6.1 billion

6.1 billion

The number of packets dropped across the Internet in the past minute[1]

[1] Estimated conservatively at 0.1% loss rate

Cisco. Cisco Predicts More IP Traffic in the Next Five Years Than in the History of the Internet.

APNIC. The Size of Packets.

Should we avoid the 6.1 billion drops?

“Yes, packet drops are undesirable”

- Hurt throughput as senders back off
- Waste bandwidth with retransmissions
- Degrade user experience for real-time applications

Should we avoid the 6.1 billion drops?

“Yes, packet drops are undesirable”

Memory is limited & shared across:

- Flows
- Queues
- Ports

Why not enqueuing all packets?

- Longer queuing latency for all flows in the queue
- No buffer to absorb bursts that come later
- Buffer pressure & low throughput on other ports

The goal is not to eliminate drops

—— but to control the queues

Controlling Arbitrary Internet Queues with Titrate

Anchengcheng Zhou
Princeton University

Joshua Lau, Brighten Godfrey, Maria Apostolaki



Controlling Arbitrary Internet Queues with Titrate

- **Motivation** Queue management is important
- **Challenge** Queue management is challenging
- **Titrate** A control loop dynamically adapts queue threshold
- **Evaluation** Experiments demonstrate Titrate’s effectiveness

Controlling Arbitrary Internet Queues with Titrate

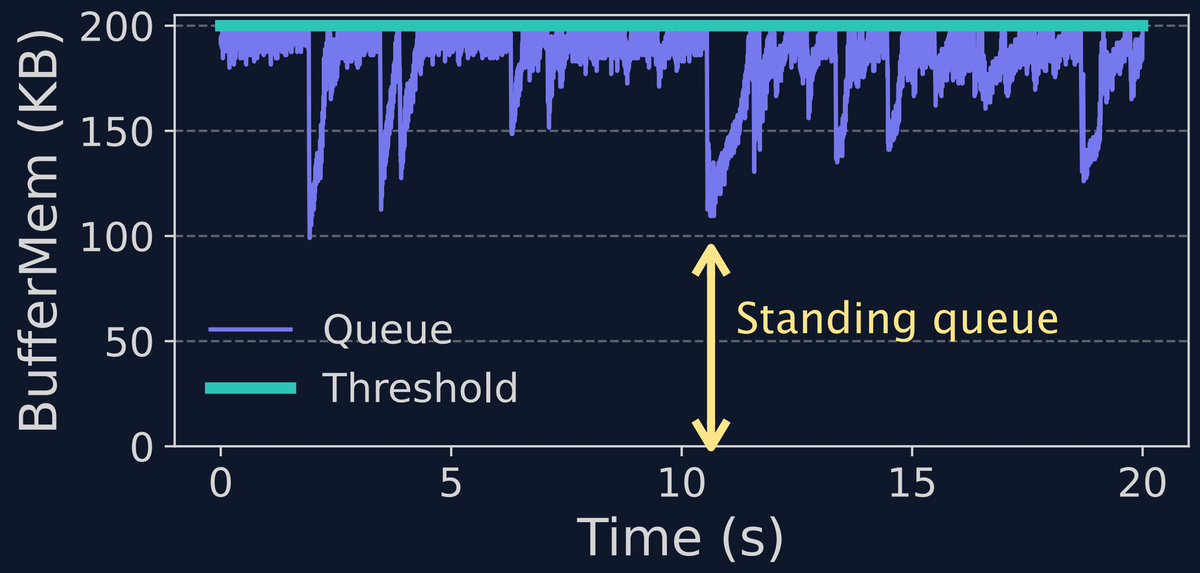
- **Motivation** Queue management is important
- **Challenge** Queue management is challenging
- **Titrate** A control loop dynamically adapts queue threshold
- **Evaluation** Experiments demonstrate Titrate's effectiveness

Controlling Arbitrary Internet Queues with Titrate

- **Motivation** Queue management is important
- **Challenge** Queue management is challenging
- **Titrate** A control loop dynamically adapts queue threshold
- **Evaluation** Experiments demonstrate Titrate's effectiveness

Queues should occupy min buffer needed for full throughput

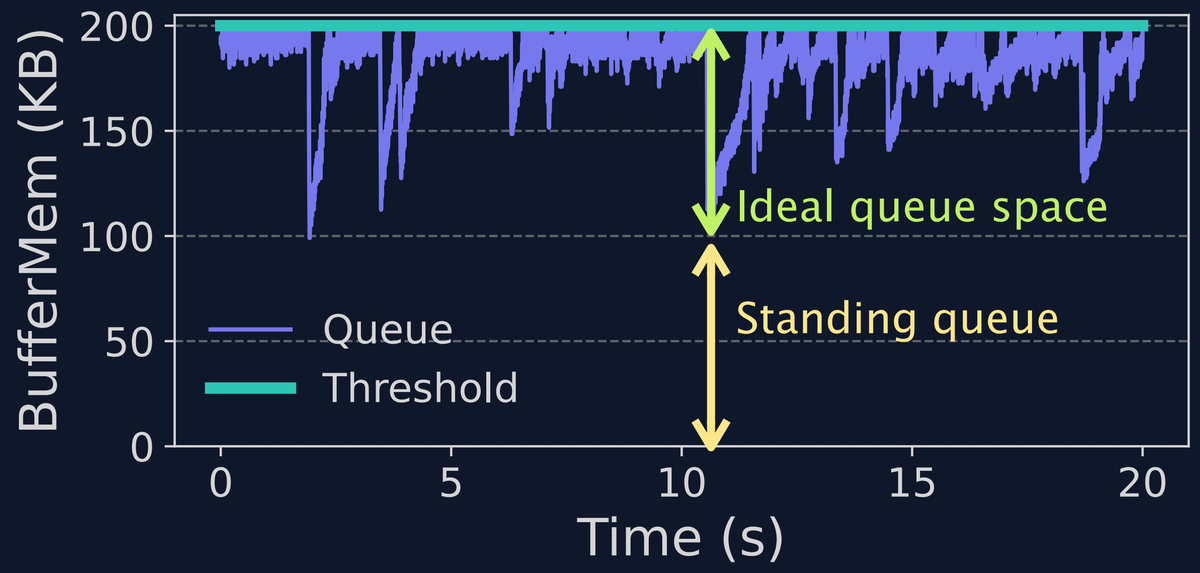
We want to avoid standing queues



Queues should occupy min buffer needed for full throughput

We want to avoid standing queues

BUT we want to allow natural oscillations of queue lengths



Queues should occupy min buffer needed for full throughput

We want to avoid standing queues

BUT we want to allow natural fluctuations of queue lengths

What's the ideal queuing situation here?

To place the threshold such that:

- It *just* accommodates the queue oscillation
- *i.e.* it is the minimum buffer space needed for achieving full throughput

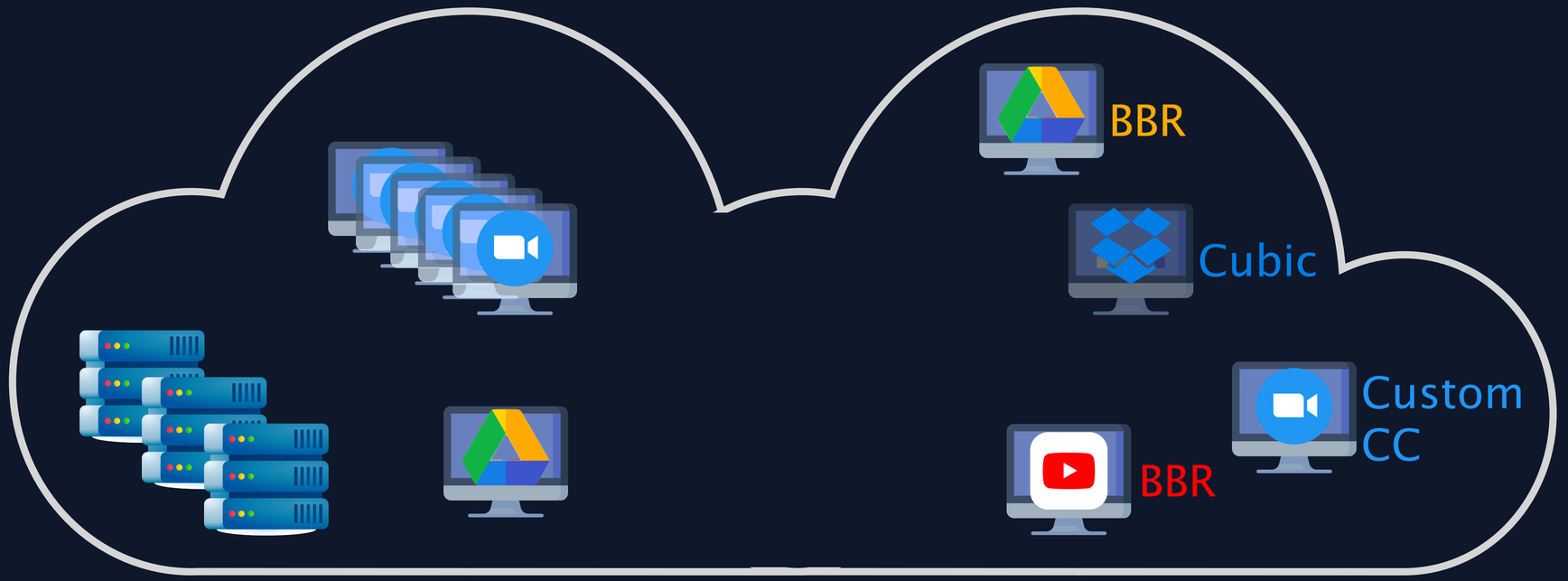
Different queues have different buffer needs

Different types of applications

Different congestion controls

Different RTTs

Different numbers of flows



Different queues have different buffer needs

Different types of applications

Different congestion controls

Different RTTs

Different numbers of flows

} Internet queues have different flow compositions

Different queues have different buffer needs

Different types of applications

Different congestion controls

Different RTTs

Different numbers of flows

Internet queues have different flow compositions

Ideal: Min buffer needed for full throughput per queue

Different queues have different buffer needs

Different types of applications

Different congestion controls

Different RTTs

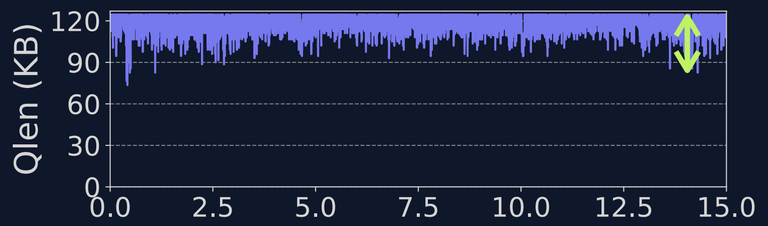
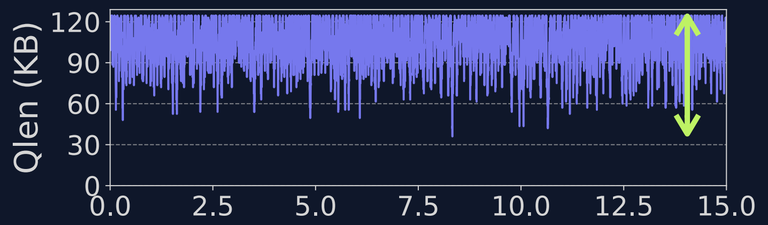
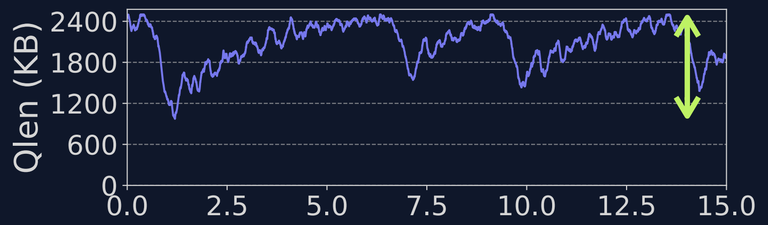
Different numbers of flows

Internet queues have different flow compositions

Ideal: Min buffer needed for full throughput per queue

Challenge: Different for different queues

Different queues have different buffer needs

Queue composition	Queue length time series	Ideal queue space
20 10ms-RTT Cubic flows		100KB
200 10ms-RTT Cubic flows		45KB
20 10ms-RTT Reno flows		90KB
20 200ms-RTT Cubic flows		1250KB

Existing work does not tackle this challenge

Can we pick a static threshold?

Different queues have different ideals
No one-size-fits-all threshold

Can we pre-compute a threshold?

e.g., Buffer sizing

Queue compositions are dynamic/unknown
Mixed/unknown congestion controls

Can we ask hosts to mark their flows?

e.g., DC buffer sharing, L4S

Markings are not reliable in Internet

How about AQMs?

e.g., RED, CoDel, COBALT, PIE

Rigid heuristics for identifying standing queues do not generalize to arbitrary queues

Controlling Arbitrary Internet Queues with Titrate

- **Motivation** Queue management is important
- **Challenge** Queue management is challenging
- **Titrate** A control loop dynamically adapts queue threshold
- **Evaluation** Experiments demonstrate Titrate’s effectiveness

Controlling Arbitrary Internet Queues with Titrate

- **Motivation** Queue management is important
- **Challenge** Queue management is challenging
- **Titrate** A control loop dynamically adapts queue threshold
- **Evaluation** Experiments demonstrate Titrate's effectiveness

Titrate draws inspiration from TCP

I want to allocate the min buffer space needed by a queue

Why don’t routers “probe” too?

- Use queue_threshold_0
- Wait for end host feedback
- Adjust to queue_threshold_1
- Iterate

end host feedback

sending_rate

network feedback

queue_threshold

I want to send at the rate of the max available bandwidth

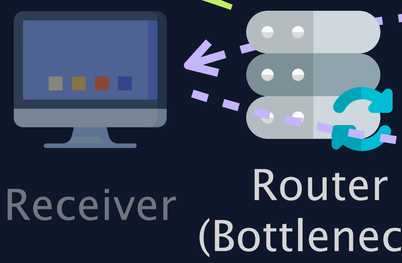
- Use sending_rate_0
- Wait for network feedback
- Adjust to sending_rate_1
- Iterate

Titrate:
a closed-loop controller that adapts threshold in real-time

I want to allocate the min buffer space needed by a queue

Why don't routers "probe" too?

- Use queue_threshold_0
- Wait for end host feedback
- Adjust to queue_threshold_1
- Iterate



✓ **Performance**

High thpt, low latency, effective burst absorptn

✓ **Generality**

Applicability to arbitrary queue

✓ **Practicality**

No reliance on end host or hardware updates

Titrate:
a closed-loop controller that adapts threshold in real-time

- What constitutes useful end host feedback?

  What to sense?

- When to collect end host feedback?

  When to sense?

- How to adjust queue threshold?

  How to adjust?

I want to allocate the min buffer space needed by a queue

Why don't routers "probe" too?

Router (Bottleneck)

- Use queue_threshold_0
- Wait for end host feedback
- Adjust to queue_threshold_1
- Iterate

Titrate jointly monitors two signals for thpt & latency

What to sense?

When to sense?

How to adjust?

- Duration of queue length reaching zero

  ⇨ Magnitude of throughput

- [Minimum queue length > 2MTU]

  ⇨ Existence of standing queue

Titrate filters out network noises when collecting signals

What to sense?

When to sense?

How to adjust?

- Keep the monitoring window relatively long

  ⇨ To avoid disrupting congestion control probing

- Collect signals after a packet drop

  ⇨ To avoid reacting to network noises

- Ignore temporary spikes via computed *effective threshold*

  ⇨ To avoid dropping burst packets

Titrate balances convergence speed & stability

What to sense?

When to sense?

How to adjust?

Cautious, small adjustments are safe, but can prolong under-performance

Blind large adjustments risk overshooting, inducing oscillations

Titrate balances convergence speed & stability

Cautious, small adjustments are safe, but can prolong under-performance

Blind large adjustments risk overshooting, inducing oscillations

What to sense?

When to sense?

How to adjust?

- Multiplicative-increase
  - ⇨ Threshold increase is more urgent, less risky
- Additive-decrease
  - ⇨ Threshold decrease is less urgent, more risky
- An ssthresh-like variable
  - ⇨ Rapid threshold decrease when deemed safe

Controlling Arbitrary Internet Queues with Titrate

- **Motivation** Queue management is important
- **Challenge** Queue management is challenging
- **Titrate** A control loop dynamically adapts queue threshold
- **Evaluation** Experiments demonstrate Titrate’s effectiveness

Controlling Arbitrary Internet Queues with Titrate

- **Motivation** Queue management is important
- **Challenge** Queue management is challenging
- **Titrate** A control loop dynamically adapts queue threshold
- **Evaluation** Experiments demonstrate Titrate’s effectiveness

Key evaluation takeaways

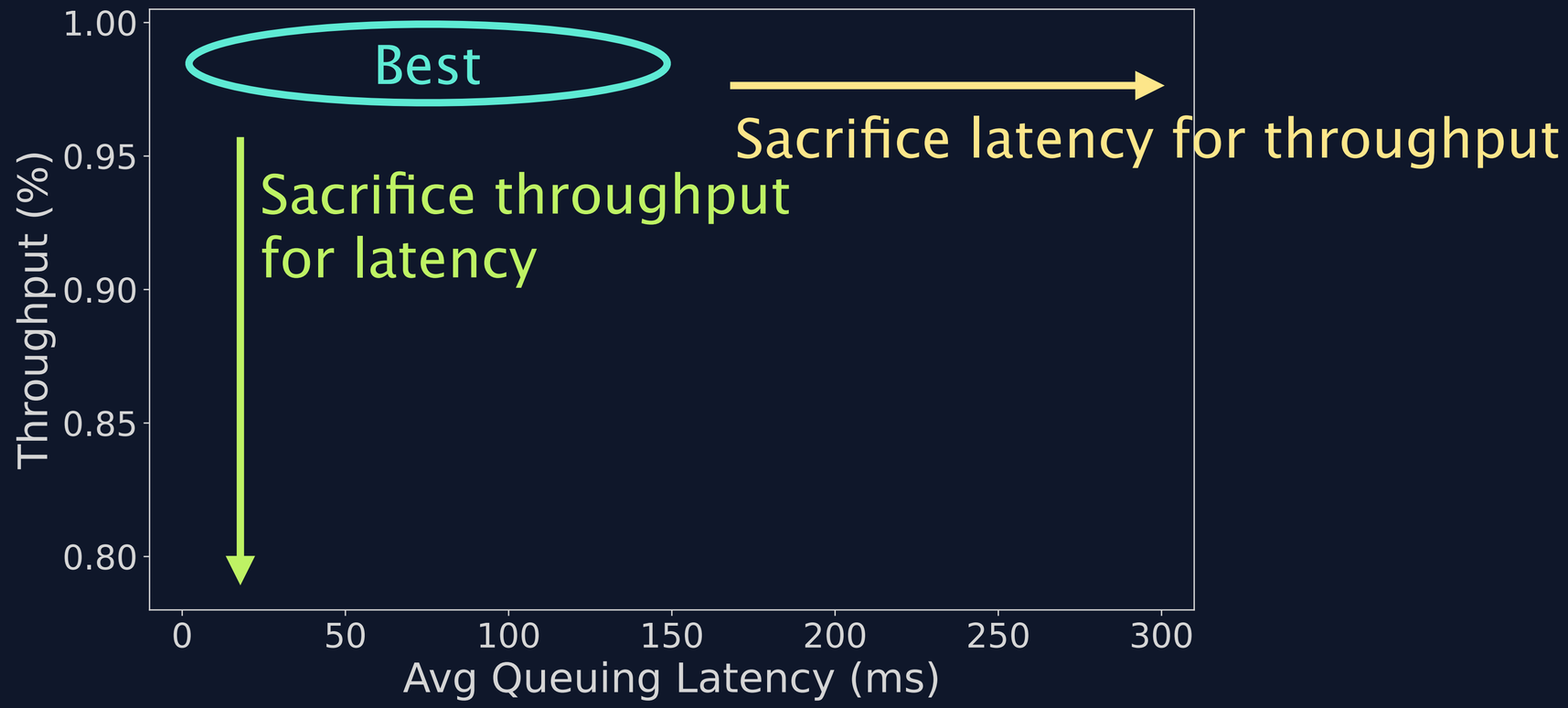
- Titrate achieves high throughput, low latency & effective burst absorption for diverse queues
- Titrate offers device-wide benefits
- Titrate adapts to dynamic network conditions efficiently

Key evaluation takeaways

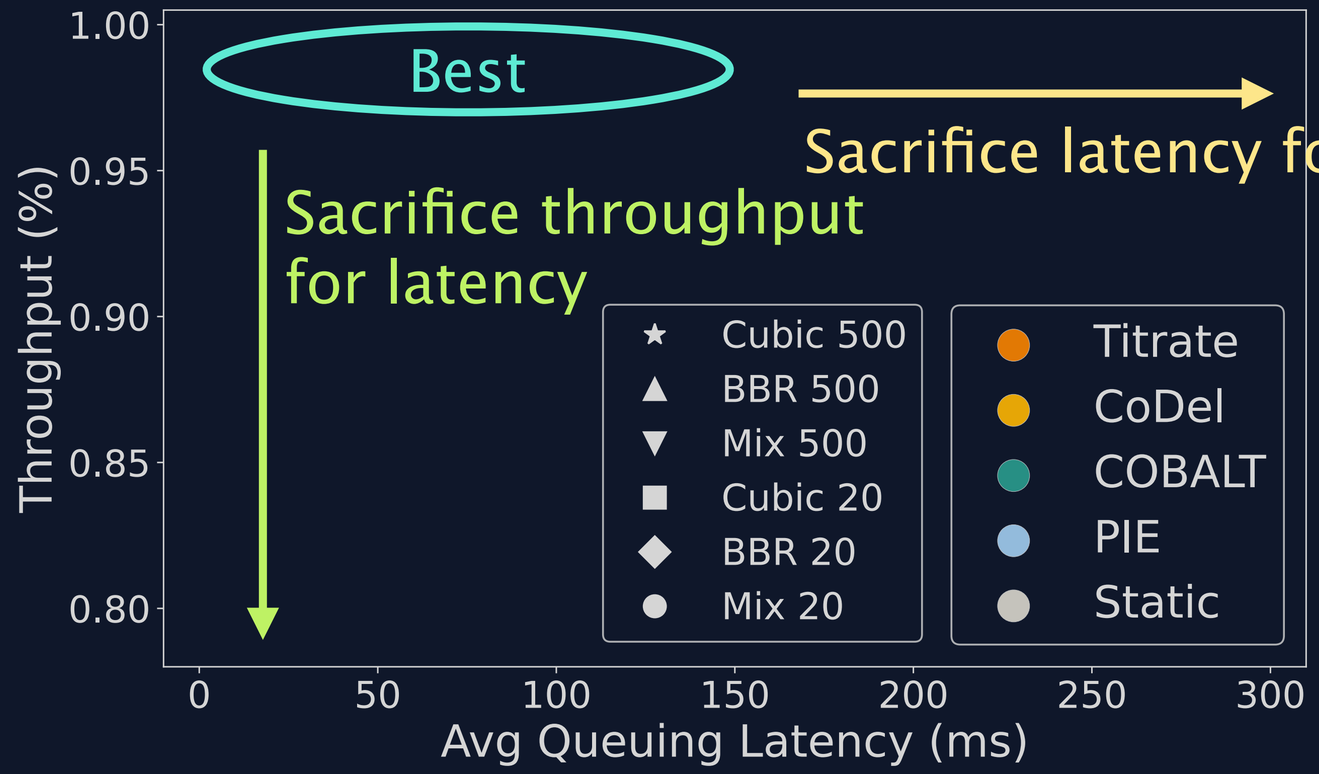
- Titrate achieves high throughput, low latency & effective burst absorption for diverse queues
- Titrate offers device-wide benefits
- Titrate adapts to dynamic network con

- ns-3 packet-level simulation
- 1Gbps bottleneck link
- 2*BDP total buffer memory

Titrate achieves high throughput & low latency for diverse queues



Titrate achieves high throughput & low latency for diverse queues



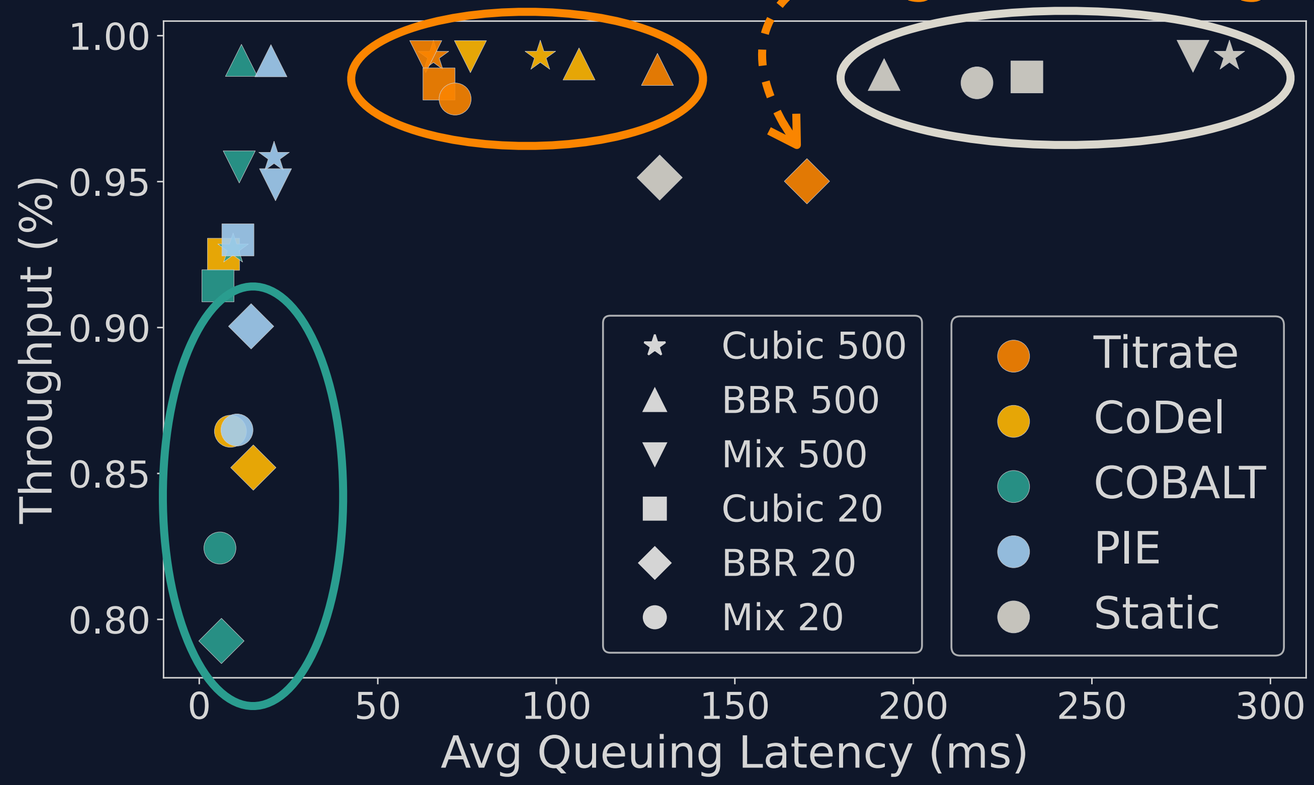
5 schemes

6 queue compositions

Titrate achieves high throughput & low latency for diverse queues

Titrate: Consistently minimizes queuing latency while achieving ~full throughput

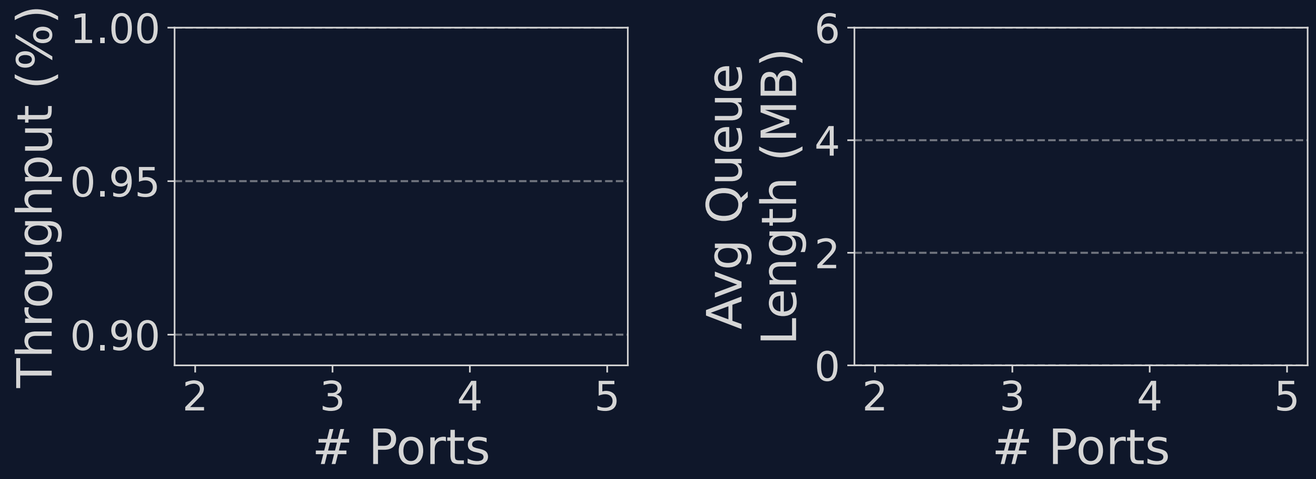
Highest throughput among baselines



Static threshold: Unnecessarily long queues

COBALT, CoDel, PIE: Maintain high throughput only for a subset of queues (Similar w/ latency)

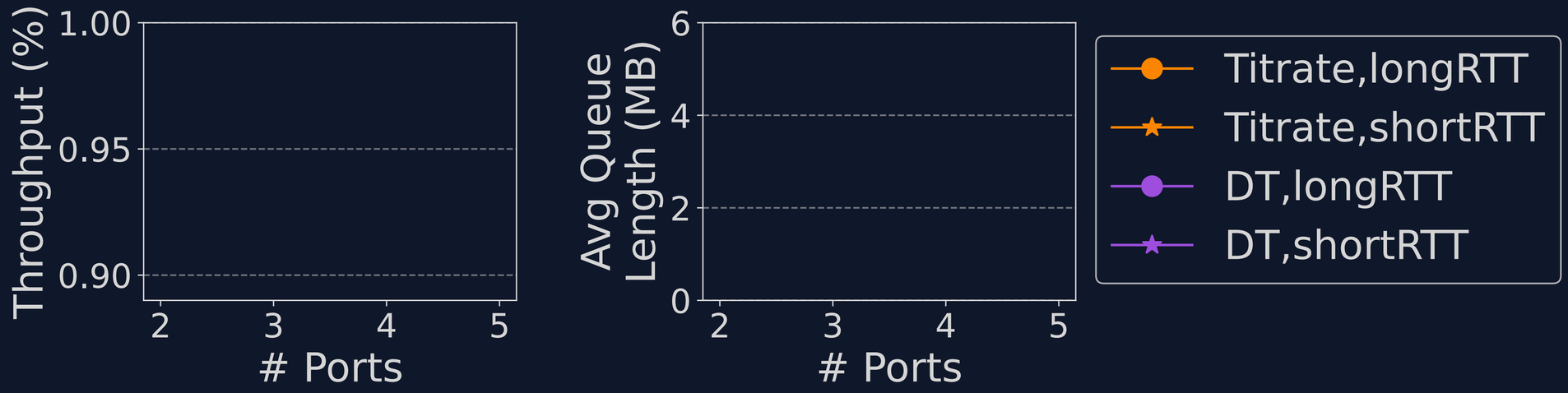
Titrate offers device-wide benefits



One queue per port

One port has long-RTT flows; all other ports have short-RTT flows

Titrate offers device-wide benefits



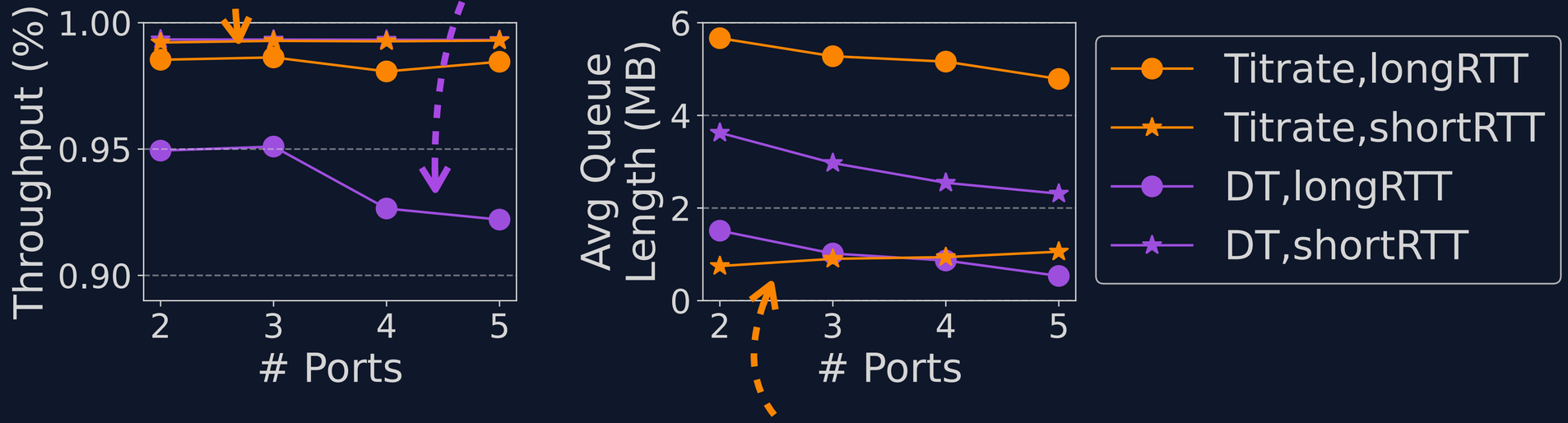
One queue per port

One port has long-RTT flows; all other ports have short-RTT flows

Titrate offers device-wide benefits

Thus Titrate achieves ~full throughput for all queues

DT suffers from low throughput



Titrate allocates memory to a queue when the queue needs it

thus saving memory for other queues that actually need it

6.1 billion

Conclusion

Titrate increases throughput & reduces latency for arbitrary Internet queues

- Operates a control loop that senses queue dynamics and adapts queue thresholds in real time
- Achieves:
  - ✔ Performance
  - ✔ Generality
  - ✔ Practicality
- Find out more about Titrate:

Paper

Artifact

Image References

- Icon made by Freepik from ww.flaticon.com
- Icon made by Valeria from ww.flaticon.com
- Icon made by meaicon from ww.flaticon.com

Thank you!

Controlling Arbitrary Internet Queues with Titrate

Anchengcheng Zhou
Princeton University