How BBR Congestion Control Tackles Latency

The Google BBR Team

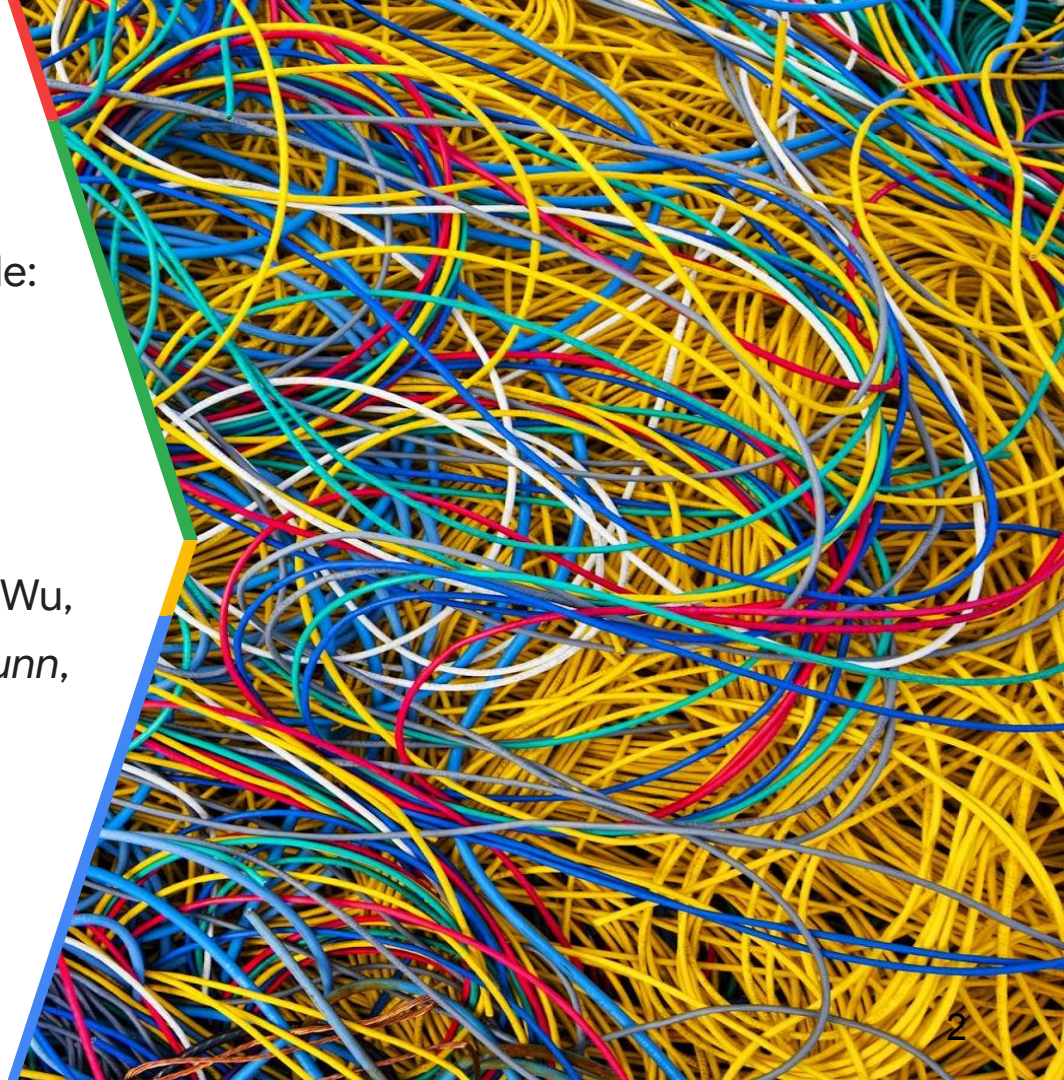
speaker: Neal Cardwell <neal@google.com>

2023-12-10
Understanding Latency 3.0 - understandinglatency.com

This is joint work by the BBR team at Google:

Neal Cardwell, Yuchung Cheng,
Kevin Yang, David Morley,
Soheil Hassas Yeganeh,
Ian Swett, *Jana Iyengar*, Victor Vasiliev, Bin Wu,
Priyaranjan Jha, Yousuk Seung, *Stephen Gunn*,
Matt Mathis,
Van Jacobson

[*ex-Googlers in italics*]



Part 1:
The Problems

The problems that motivated BBR

The story of BBR goes back to 2013...

- Many Google services complained about TCP performance...
  - Internal B4 backbone TCP throughput often < 10Mbps
  - Youtube.com: sometimes terrible video quality, with RTT > 10 secs
  - Google.com: poor latency in developing regions
- Services started to “work around” TCP
  - Use parallel connections, tweak TCP knobs, add more buffer to the network, ...

What was the root cause of these problems?

- TCP congestion control at Google in 2013 was CUBIC (Linux default)
  - CUBIC is a loss-based congestion control algorithm
  - Packet loss was the sole signal

The problem: loss-based congestion control

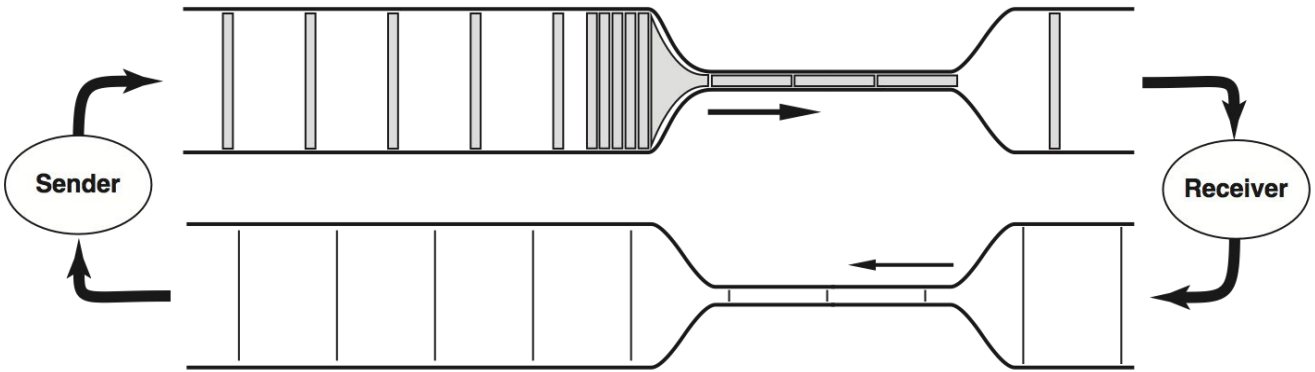
- Loss-based congestion control
  - Keeps sending faster until it sees a packet loss
  - Widely deployed: Reno (NetFlix); CUBIC (Linux/iOS/MacOS/Windows/FreeBSD)

- But packet loss **alone** is **not** a good proxy for congestion

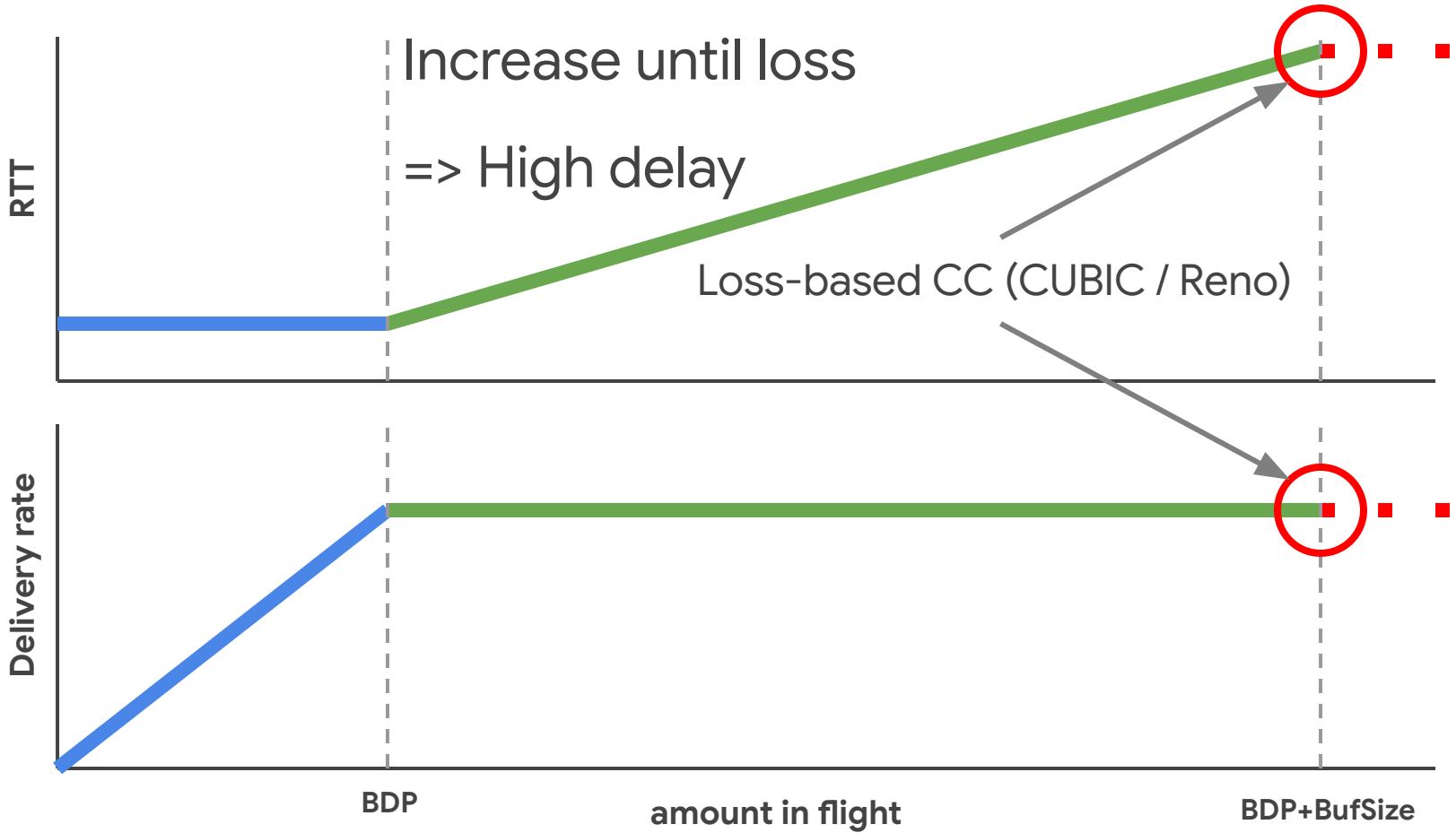
- In **shallow buffers**, flows frequently suffer random loss and **low throughput**
  - Loss comes **before** sustained full utilization, loss-based CC gets low throughput
  - 10Gbps over 100ms RTT needs < 0.0000029% (2.9e-8) packet loss (infeasible)
  - 1% loss (feasible) over 100ms RTT gets < 3Mbps

- In **deep buffers**, flows suffer low loss while maintaining **high delay**
  - Loss comes **after** congestion, loss-based CC suffers high delays ("bufferbloat")

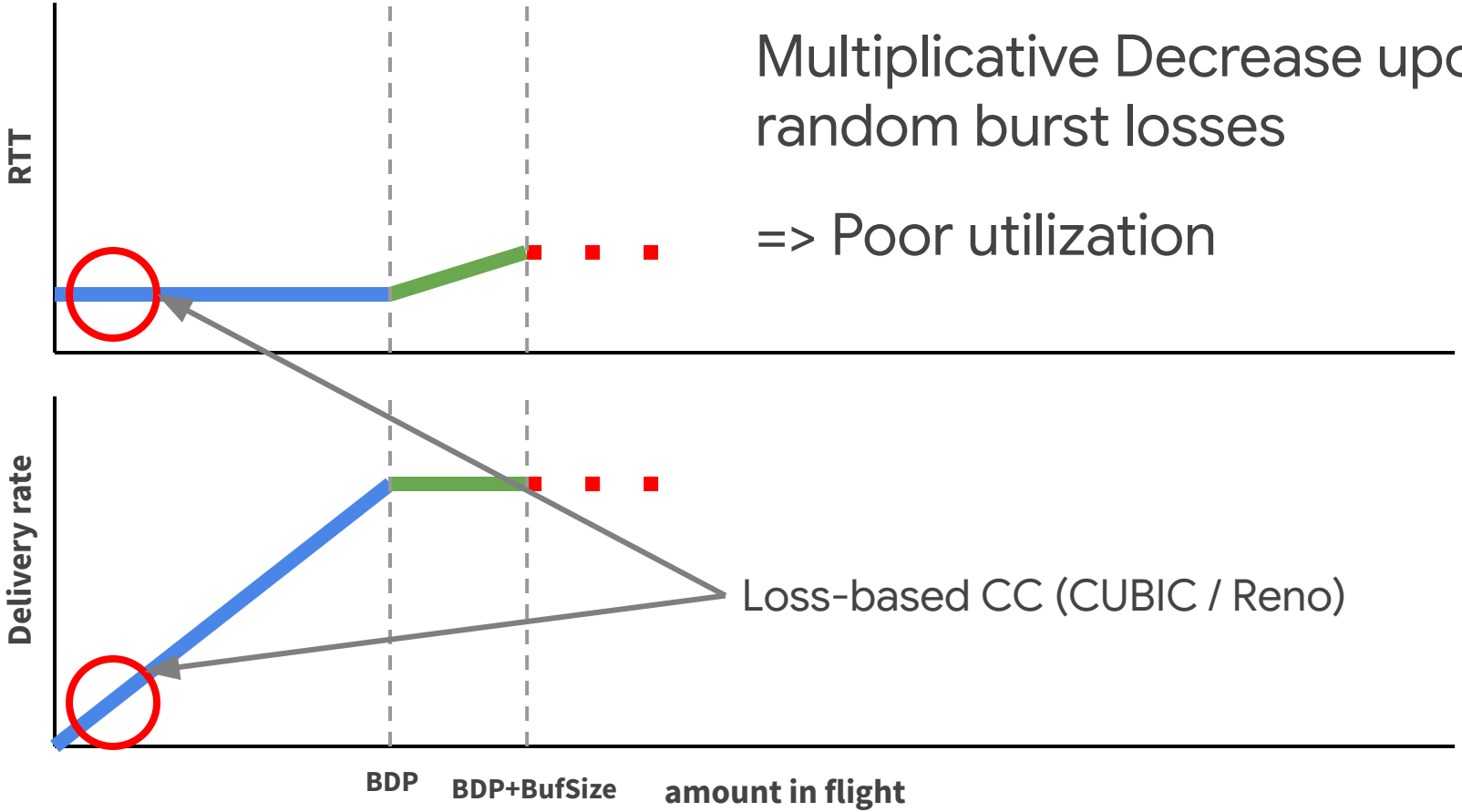
Network congestion and bottlenecks: a model



Loss-based congestion control in deep buffers



Loss-based congestion control in shallow buffers

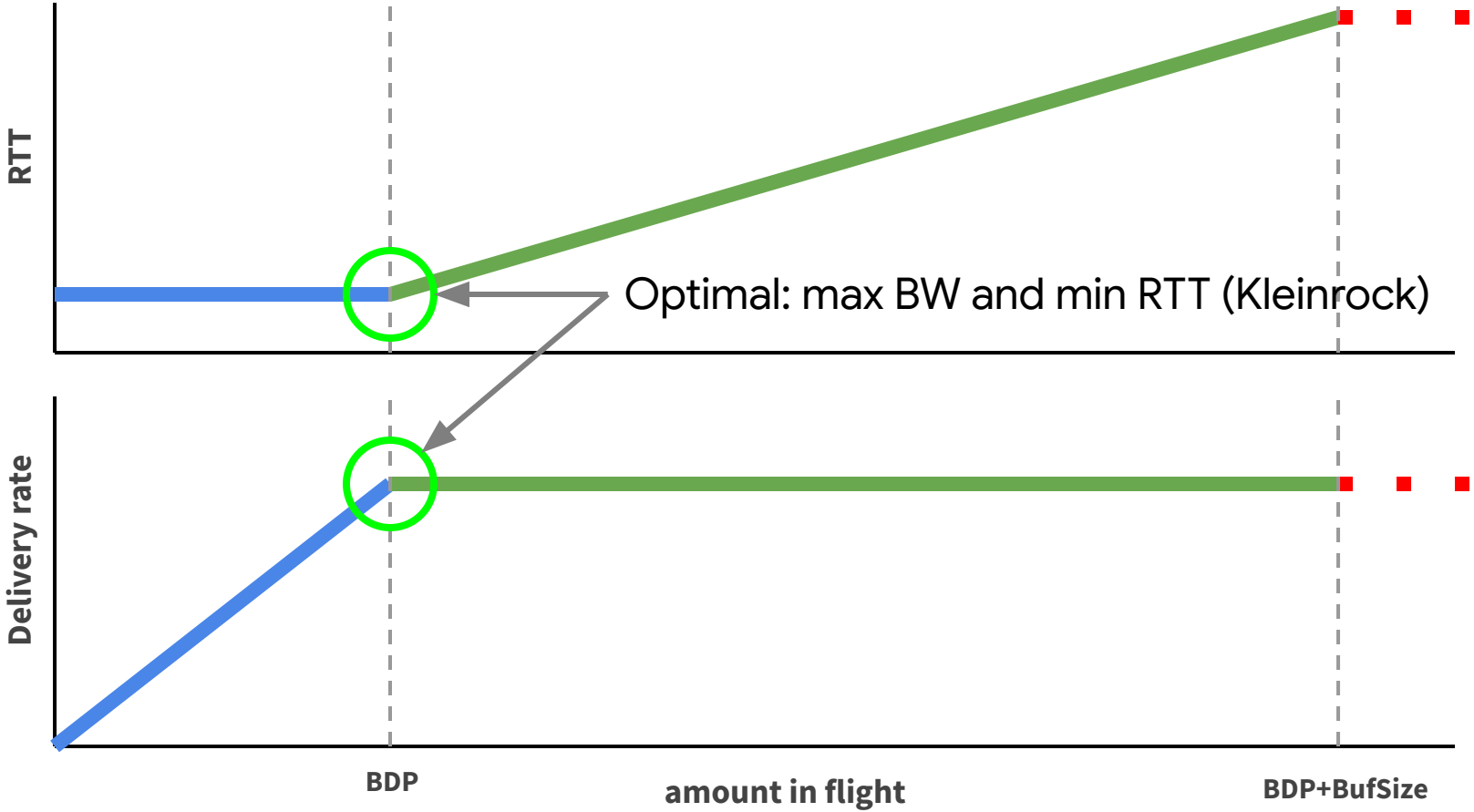


Multiplicative Decrease upon random burst losses

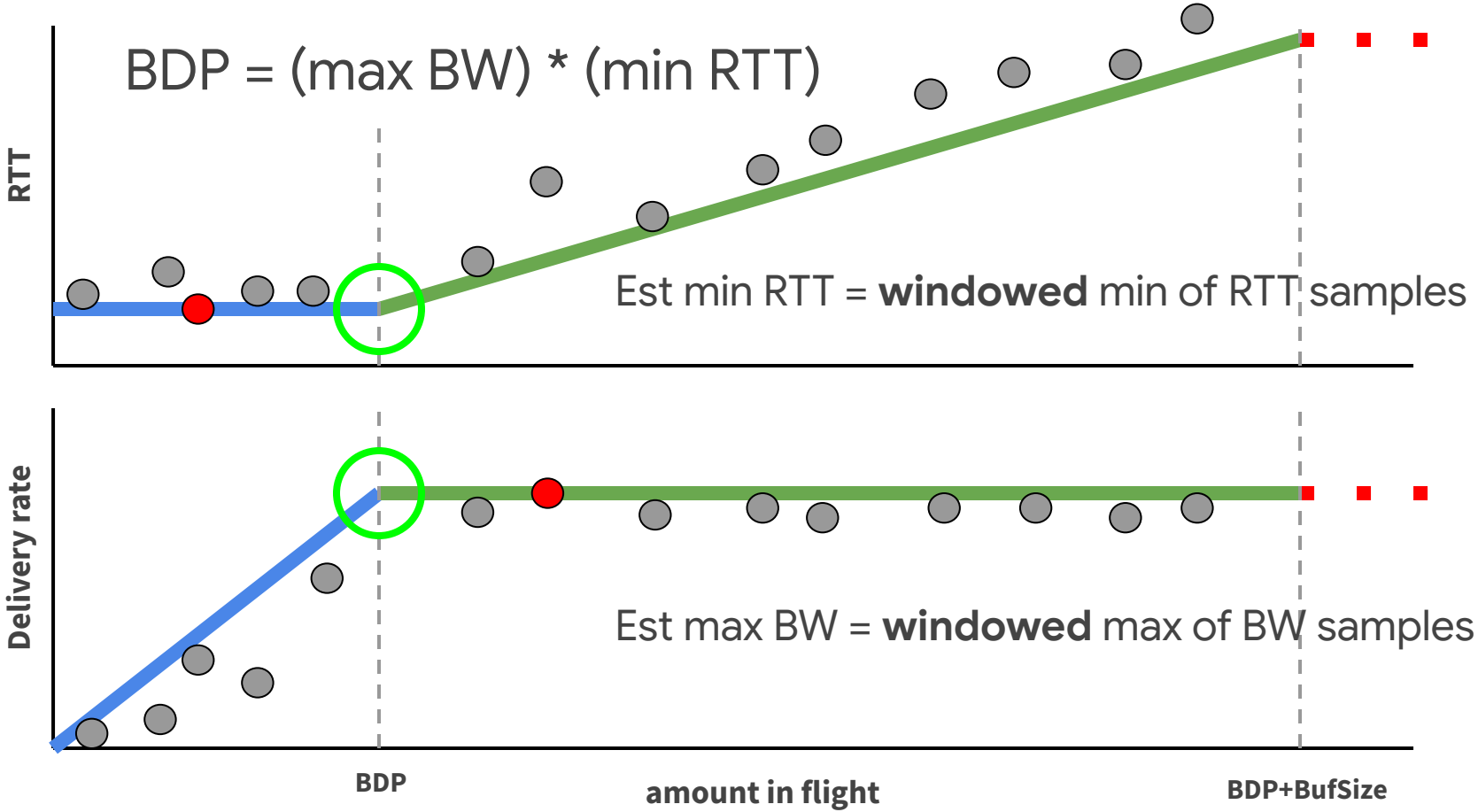
=> Poor utilization

Loss-based CC (CUBIC / Reno)

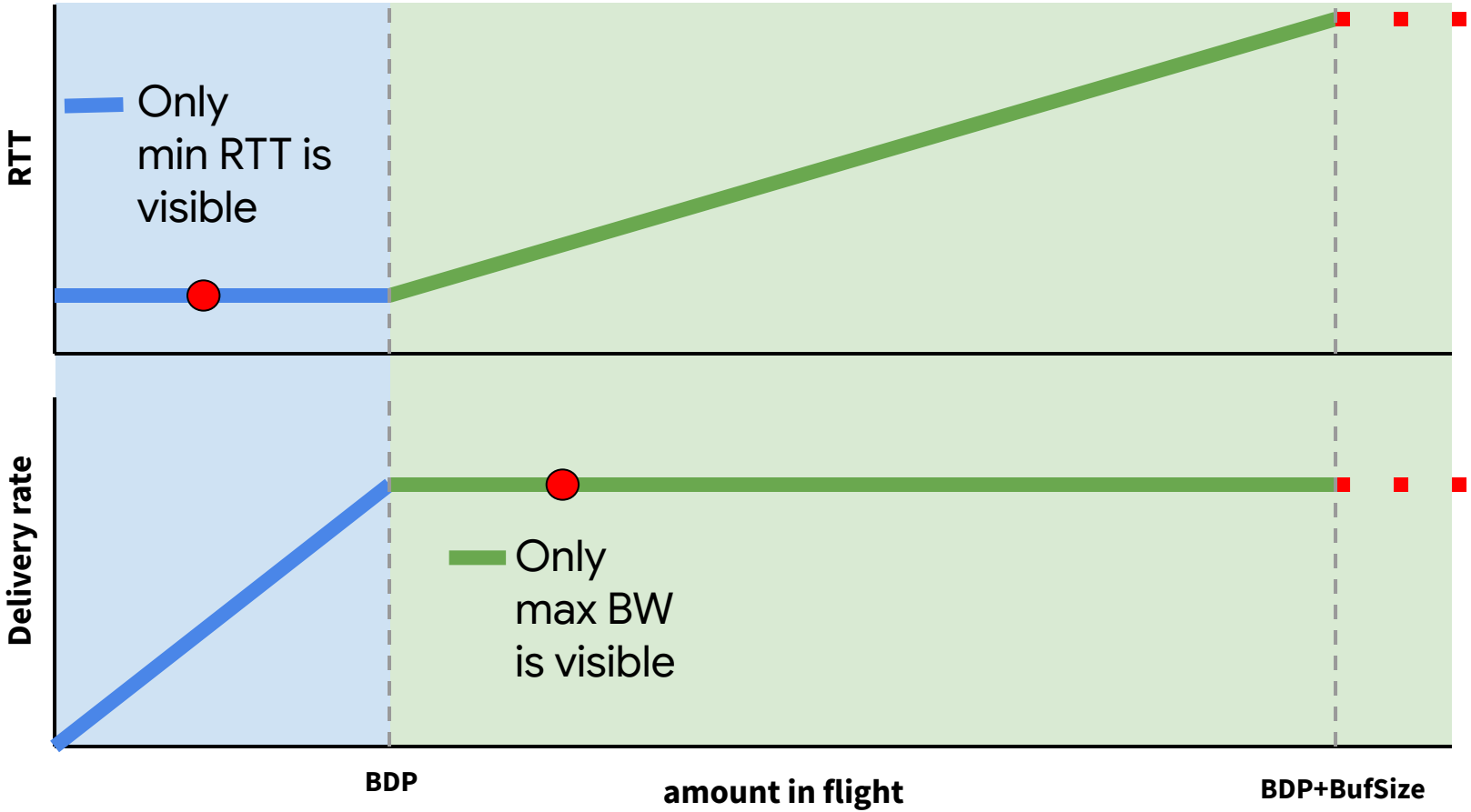
What is the optimal operating point?



Estimating optimal point (max BW, min RTT)



To see max BW, min RTT: probe both sides of BDP



Part 2:
BBR Congestion Control

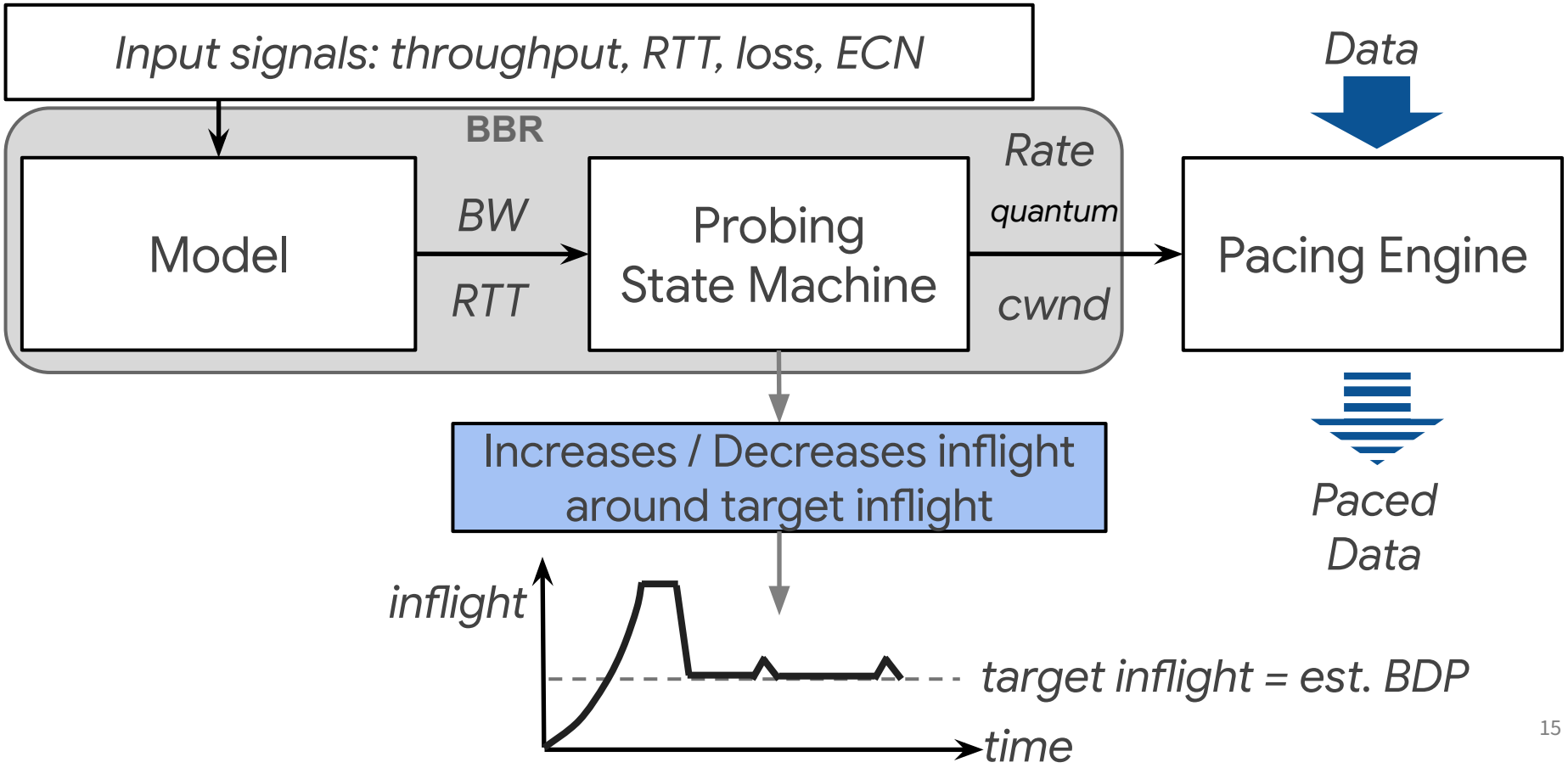
BBR (*B*ottleneck *B*andwidth and *R*ound-Trip Time)

- Dynamically **Model** network path: track windowed max BW and min RTT on each ACK
- **Control sending rate based on the model**
  - Pace near the estimated max BW
  - Bound in-flight data as a function of the estimated BDP

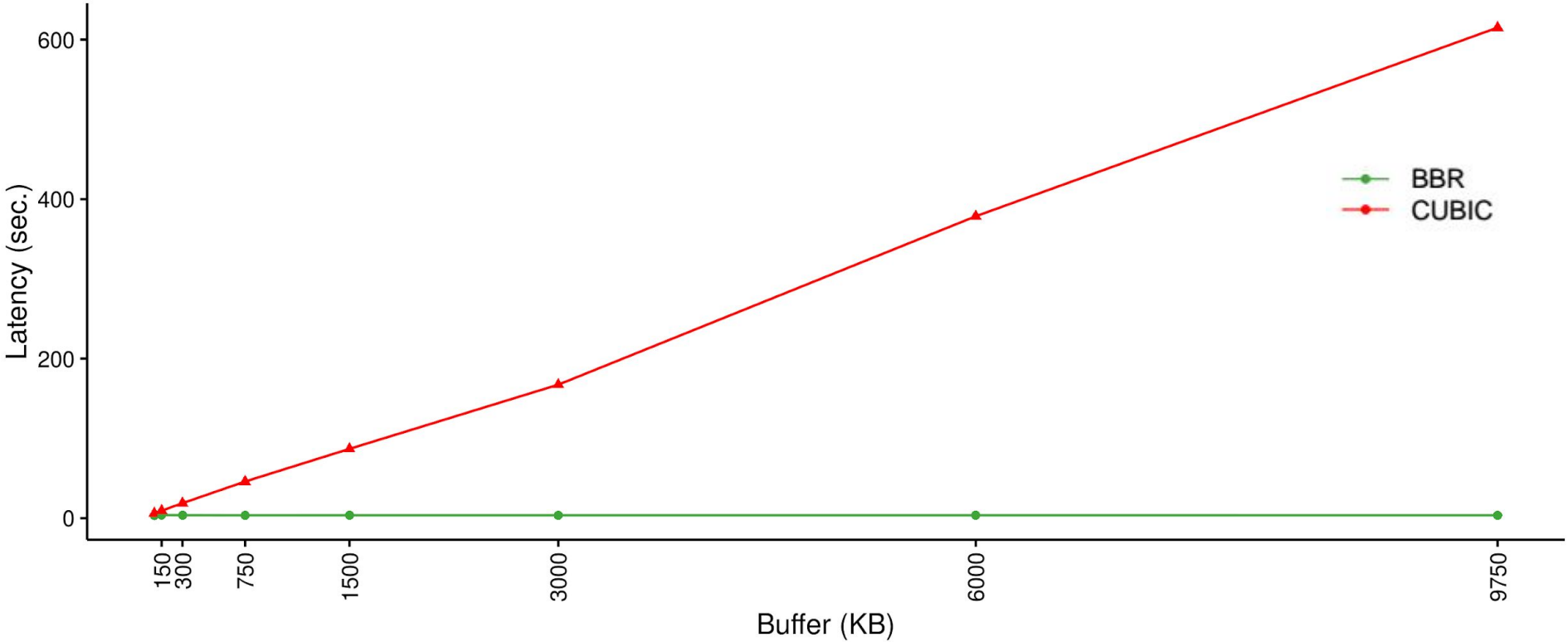
- **Sequentially** probe max BW and min RTT, to feed the model samples

- **Seek high throughput with a small queue**
  - Approaches maximum available throughput for a targeted level of random loss
  - Maintains small, bounded queue independent of buffer depth

BBR congestion control: the big picture

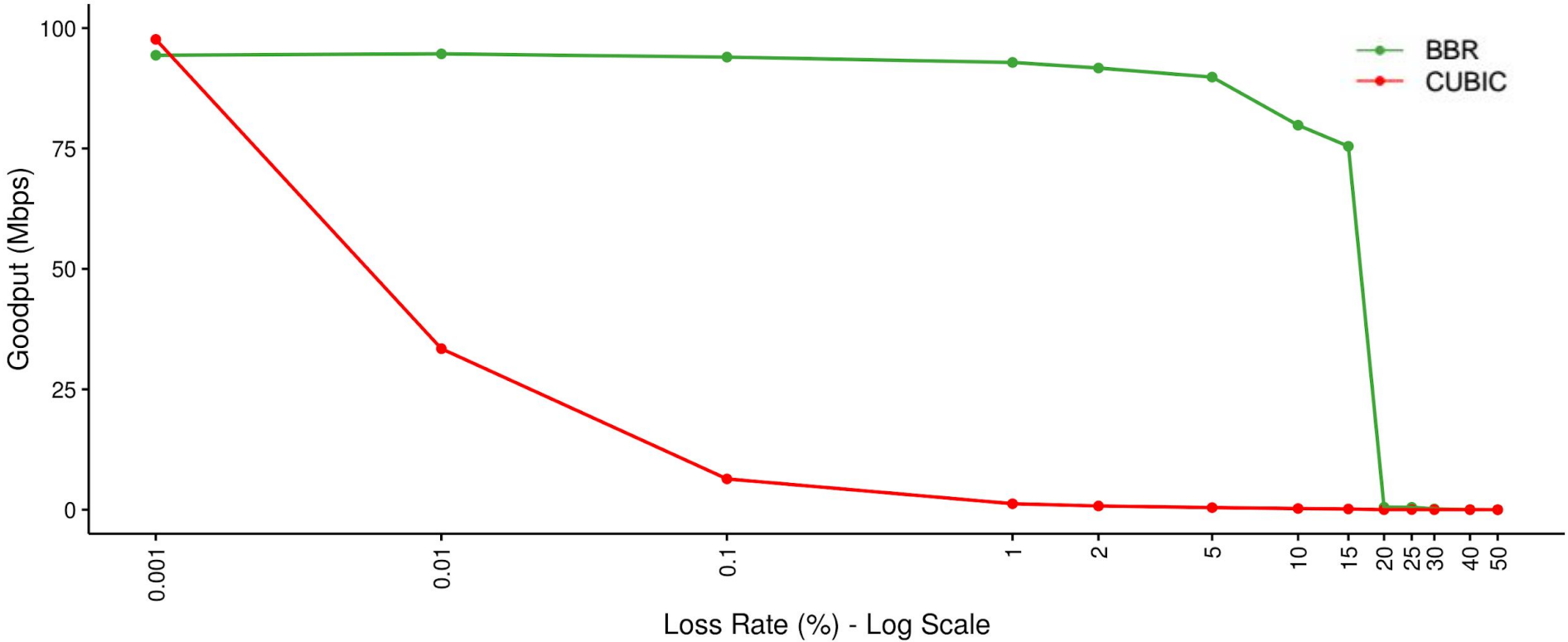


BBR: low queue delay, despite bloated buffers



BBR vs CUBIC: synthetic bulk TCP test with 8 flows, bottleneck_bw=128kbps, RTT=40ms

BBR: fully use bandwidth, despite targeted loss



BBR vs CUBIC: synthetic bulk TCP test with 1 flow, bottleneck_bw 100Mbps, RTT 100ms

BBRv2 in a nutshell

- Design goals:
  - High throughput with a targeted level of random packet loss
  - Bounded queuing delay, despite bloated buffers
  - **Usable coexistence when sharing Reno and CUBIC congestion control**
- Mechanisms:
  - Model-based: dynamically probes and models the network path
    - Models max bandwidth, min RTT, **max aggregation, max inflight**
  - Signals:
    - Bandwidth, RTT
    - **ECN (like DCTCP, L4S)**
    - **Loss (explicit loss rate cap of 2%)**

[New aspects since BBRv1 in **Bold**]

BBR history in a nutshell

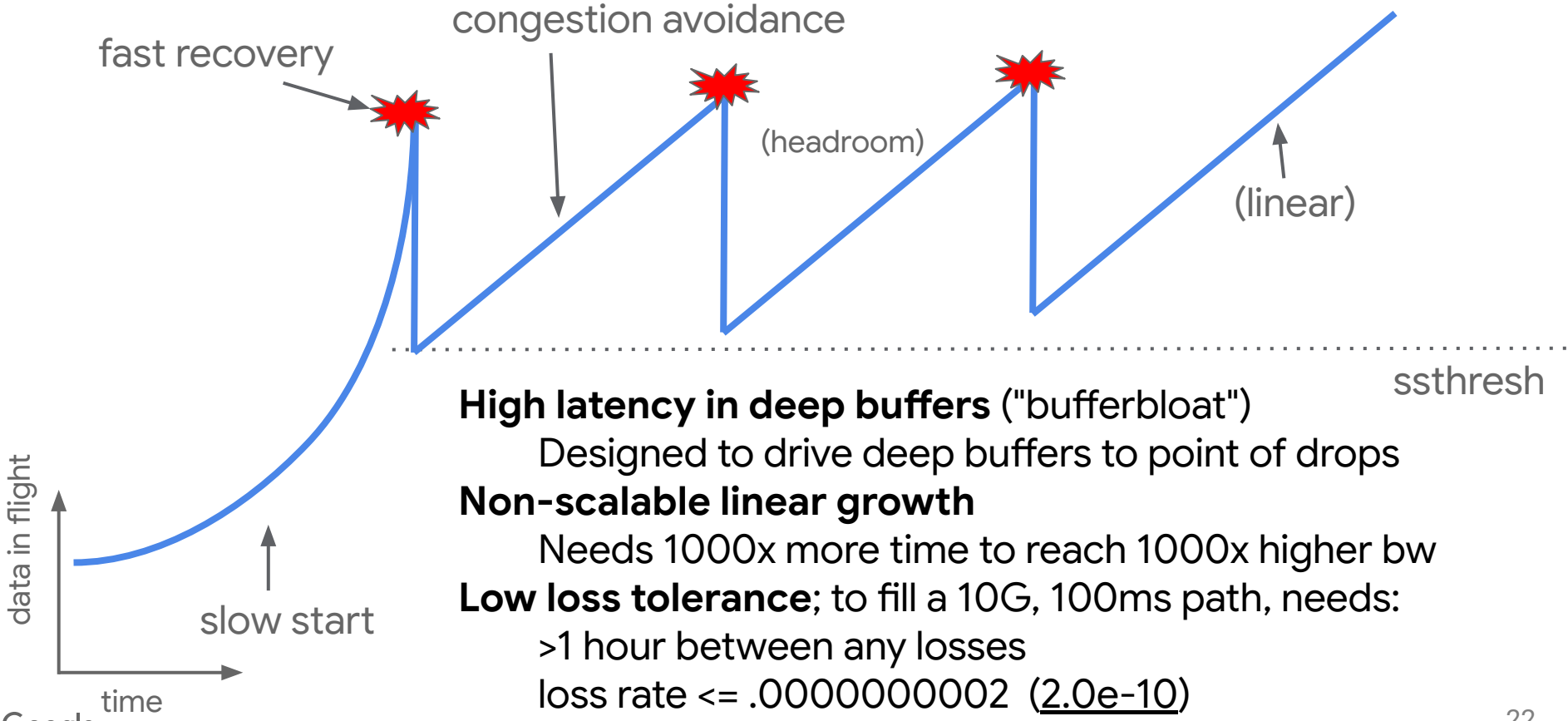
- BBRv1
  - Deployed for Google internal TCP WAN traffic in 2016
  - Open-sourced in 2016
  - Deployed for YouTube, google.com public Internet traffic in 2017
  - "The Great Internet TCP Congestion Control Census" [paper] [slides] (Dec 2019) estimated ~25% of top sites, ~40% of downstream Internet traffic use BBRv1
  - Deployed for Amazon CloudFront
- BBRv2
  - Open-sourced in 2019: https://github.com/google/bbr/blob/v2alpha/README.md
  - Was deployed at Google for >99% of *internal* TCP traffic 2021Q1 - 2022Q1
  - Currently used by Akamai and Dropbox
- BBRv3
  - BBRv2 + significant bug fixes described at CCWG session at IETF 117
  - Open-sourced in July 2023: https://github.com/google/bbr/blob/v3/README.md
  - Used for Google's internal TCP WAN traffic, and YouTube and google.com TCP

BBR at the IETF

- BBR is a Congestion Control Working Group (CCWG) "working group item"
- Target: publish an experimental RFC documenting the algorithm
- IETF working group members are collaborating on github
  - https://github.com/ietf-wg-ccwg/draft-ietf-ccwg-bbr
  - Ideas or suggestions? Feel free to file a github issue.
  - Specific editorial suggestions? Feel free to propose a pull request.
- BBR Internet Draft name: draft-ietf-ccwg-bbr
  - https://datatracker.ietf.org/doc/draft-ietf-ccwg-bbr/
- Editors:
  - Neal Cardwell (Google)
  - Ian Swett (Google)
  - Joseph Beshay (Meta)

Part 3:
Comparing
Congestion Control Algorithms
For the Public Internet

Reno (1988-now; still IETF standard; Netflix)



High latency in deep buffers ("bufferbloat")

Designed to drive deep buffers to point of drops

Non-scalable linear growth

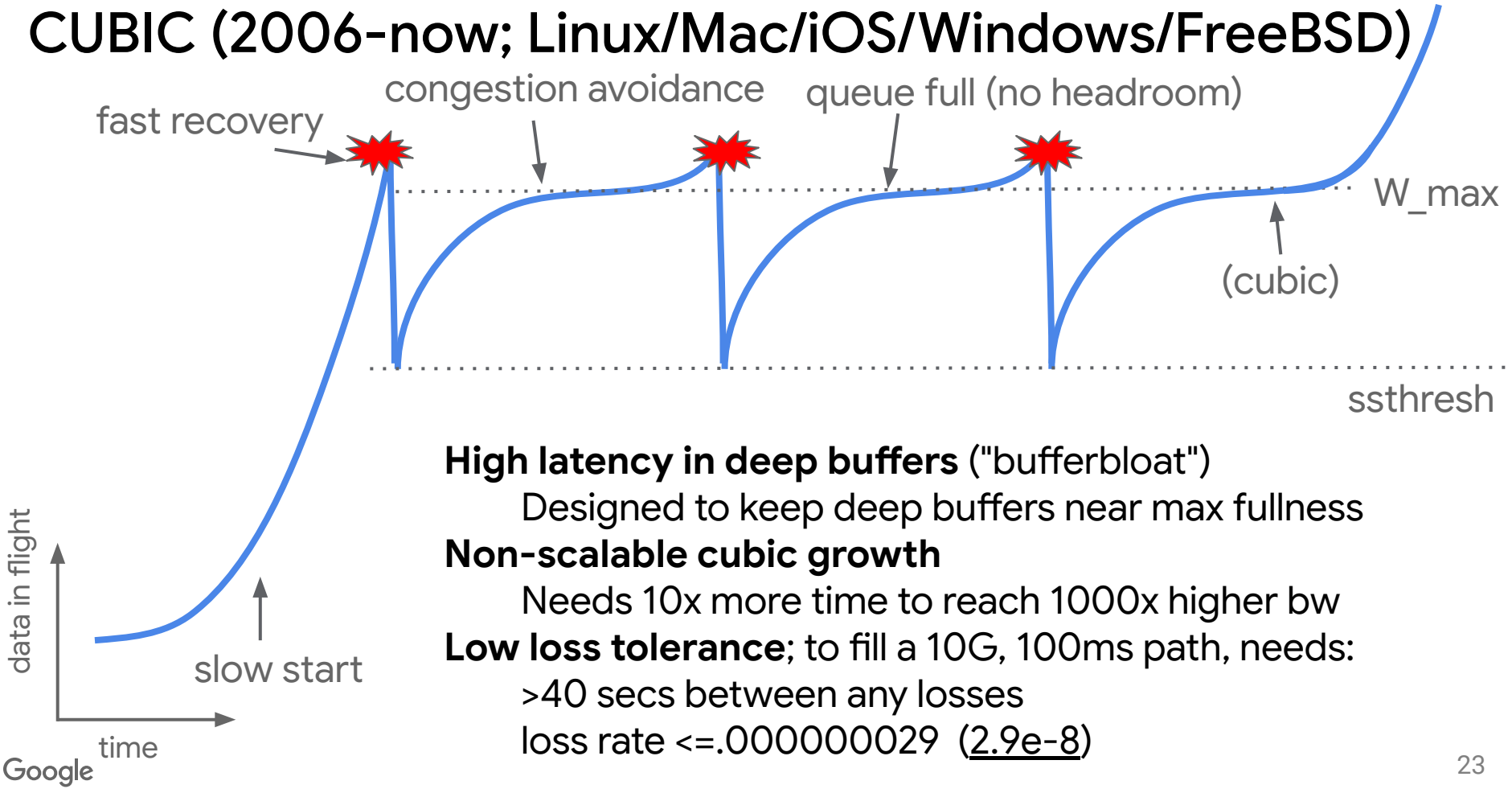
Needs 1000x more time to reach 1000x higher bw

Low loss tolerance; to fill a 10G, 100ms path, needs:

>1 hour between any losses

loss rate <= .0000000002 (2.0e-10)

CUBIC (2006-now; Linux/Mac/iOS/Windows/FreeBSD)



High latency in deep buffers ("bufferbloat")

Designed to keep deep buffers near max fullness

Non-scalable cubic growth

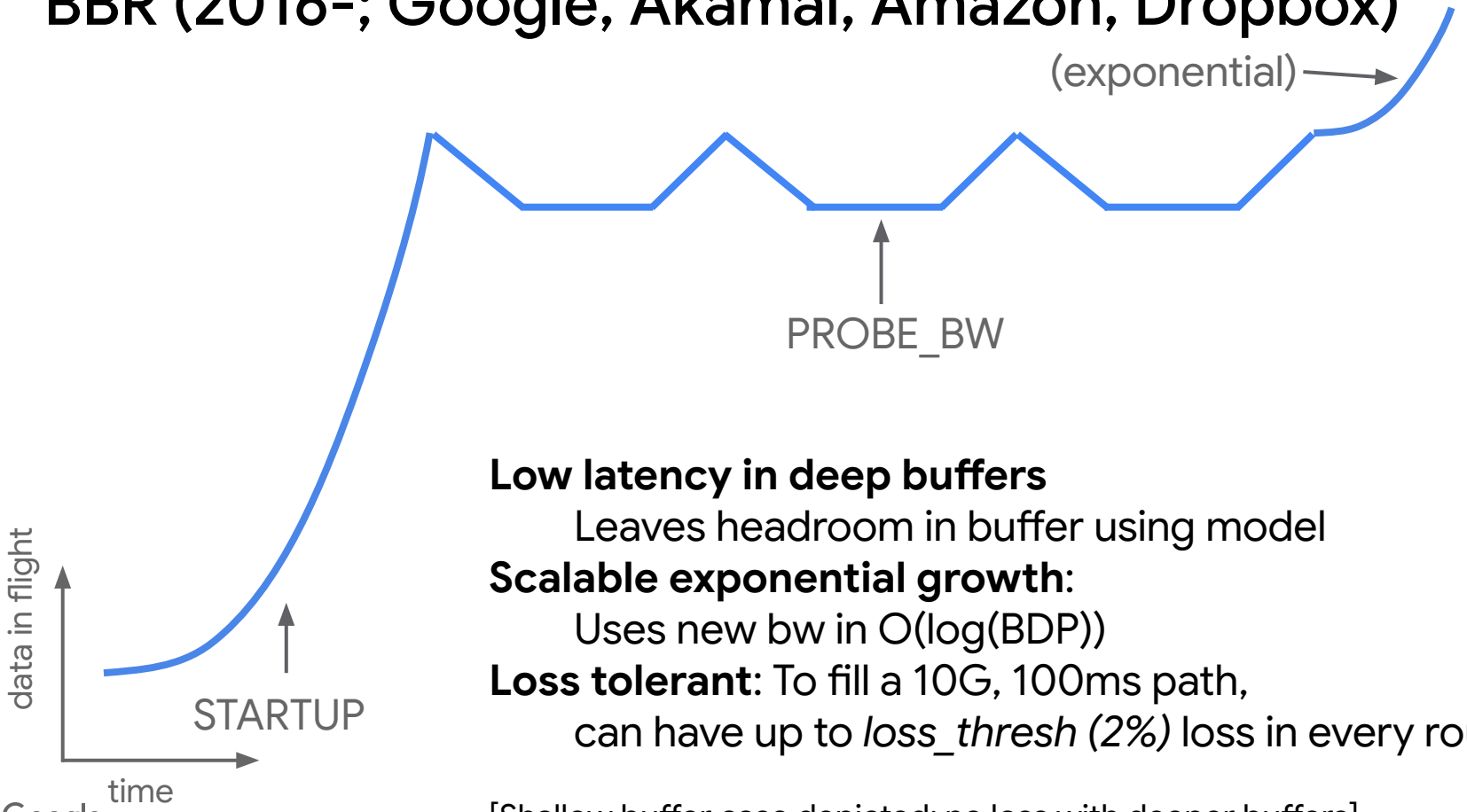
Needs 10x more time to reach 1000x higher bw

Low loss tolerance; to fill a 10G, 100ms path, needs:

>40 secs between any losses

loss rate <=.000000029 (2.9e-8)

BBR (2016-; Google, Akamai, Amazon, Dropbox)



Low latency in deep buffers
Leaves headroom in buffer using model

Scalable exponential growth:
Uses new bw in O(log(BDP))

Loss tolerant: To fill a 10G, 100ms path,
can have up to *loss_thresh (2%)* loss in every round

[Shallow buffer case depicted; no loss with deeper buffers]

Conclusion

- BBR is a model-based congestion control
  - Evolved to use a rich model and diverse signals for diverse environments
- Deployed at Google, YouTube, Amazon, Akamai, Dropbox
- Code status: open source in Linux TCP, QUIC; latest is BBRv3 on GitHub
- IETF status: CCWG "working group item" (target: experimental RFC)
- Documentation and discussion:
  - Paper in Feb 2017 CACM
  - Independent study of BBR and other C.C.s by Stanford: pantheon.stanford.edu
  - Mailing list and landing page: https://groups.google.com/g/bbr-dev
- In progress: support for L4S ECN, reducing latency/loss, improving coexistence
- We welcome feedback and contributions from the research and user community...

... Thanks!