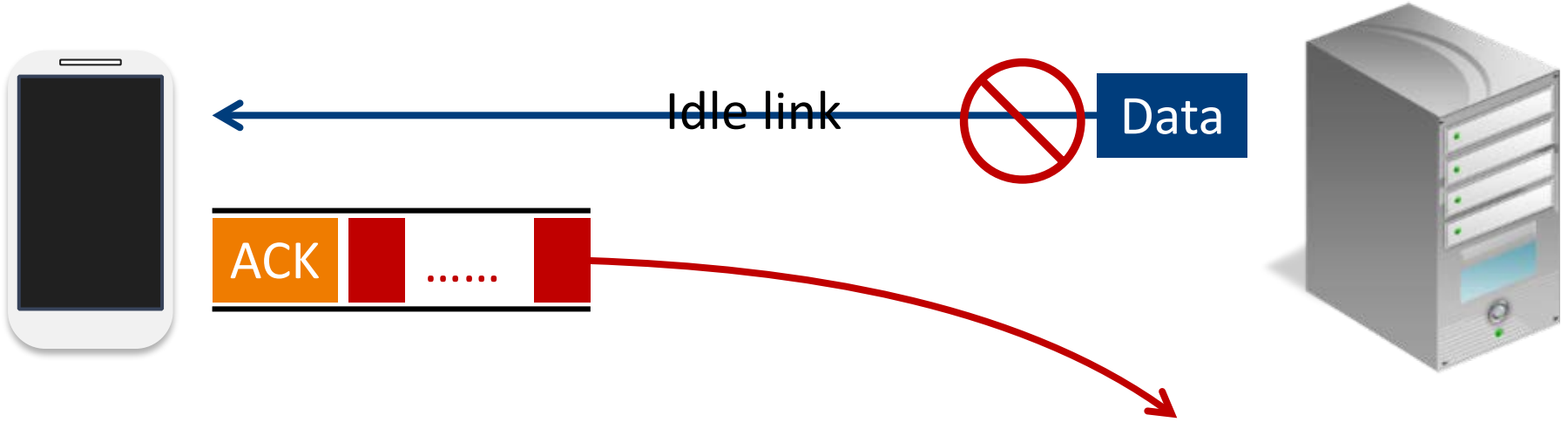
Deep buffer causes high latency (hundreds of ms)

2. Uplink Congestion

More predominant in slower 3G/HSPA networks

ACK gets delayed in return uplink

- Stuck in deep buffer/ high volume of users
- Server is prevented from sending new packet even though downlink is clear



Rethink congestion control for mobile networks

Traditional TCP congestion control

- Lack of congestion signal (ECN not popular)
- Long delay/high latency (CUBIC)
- ACK clocked

Rise in new mobile TCP algorithms

- Sprout
- Verus
- PCC
- BBR

Key Idea

Purpose of congestion control is to

- avoid congestion
- finding the correct rate to send packets
- ideally keep 1×BDP packets in transit

Why not just send at the correct rate?

- Vary conditions of mobile networks
- Try to forecast the condition (Sprout, PR
- Try to build a model (PCC, Remy)

Our Insight:
Timely estimation of the bandwidth + quick reaction to new network condition is sufficient

Our Approach

Abandon ACK clocking

Pure rate-based sending of packets

1. Estimate current bandwidth/receive rate
2. Send packets at estimated rate
3. Observe buffer delay
4. Update send rate

Takes advantage of large buffer

Congestion with others mitigated by fair scheduling in base station

We need a means to

1. Estimate the bandwidth/receive rate
2. Detect congestion by measuring one-way delay

Make use of TCP timestamp option

- Enabled by default on most servers and phones

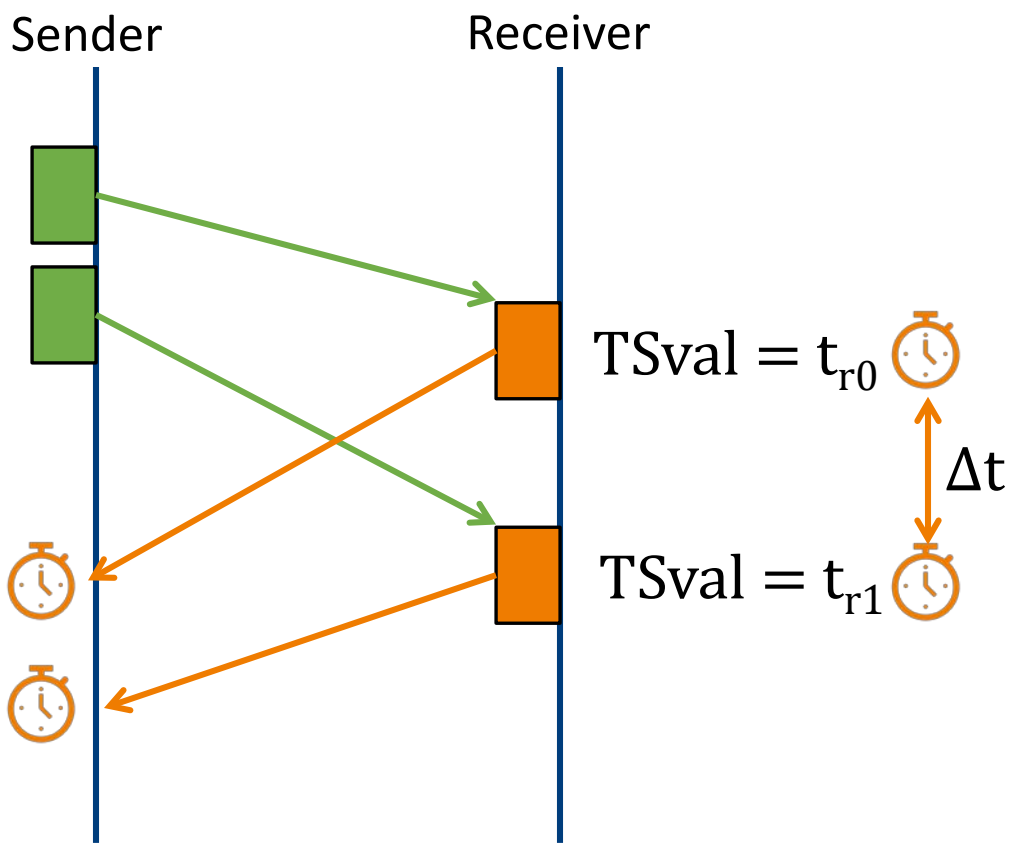
Estimating Receive Rate

Receiver will send ACK when packet is received

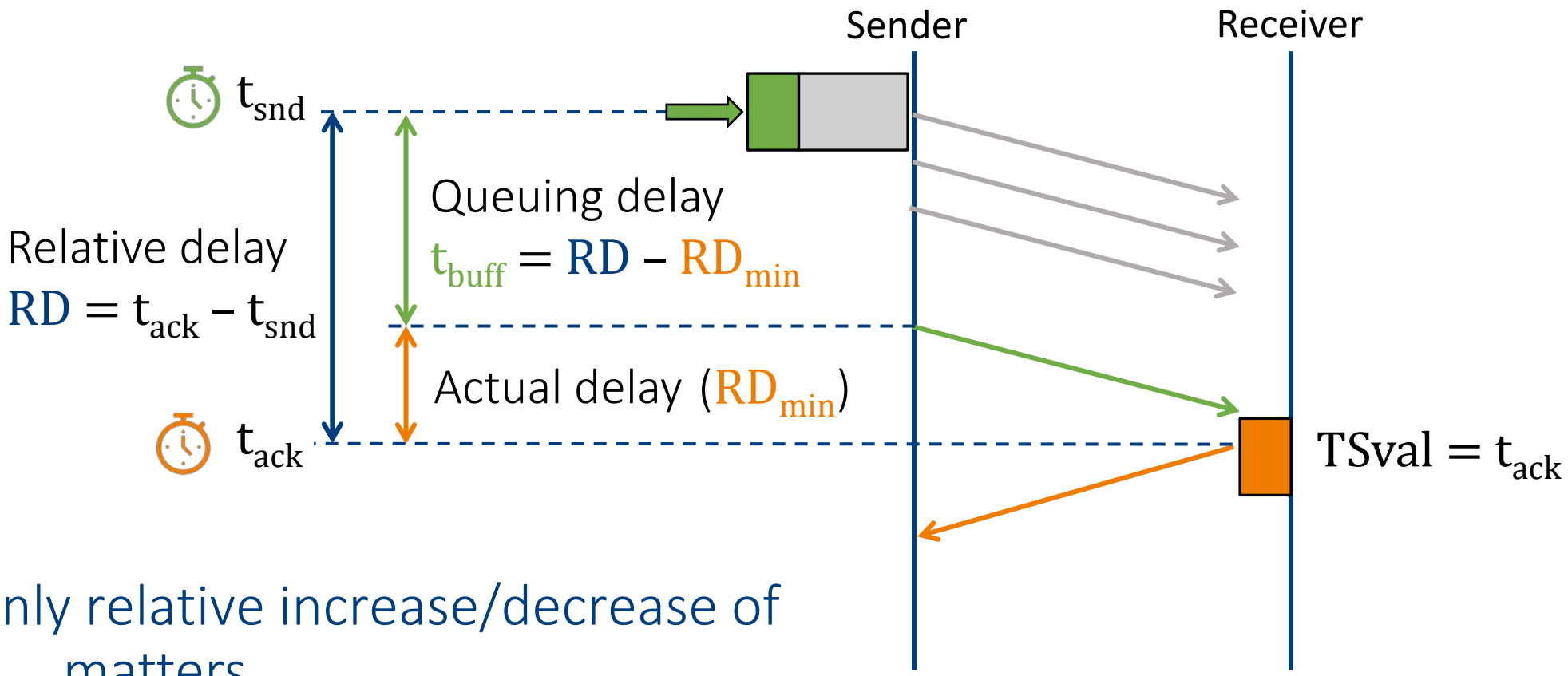
ACK will be timestamped

Compute rate by

- comparing timestamps: $t_{r1} - t_{r0} = \Delta t$
- and bytes ACK: $\Delta ACK / \Delta t = \rho$

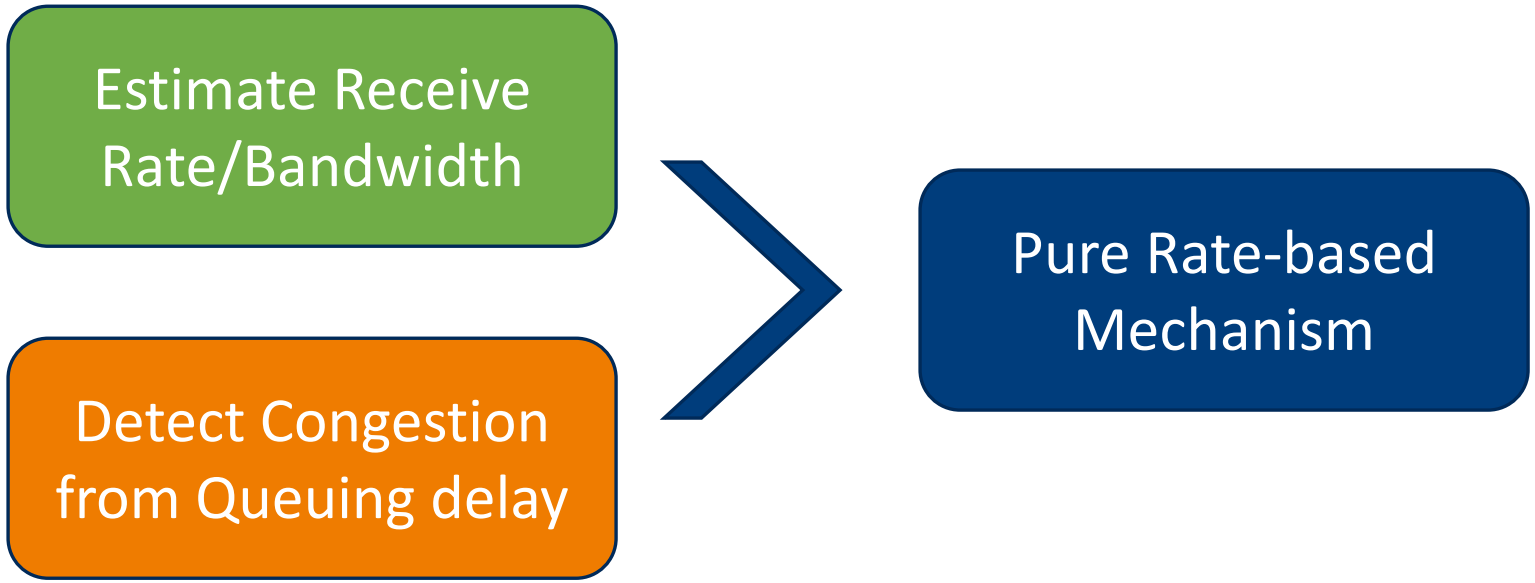


Estimating Buffer/Queuing Delay

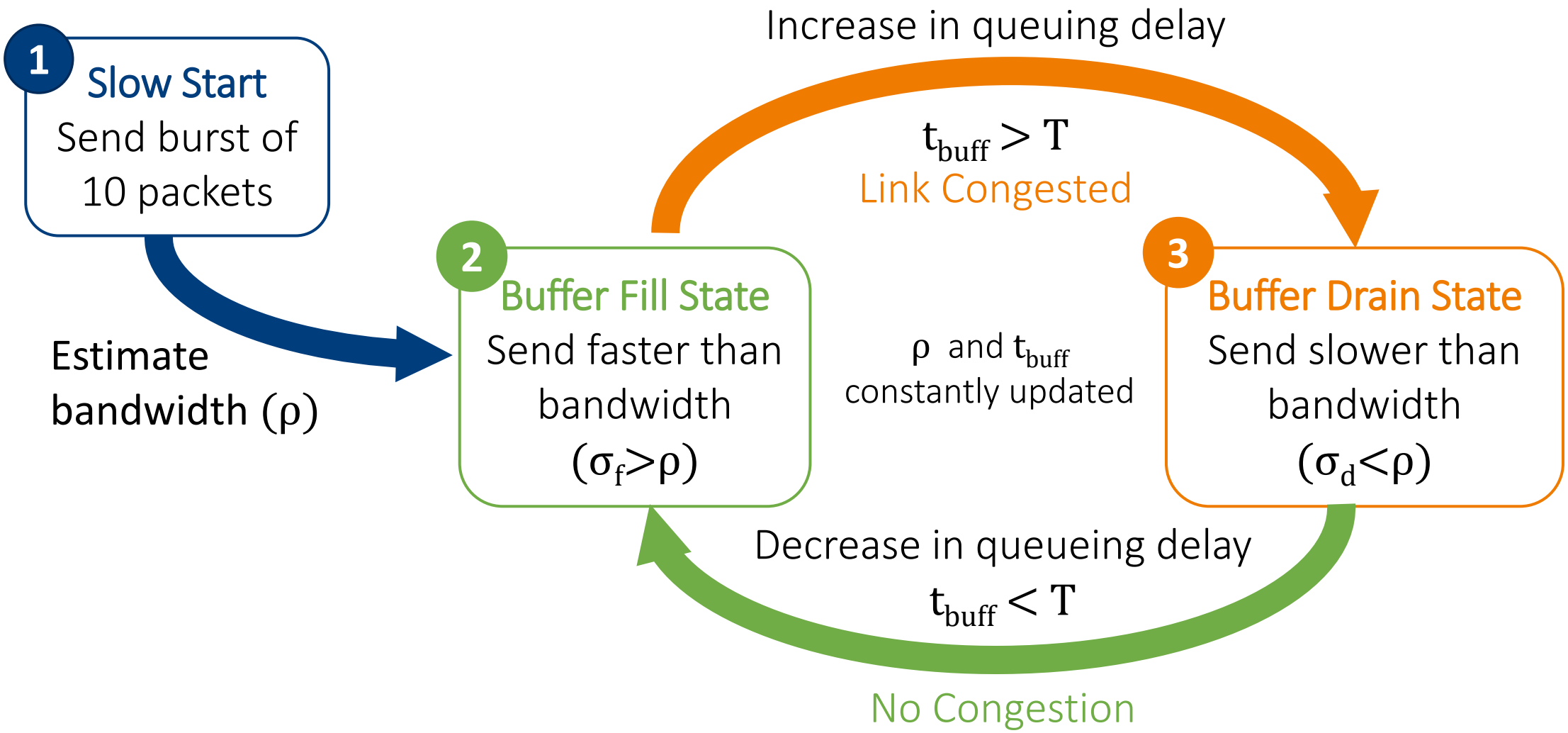


Only relative increase/decrease of t_{buff} matters

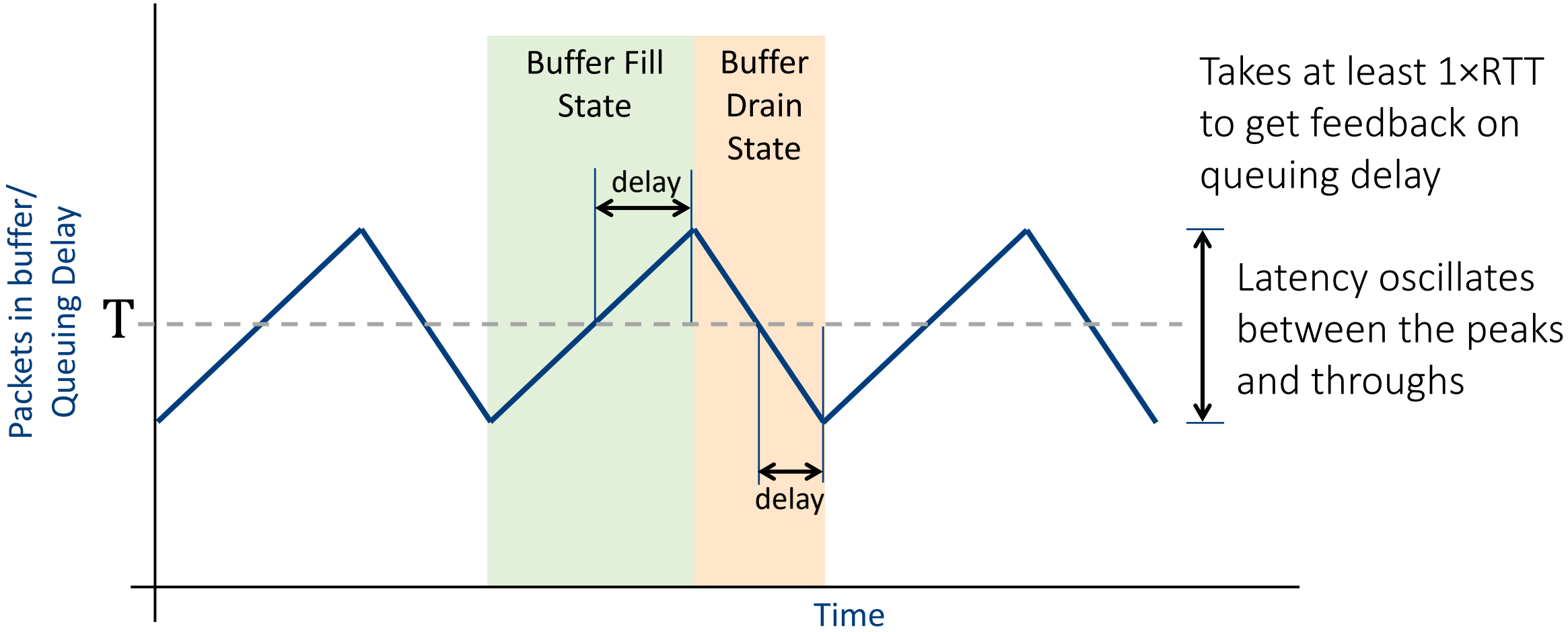
Putting it together



Self-oscillating feedback loop



Packet Trace, aka Sawtooth



PropRate

Sending rate is a proportion of bandwidth/receive rate

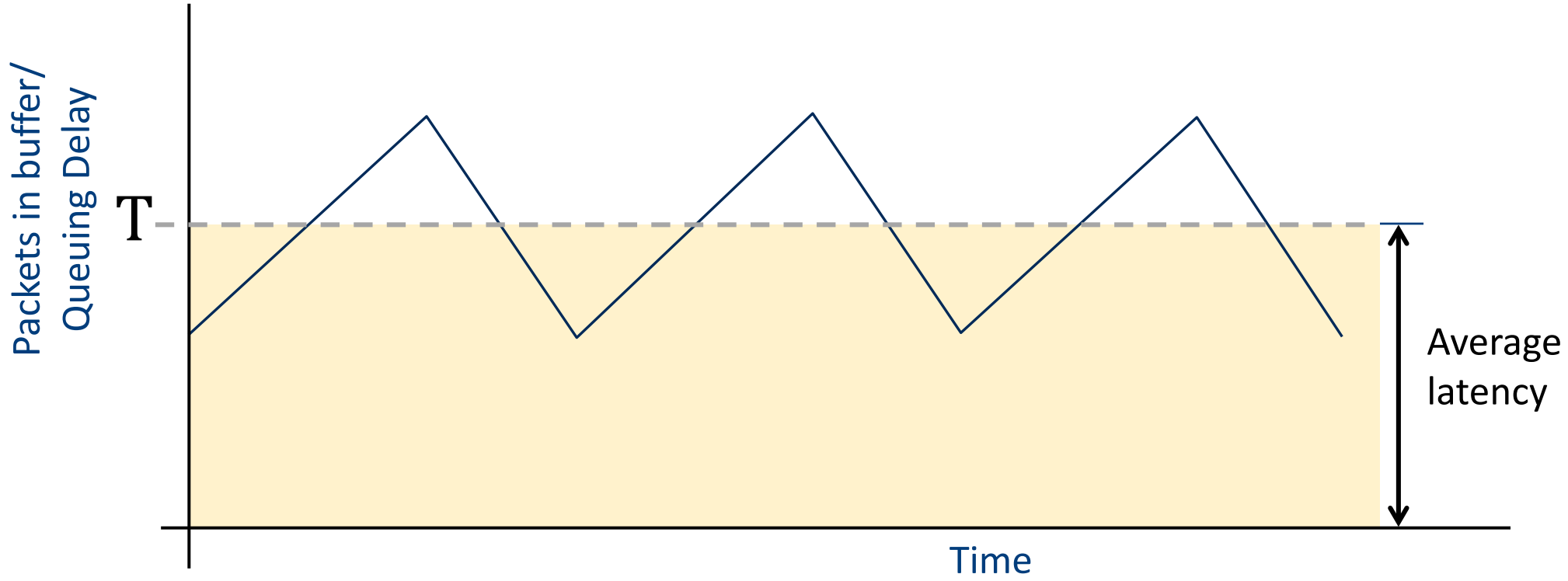
- $\sigma_f = k_f \rho$
- $\sigma_d = k_d \rho$

Three parameters controls the sawtooth

- k_f – proportion to fill buffer
- k_d – proportion to drain buffer
- T – threshold for switching state

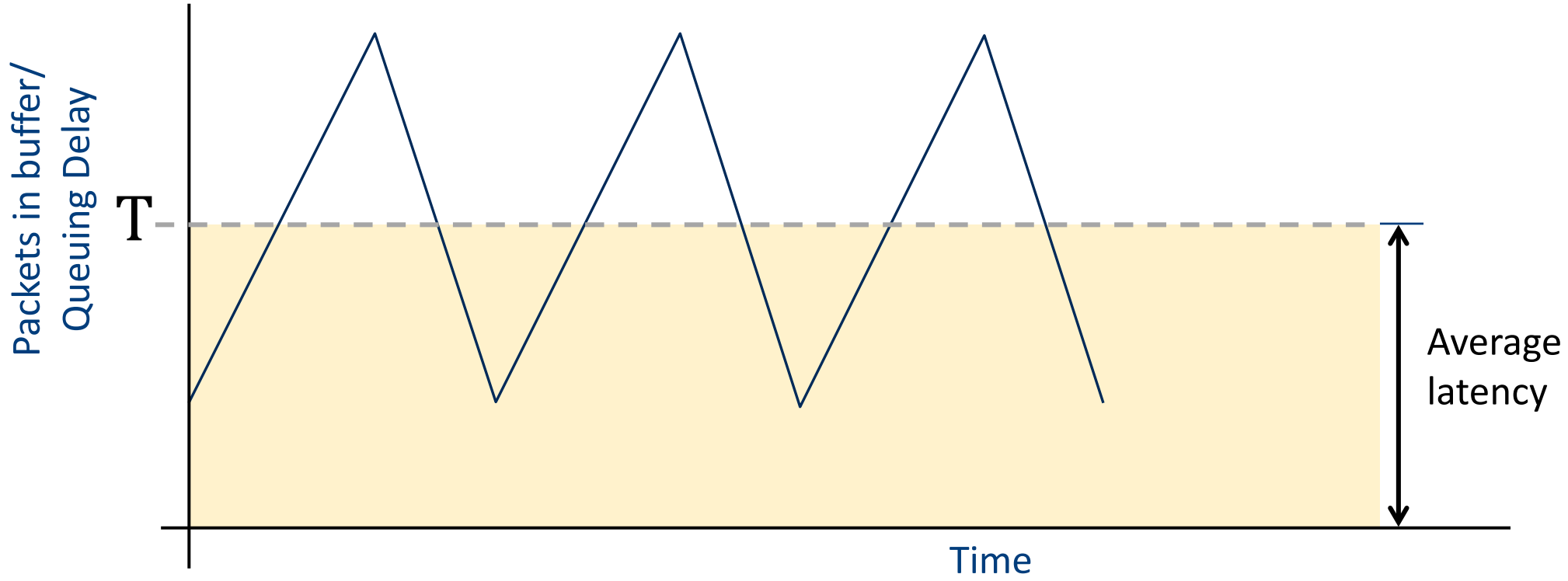
Parameters

By adjusting the parameters, k_f, k_d and T, we can change the shape of the sawtooth.



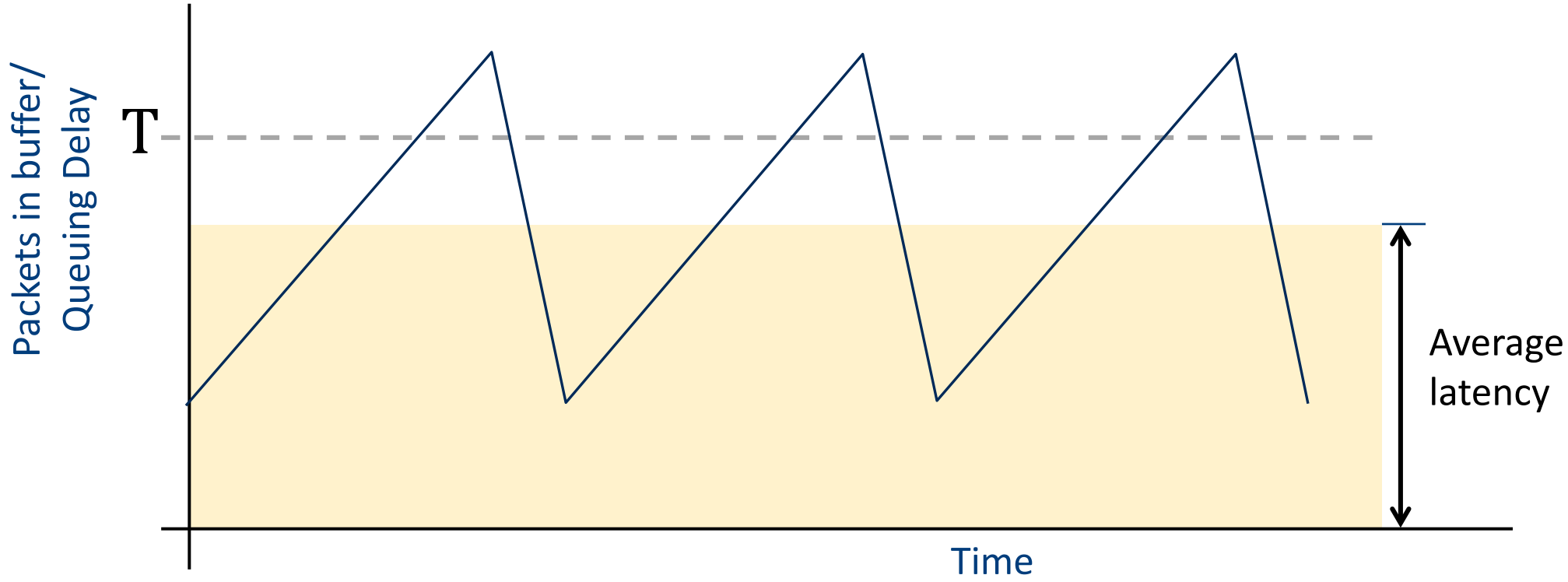
Parameters

By adjusting the parameters, k_f, k_d and T, we can change the shape of the sawtooth.



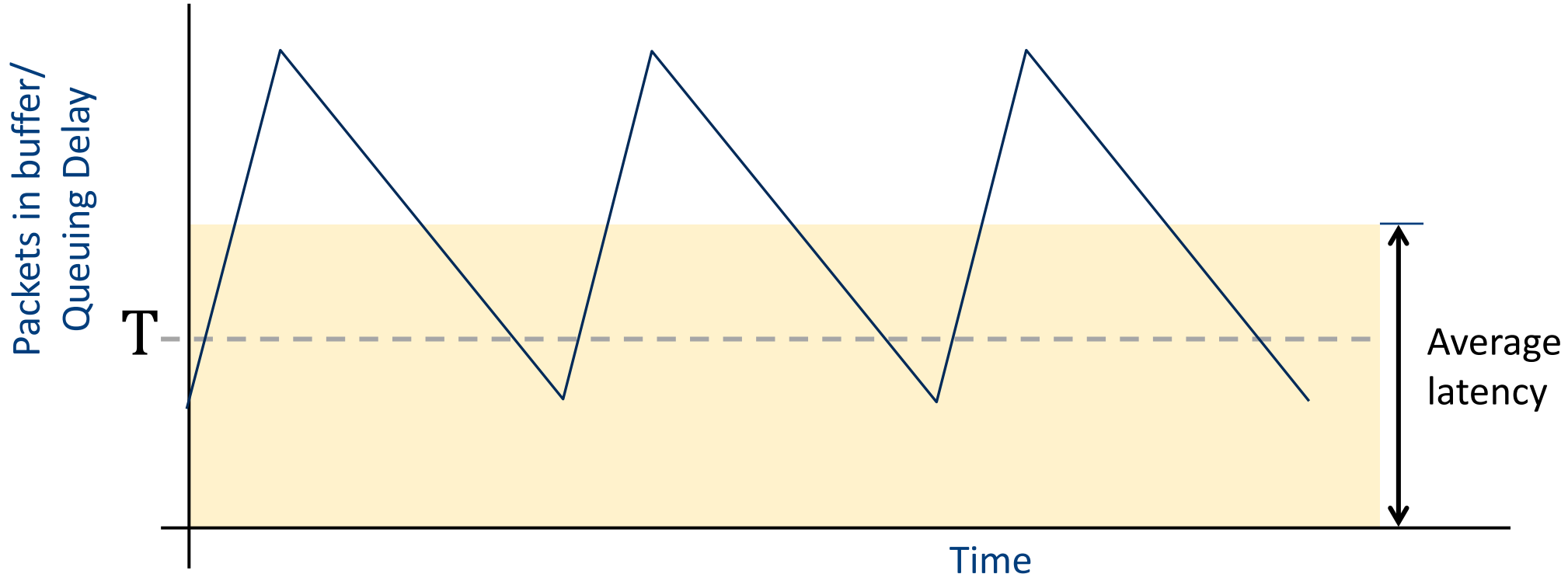
Parameters

By adjusting the parameters, k_f, k_d and T, we can change the shape of the sawtooth.



Parameters

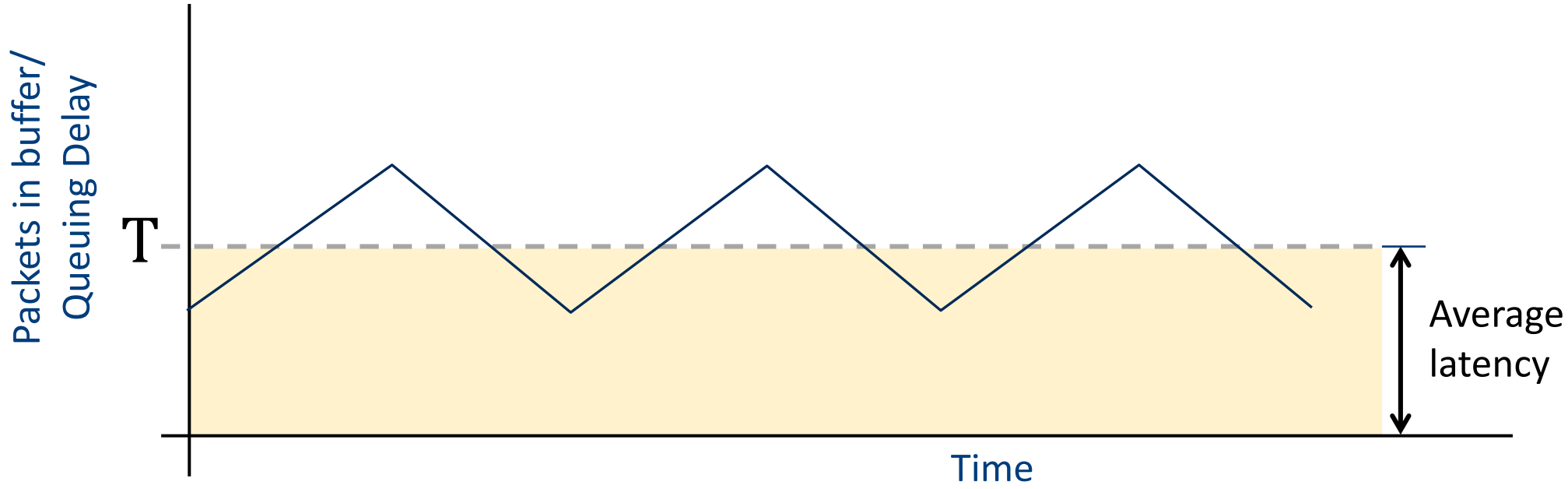
By adjusting the parameters, k_f, k_d and T, we can change the shape of the sawtooth.



Parameters

By adjusting the parameters, k_f, k_d and T, we can change the shape of the sawtooth.

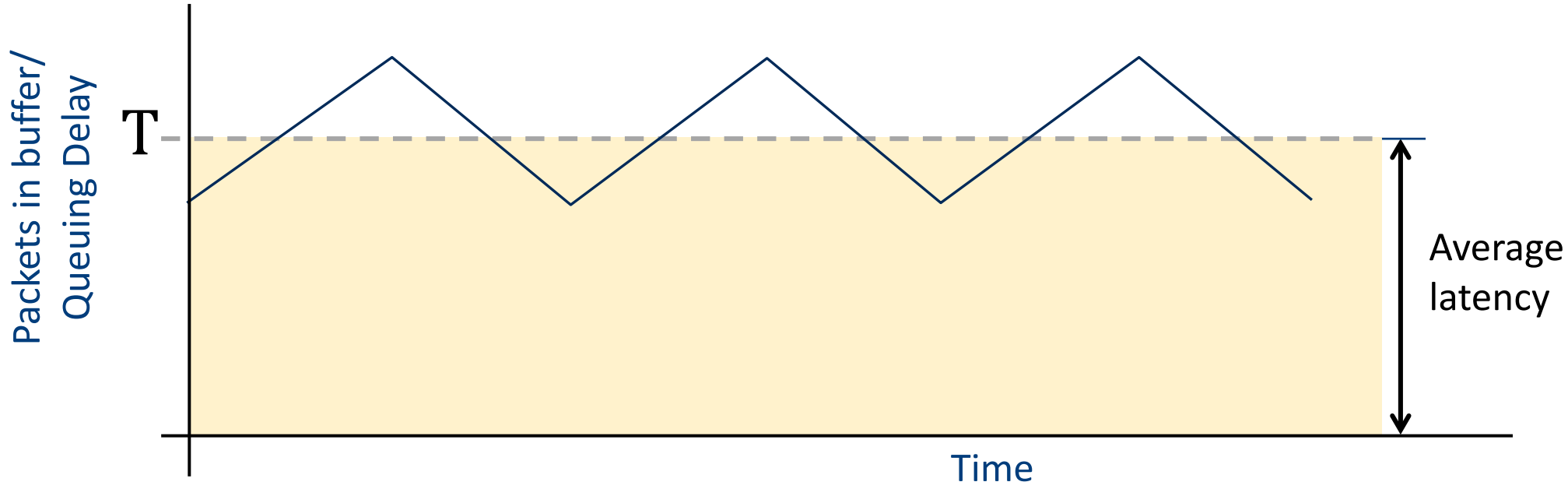
Throughput is maximum because buffer is always filled



Parameters

Throughput is maximum because buffer is always filled

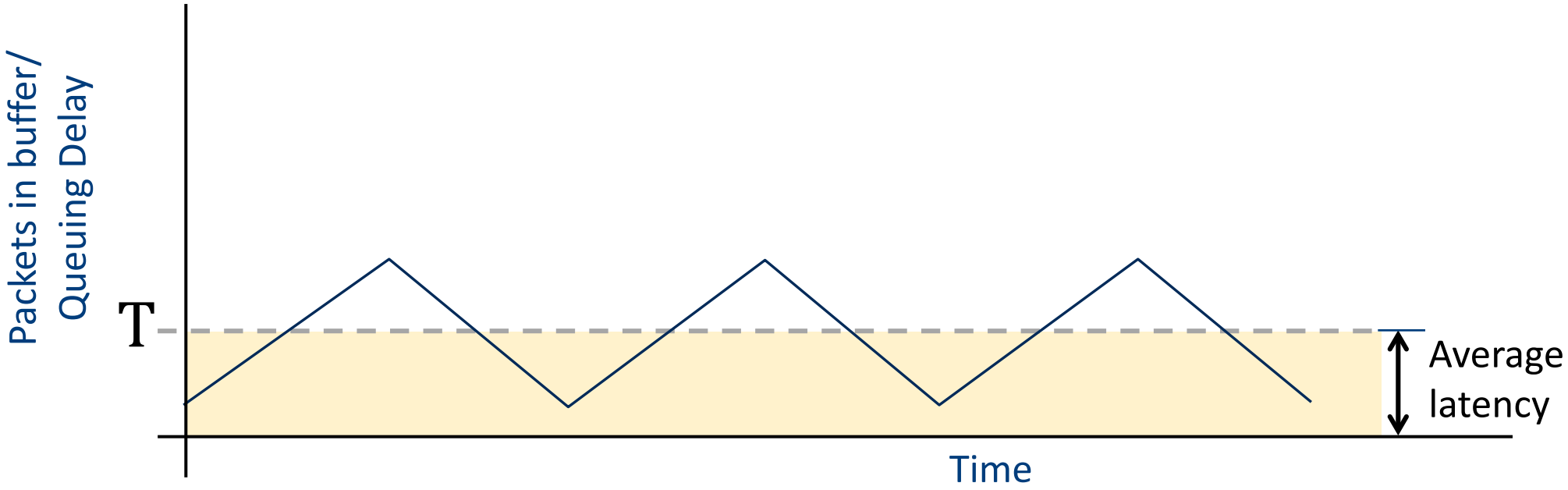
Average latency can be adjusted



Parameters

Throughput is maximum because buffer is always filled

Average latency can be adjusted



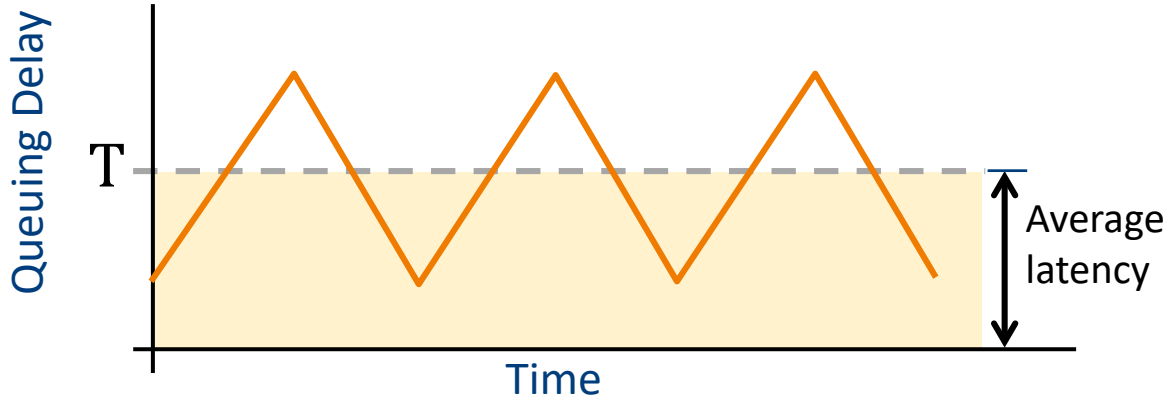
Two optimization modes

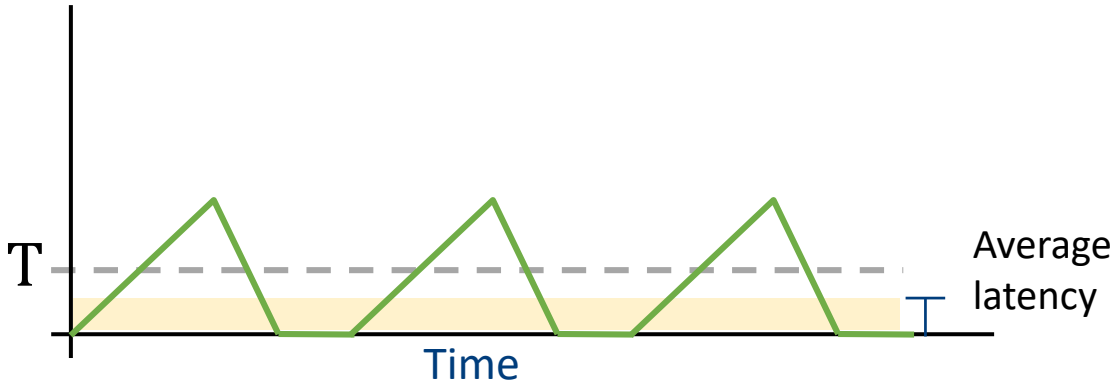
Optimizing for Throughput

- Buffer to be kept filled
- Implies maximum throughput
- Latency suffers due to queuing delay

Optimizing for Latency

- Buffer needs to be emptied
- Reduced utilization → reduced throughput
- More responsive latencies





Please read our paper

Parameter tuning

- Specify target latency to set the parameter

Updating Threshold

- Due to network volatility

Some math

읽으십시오

Evaluation

Performance Evaluation

1. Compare with other TCP protocols
   - Traditional TCP: CUBIC, Vegas, Westwood, LEDBAT
   - State-of-art Mobile: Sprout, PCC, Verus, BBR
2. Delayed ACK/Saturated Uplink
3. Throughput vs Delay tradeoff
4. Fairness/Contention
5. Computation overhead

Two Scenarios

1. Emulated networks
2. Real cellular networks

Three flavours of PropRate

Low, Medium, High

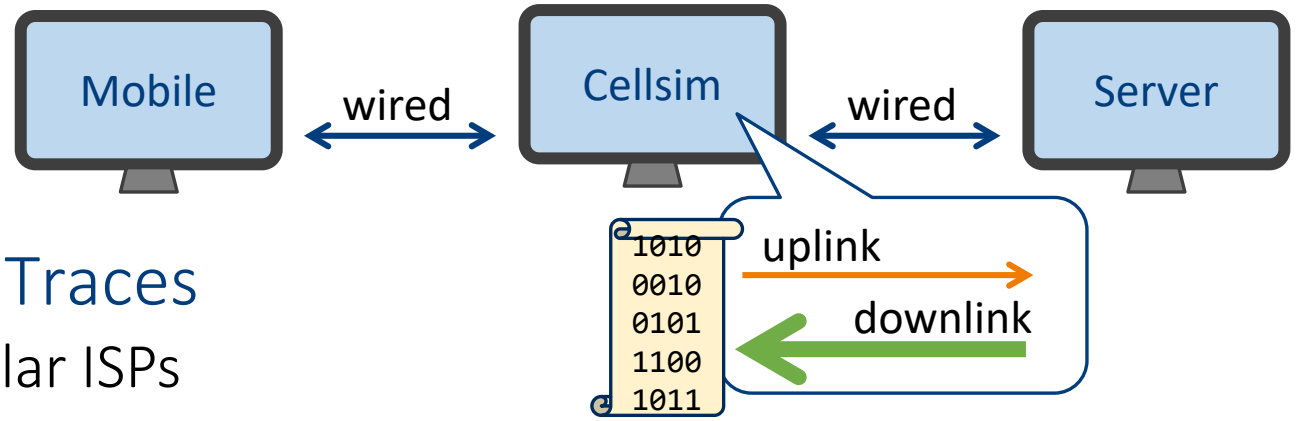
+ Frontier

Enumerate parameter space

Trace-based Emulation

Keep network constant – for fair comparison

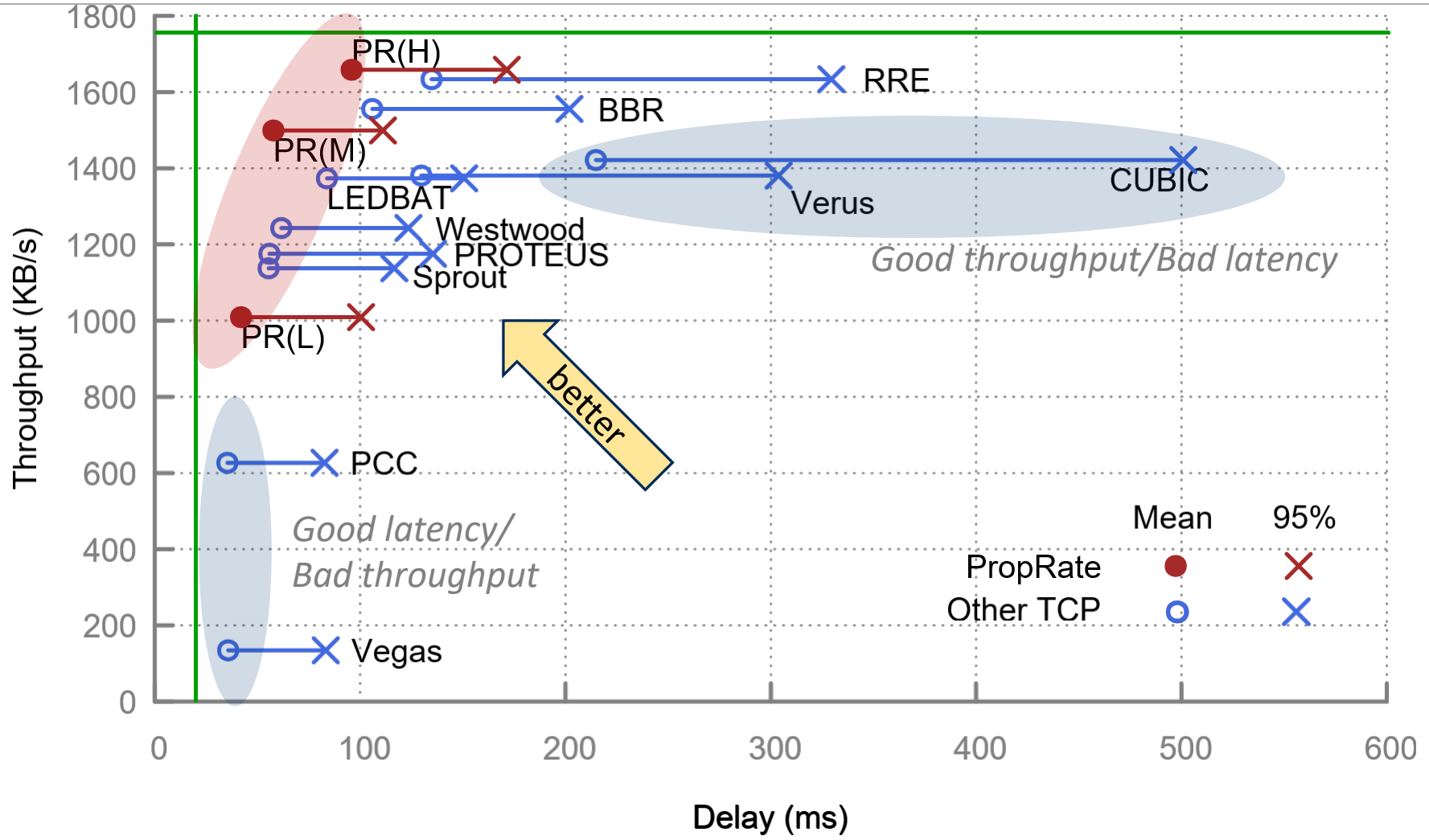
Cellsim Emulator (from MIT)



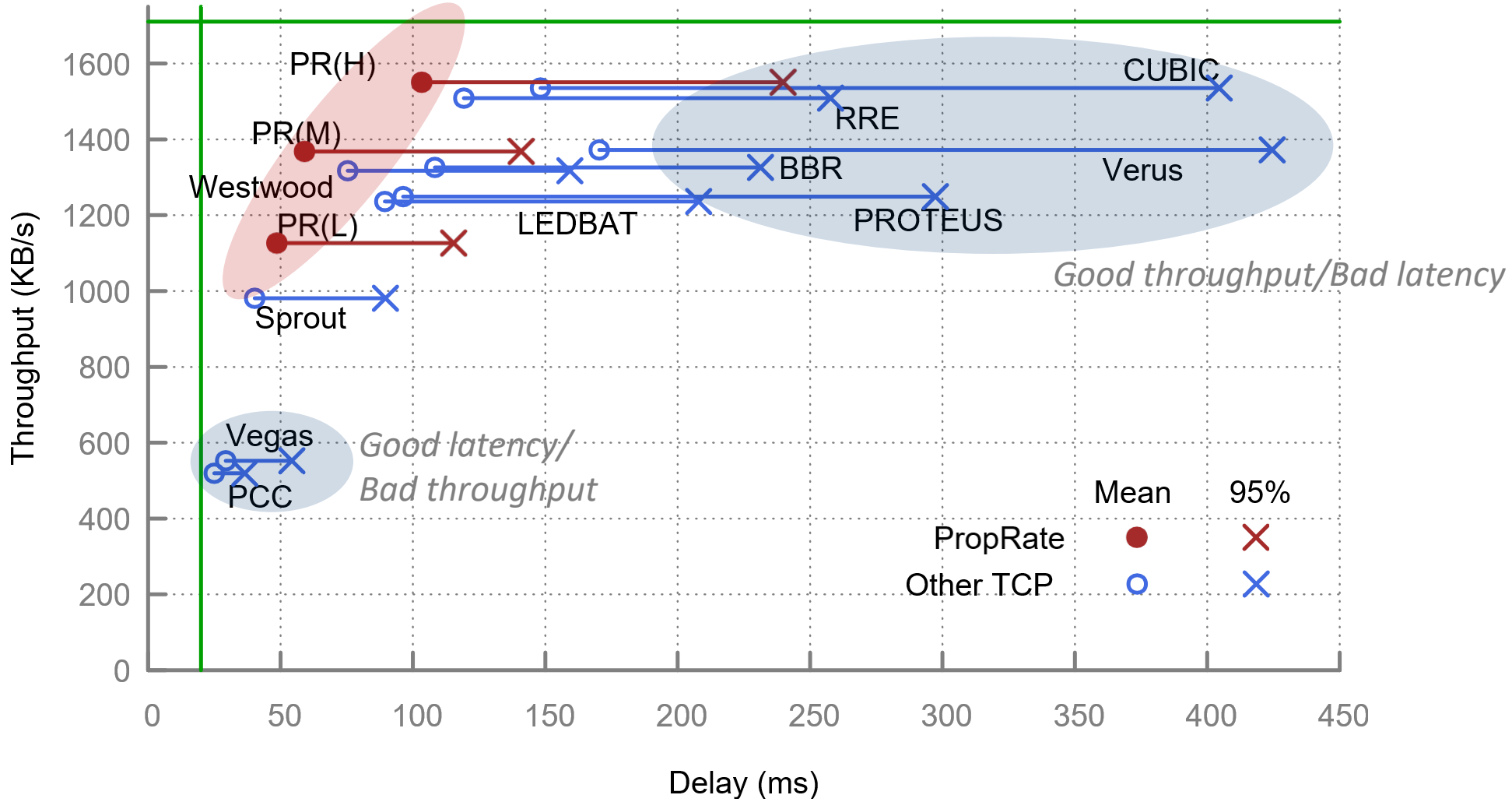
Actual Network Traces

- Three local cellular ISPs
- Two scenarios: stationary (in our lab) and mobile (on a bus)
- MIT traces [Winstein et al.]

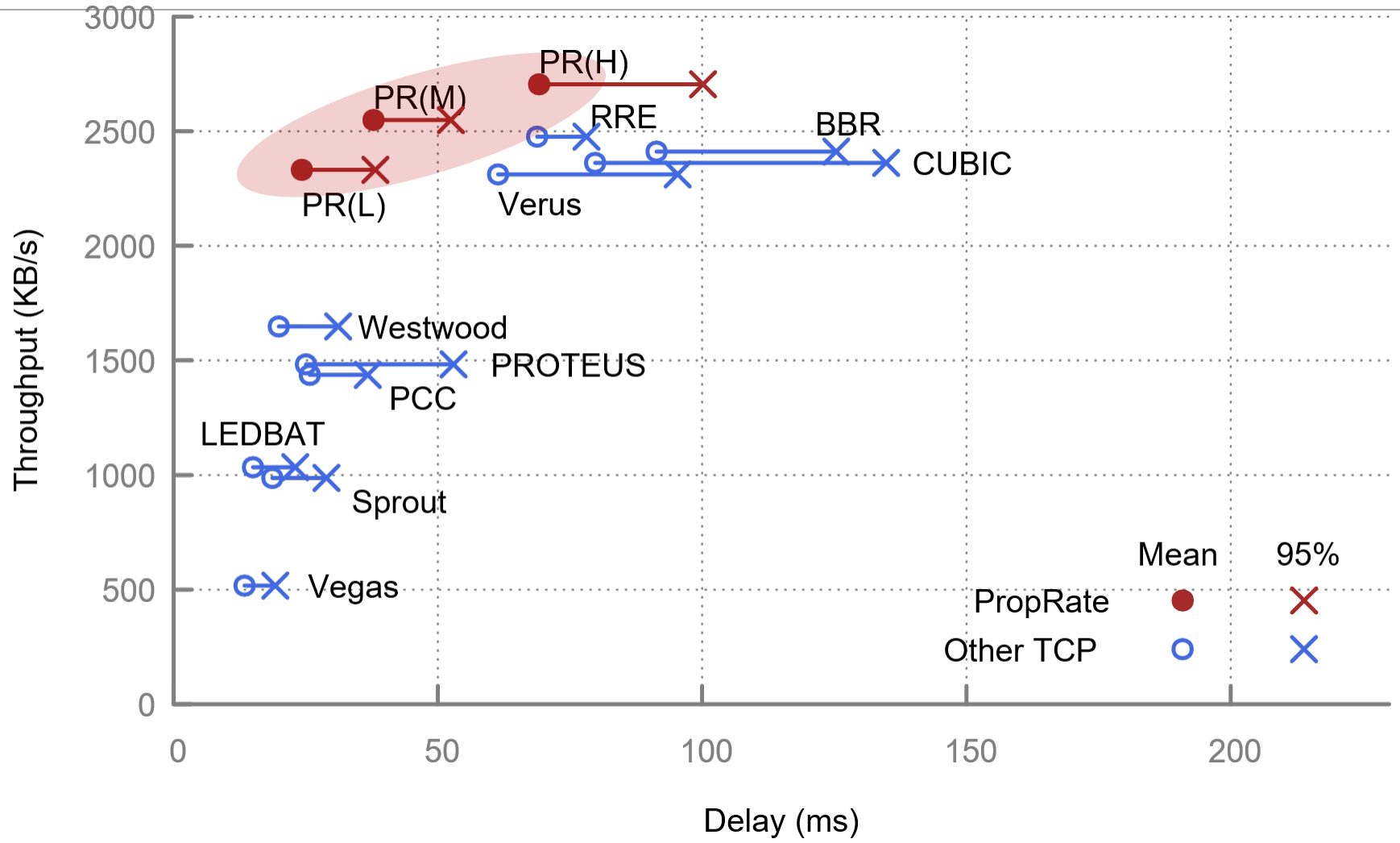
Results – Local ISP, Stationary



Results – Local ISP, Mobile



Results – Real LTE Network

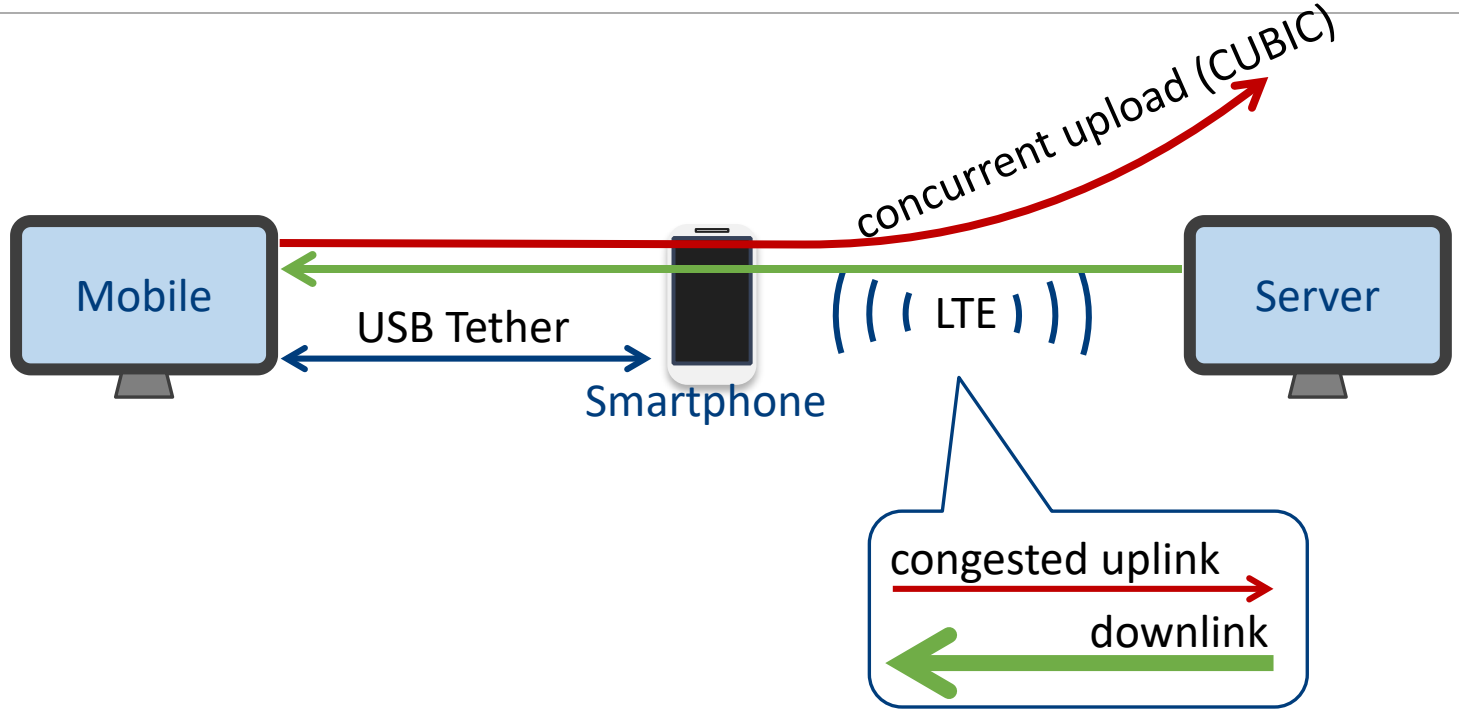


Results

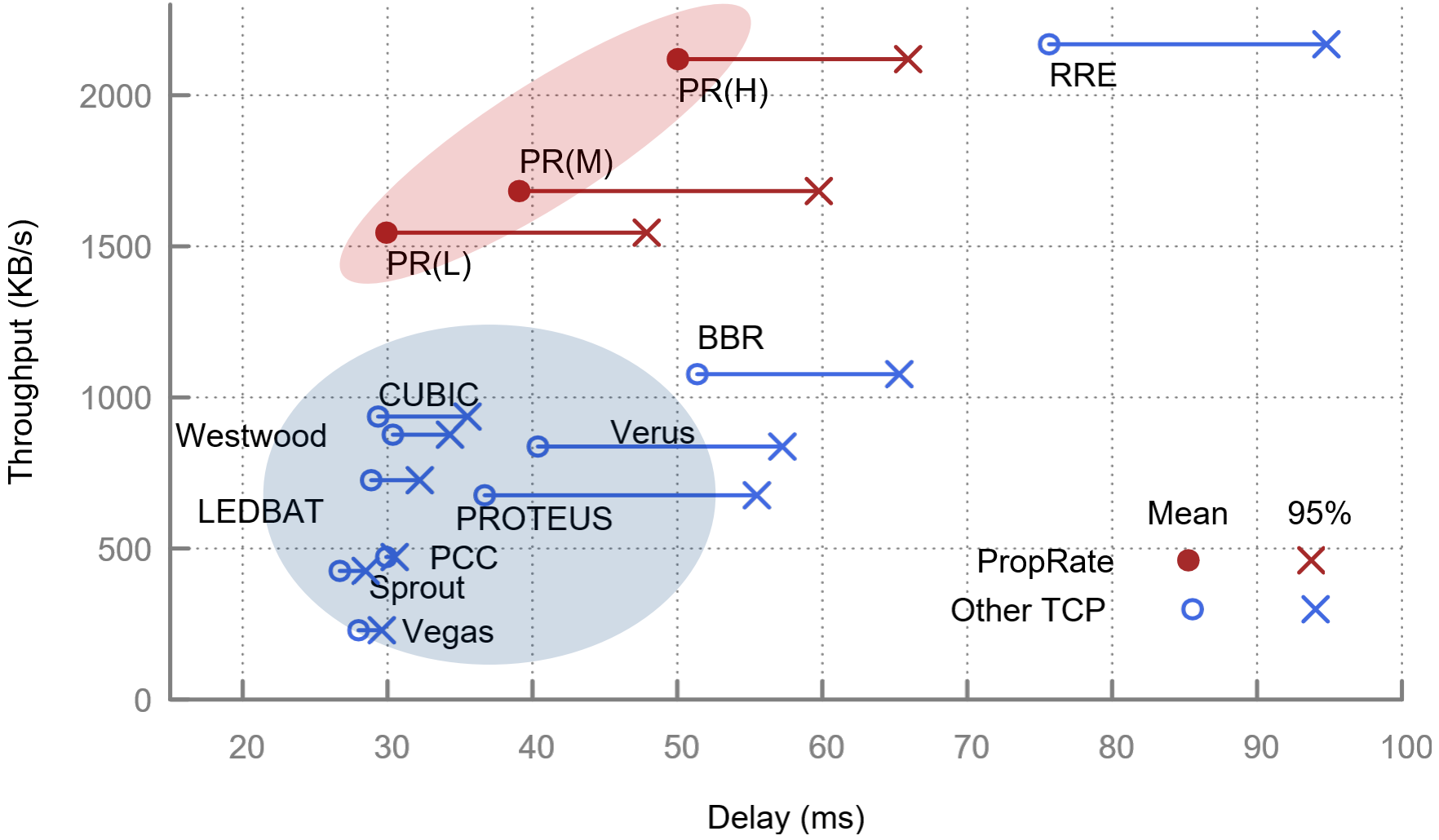
PropRate more optimal than other TCP variants

- Achieves higher throughput
- or, lower latency

Congested/Saturated Uplink



Congested Uplink – Real LTE Network



Results

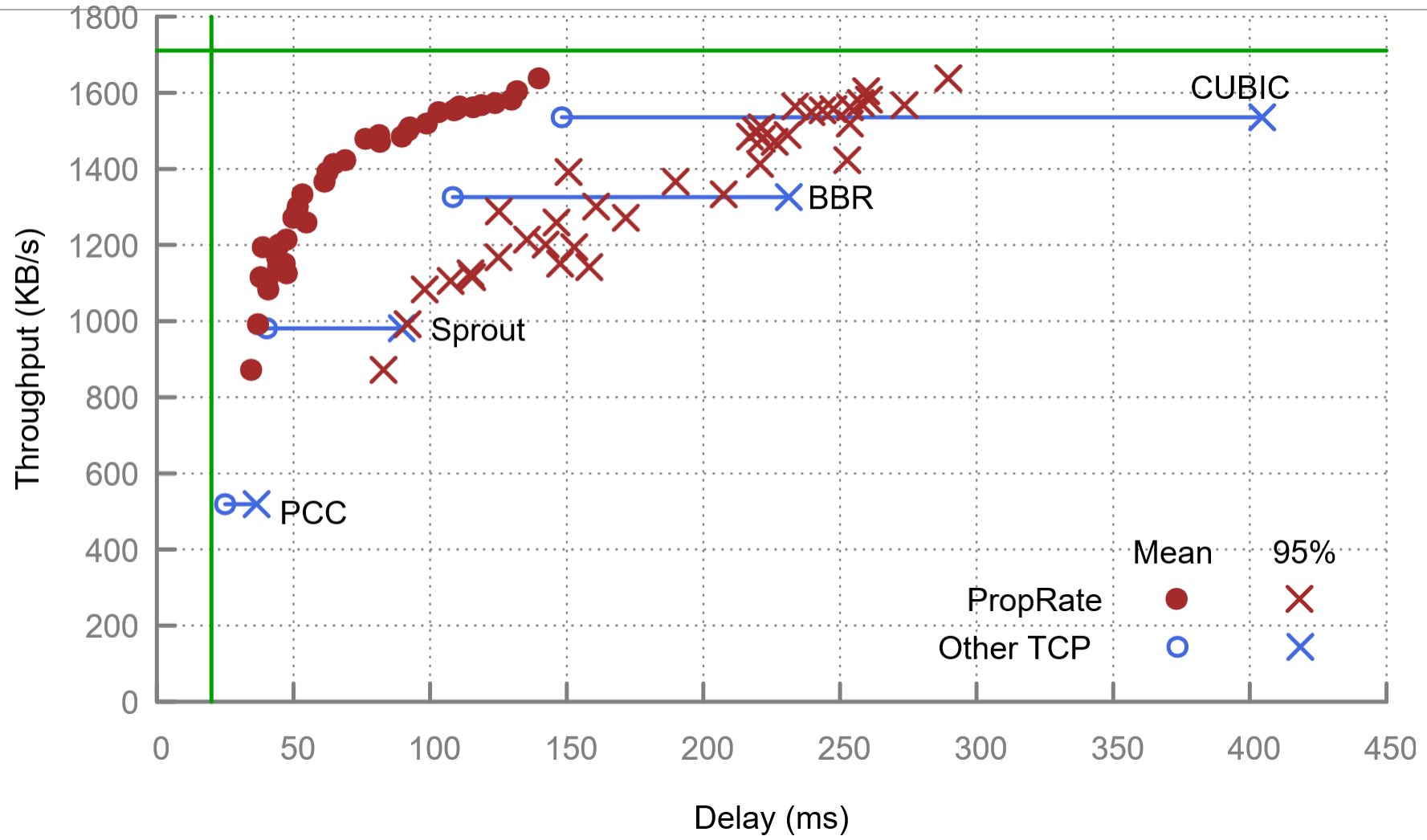
PropRate more optimal than other TCP variants

- Achieves higher throughput
- or, lower latency

Decoupling ACK clocking improves resilience

- Towards asymmetric links

Performance Frontiers



Results

PropRate more optimal than other TCP variants

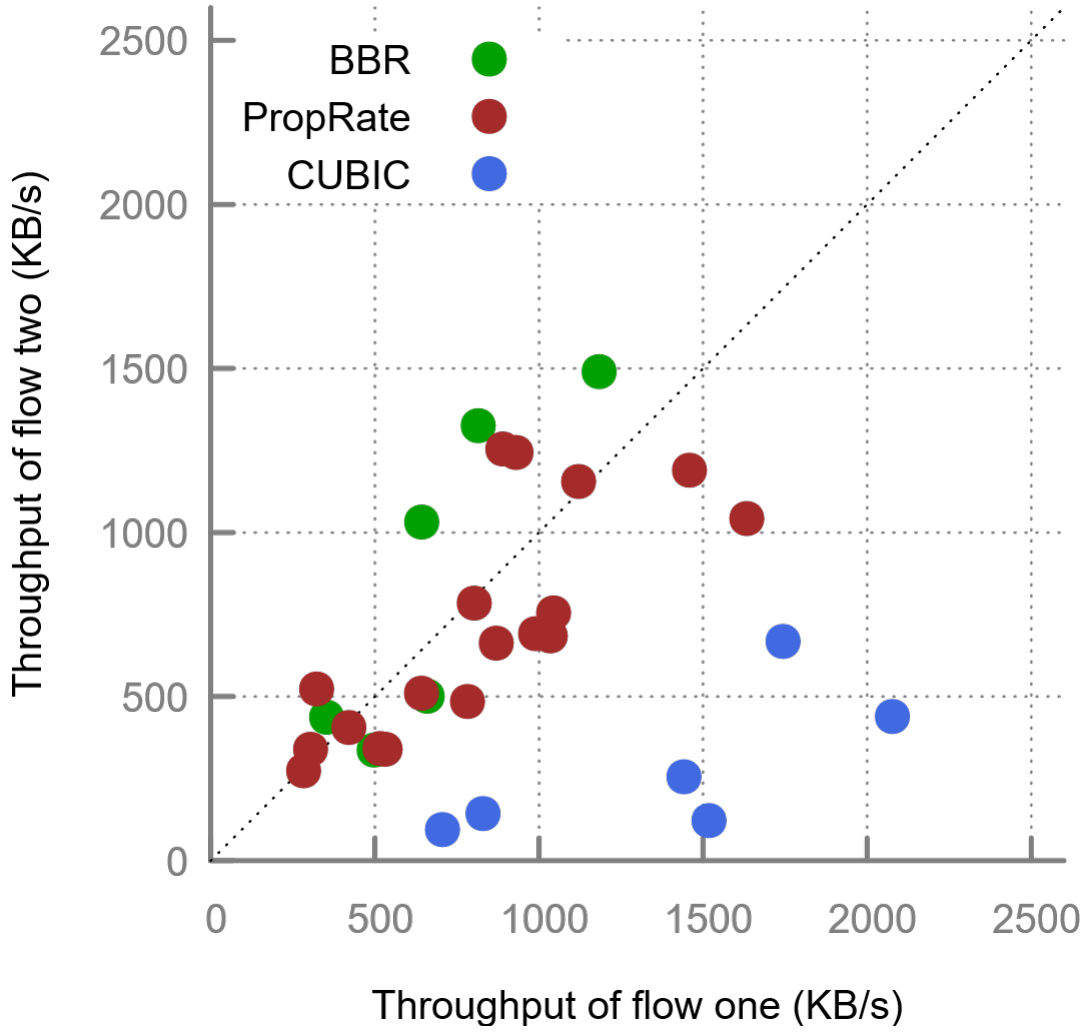
- Achieves higher throughput
- or, lower latency

Decoupling ACK clocking improves resilience

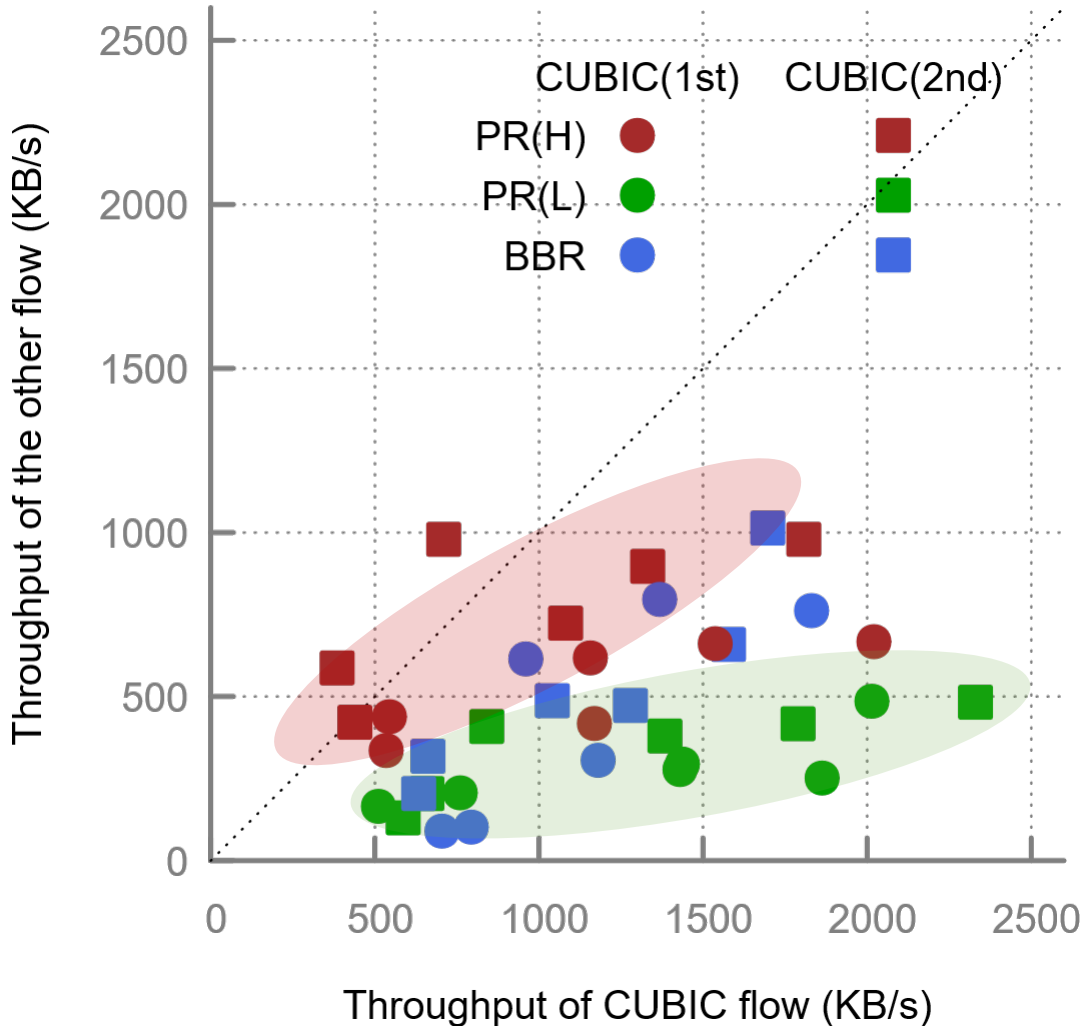
- Towards asymmetric links

Frontier hull shows PropRate is always most optimal

Fairness – Self Contention



Fairness – Contention from others



Results

PropRate more optimal than other TCP variants

- Achieves higher throughput
- or, lower latency

Decoupling ACK clocking improves resilience

- Towards asymmetric links

Frontier hull shows PropRate is always most optimal

PropRate can compete with CUBIC flows

Whither the future?

Resurgence in interest in TCP

- Different emergent networks: Datacenter, Wi-Fi, Cellular, etc.

Traditional TCP: CUBIC/Compound

- Floods buffer → Increased latency

Delay-based algorithms: Vegas, Westwood, etc.

- Good latency
- Starved by CUBIC

Is TCP ready for rate-based algorithms?

Pure rate-based algorithms: PropRate & BBR

- Handles bufferbloat
- Compete well against CUBIC

Can co-exist with CUBIC

- Facilitate transition to better rate-based TCP algorithms

PropRate builds on a framework

- More optimal algorithms in the future?
- Better integration with TCP stack in the future?

Thank You

감사합니다

QUESTIONS?