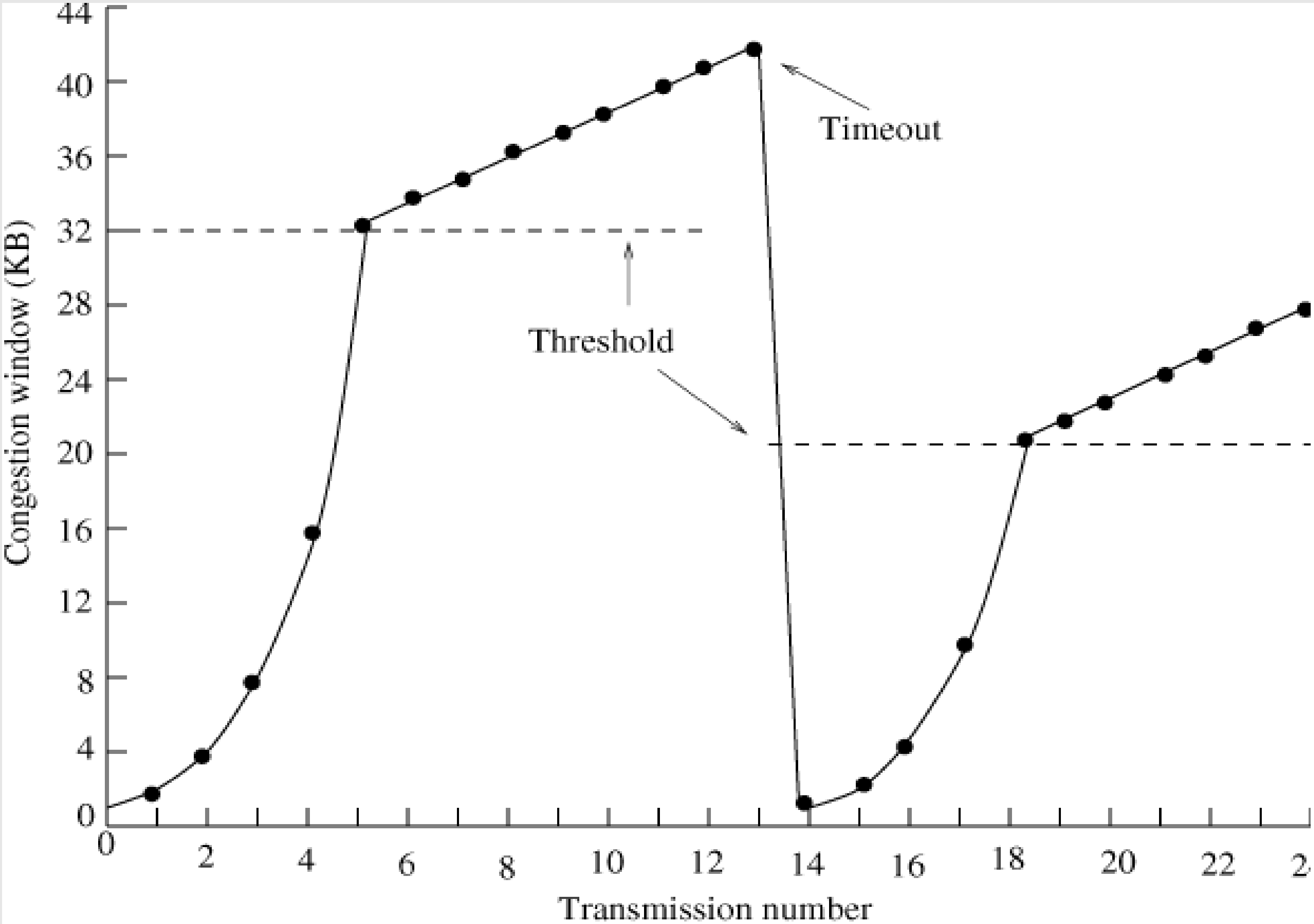
Transmission Control Protocol

Data Transfer

Quick Review: TCP Tahoe

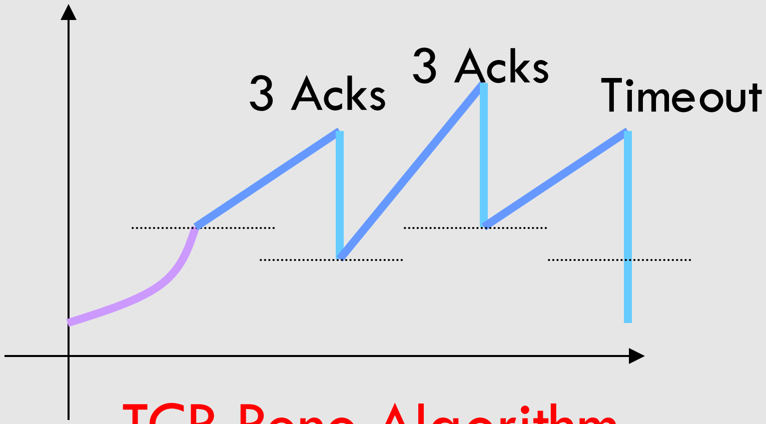


Congestion Control

Alternative: Fall to W/2 (+n*MSS)
and start congestion avoidance directly

If (Timeout)
  W = 1
  ssThresh = w/2
  slow start …

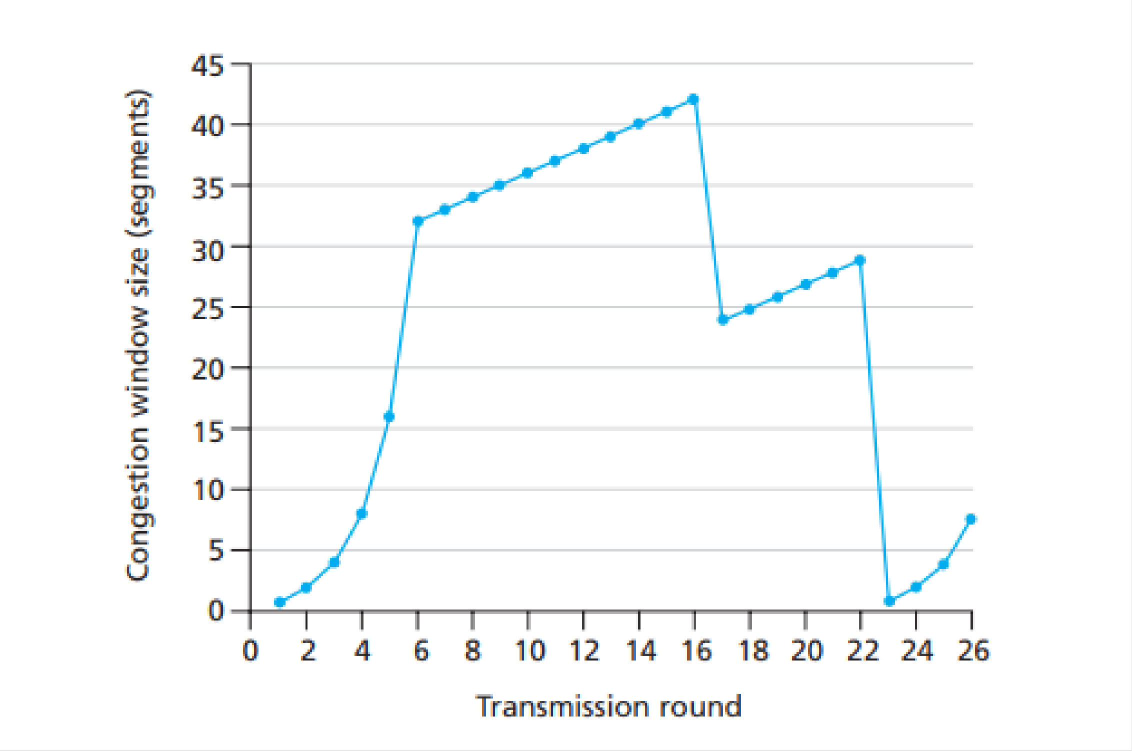
If (3 Acks)
  W = w/2
  linear/additive increase …
  (fast retransmission)



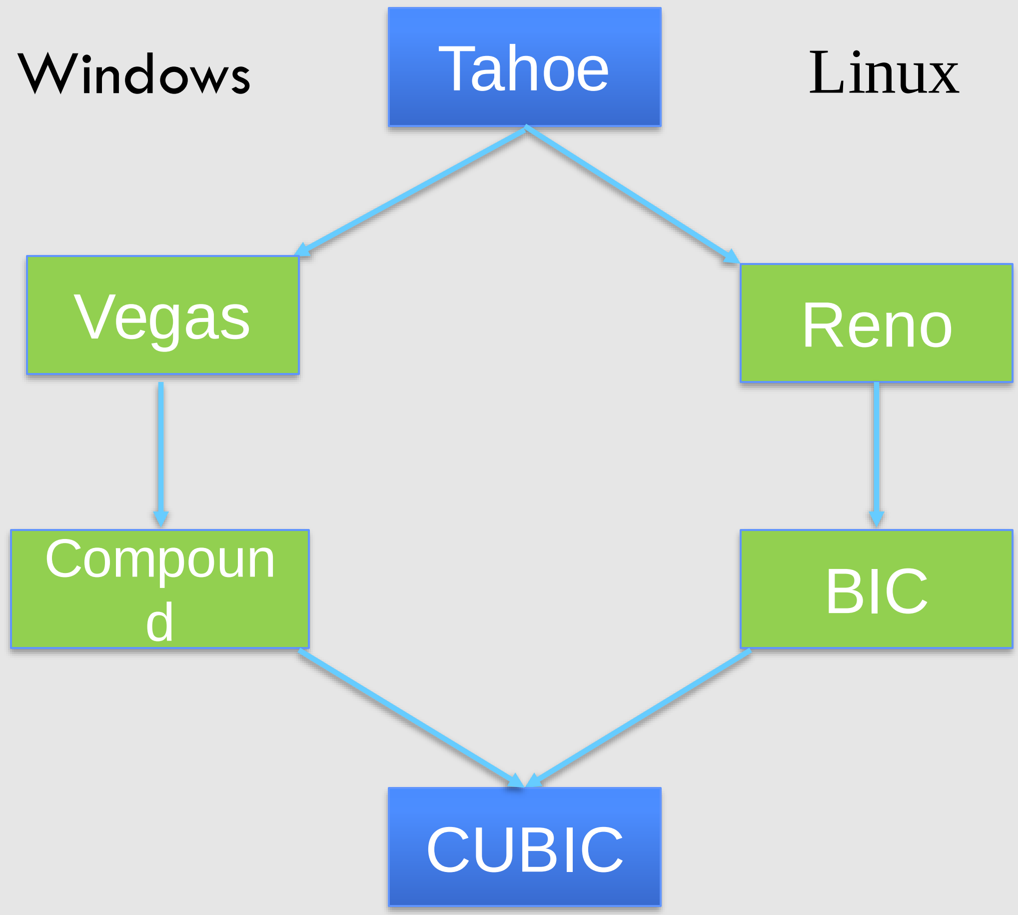
TCP Reno Algorithm

Example

Rule: Fall to W/2 (+n*MSS)
and start congestion avoidance directly



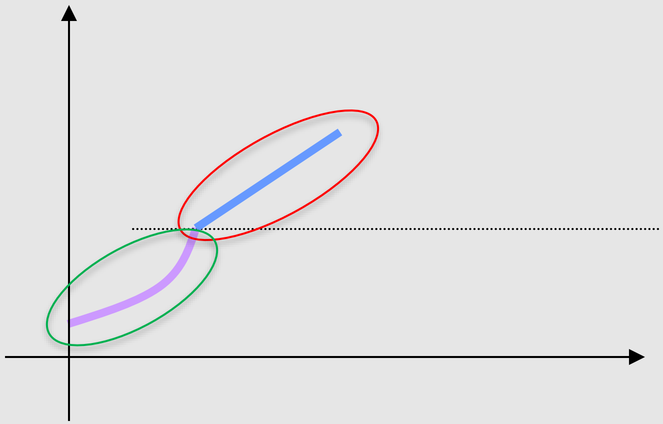
History of TCP



TCP CUBIC[1]

- Problems of earlier versions
  - poor utilization of bandwidth

BW =10 Gbps, RTT =100 ms, pkt =1250 bytes



BDP =100,000 packets. For TCP to grow its window from the mid-point of the BDP, say 50,000, it takes about 50,000 RTTs which amounts to 5000 seconds (1.4 hours). If a flow finishes before that time, it severely under-utilizes the path.

- Overshooting problem in SS

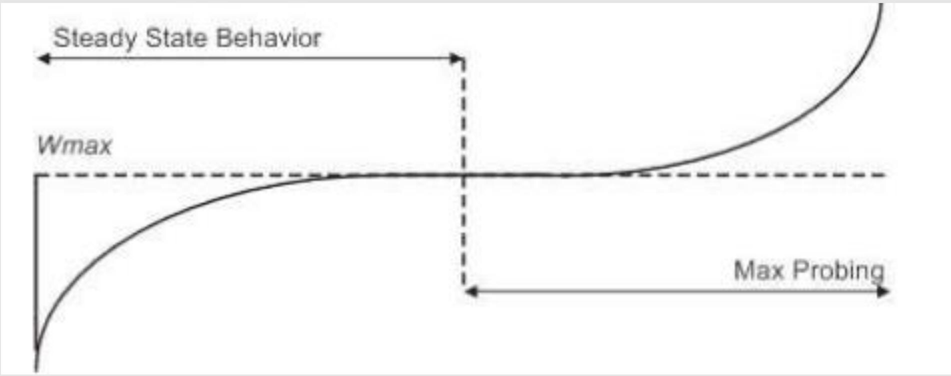
[1]Sangtae Ha, Injong Rhee and Lisong Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant".

CUBIC

- Basic idea

  -- it uses cubic function for window growth

  -- it uses real time (available BW) instead of RTT to increase the window size (congestion epoch)



CUBIC

- Algorithm

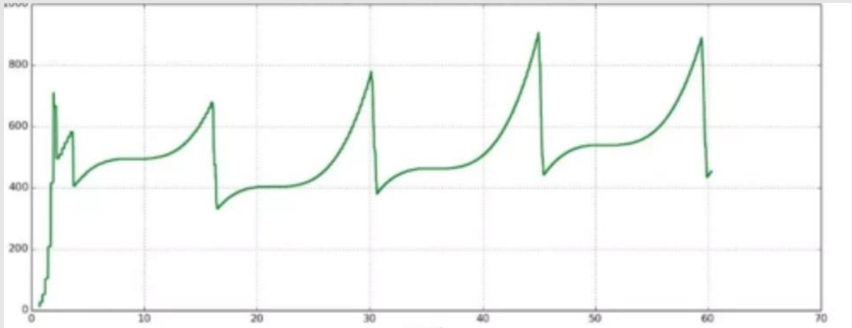
-- after a packet loss, reduces its window by a multiplicative factor of β (<1)

-- the window size just before reduction is set to Wmax

-- after it enters into congestion avoidance, it starts to increase the window using a cubic function

-- the plateau of cubic function is set to Wmax

-- size of the window grows in concave mode to reach Wmax, then it enters the convex part



CUBIC

- Algorithm

  -- the window growth function uses the formula:

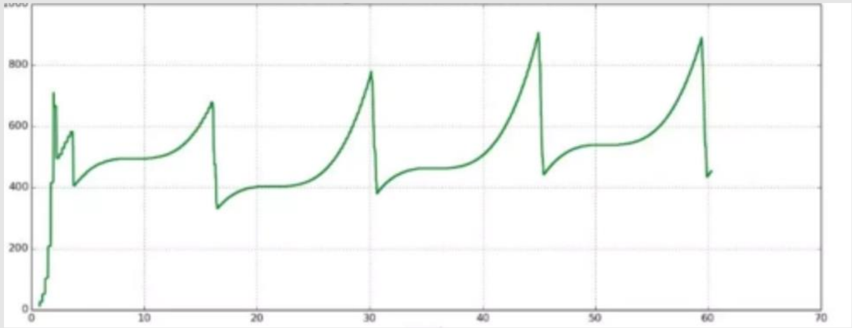
$$W(t) = C(t - K)^3 + W_{max}$$

$$K = \sqrt[3]{\frac{W_{max}\beta}{C}}$$

-- C is cubic constant

-- *t* is elapsed time from the last window reduction

-- *K* is the time period takes to get from W to Wmax while no other loss occurs



CUBIC

- Algorithm

while (receive ACK during congestion avoidance)

compute W(t+RTT) as congestion window

if CW < Wtcp[1]

$$W_{tcp(t)} = W_{max}(1-\beta) + 3\frac{\beta}{2-\beta}\frac{t}{RTT}$$

CUBIC is in TCP mode

else if Wtcp < CW < Wmax

CUBIC is in concave mode

else

CUBIC is in convex mode

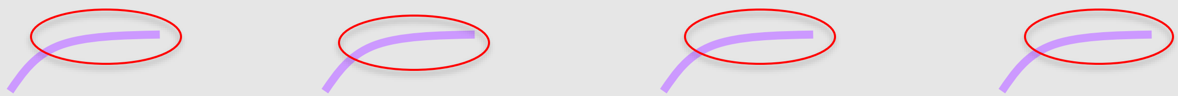
$$W(t) = C(t-K)^3 + W_{\max}$$

$$K = \sqrt[3]{\frac{W_{\max}\beta}{C}}$$

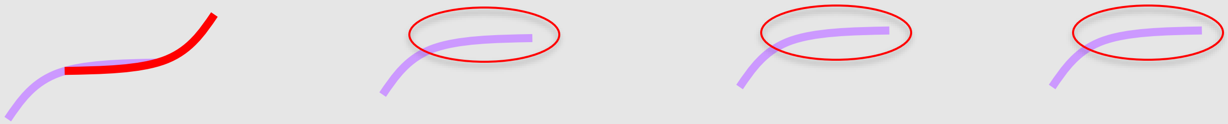
[1] Floyd, S., Handley, M., and Padhye, J. "A Comparison of Equation-Based and AIMD Congestion Control".

CUBIC

- Network condition stable



- Congestion is alleviated



- Congestion is created

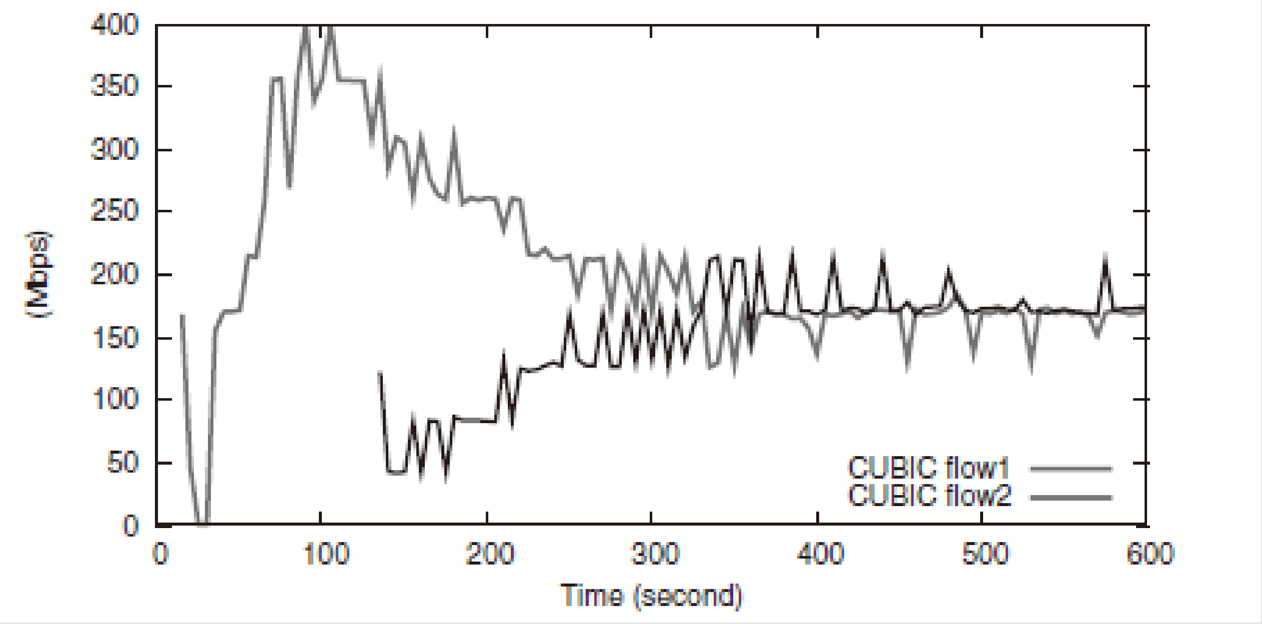
CUBIC

- TCP Friendliness
  - Standard TCP works well with short RTT, and CUBIC is designed to work similarly in these conditions.

- Fast Convergence
  - directly reduce a larger amount of W_{max} to achieve the stable point

- RTT Fairness
  - multiple flows: longer, shorter RTT

CUBIC


400
350
300
250
200
150
100
50
0
(Mbps)
0
100
200
300
400
500
600
Time (second)
CUBIC flow1
CUBIC flow2

NC STATE UNIVERSITY
Department of Computer Science

BBR

(Bottleneck Bandwidth and Round-trip propagation time)

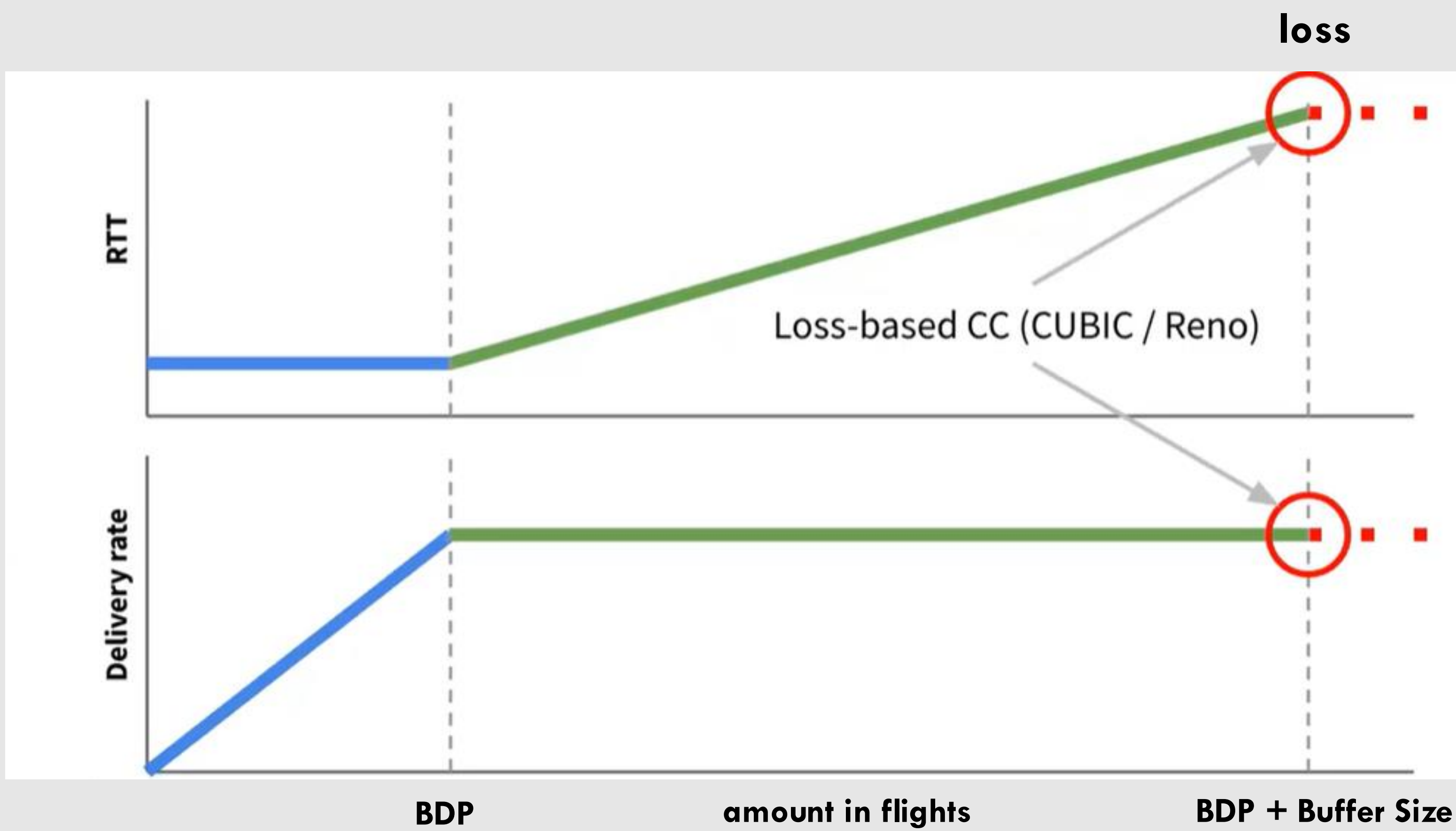
Cardwell, N., Cheng, Y., Gunn, C. S., Yeganeh, S. H., & Jacobson, V. (2017).
BBR: congestion-based congestion control. *Communications of the ACM*, *60*(2), 58-66.

Problems

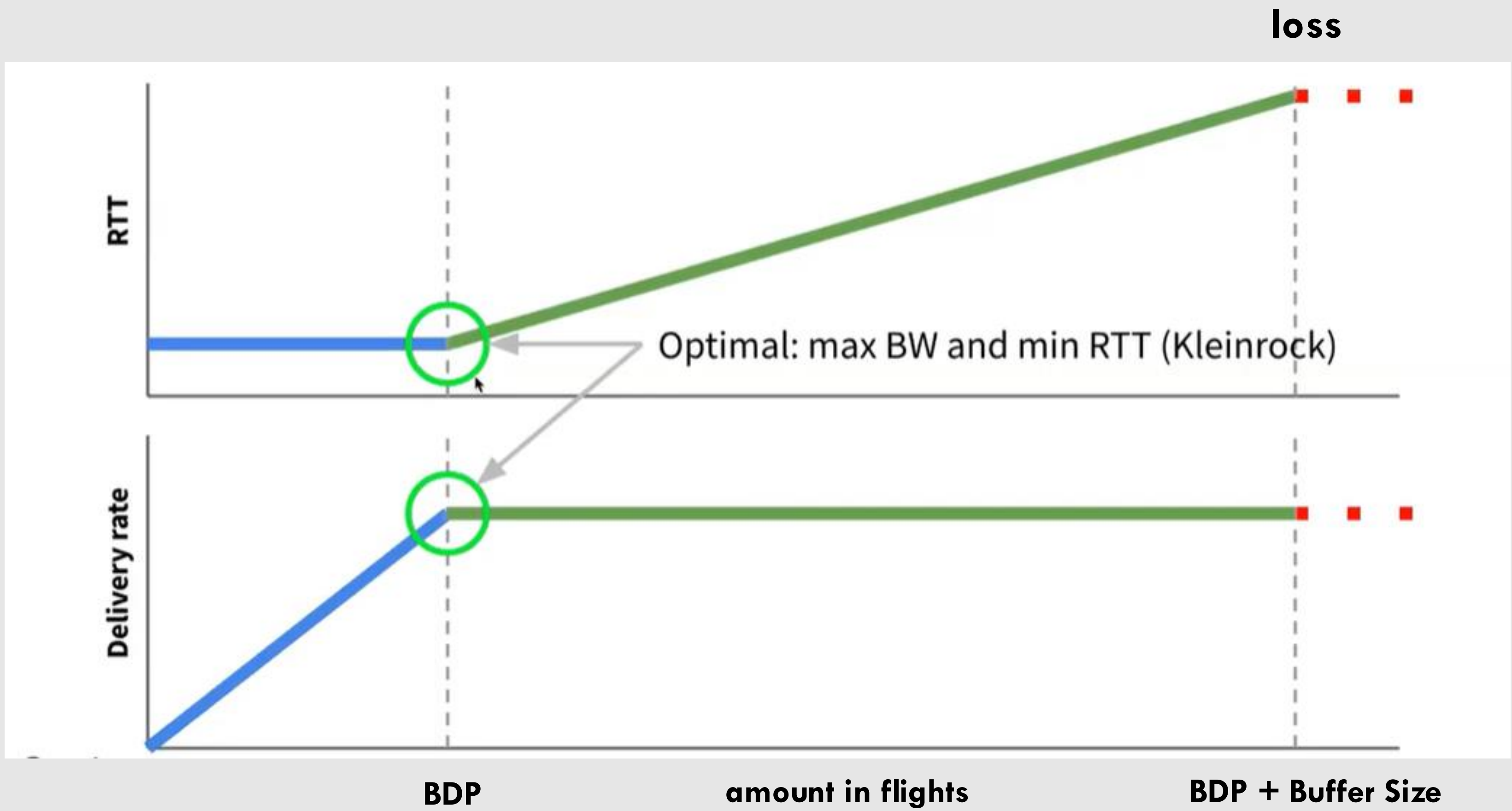
- Loss-based congestion control:
  - Tahoe, Reno, NewReno, CUBIC
  - Packet loss is not a good indicator
  - Overly sensitive to losses that come before congestion

- Why this important?
  - higher speed (5G/beyond, mmWave)
  - e.g., 10Gbps over 100ms RTT needs < 0.000003% packet loss

Loss-Based Method

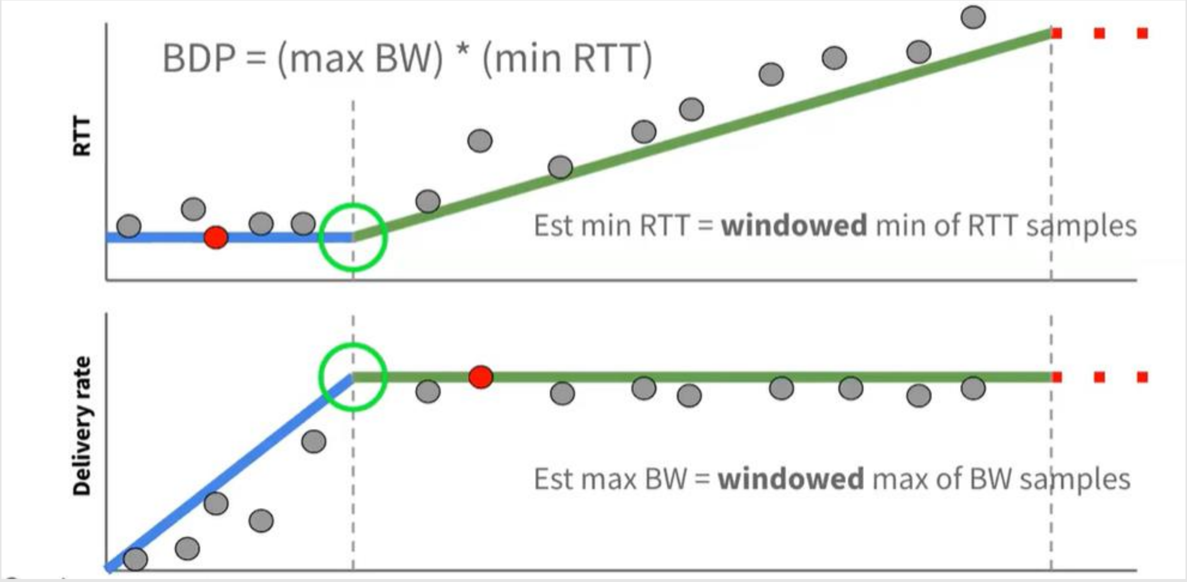


What is the Optimal Point?



BBR

- Model-based congestion control:
  - Proactively doing countermeasures
  - Estimate the best time to control



Dilemma

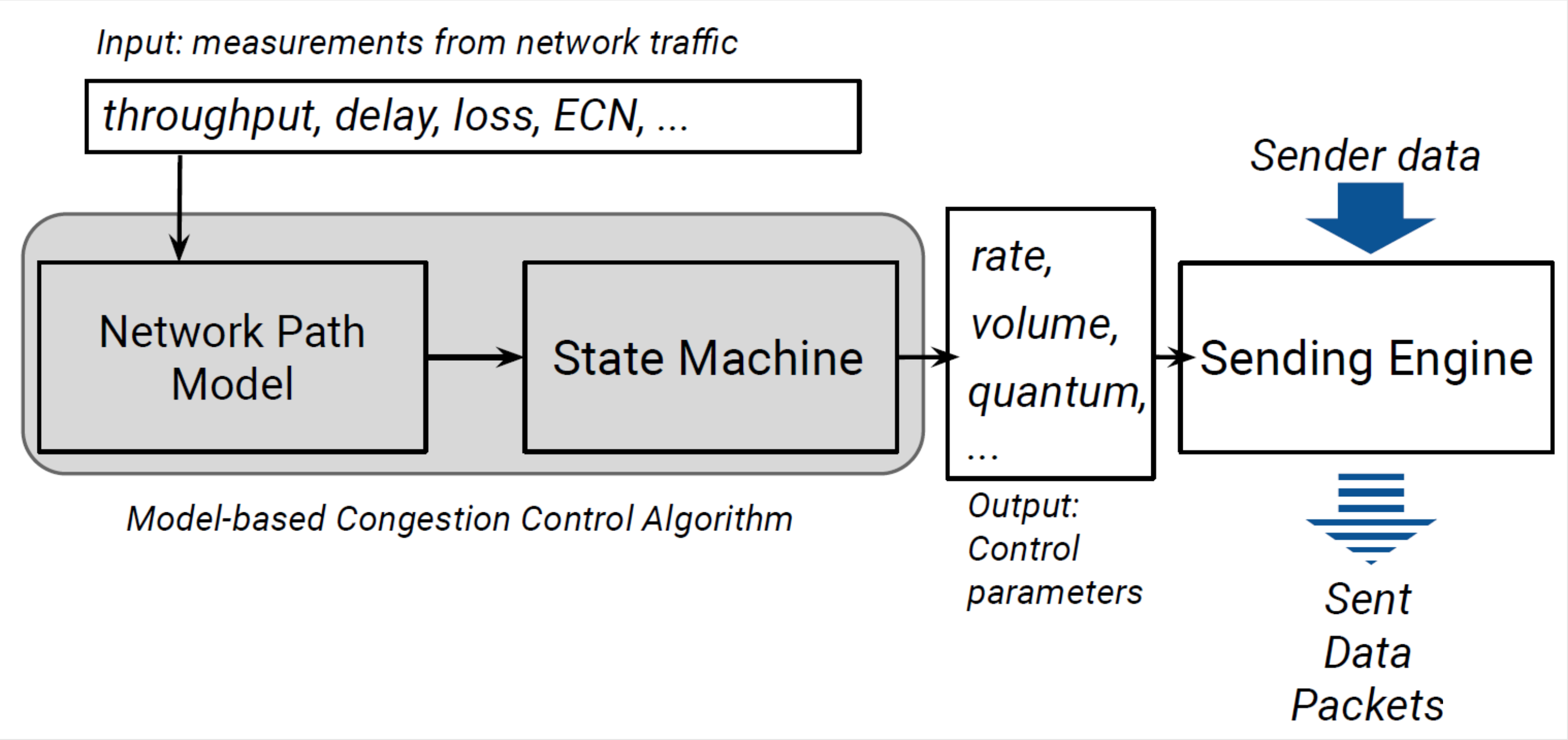
- Measure min RTT & max BW
  - Measure on both sides of BDP

BBR Design

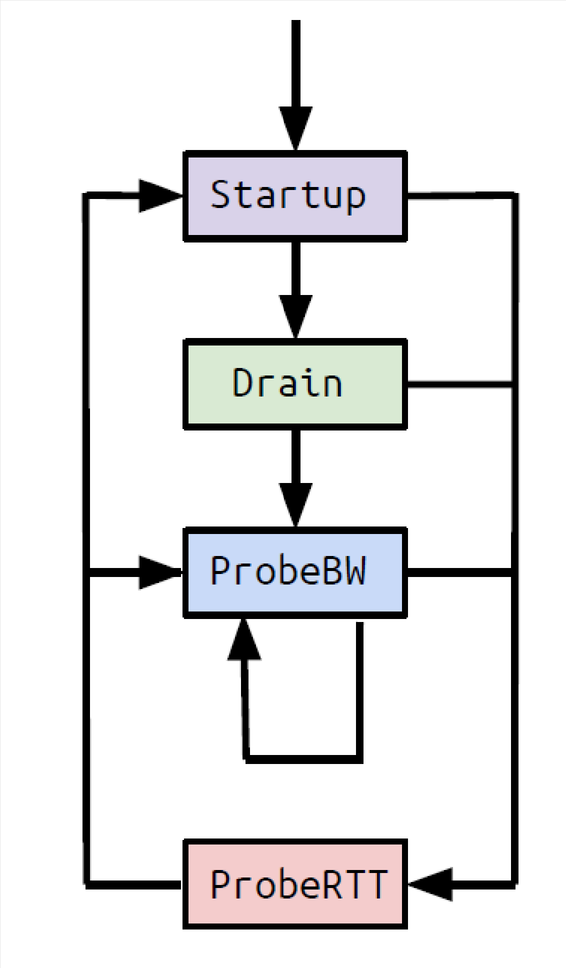
- Dynamically estimate windowed max BW & min RTT
- Sequentially probe max BW & min RTT

	CUBIC	BBR v1	BBR v2
Model parameters to the state machine	N/A	Throughput, RTT	Throughput, RTT, max aggregation, max inflight
Loss	Reduce cwnd by 30% on window with any loss	N/A	Explicit loss rate target
ECN	RFC3168 (Classic ECN)	N/A	DCTCP-inspired ECN
Startup	Slow-start until RTT rises (Hystart) or any loss	Slow-start until tput plateaus	Slow-start until tput plateaus or ECN/loss rate > target

BBR Algorithm



BBR State Machine

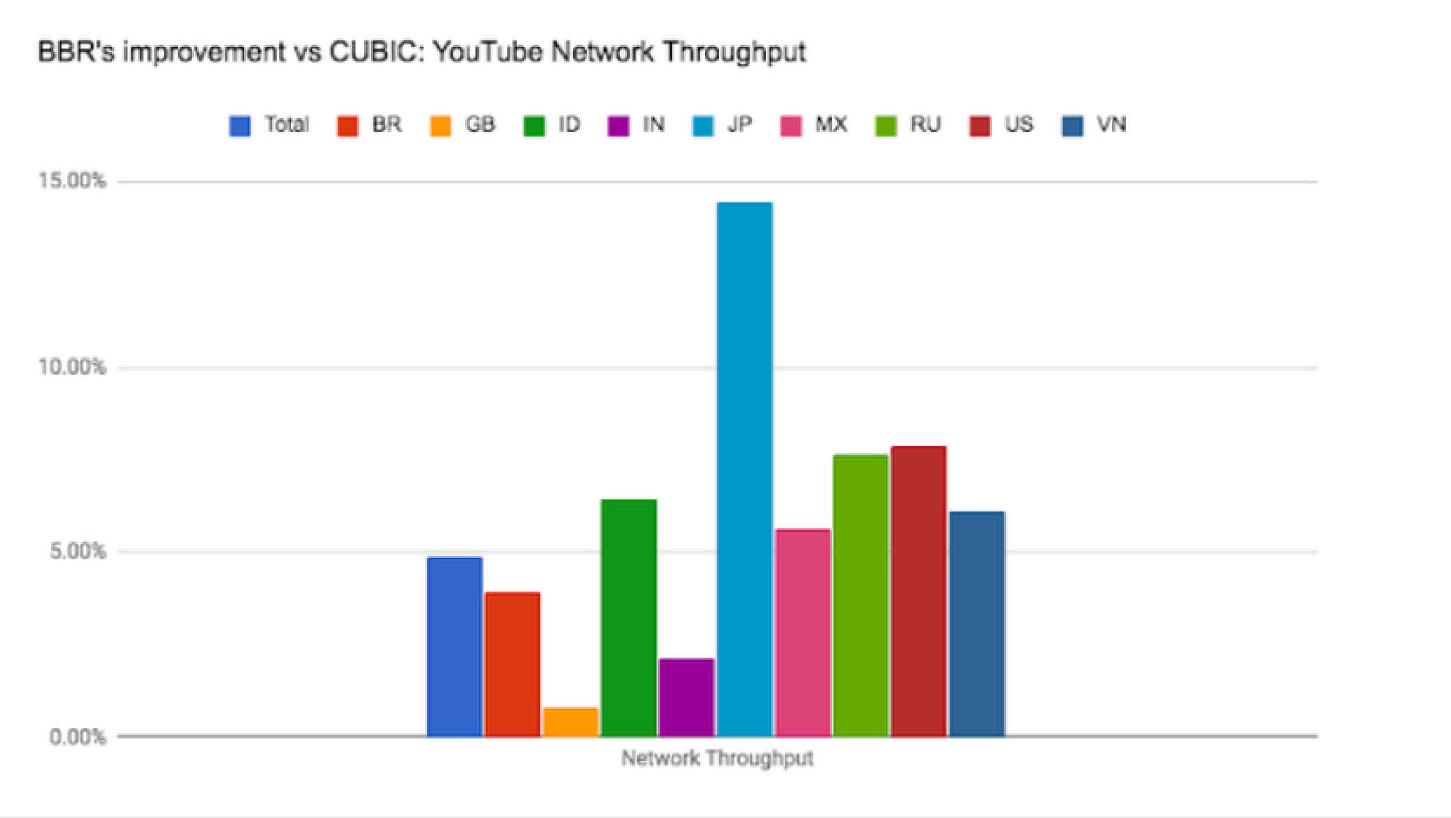


- **Startup:** ramp up quickly until we estimate pipe is full
- **Drain:** drain the estimated queue from the bottleneck

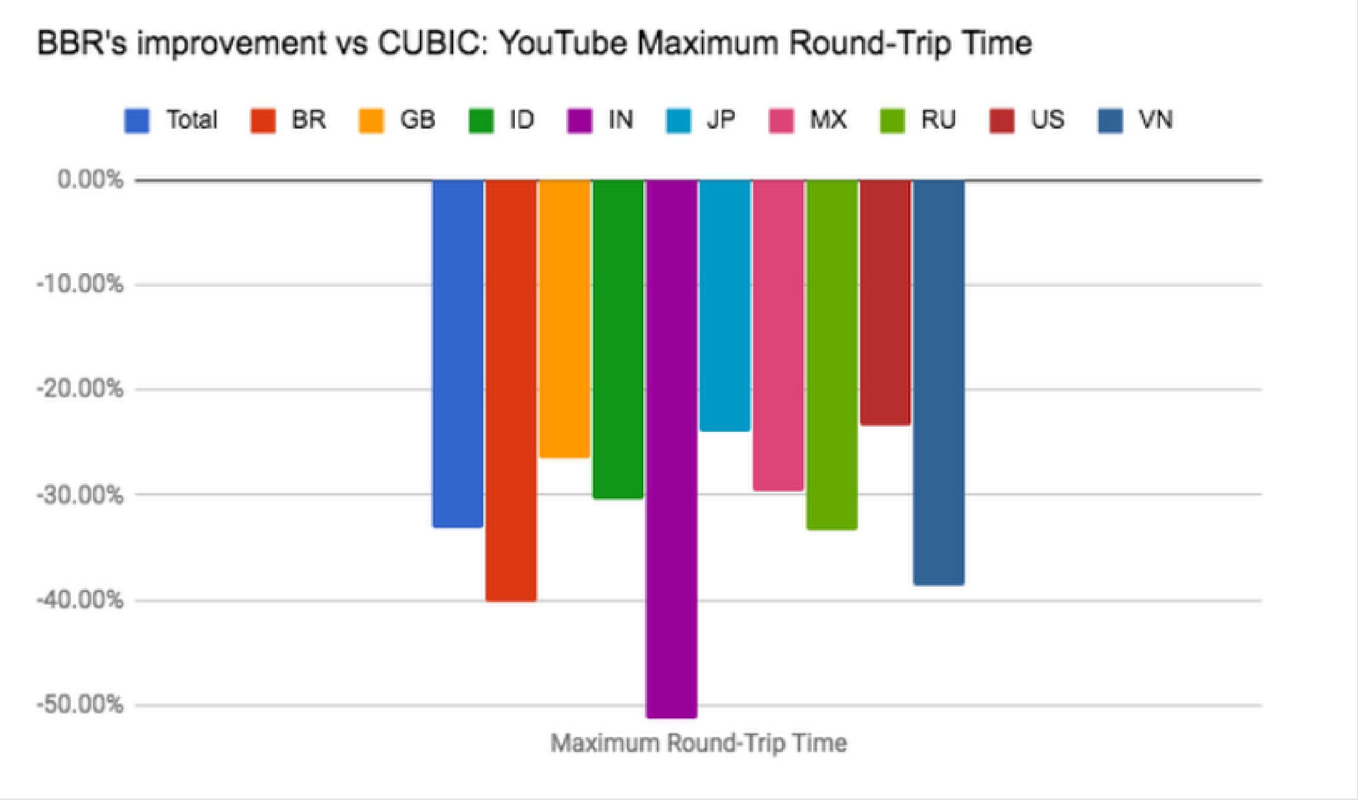
Steady-state:

- **ProbeBW:** cycle pacing rate to vary inflight, probe BW
- **ProbeRTT:** if needed, a coordinated dip to probe RTT

BBR Evaluation

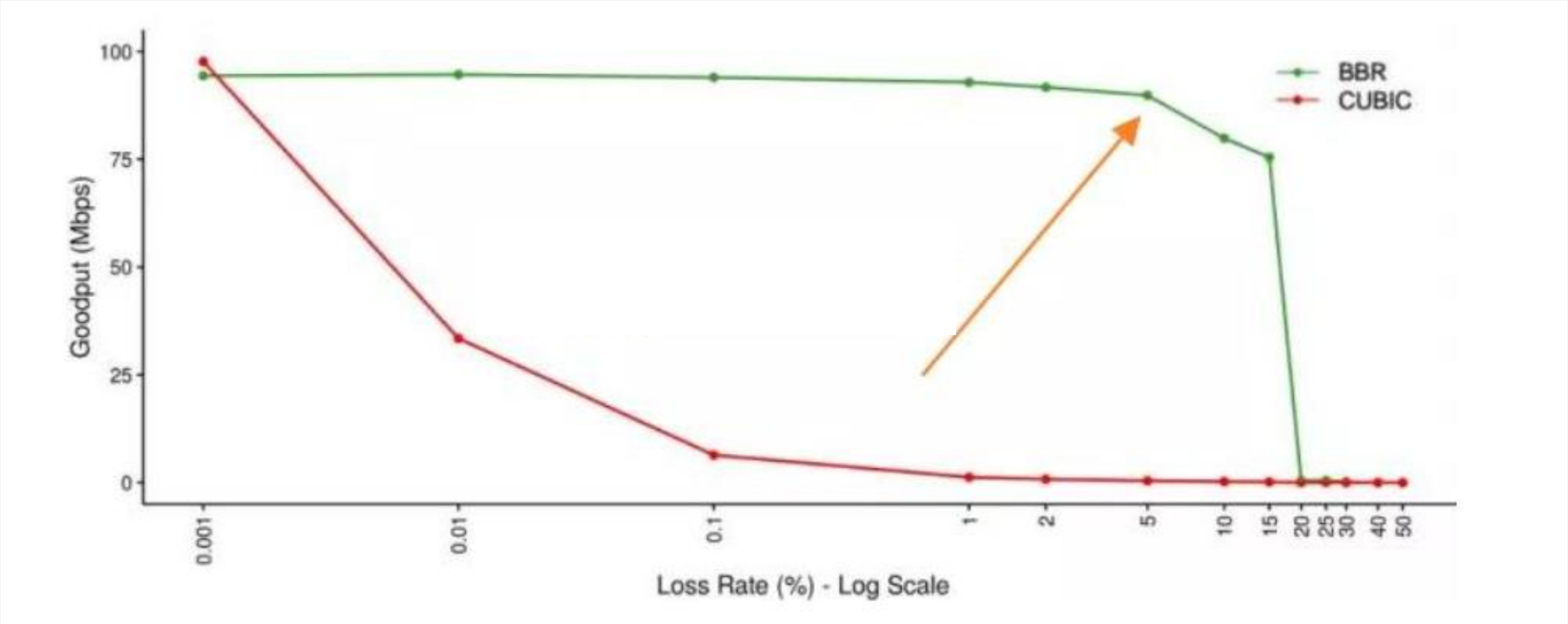


BBR Evaluation



Result in higher throughput, lower latency and better quality of experience.

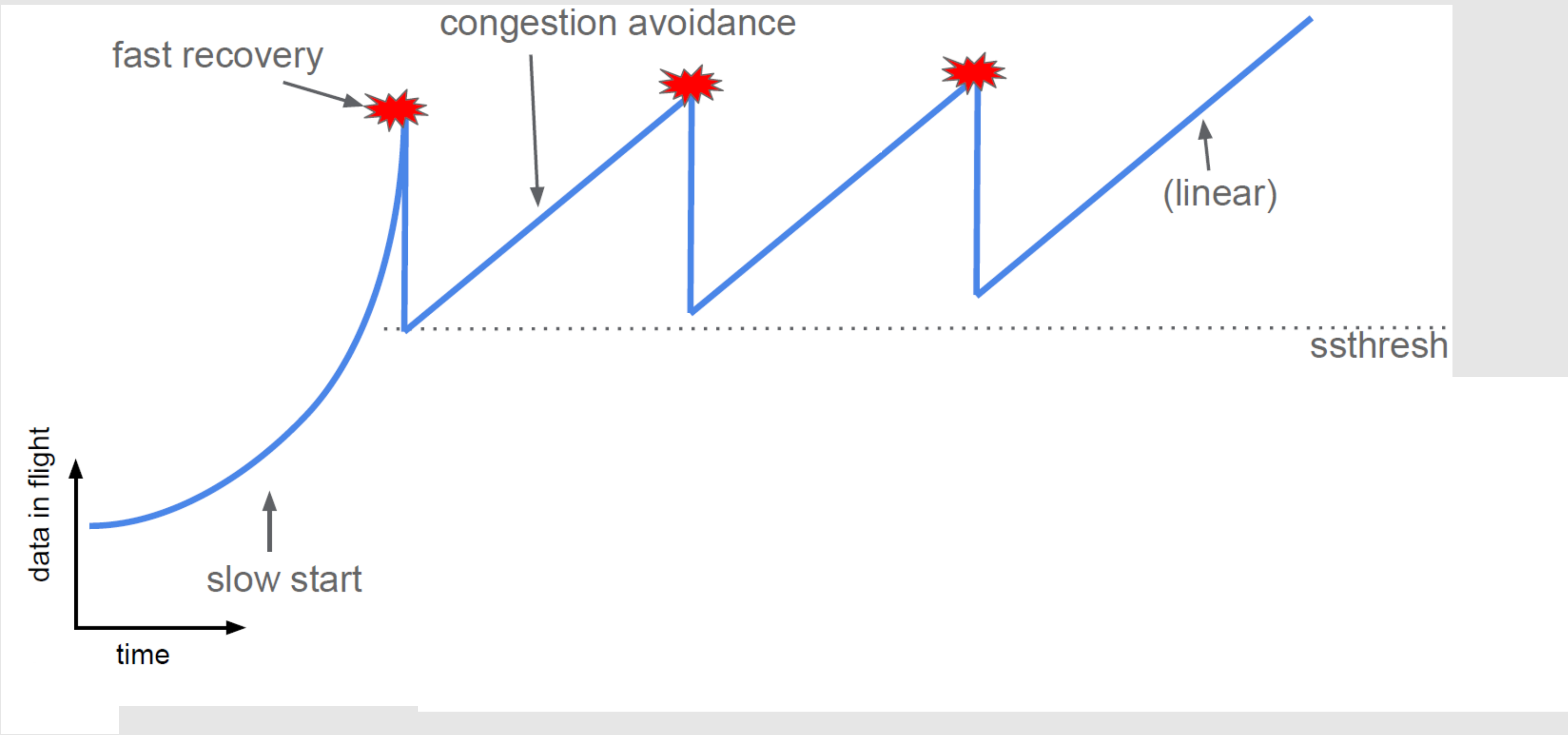
BBR Evaluation



Better packet-loss tolerance

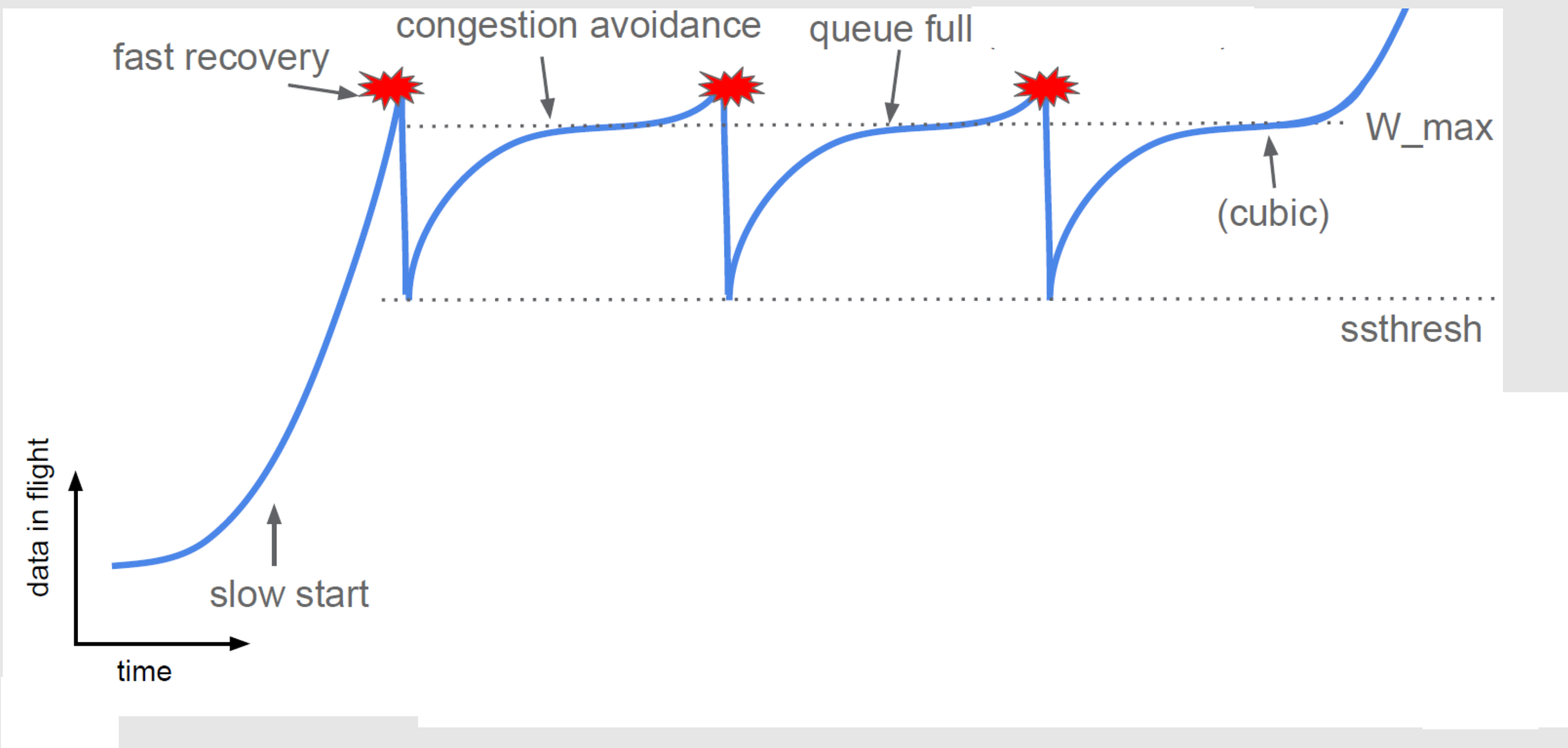
Recap

- Reno


congestion avoidance
fast recovery
(linear)
ssthresh
data in flight
slow start
time

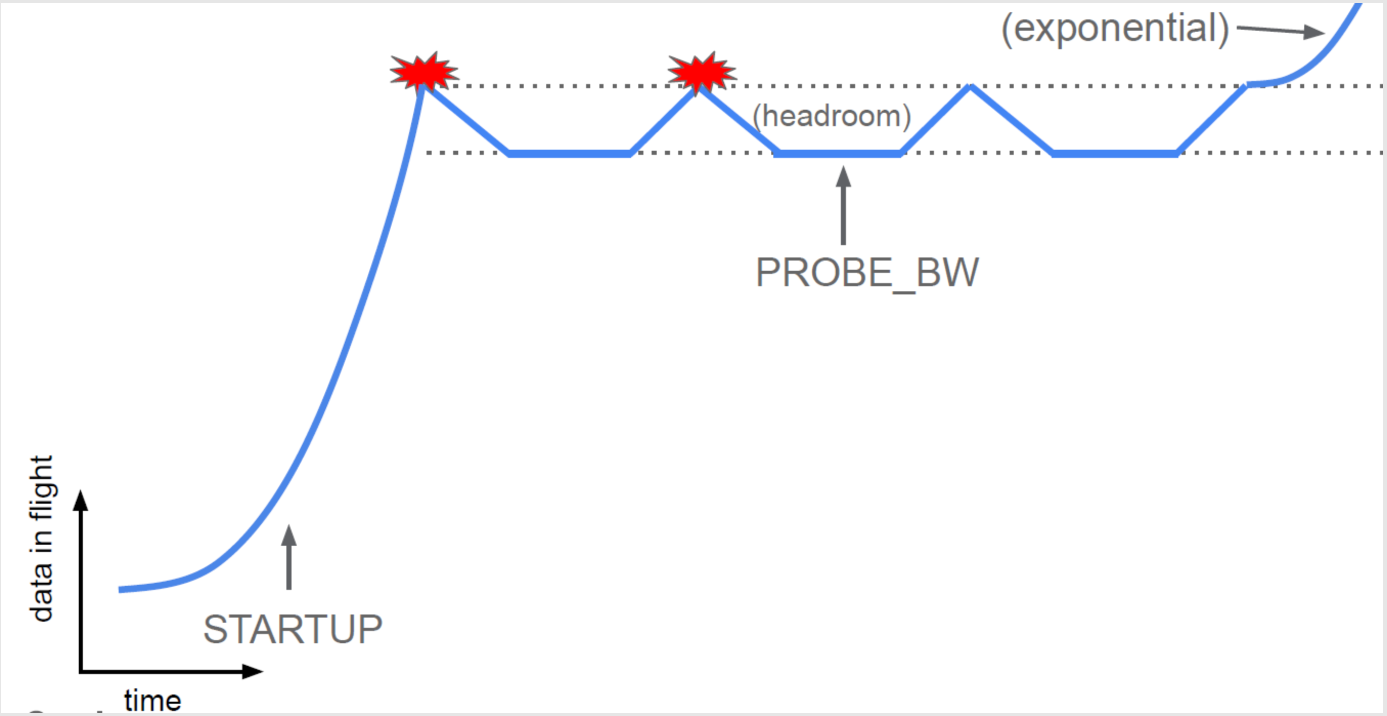
Recap

- Cubic


congestion avoidance
queue full
fast recovery
W_max
(cubic)
ssthresh
data in flight
slow start
time

Recap

- BBR



ns-3 TCP

- CUBIC by default
  - support: TcpNewReno, TcpHybla, TcpHighSpeed, TcpHtcp, TcpVegas, TcpScalable, TcpVeno, TcpBic, TcpYeah, TcpIllinois, TcpWestwood, TcpWestwoodPlus,TcpLedbat, TcpLp, TcpDctcp, TcpCubic, TcpBbr
  - ~\ns-3.37\examples\tcp\tcp-variants-comparison.cc
  - ~\ns-3.37\examples\tutorial\fifth.cc (ns-3 tutorial, Tracing\real examples)
  - TraceCwnd ()
  - TraceRtt ()

Reference

- BBR v1 is (only) available in Linux
  - need to enable it
- BBR v2 release for research study
  - github.com/google/bbr
- ns-3 (TcpBbr)
  - tcp-bbr example.cc

Summary

- Transport layer is logically application's interface to network
  - Must create endpoint abstractions (ports)
  - Must maintain state
- In the Internet,
  - TCP attempts to impose reliability on unreliable network layer
    - Requires sliding window management
  - TCP attempts to perform congestion control
    - Slow down transmission rate in response to lost segments