OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

CT 420 Real-Time Systems

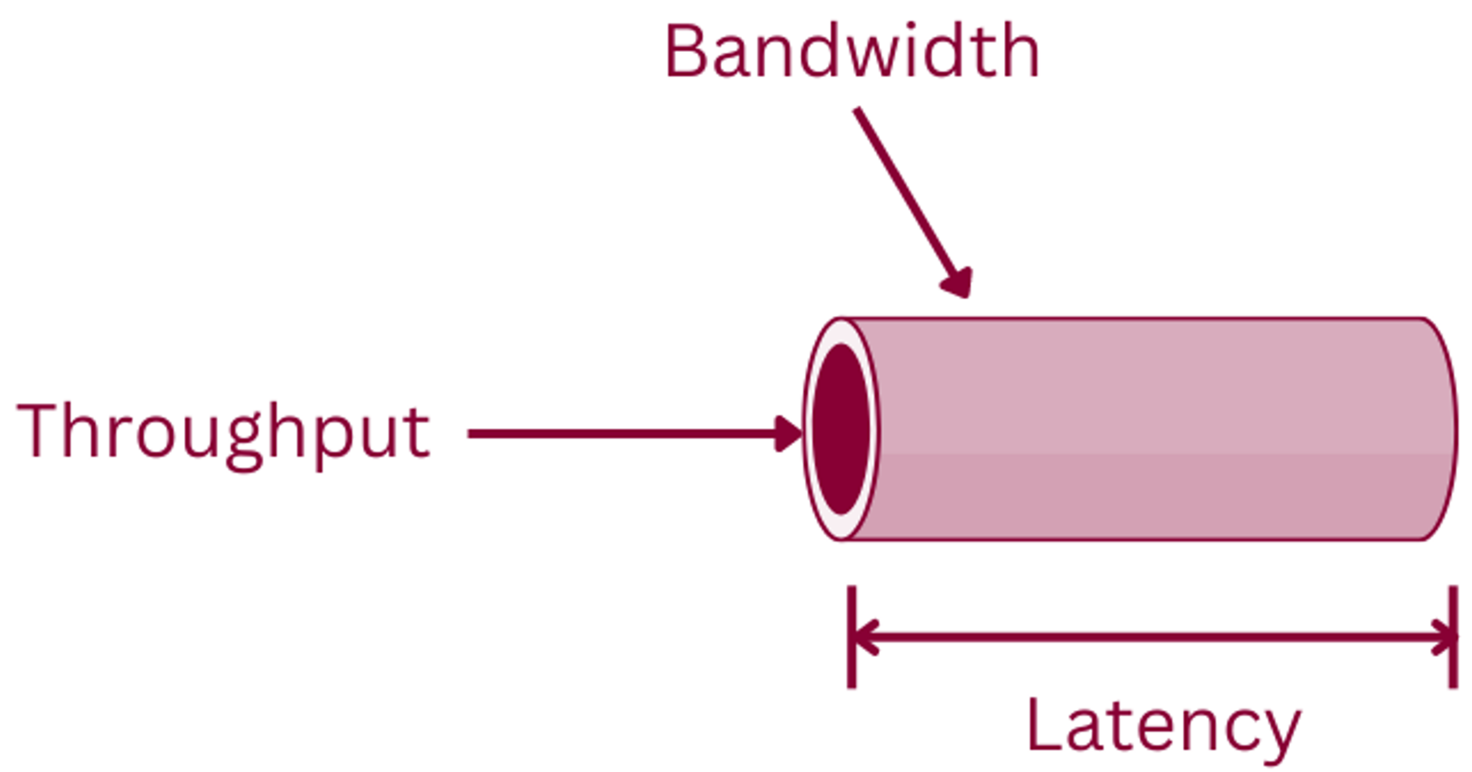
Congestion Control in QUIC

Dr. Jawad Manzoor
Assistant Professor
School of Computer Science

Contents

Web Performance Metrics

- Web performance is usually measured using the following network aspects:
  - Latency
  - Bandwidth
  - Throughput



Latency

- We will often need quite a few round trips to load even a single file, due to features such as congestion control.
- Even low latencies of less than 50 milliseconds can add up to considerable delays.
- This is one of the main reasons why content delivery networks (CDNs) exist: They place servers physically closer to the end user in order to reduce latency, and thus delay, as much as possible.

Bandwidth

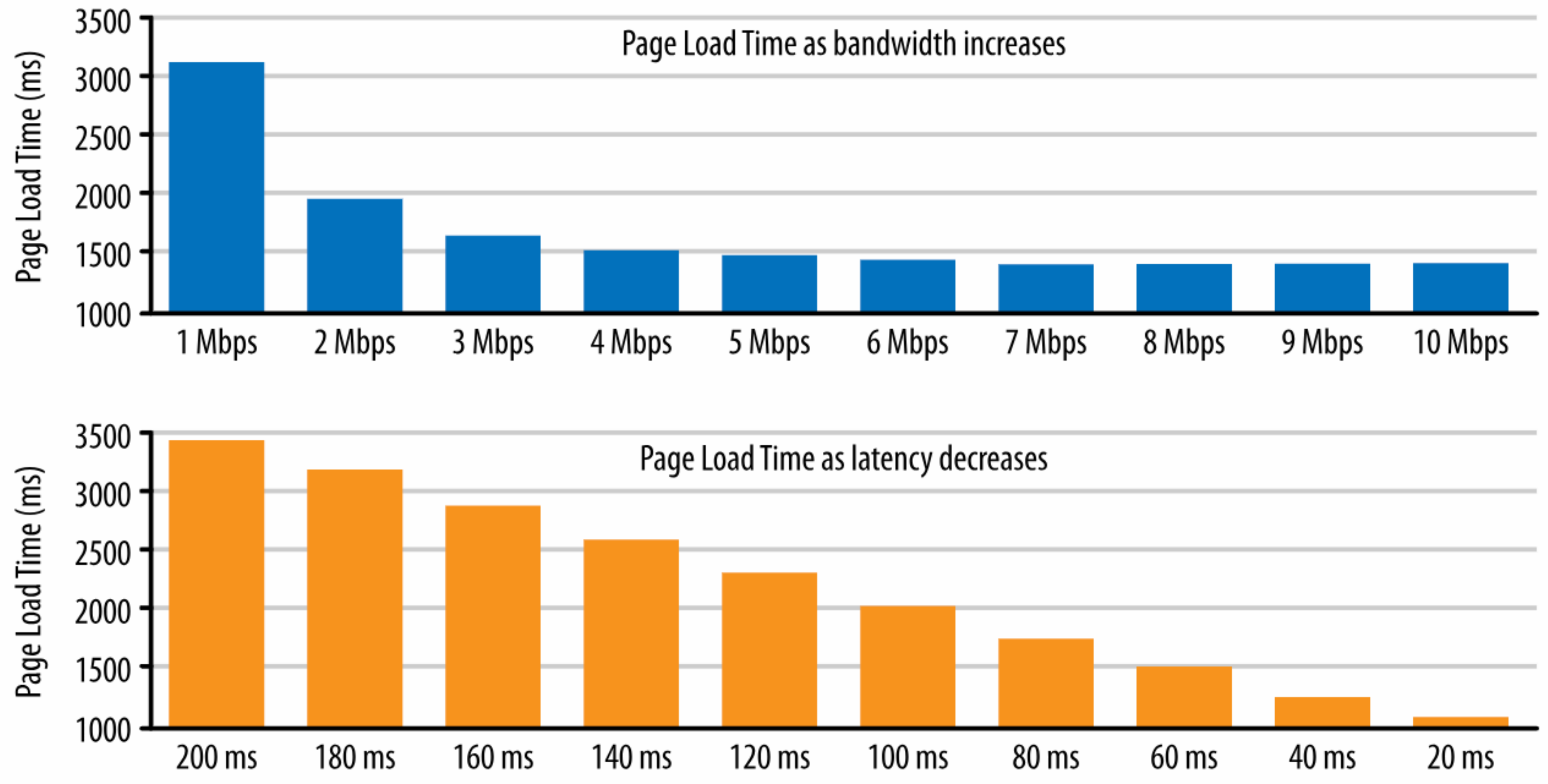
- Bandwidth measures the amount of data that is able to pass through a network at a given time.
- It is measured in bits per second (bps), such as megabits per second (Mbps) and gigabits per second (Gbps).

Throughput

- Throughput is the number of data packets that are able to reach their destination within a specific period of time.
- Throughput can be affected by a number of factors, including bus or network congestion, latency, packet loss/errors, and the protocol used.
  - The system will likely experience low throughput even with a high bandwidth

Performance Bottleneck

- Access to higher bandwidth data rates is always good
  - video and audio streaming
  - large data transfer
- However, latency is the limiting factor for everyday web browsing
  - requires fetching hundreds of relatively small resources from dozens of different hosts



Class Activity

- How do bandwidth and latency affect these applications?
- Cloud Gaming
  - Bandwidth: Medium (Requires around 5Mbps per device)
  - Latency: High impact (vital to a good experience of online games—especially in fast-paced games)
- Streaming
  - Bandwidth: High (4K stream averages around 25Mbps)
  - Latency: Medium impact
- Video chat
  - Bandwidth: Medium impact (Requires around 5Mbps. Low bandwidth will reduce the quality of your chat e.g. FaceTime or Skype
  - Latency: High impact (High latency causes sync issues and freezing)
- Browsing
  - Bandwidth: Medium impact
  - Latency: High impact (High delay causes long page load times and makes websites feel unresponsive)

Deciding Sending Rate

- **Why can't we transmit data between end points at the highest rate?**

- One aspect of performance is about how efficiently a transport protocol can use a network's full (physical) bandwidth.
- Reliable transports don't just start sending data at full rate, because this could end up congesting the network.
- Each network link has only a certain amount of data it can process every second.
  - Sending excessive data will lead to packet loss
  - Lost packets need to be retransmitted and can seriously affect performance in high latency networks.

Estimating Link Capacity

- **How to know about the available bandwidth between end-points?**

- We don’t know up front how much the maximum bandwidth will be.
- It depends on a bottleneck somewhere in the end-to-end connection, but we cannot predict or know where this will be. The Internet also doesn’t have mechanisms (yet) to signal link capacities back to the endpoints.
- Even if we knew the available physical bandwidth, that wouldn’t mean we could use all of it ourselves.
  - Several users are typically active on a network concurrently, each of whom need a fair share of the available bandwidth.

Congestion Control

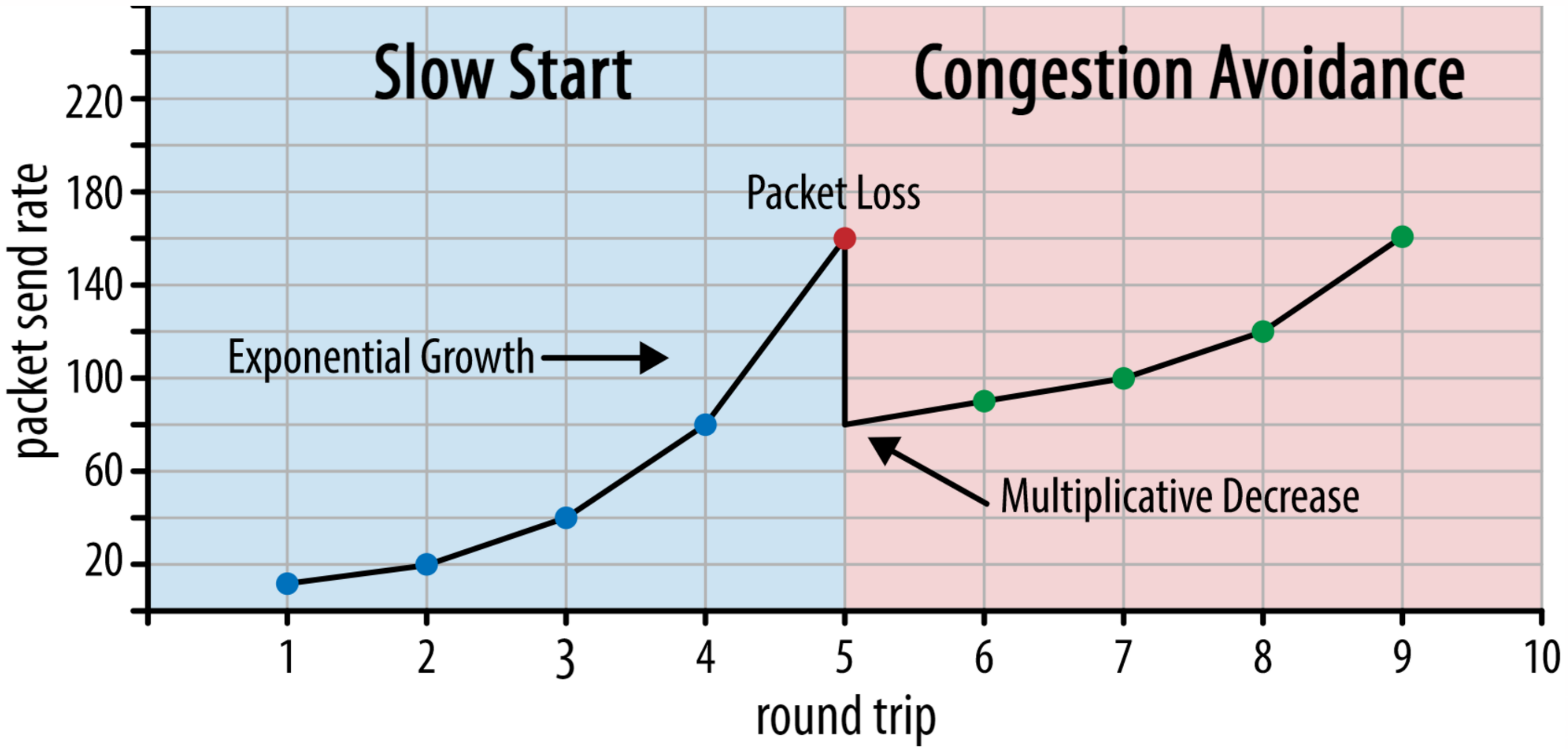
- In the 1980s when the internet was still run by the government and TCP was young, engineers were learning about bad TCP behavior and networks.
- At the time TCP with no congestion control, would occasionally "collapse" whole segments of the internet.
- Clients were programmed to respect the capacity of the machine at the other end of the connection (flow control) but not the capacity of the network.
- In 1986 in an especially bad incident, backbone links across the network, under incredible congestion, passed only 40bps, 1/1000th of their rated 32kbps capacity.
- The internet was suffering congestion collapse.

Congestion Control

- In the network transport area, congestion control is how to decide how much data the connection can send into the network.
- Reliable transport protocols such as TCP and QUIC constantly try to discover the available bandwidth over time by using congestion control algorithm.

- Terminology
- MSS
  - Maximum segment size, is the largest data payload that a device will accept from a network connection
- CWND size
  - The congestion window (cwnd) is a sender-side limit on the amount of data the sender can transmit into the network before receiving an acknowledgment (ACK)

Congestion Control



Slow Start Phase

- A connection is started slow
- It waits one round trip to receive acknowledgements of these packets
- If they are all acknowledged, this means the network has capacity, and the send rate is increased every iteration (usually doubled).

CWND Growth in Slow Start

- MSS = 1460 bytes
- CWND = 10 segments
- What is the CWND size in KB after 5 round trips?

Round Trip	MSS	CWND (# segments)	CWND (KB)
1	1460	10	14
2	1460	20	28
3	1460	40	57
4	1460	80	114
5	1460	160	228

Congestion Avoidance Phase

- The send rate continues to grow until some packets are not acknowledged (which indicates packet loss and network congestion).
- On packet loss, the send rate is slashed, and afterward it is gradually increased in much smaller increments.
- This reduce-then-grow logic is repeated for every packet loss.

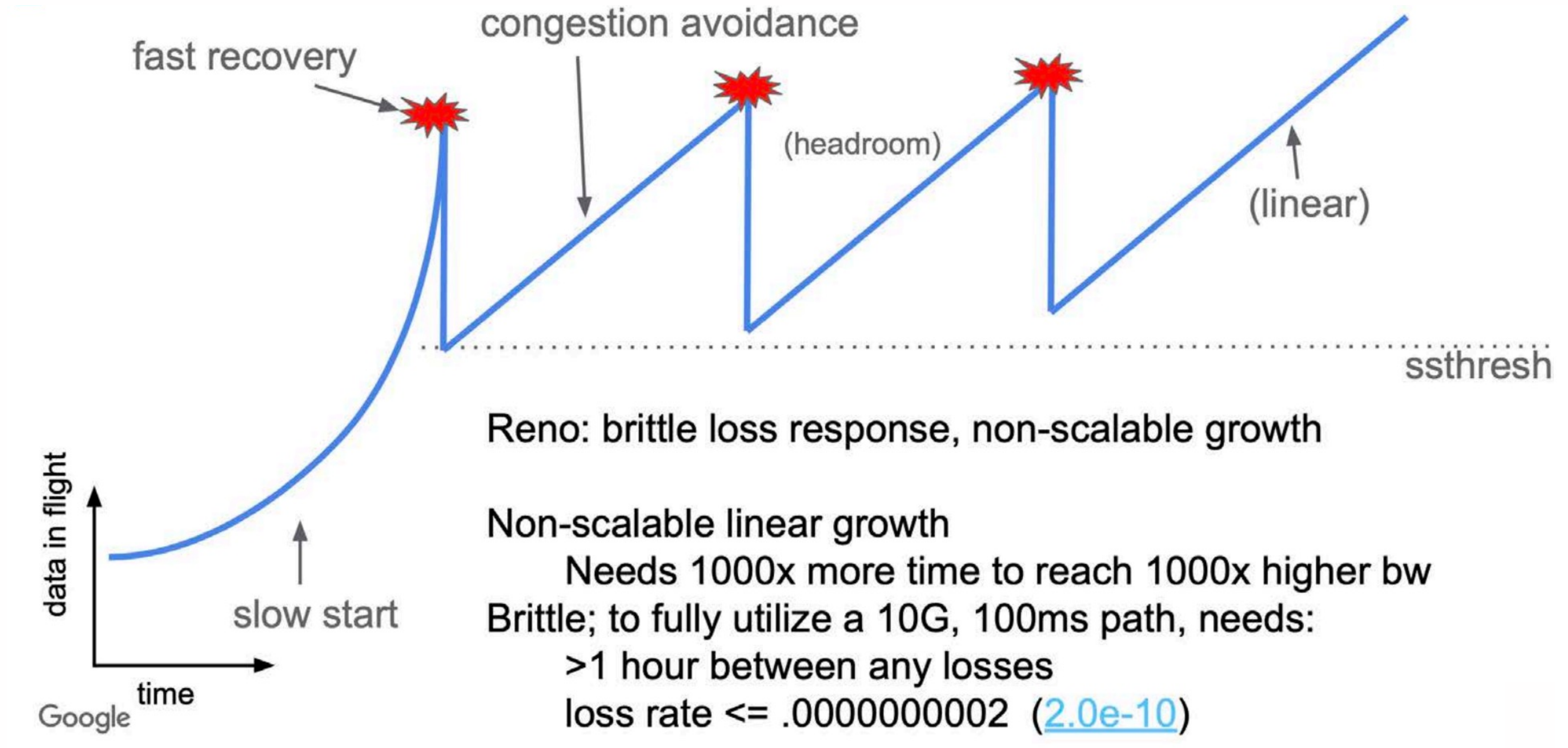
Congestion Control Strategies

- Largely there are two categories:
- Loss-based congestion control - where the congestion control responds to a packet loss event. E.g. Reno and CUBIC
- Delay-based congestion control - the algorithm tries to find a balance between the bandwidth and RTT increase and tune the packet send rate. E.g. Vegas and BBR

Reno

- Reno (often referred as NewReno) is a standard congestion control for TCP and QUIC .
- Reno starts from "slow start" mode which increases the CWND (limits the amount of data a TCP can send) roughly 2x for every RTT until the congestion is detected.
- When packet loss is detected, it enters into the “recovery” mode until packet loss is recovered.
- When it exits from recovery (no lost ranges) and CWND > SSTHRESH (slow start threshold), it enters into the "congestion avoidance" mode where the CWND grows slowly (roughly a full packet per RTT) and tries to converge on a stable CWND.
- A “sawtooth” pattern is observed when you make a graph of the CWND over time.

Reno



Reno: brittle loss response, non-scalable growth

Non-scalable linear growth

Needs 1000x more time to reach 1000x higher bw

Brittle; to fully utilize a 10G, 100ms path, needs:

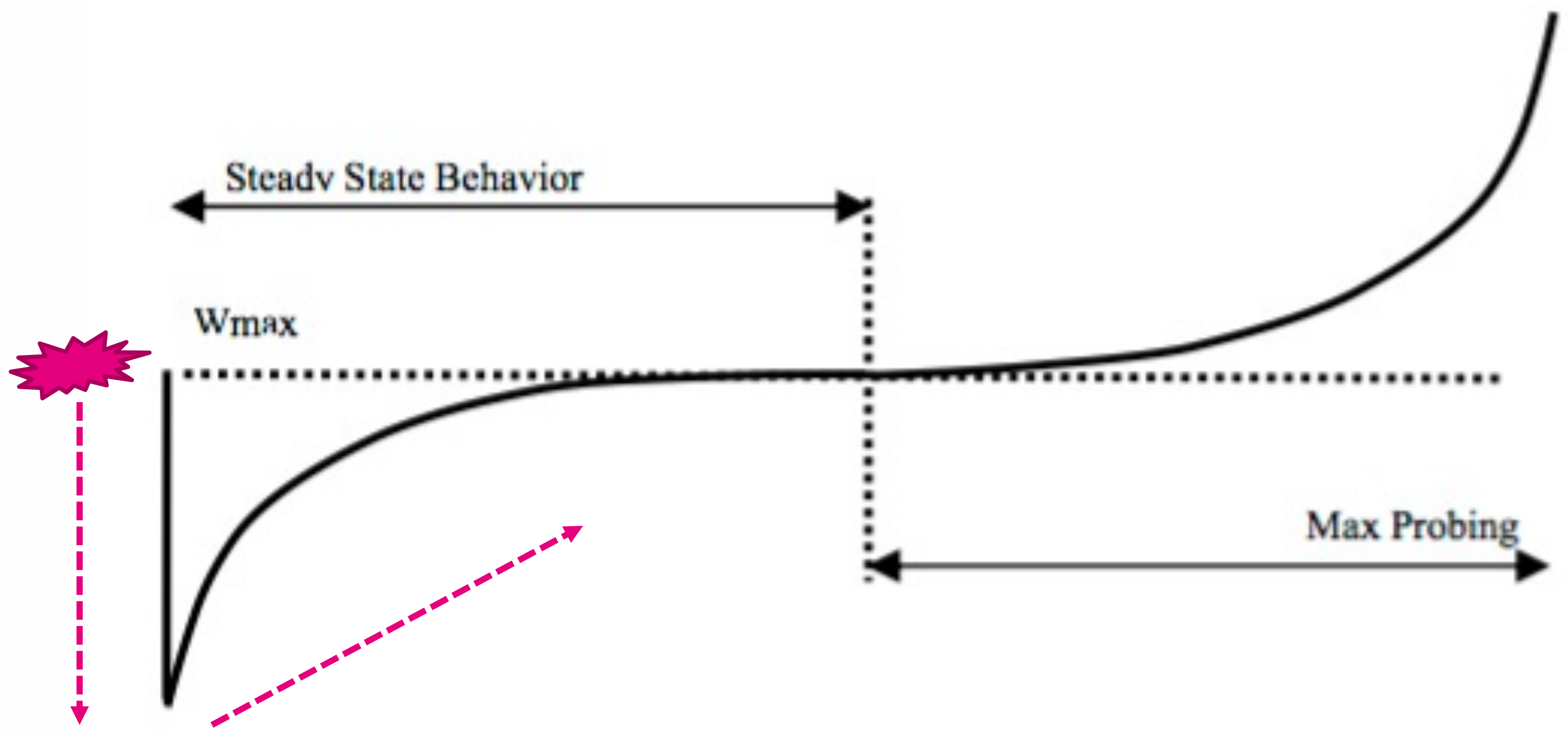
>1 hour between any losses

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

Slide from Google presentation at IETF 104

Cubic

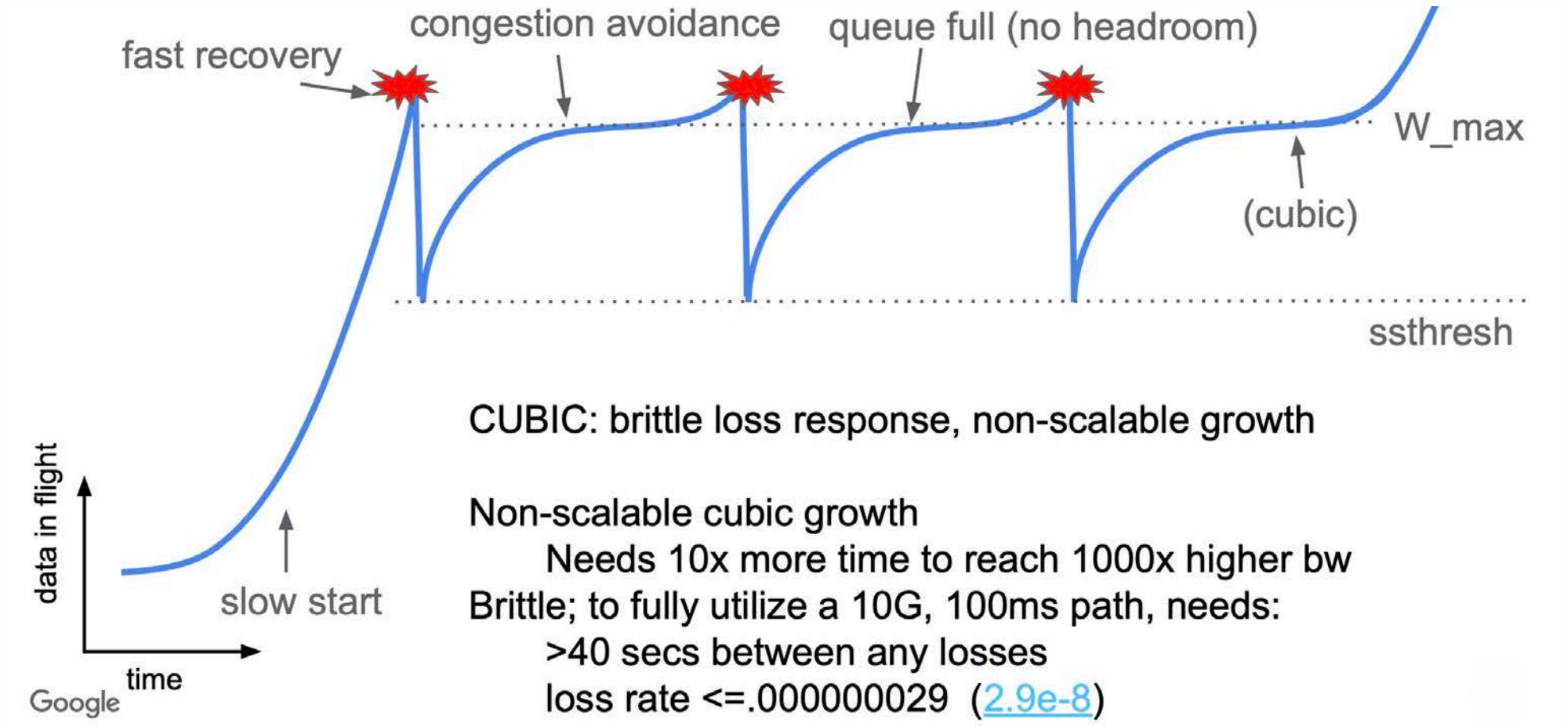
- Cubic is defined in RFC8312 and implemented in many OS including Linux, BSD and Windows.
- The difference from Reno is that during congestion avoidance its CWND growth is based on a cubic function instead of a linear function as follows:



Cubic

- Wmax is the value of CWND when the congestion is detected.
- Then it will reduce the CWND by 30% and then the CWND starts to grow again using a cubic function as in the graph, approaching Wmax aggressively in the beginning in the first half but slowly converging to Wmax later.
- This makes sure that CWND growth approaches the previous point carefully and once we pass Wmax, it starts to grow aggressively again after some time to find a new CWND (this is called "Max Probing").

Cubic



CUBIC: brittle loss response, non-scalable growth

Non-scalable cubic growth

Needs 10x more time to reach 1000x higher bw

Brittle; to fully utilize a 10G, 100ms path, needs:

>40 secs between any losses

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

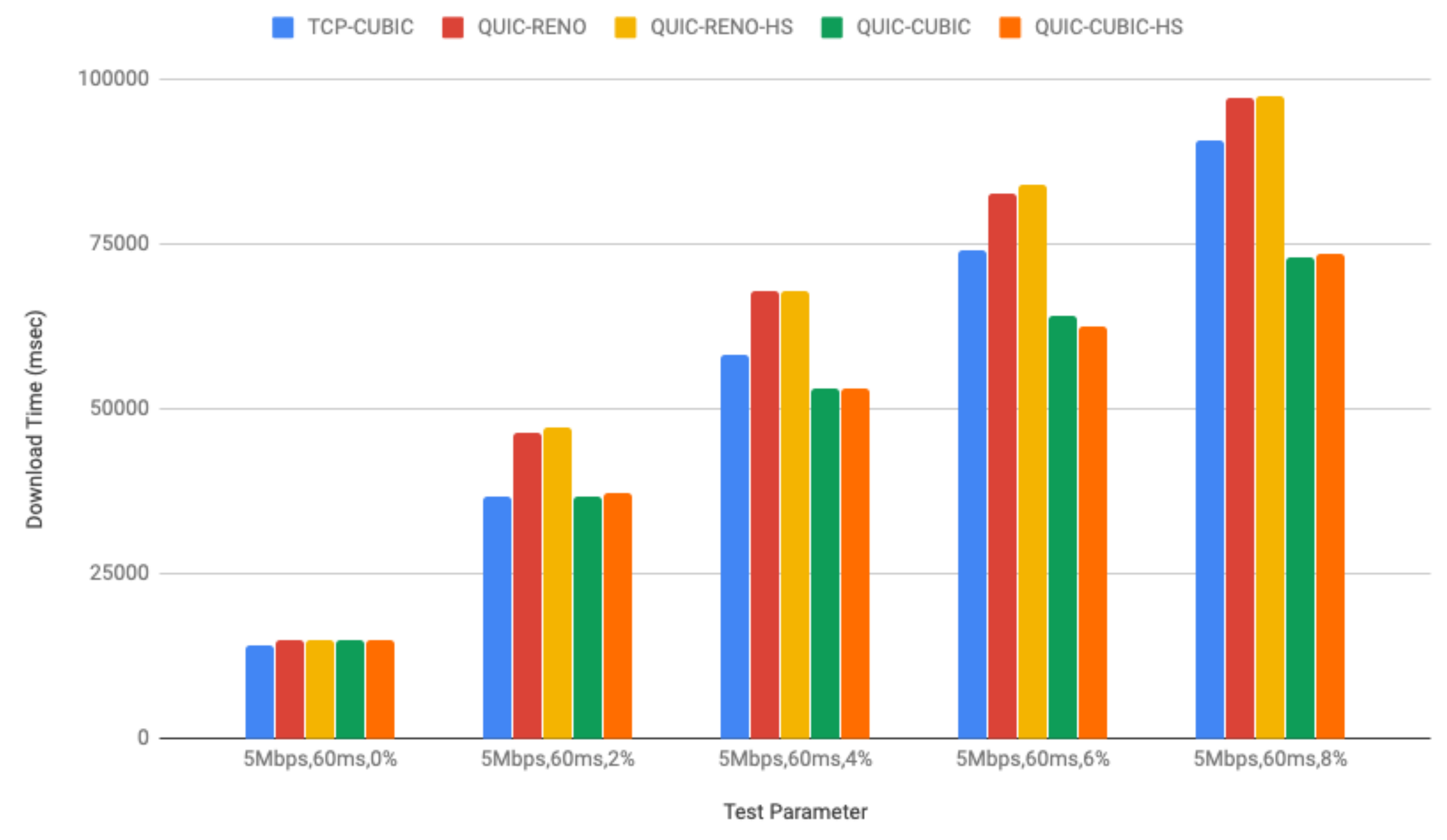
Slide from Google presentation at IETF 104

HyStart++

- The authors of CUBIC made a separate effort to improve slow start
- It is based on RTT delay samples - when the RTT is increased during slow start and over the threshold, it exits slow start early and enters into LSS (Limited Slow Start).
- LSS grows the CWND faster than congestion avoidance but slower than Reno slow start.

QUIC vs TCP congestion control

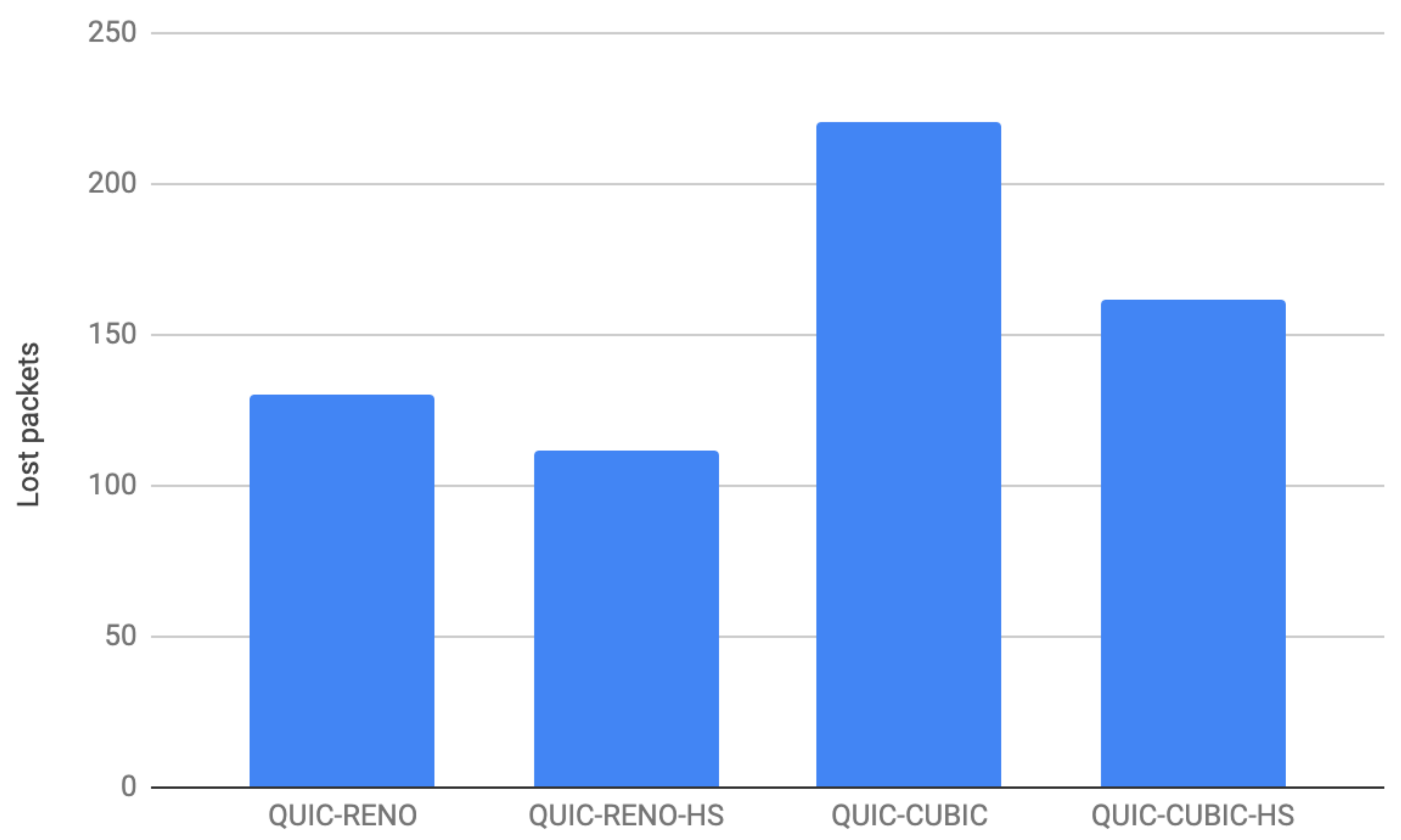
- Performance comparison of QUIC congestion control variants.
  - with 0% packet loss TCP and QUIC are almost doing the same
  - As packet loss increases QUIC CUBIC performs better than TCP CUBIC.
  - With HyStart++, overall performance is the same



Source: cloudflare.com

QUIC vs TCP congestion control

- Performance comparison of QUIC congestion control variants.
  - Compared with Reno, CUBIC can create more packet loss in general.
  - HyStart++ significantly reduces packet loss.



Source: cloudflare.com

Question

- Is packet loss always an indication of congestion?
- Are there any other causes of packet loss?

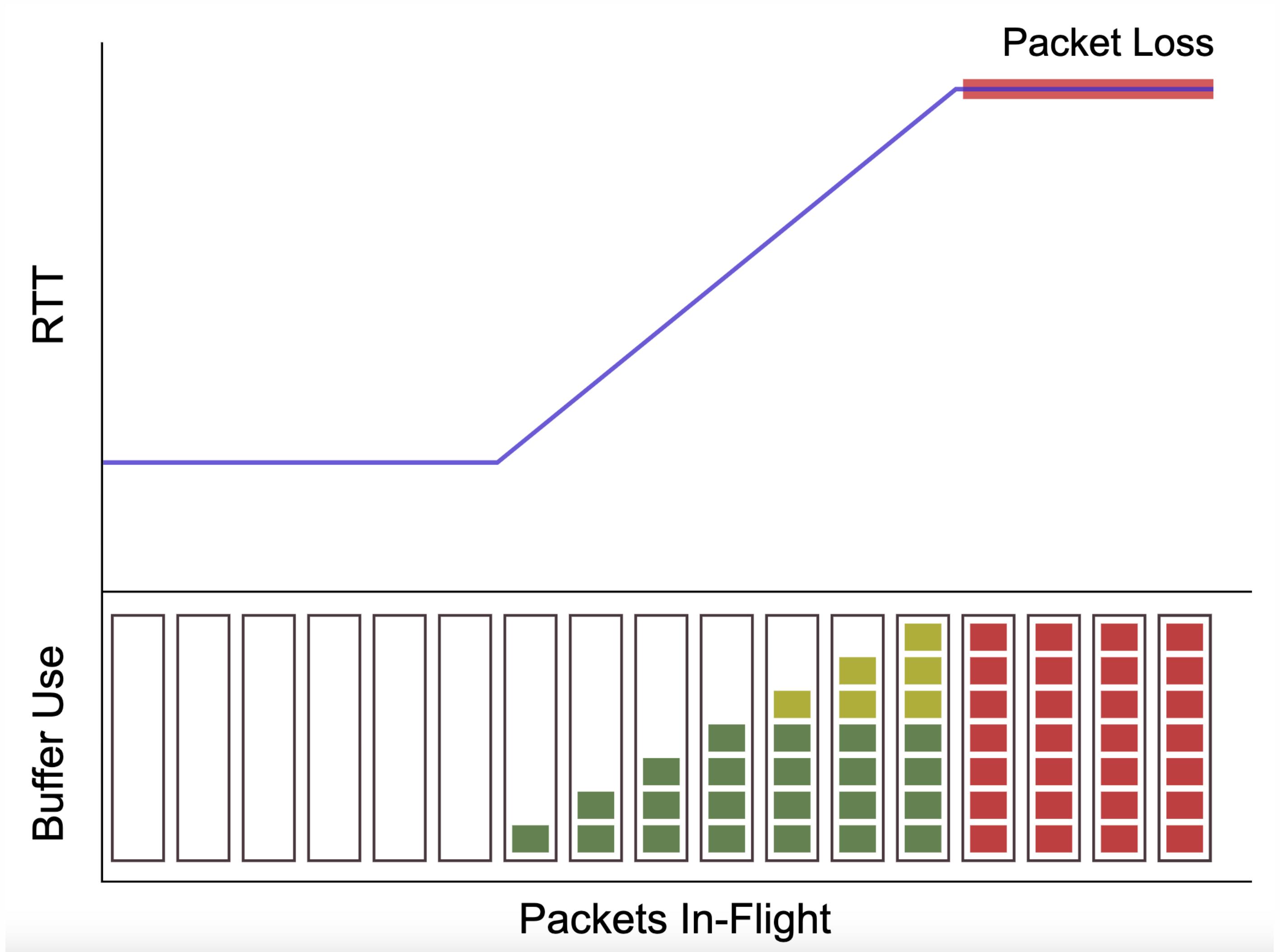
- Packet loss may not always mean that there is congestion in the network.
  - For example, a packet may be lost due:
  - Transient radio interference
  - Software bugs
  - Hardware issues

Delay based Congestion Control

- So far, we have discussed loss-based congestion control
- Cubic and other loss-based algorithms do not distinguish between spurious packet losses and real congestion, reducing their send rate in both cases.
- Another issue is that when bottleneck buffers are large, loss-based congestion control keeps them full, causing bufferbloat.
- The main idea behind delay-based congestion control is to detect network congestion using packet delays.

BBR

- BBR (Bottleneck Bandwidth and RTT) is new congestion control algorithm developed by Google in 2016.
- BBR monitors RTT and attempts to detect when packets are getting getting buffered based on timing. The larger the increase in RTT, the longer packets are being buffered, and the higher the likelihood the network is entering congestion.

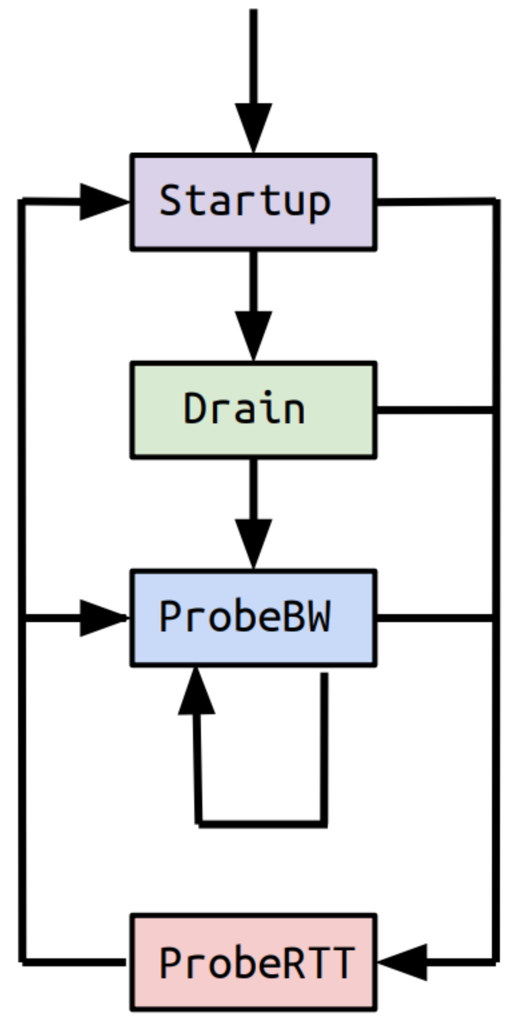


BBR

- BBR limits its number of packets in flight to be a multiple of the bandwidth-delay product (BDP).
- BBR strives to optimize both throughput and latency by estimating the bottleneck bandwidth and RTT to compute a pacing rate.
- The goal is is to avoid filling up the bottleneck buffer, which might induce bufferbloat.
- A description of the BBR algorithm was published in the 2016.
- The latest version is BBR v3, that includes several improvements over BBR v1 and v2

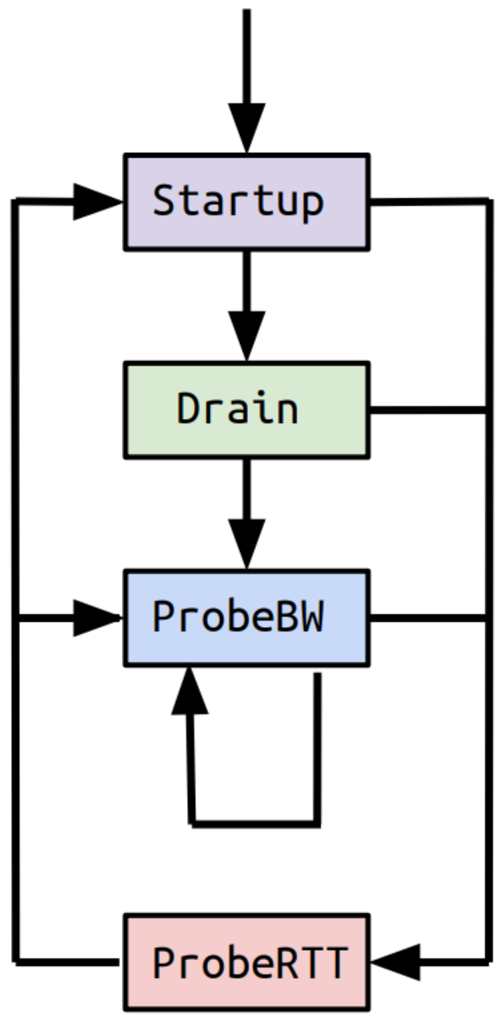
BBR Phases

- BBR is split up into four different phases; Startup, Drain, ProbeBW and ProbeRTT.
- The first phase, Startup, uses the same exponential starting behavior as CUBIC, which doubles the sending rate with each round-trip.
- BBR assumes the bottleneck bandwidth to have been reached once the measured bandwidth does not increase any further.
- This observation causes a queue to form at the bottleneck, due to it being delayed by one RTT.
- BBR then moves to the Drain phase, where it attempts to drain the queue by reducing the pacing gain temporarily.

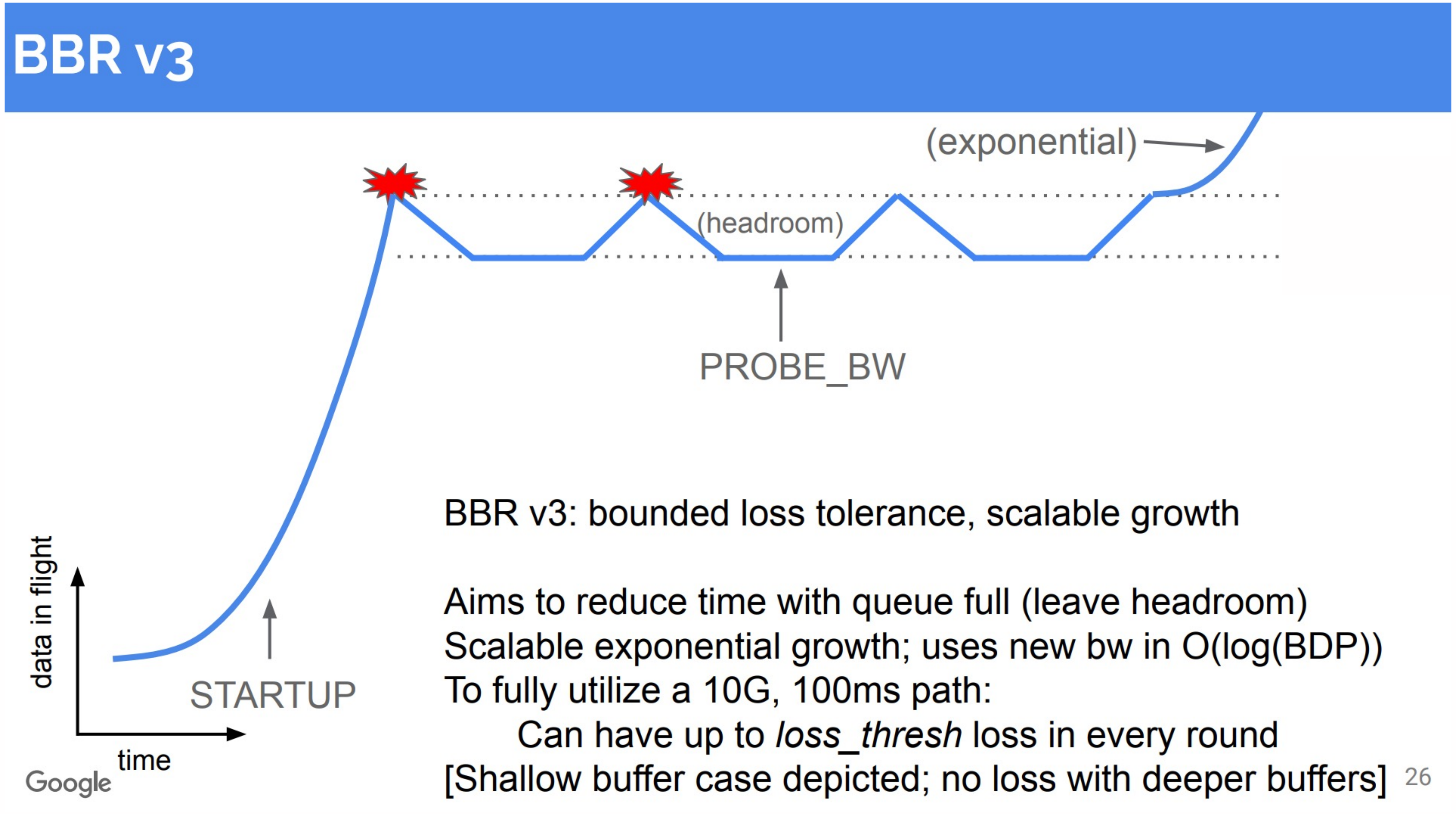


BBR Phases

- Next, BBR moves to the third phase, ProbeBW, which is also its steady state, since it spends most of its time in it probing and utilizing the pipe's bandwidth in a fair manner, with a small, bounded queue.
- If a flow hasn't seen an RTT sample that matches or decreases its min_rtt estimate for 10 seconds, BBR stops the bandwidth probing and moves on to the ProbeRTT phase, where the bandwidth gets reduced to four packets to drain any queues and get a more accurate estimation of the RTT.
- ProbeRTT is run for 200 ms + one RTT, where it will afterwards go back to the steady state, ProbeBW.



BBR v3



BBR v3

(exponential)
(headroom)
PROBE_BW
data in flight
time
STARTUP
Google

BBR v3: bounded loss tolerance, scalable growth

Aims to reduce time with queue full (leave headroom)
Scalable exponential growth; uses new bw in O(log(BDP))
To fully utilize a 10G, 100ms path:
    Can have up to *loss_thresh* loss in every round
[Shallow buffer case depicted; no loss with deeper buffers] 26

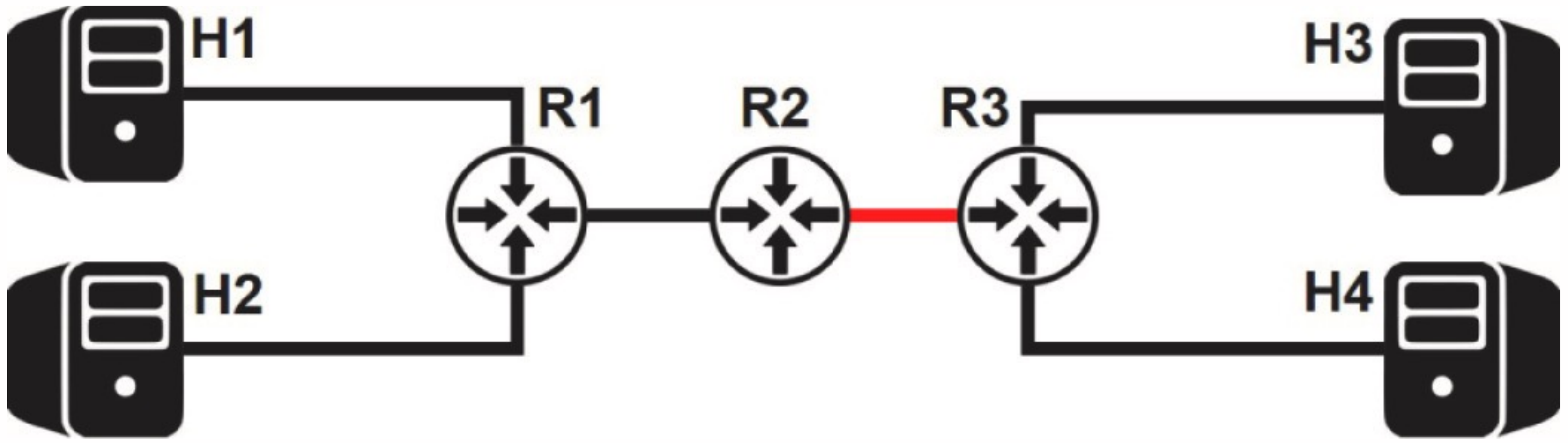
Slide from Google presentation at IETF 104

BBRv3 performance impact for public Internet traffic

- Impact of BBRv3 vs BBRv1 on Google.com and YouTube TCP public Internet traffic:
  - Lower retransmit rate (12% reduction)
  - Slight latency improvement (0.2% reduction) for:
    - Google.com web search
    - Starting YouTube video playback
  - Latency wins seem to be from lower loss rate (less/faster loss recovery)

Performance Evaluation

- You can test the performance of various congestion control algorithms using simulations
- Mininet testbed
  - The bottleneck link is the link between R2 and R3 (marked in red)
  - Delay and packet loss are configured on the link between R1 and R2
  - Traffic is generated between sender and receiver pairs (H1 and H3, H2 and H4)
  - Different congestion control algorithms are used for each pair



QUIC Congestion Control

- Incorporating existing algorithms
  - Similar to TCP congestion control, QUIC utilizes a window-based congestion control scheme
  - QUIC does not aim to develop its own new congestion control algorithms, nor use any specific one.
  - QUIC has pluggable congestion control which allows the sender to choose the best algorithm for its application.
  - To avoid unnecessary congestion window reduction, QUIC does not collapse the congestion window unless it detects persistent congestion using both loss-based and delay-based mechanisms.

QUIC RTT Estimation

- Estimating the Round-Trip Time
  - At a high level, an endpoint measures the time from when a packet was sent to when it is acknowledged, which allows to calculate the actual time used in transmitting a packet over the network.
  - An endpoint computes the following three values for each path:
    - smoothed_rtt – a stable average of the RTT measurements over a period of time
    - min_rtt - the minimum value over a period of time
    - rttvar - and the mean deviation (referred to as "variation") in the observed RTT samples

QUIC Loss Detection

- Detecting packet loss
  - QUIC uses two main signals to detect packet loss:
    1. Packet threshold loss detection
    2. Time threshold loss detection

QUIC Loss Detection

Packet Threshold Loss Detection

- QUIC considers a packet lost if a later packet has been acknowledged, and it was sent at least `kPacketThreshold` packets after the one we're considering.
- The recommended initial value for the packet reordering threshold is 3.

- Example:
  - You send packets 1, 2, 3, 4, 5.
  - You receive an ACK for 5, but 2 is still unacknowledged.
  - Since 5 was acknowledged, it means 3 new packets have been acknowledged since 2 was sent, therefore 2 is declared lost.

```
if (largest_acked_packet_number - packet_number >= kPacketThreshold)
           => packet is lost
```

QUIC Loss Detection

Time Threshold Loss Detection

- A packet is considered lost if enough time has passed since it was sent, without being acknowledged.
- Example:
  - If a packet was sent earlier than:

```
current_time - max(kTimeThreshold * max(smoothed_rtt, latest_rtt), kGranularity)
        => packet is lost
```

  - The recommended time threshold (kTimeThreshold), expressed as an RTT multiplier, is 9/8 (i.e., 1.125)
  - The recommended value of the timer granularity (kGranularity) is 1 millisecond

QUIC Packet Pacing

- Packet pacing means spreading out the transmission of packets evenly over time, instead of sending a burst of packets all at once, that could lead to congestion
- QUIC packet pacing is a mechanism designed to regulate the rate at which packets are sent over the network.
- The pacing rate is typically derived from the current congestion window (cwnd) and the RTT.

```
Pacing rate = cwnd / smoothed_rtt
```

- Based on the pacing rate, QUIC calculates a delay between sending packets to smooth packet bursts.
  - Example: If you can send 100 packets per second, you send one packet every 10ms, instead of all at once.
- Packet pacing mechanism adapts to changes in network conditions, such as changes in available bandwidth or variations in round-trip times.

QUIC vs TCP Congestion Control

- QUIC is more flexible and easier to evolve than TCP
- TCP is typically implemented in the operating system's kernel
  - Tuning congestion logic is usually only done by a select few developers, and evolution is slow
- QUIC implementations are in "user space"
  - They are flexible to modify
  - Experimenting with new congestion-control algorithms is easier
- RFC 9002 - QUIC Loss Detection and Congestion Control
  - https://datatracker.ietf.org/doc/rfc9002/

Acknowledgement

Ollscoil na Gaillimhe
University of Galway

- ❑ https://hpbn.co/
- ❑ https://vt200.com/
- ❑ https://www.smashingmagazine.com/

Ollscoil na Gaillimhe
University of Galway

Thank you for your attention!

University
ofGalway.ie