IN2140:
Introduction to Operating Systems and Data Communication

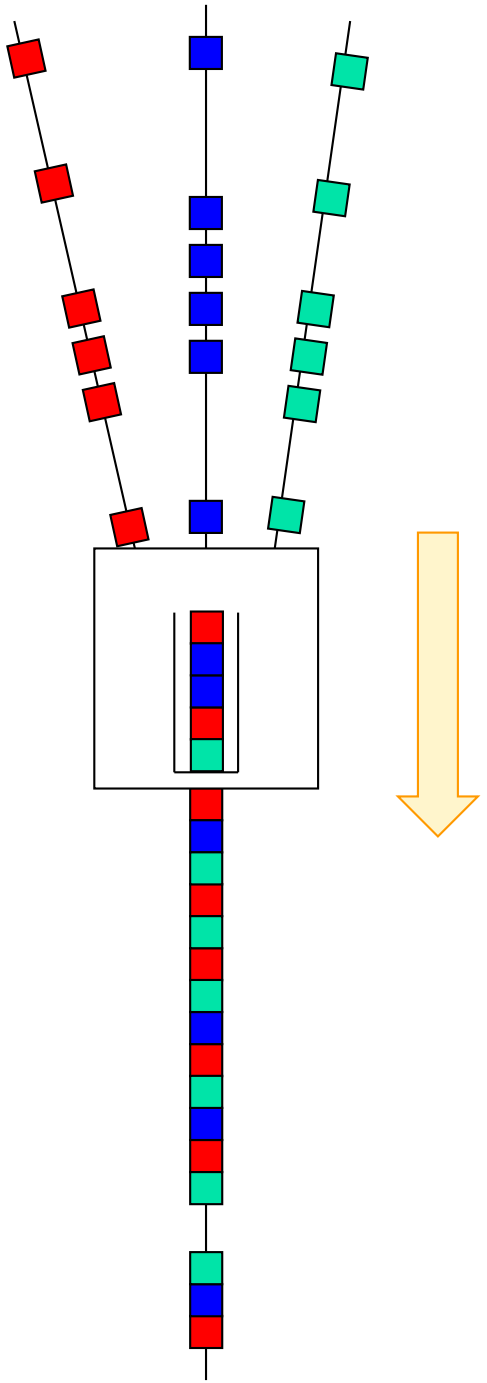


Transport layer

Introduction to Congestion Control

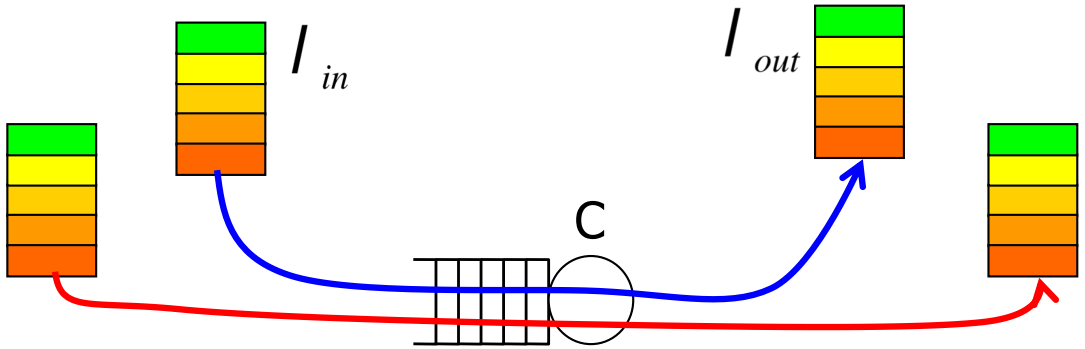
Congestion

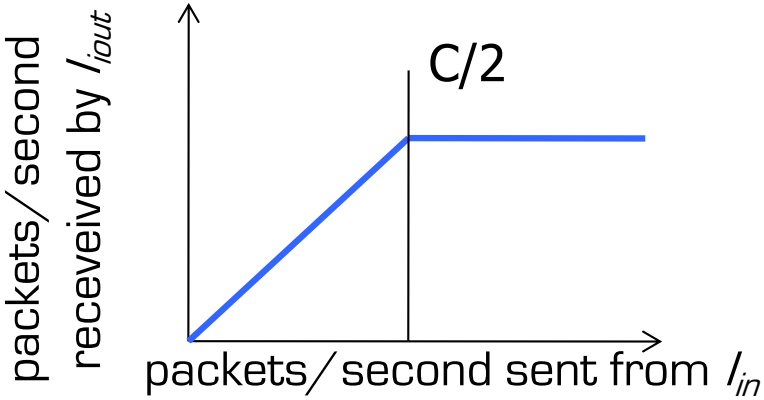
- ***Traffic***
  - All traffic from all sources
- ***Persistent congestion***
  - Router stays congested for a long time
  - Excessive traffic offered
- ***Transient congestion***
  - Congestion occurs for a while
  - Router is temporarily overloaded
  - Often due to ***burstiness***

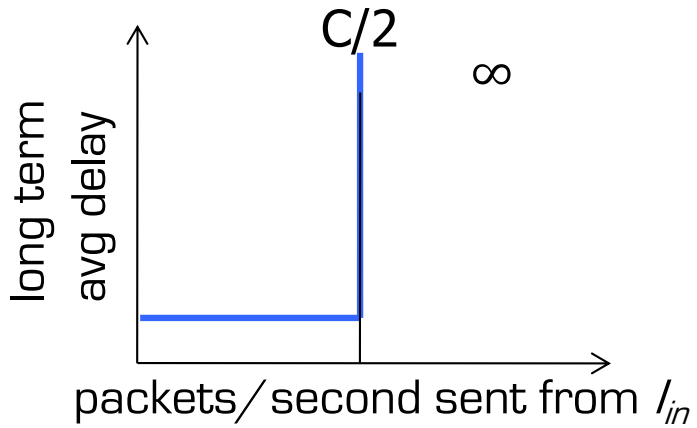


Reasons for congestion

- Two senders, two receivers
- One router, **infinite** buffers
- No retransmissions





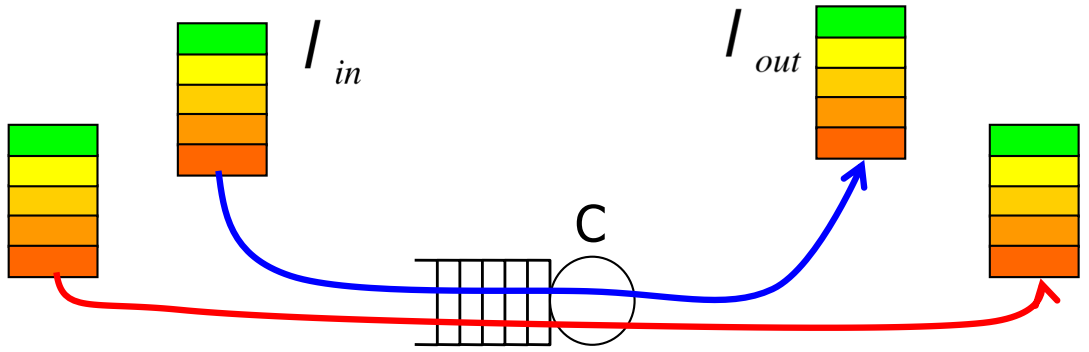


- Maximum utilization of the networks
- Very long delays in case of congestion

Average delay becomes ∞
because queue keeps growing forever

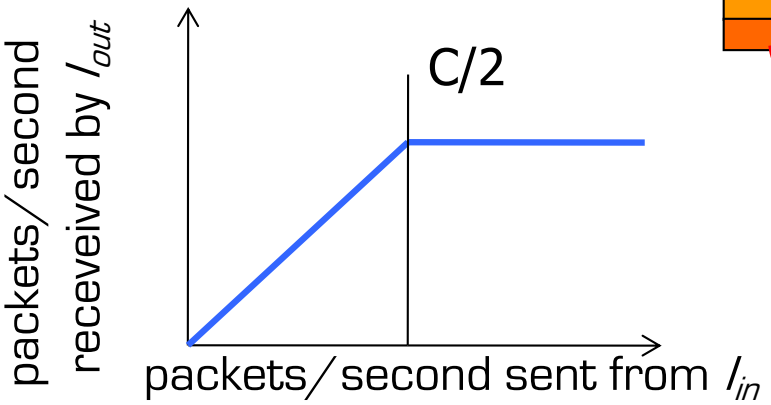
Reasons for congestion

- Two senders, two receivers
- One router, **finite** buffers
- No retransmissions



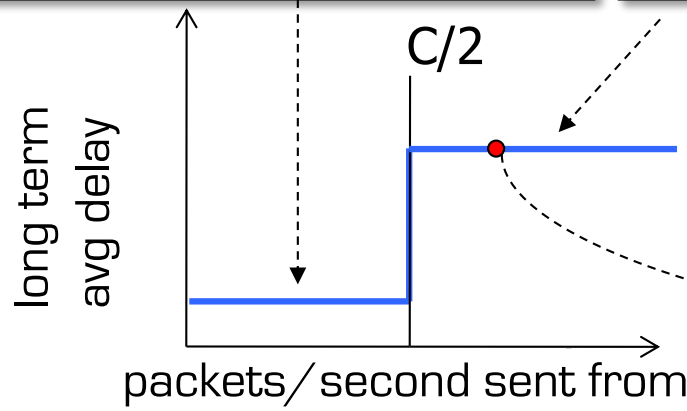
Router with finite buffers

When $\sum l_{in}$ exceeds C, l_{out} is missing some data

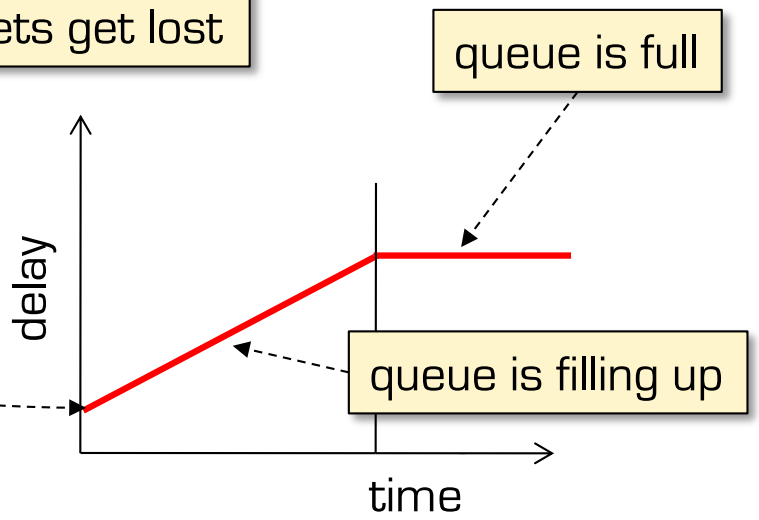


no packet losses, empty queue

queue fills up, packets get lost

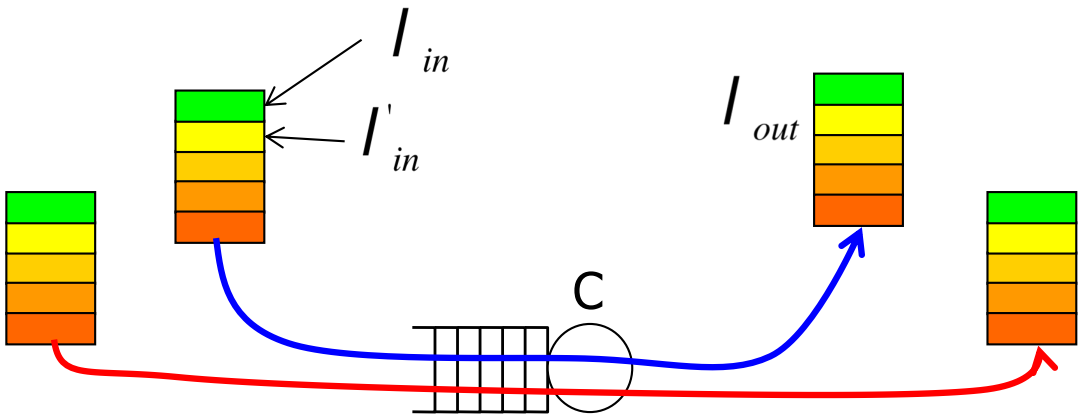


example



Reasons for congestion

- Two senders, two receivers
- One router, **finite** buffers
- **Retransmission** of lost packets



Router with finite buffers

λ_{in} Data rate sent by the application

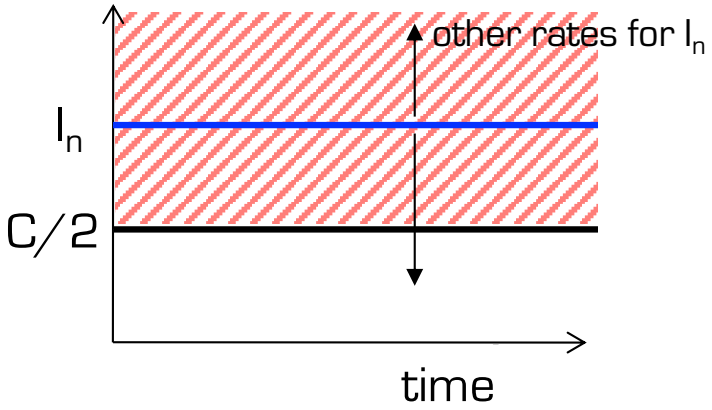
λ'_{in} Higher data rate sent by the transport layer including retransmissions

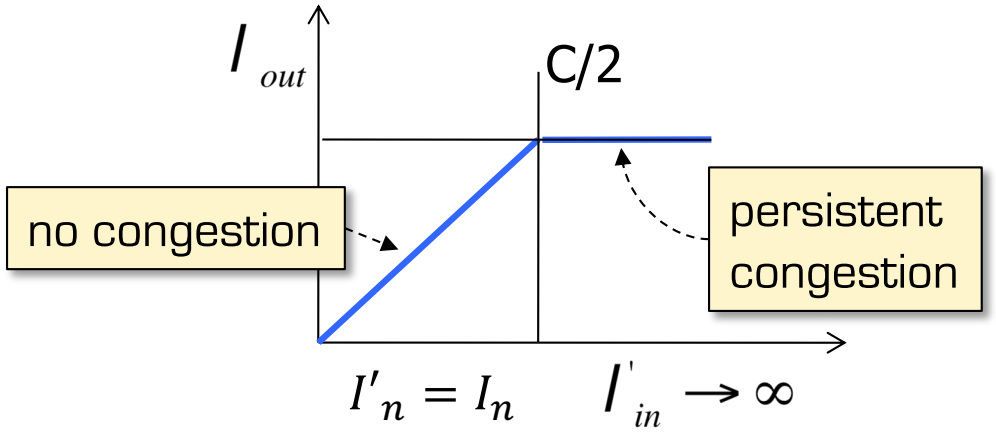
Reasons for congestion

- Retransmission means that I_{out} receives all data that I_{in} sends
- When there is loss, retransmission is happening, and $I'_{in} > I_{out}$
- Retransmission of delayed (but not lost) packets increases I'_{in} above the perfect value, without increasing I_{out}

- "Cost of congestion":
  - More work (retransmissions) for a desired throughput
  - Useless retransmissions, some links transmit several copies of the same packet

Constant bitrate (non-bursty) traffic

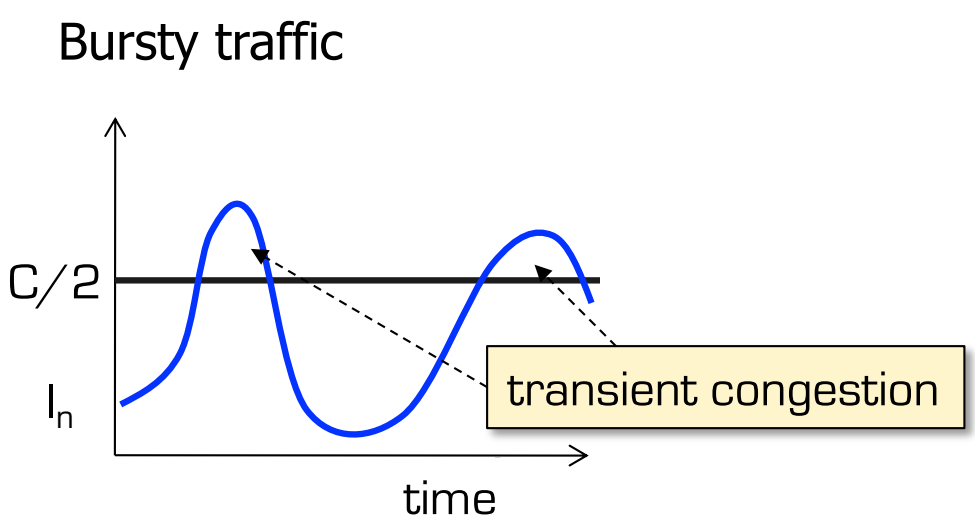


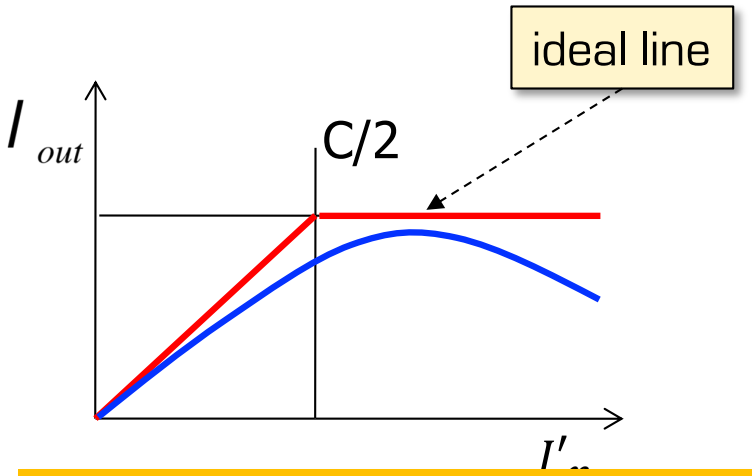


Reasons for congestion

- Retransmission means that I_{out} receives all data that I_{in} sends
- When there is loss, retransmission is happening, and $I'_{in} > I_{out}$
- Retransmission of delayed (but not lost) packets increases I'_{in} above the perfect value, without increasing I_{out}

- "Cost of congestion":
  - More work (retransmissions) for a desired throughput
  - Useless retransmissions, some links transmit several copies of the same packet





The share of retransmissions grows

Conclusion

- Protocols should implement congestion control
- Congestion control algorithms:
  - algorithms to adapt the offered load (I'_n) to the capacity at the bottleneck router
- There are quite many algorithms that can achieve this
  - at several layers
- We are going to look mainly at TCP's approach to congestion control

IN2140:
Introduction to Operating Systems and Data Communication



Transport layer

Congestion Control in TCP

Global congestion collapse

Craig Partridge, Research Director for the Internet Research Department at BBN Technologies:

> Bits of the network would fade in and out, but usually only for TCP. You could ping. You could get a UDP packet through. Telnet and FTP would fail after a while. And it depended on where you were going (some hosts were just fine, others flaky) and time of day (I did a lot of work on weekends in the late 1980s and the network was wonderfully free then).
>
> Around 1pm was bad (I was on the East Coast of the US and you could tell when those pesky folks on the West Coast decided to start work...).
>
> Another experience was that things broke in unexpected ways - we spent a lot of time making sure applications were bullet-proof against failures. (..)
>
> Finally, I remember being startled when Van Jacobson first described how truly awful network performance was in parts of the Berkeley campus. It was far worse than I was generally seeing. In some sense, I felt we were lucky that the really bad stuff hit just where Van was there to see it.

Internet congestion control: history

- **around 1986:** first congestion collapse

- **1988:** "Congestion Avoidance and Control" (Jacobson)
  Combined congestion/flow control for TCP
  (also: variation change to RTO calculation algorithm)

- Idea: packet loss = congestion, so throttle the rate; increase otherwise

- Goal: stability - in equilibrum, no packet is sent into the network until an old packet leaves
  - ack clocking, “conservation of packets“ principle
  - made possible through window based stop+go - behaviour

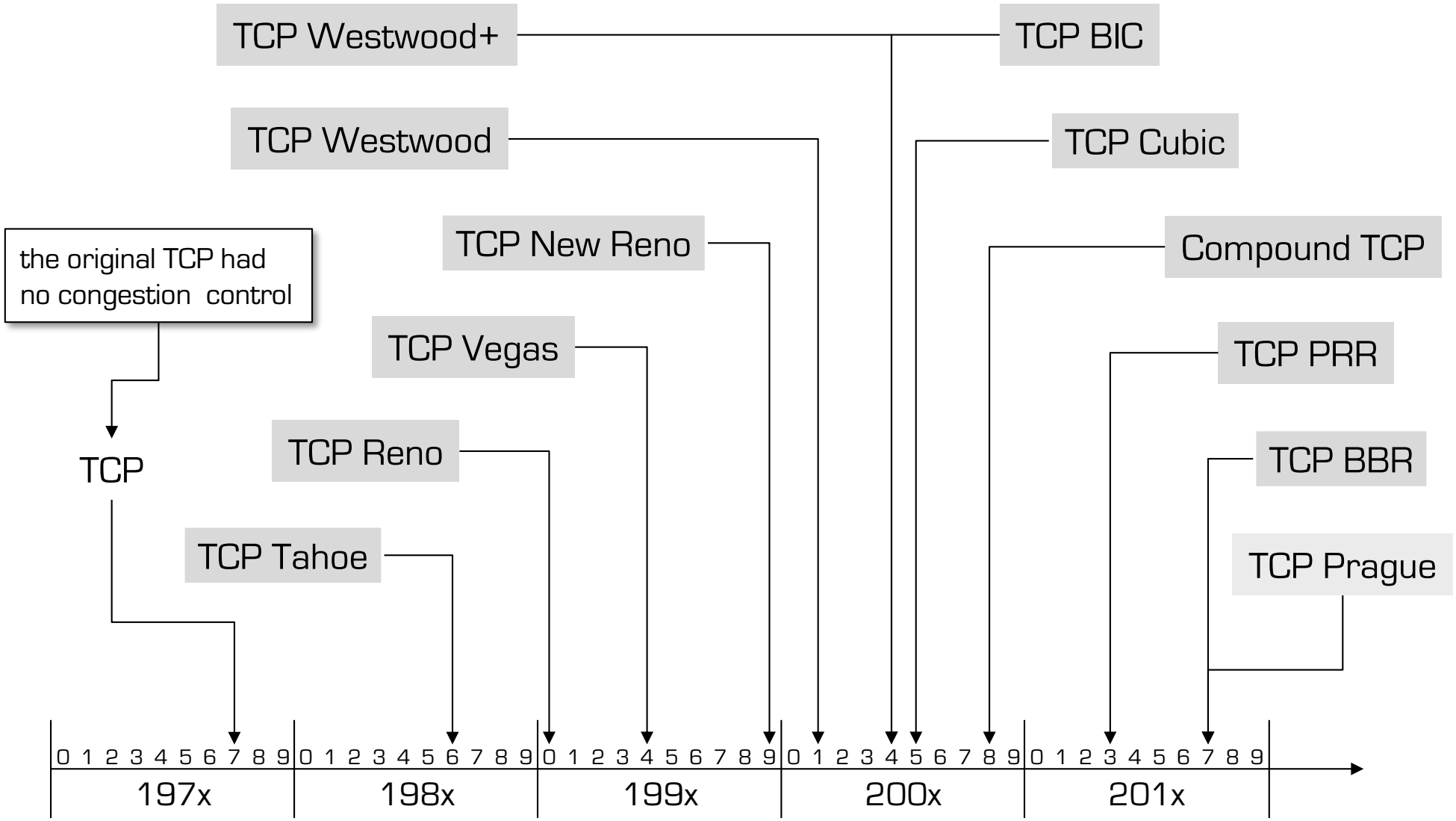
- Superposition of stable systems = stable →
  network based on TCP with congestion control = stable

TCP Congestion Control

- TCP limits sending rate as a function of perceived network congestion
  - little traffic – increase sending rate
  - much traffic – reduce sending rate

- Congestion algorithm has three major “components”:
  - additive-increase, multiplicative-decrease (AIMD)
  - slow-start
  - reaction to timeout events

TCP Congestion Control Algorithms



TCP Congestion Control

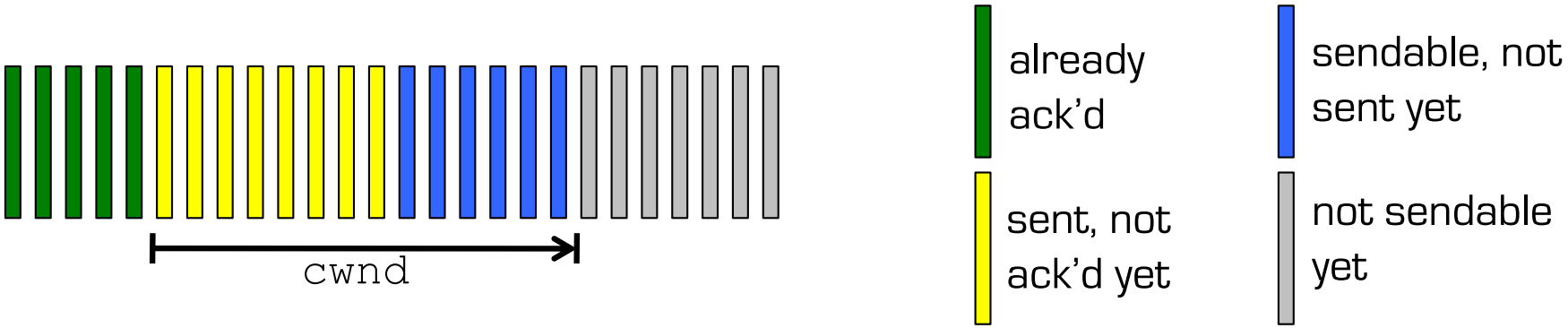
Basic terms:

- **congestion window** [`cwnd`]
  - largest amount of data (of a connection) that can be in the network at a time
- **maximum segment size** [MSS]
  - largest number of bytes that a TCP entity sends at once
  - *always* in IPv6 and *usually* in IPv4: largest payload that fits into an IP packet behind the TCP header
  - TCP header size is minimal 20 bytes
  - with options maximal 60 bytes

- Testing for available bandwidth
  - Ideally: send as fast as possible (`cwnd` as large as possible) without loss
  - Increase `cwnd` until you have loss
  - If loss, reduce `cwnd`, try increasing again

TCP Congestion Control

- End-to-end control (no support from the network layer)
- Send rate is limited by the size of a congestion window, `cwnd` (counted in bytes)

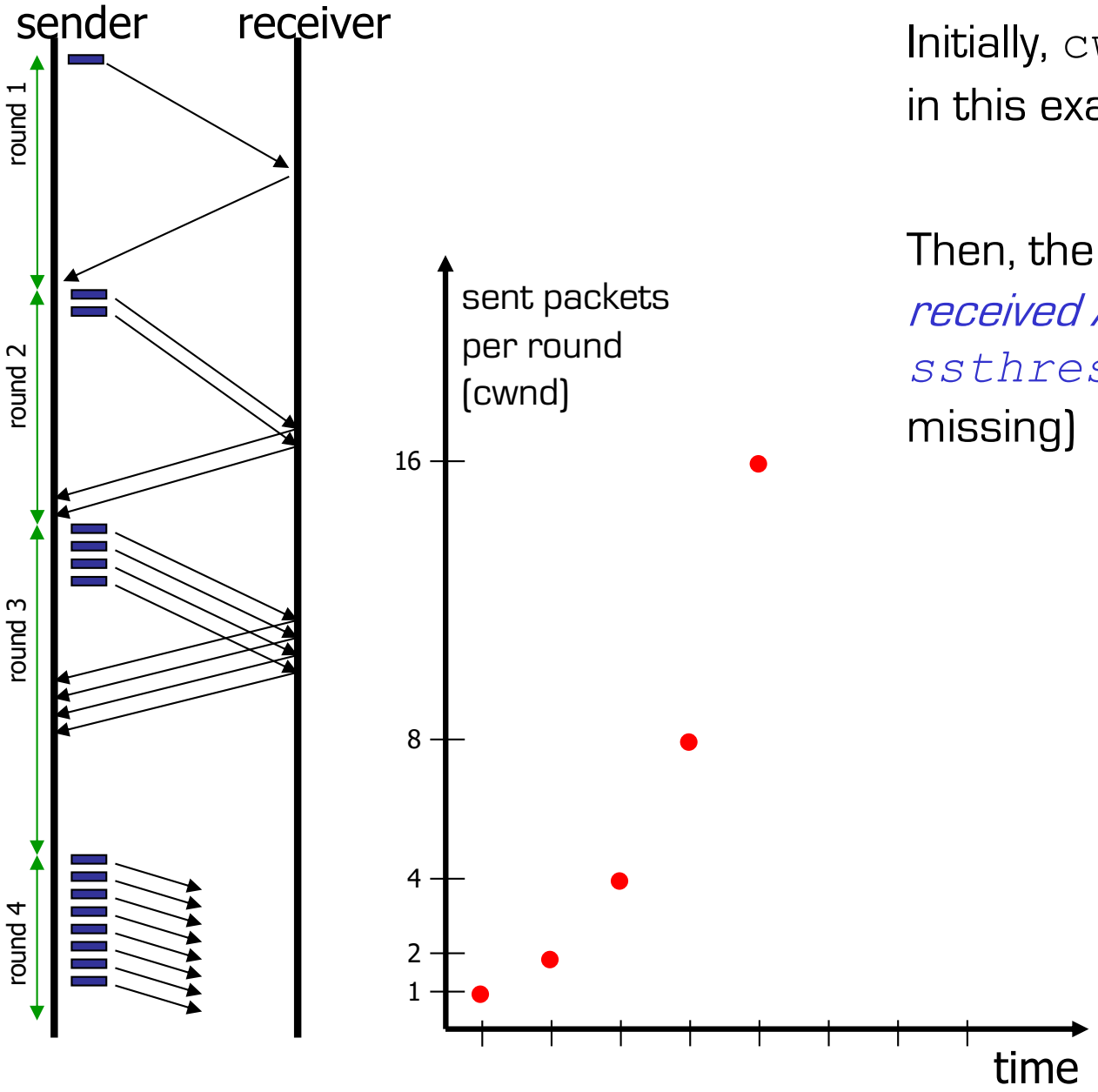


- `cwnd` bytes can be sent in each RTT:

$$sending\ rate = \frac{cwnd}{RTT}$$

- very often the permitted bytes are sent in $\frac{cwnd}{MSS}$ packets

TCP Congestion Control

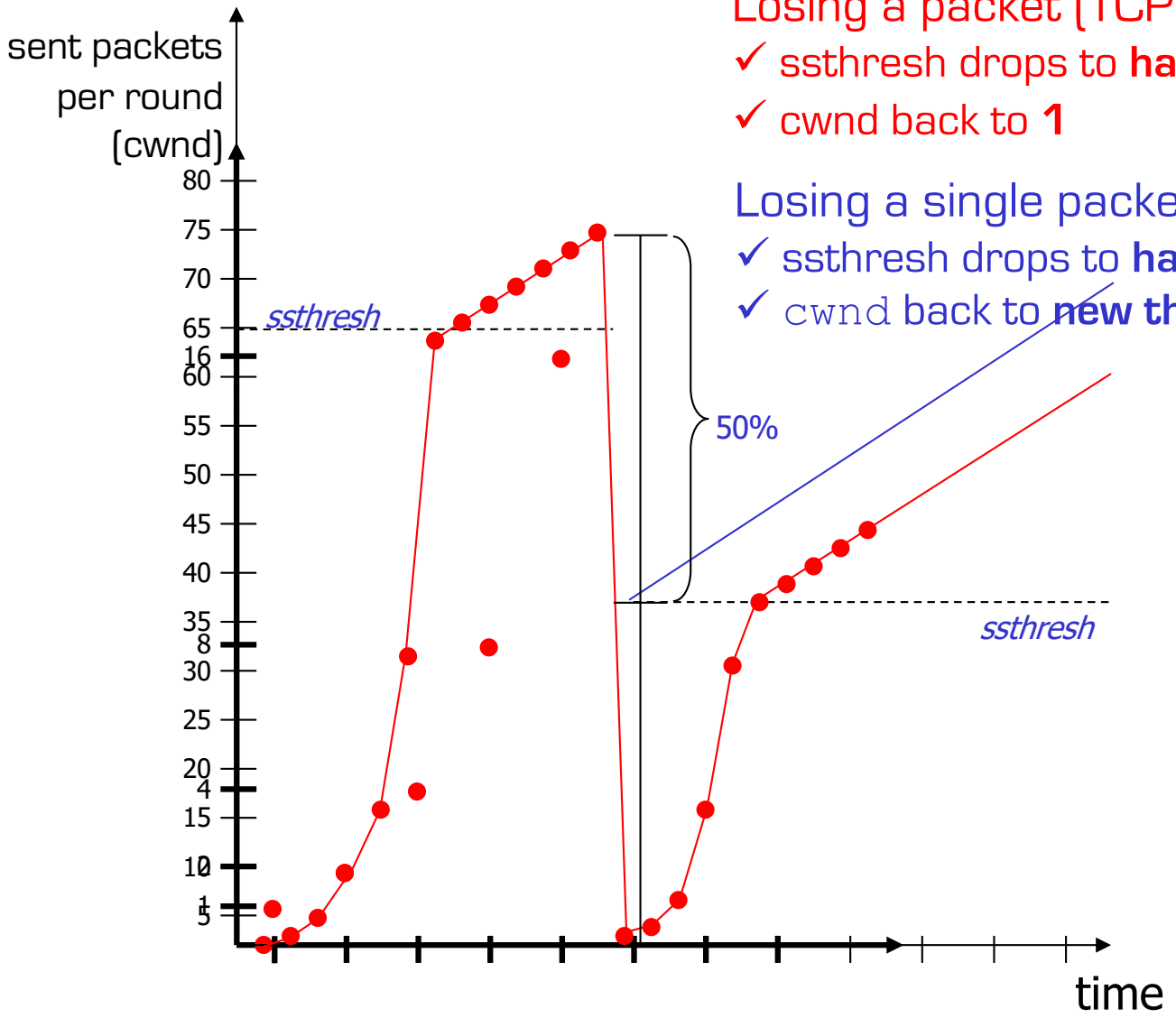


Initially, `cwnd` size is a few MSS
in this example: 1 MSS

Then, the size *increases by 1 for each received ACK* (until threshold `ssthresh` is reached or an ACK is missing)

TCP Congestion Control

Historically, initial `ssthresh` was 65 K, now in Linux it starts with ∞



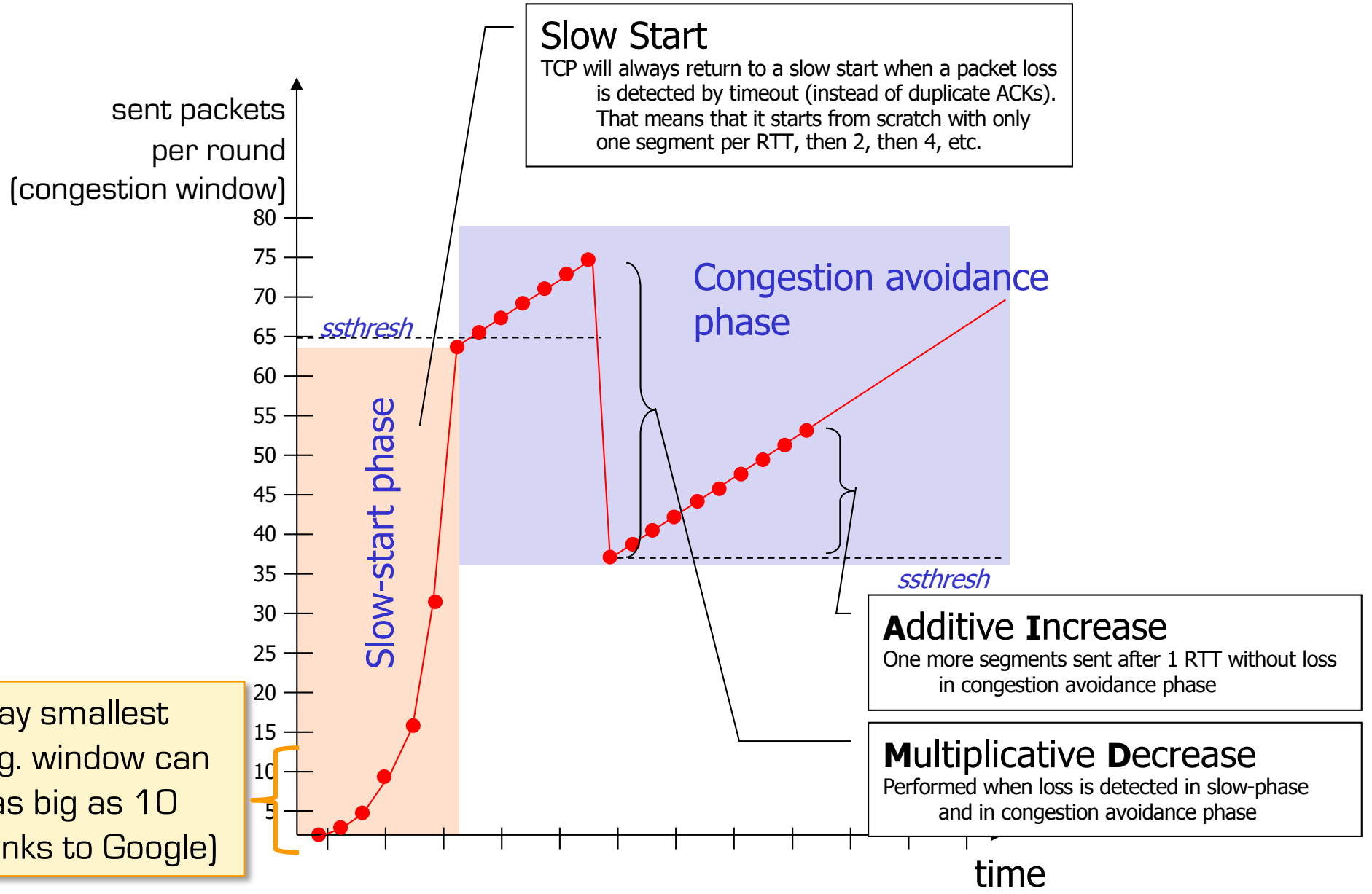
Losing a packet (TCP Tahoe):

- ✓ ssthresh drops to **half `cwnd`**
- ✓ cwnd back to **1**

Losing a single packet (TCP Reno):

- ✓ ssthresh drops to **half `cwnd`**
- ✓ `cwnd` back to **new threshold**

TCP Congestion Control



TCP Fairness

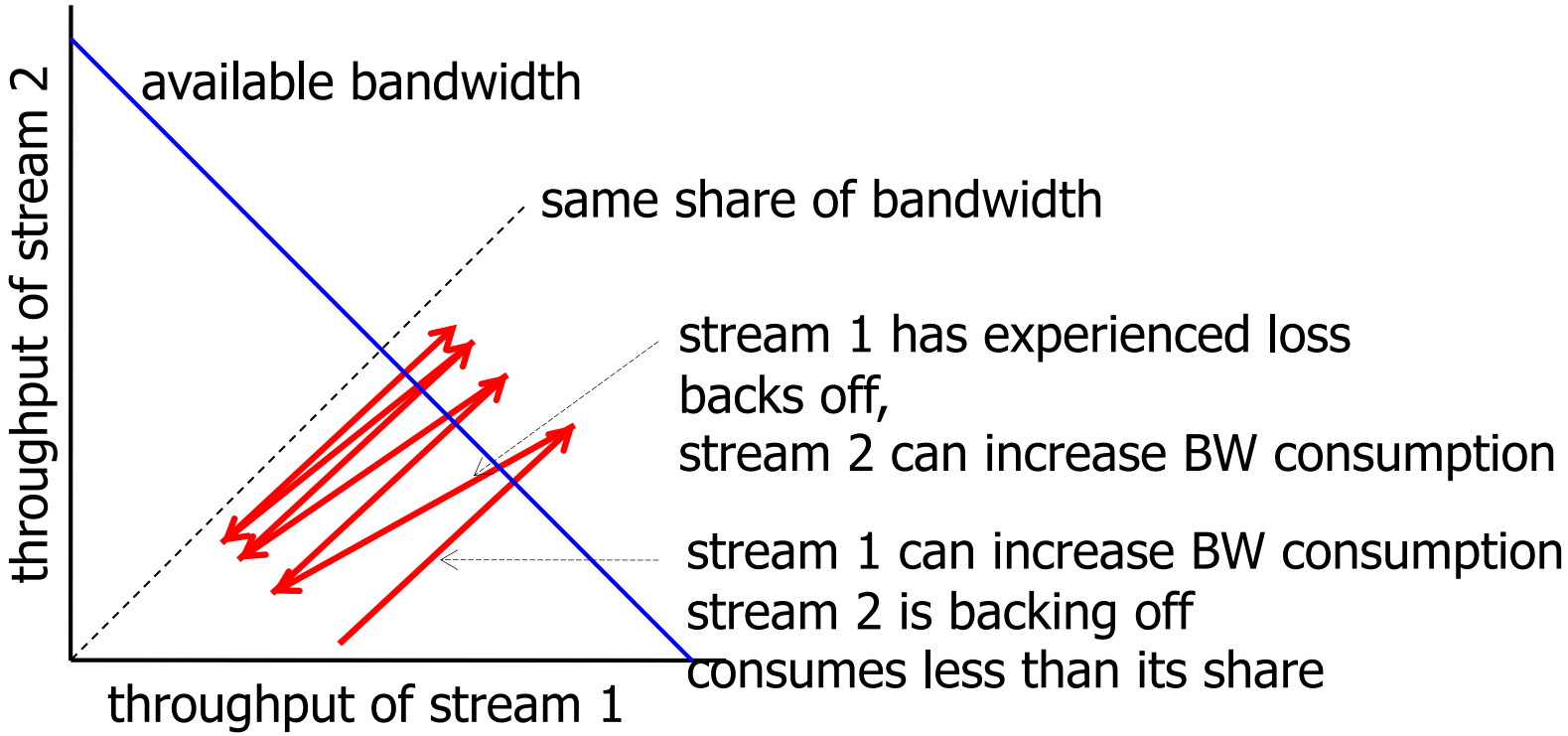
- Goal of fairness
  - When N TCP streams share a bottleneck,
    each TCP stream should receive an n^{th} of the bottleneck bandwidth

- more realistic demand
  - When N TCP streams with the same RTT and loss rate share a bottleneck,
    and they are infinitely long,
    each TCP stream receives an n^{th} of the bottleneck bandwidth

- but the approximation is in many cases good

TCP Congestion Control

- How does the AIMD algorithm achieve congestion control?
  - interpret every packet loss as a congestion signal
  - force TCP entities to slow down when loss is experienced

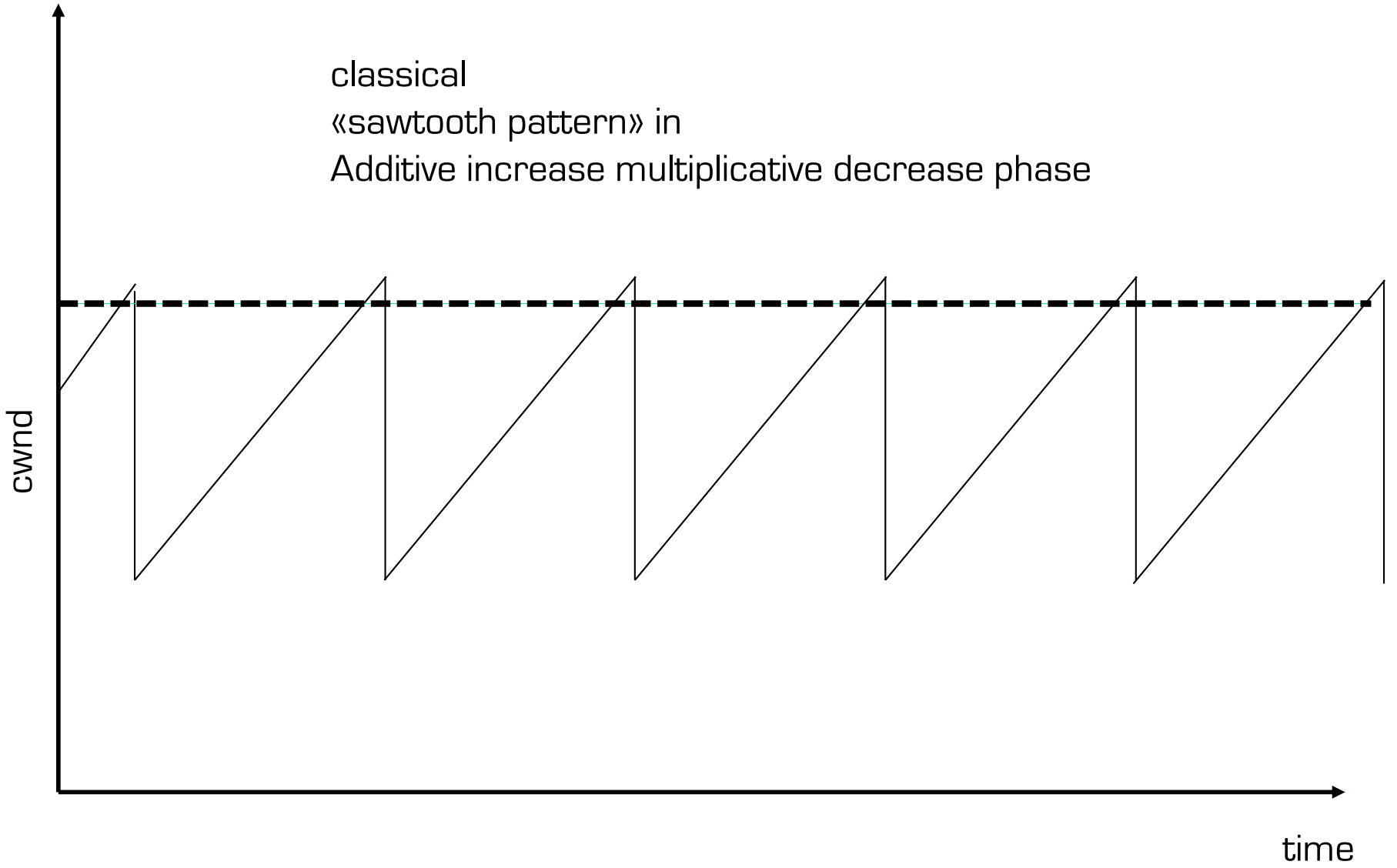




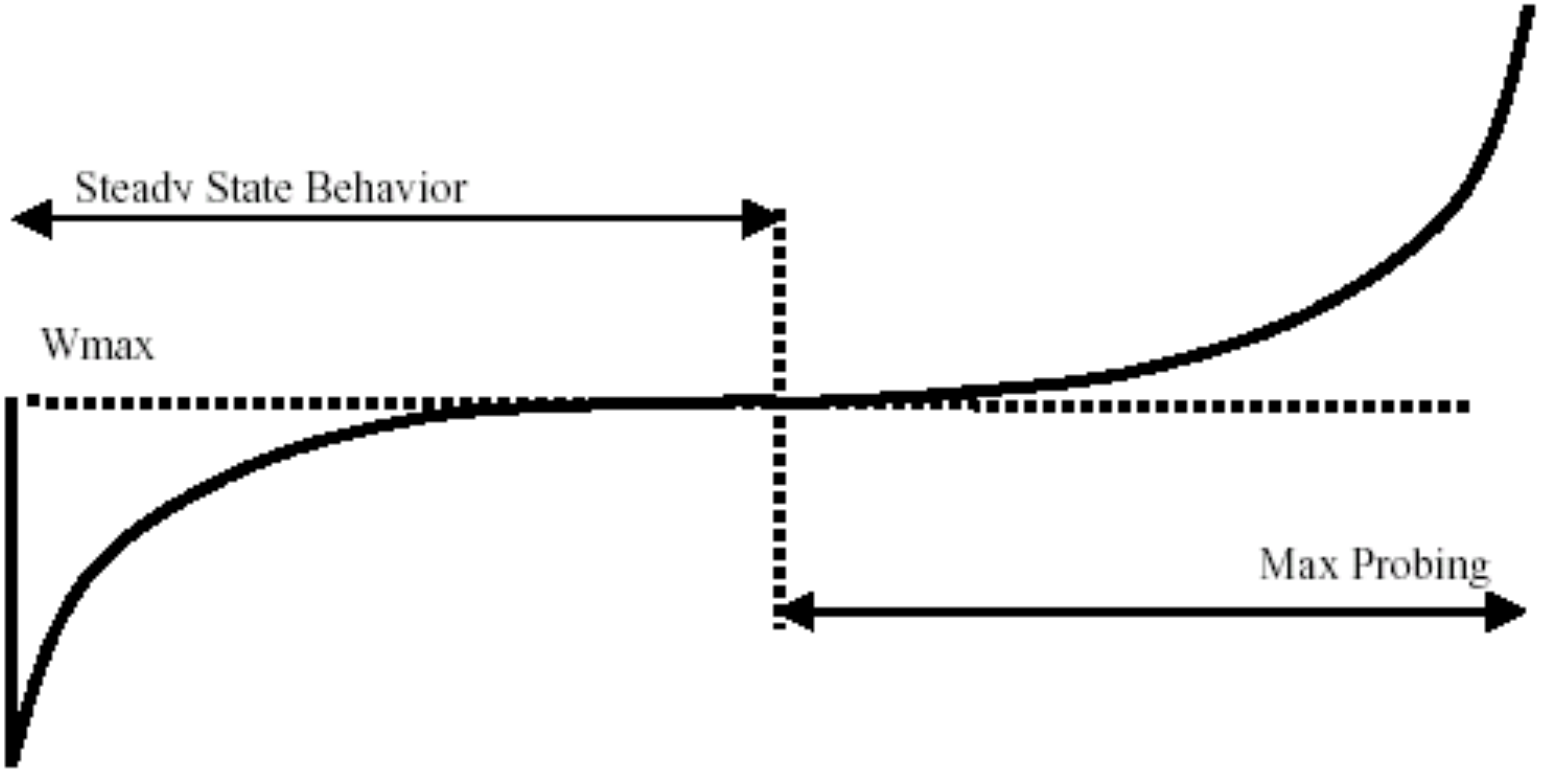
CUBIC: a new shape

use a cubic function for probing

TCP New Reno

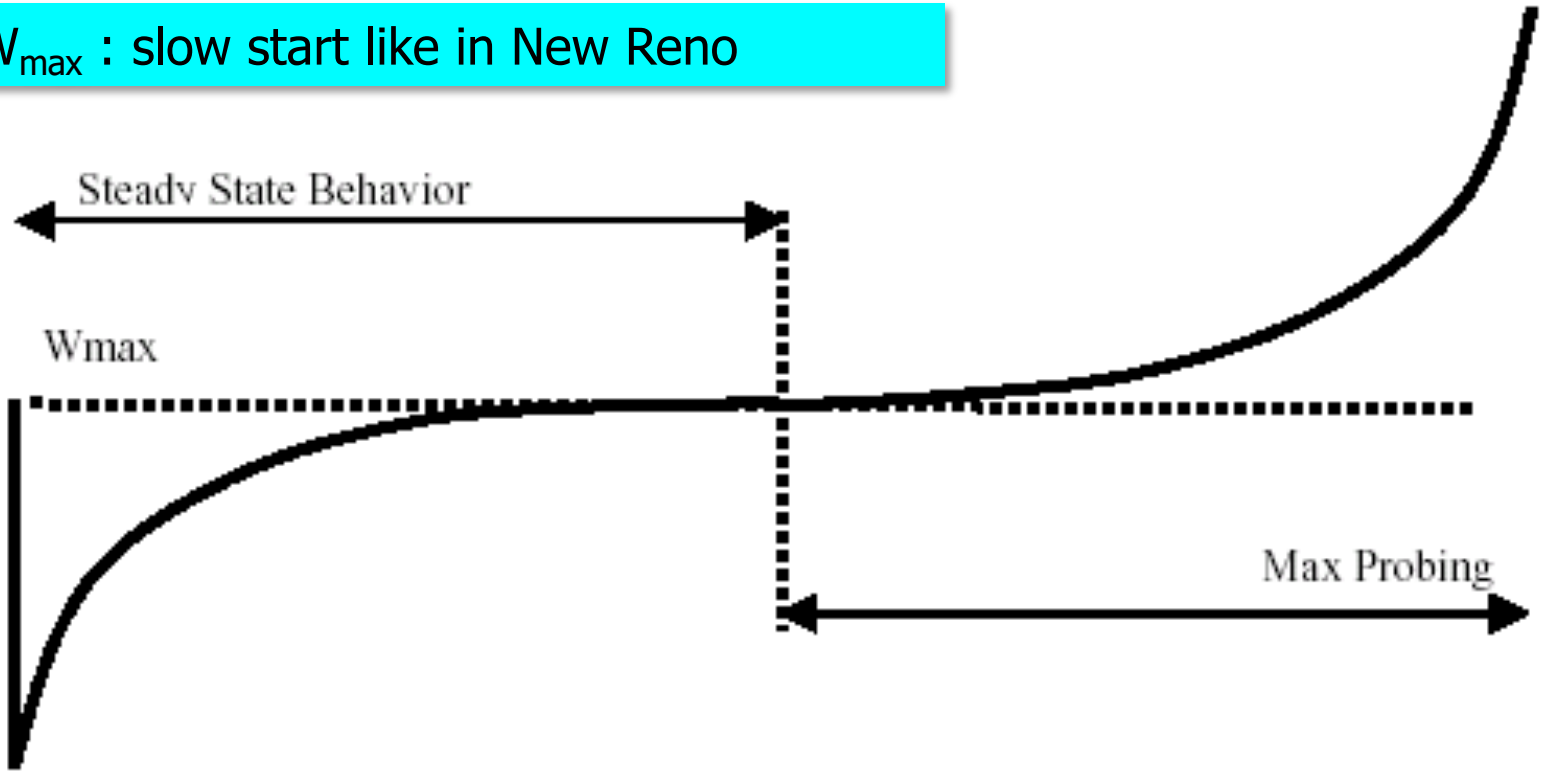

classical
«sawtooth pattern» in
Additive increase multiplicative decrease phase
cwnd
time

TCP CUBIC



TCP CUBIC

initial W_{max} : slow start like in New Reno



Concave region

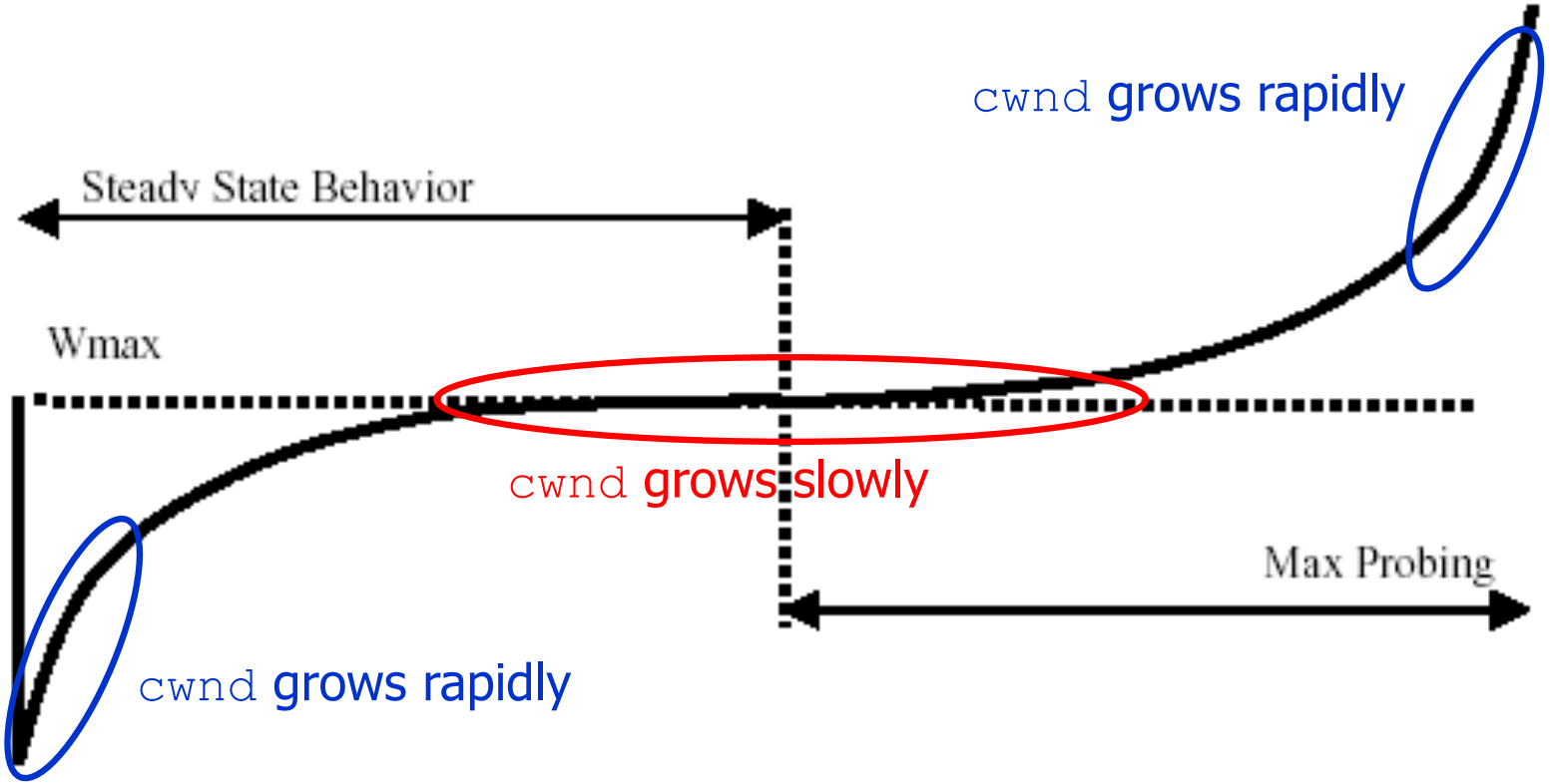
Convex region

TCP CUBIC

- The Window is a cubic function of the time since the last congestion event

- CUBIC does not rely on receiving ACKs to increase window size
  - Depends on on last congestion event
  - CUBIC is better for long-distance connections, not for short-distance ones
  - Use TCP New Reno behaviour for short-distance ones

TCP CUBIC



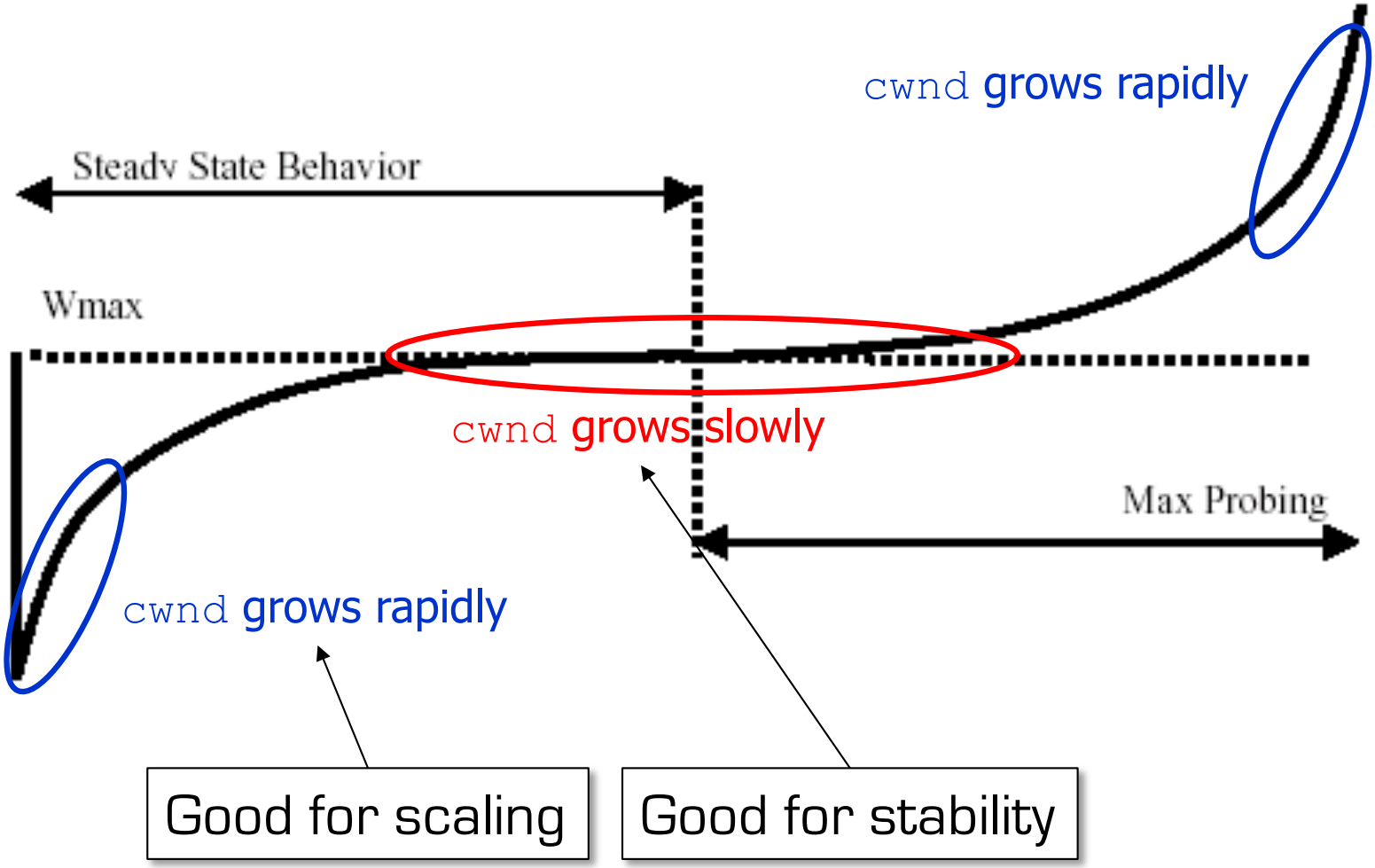
$$W_{cubic} = C(t-K)^3 + W_{\max} \qquad K = \sqrt[3]{W_{\max}\beta/C}$$

C is a scaling factor

t is the elapsed time from the last window reduction

β is a constant multiplication decrease factor

TCP CUBIC



But: CUBIC converges slower than New Reno when flows appear or disappear



Transport above transport

QUIC

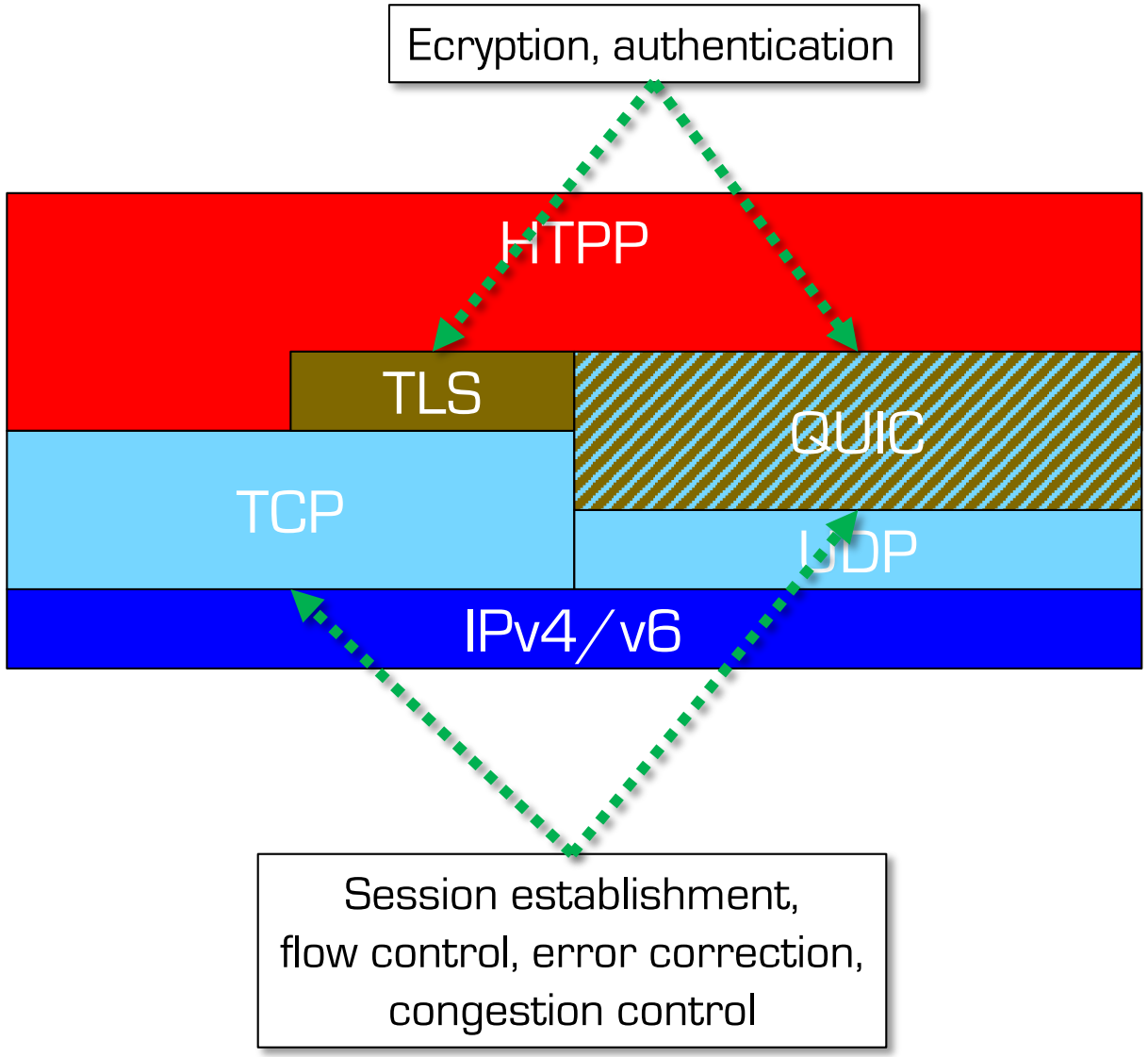
Quick UDP Internet Connections

QUIC's mission

- Replace HTTP over TCP as the main protocol for web traffic

- TCP's problems
  - Head of line blocking
  - 3-way handshake
  - Small initial congestion window and slow start
  - Large backoff caused by packet loss
  - Small socket buffer
  - NAT timeout and IP roaming
  - TCP buffer block
  - Security: TLS (transport layer security) is expensive

QUIC's mission



QUIC goals

- Deploy in today's Internet

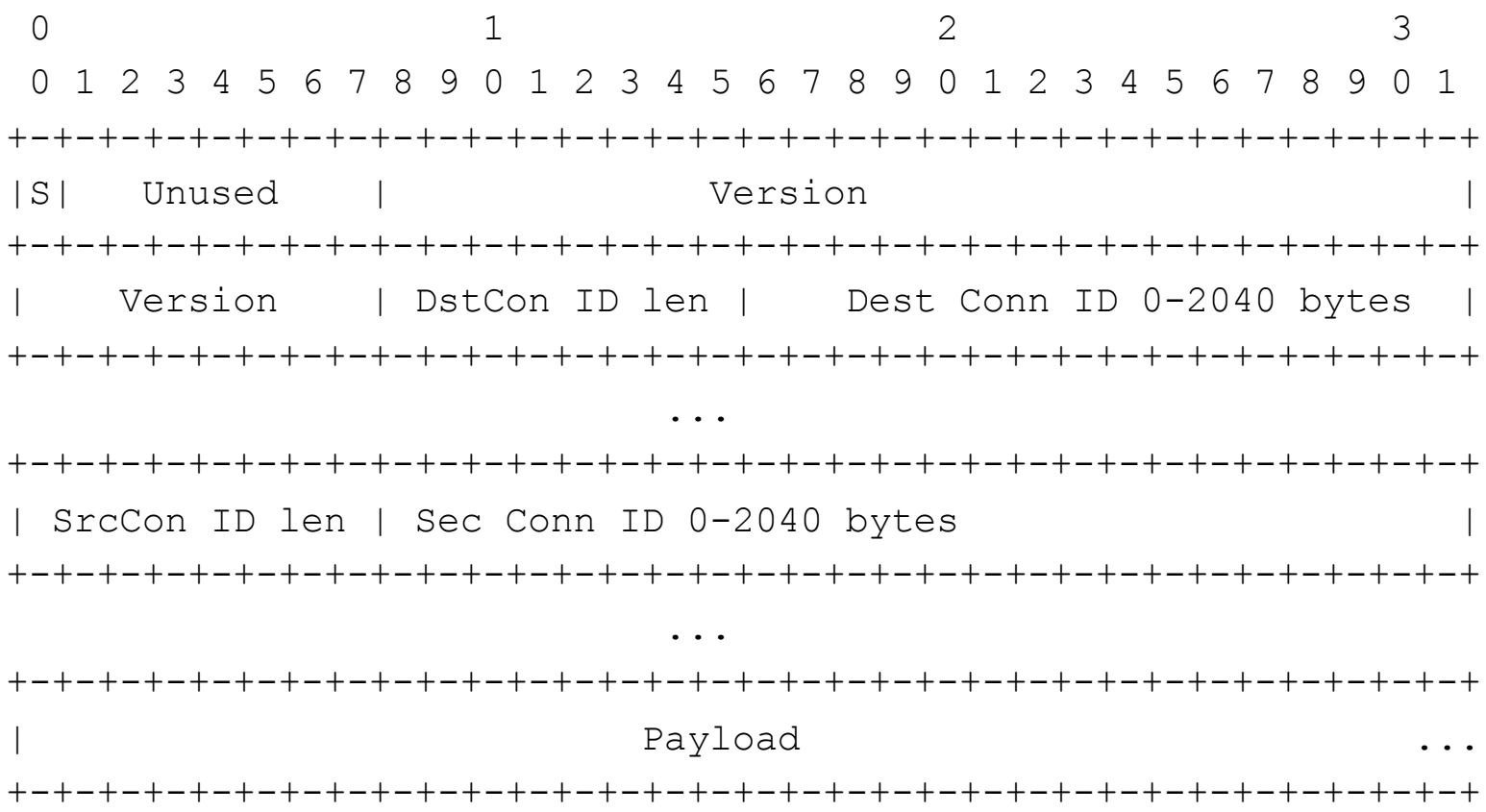
Transport layer tasks

- Low latency
  - In connect and responses
- Reliable stream support
  - Like TCP
  - But without head-of-line blocking
- Better congestion avoidance than TCP
  - Keep experimenting! This is in user space

Session layers tasks

- Privacy and security comparable to TLS
  - Always end-to-end connection
  - No benefits from caching
- Mobile interface migration

QUIC protocol header



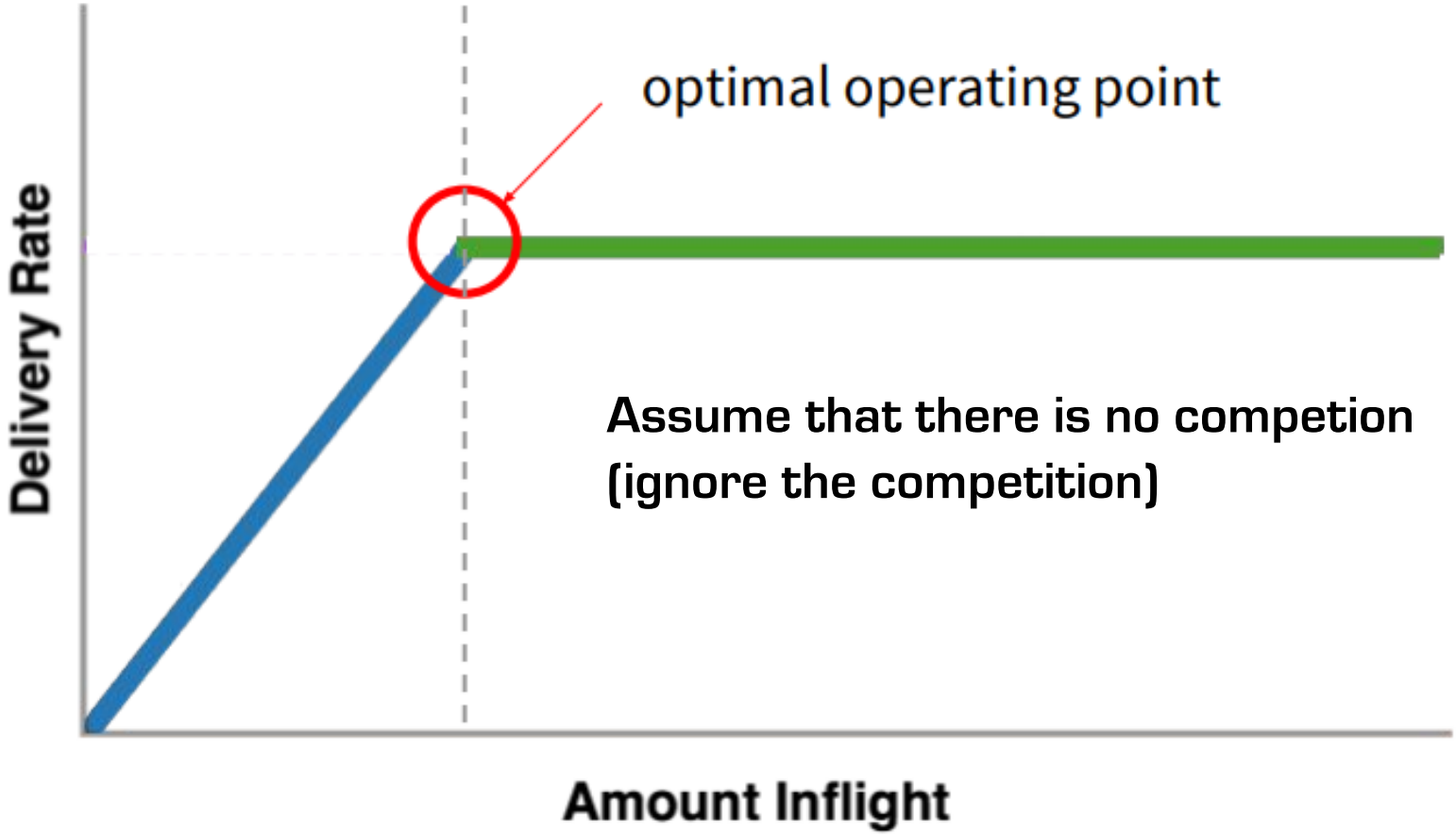
Very long and very flexible header

It is an application-layer protocol specifically targeting Chrome and Google servers

It can change every day on both sides

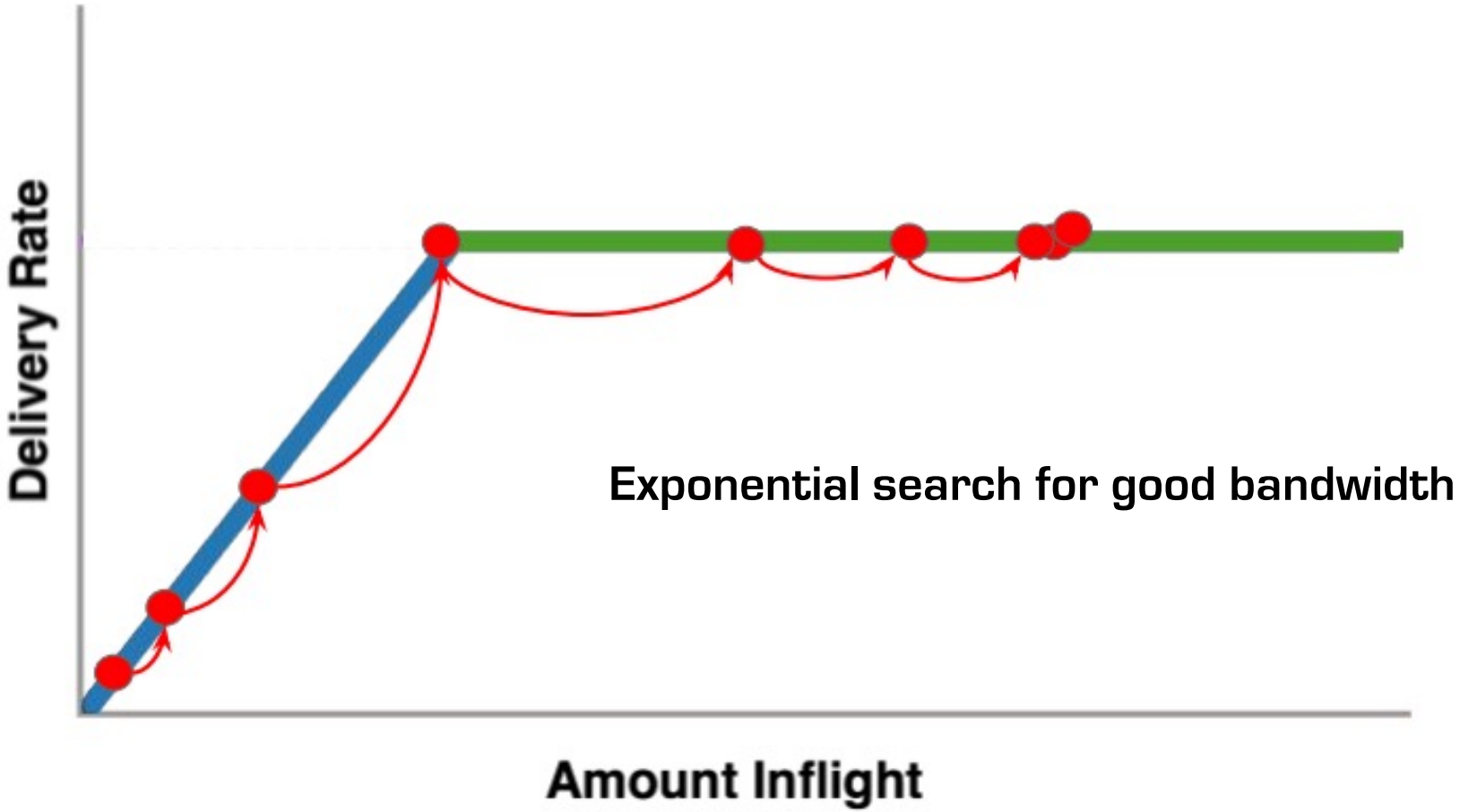
QUIC's congestion control: BBR

Searching for the ideal bandwidth



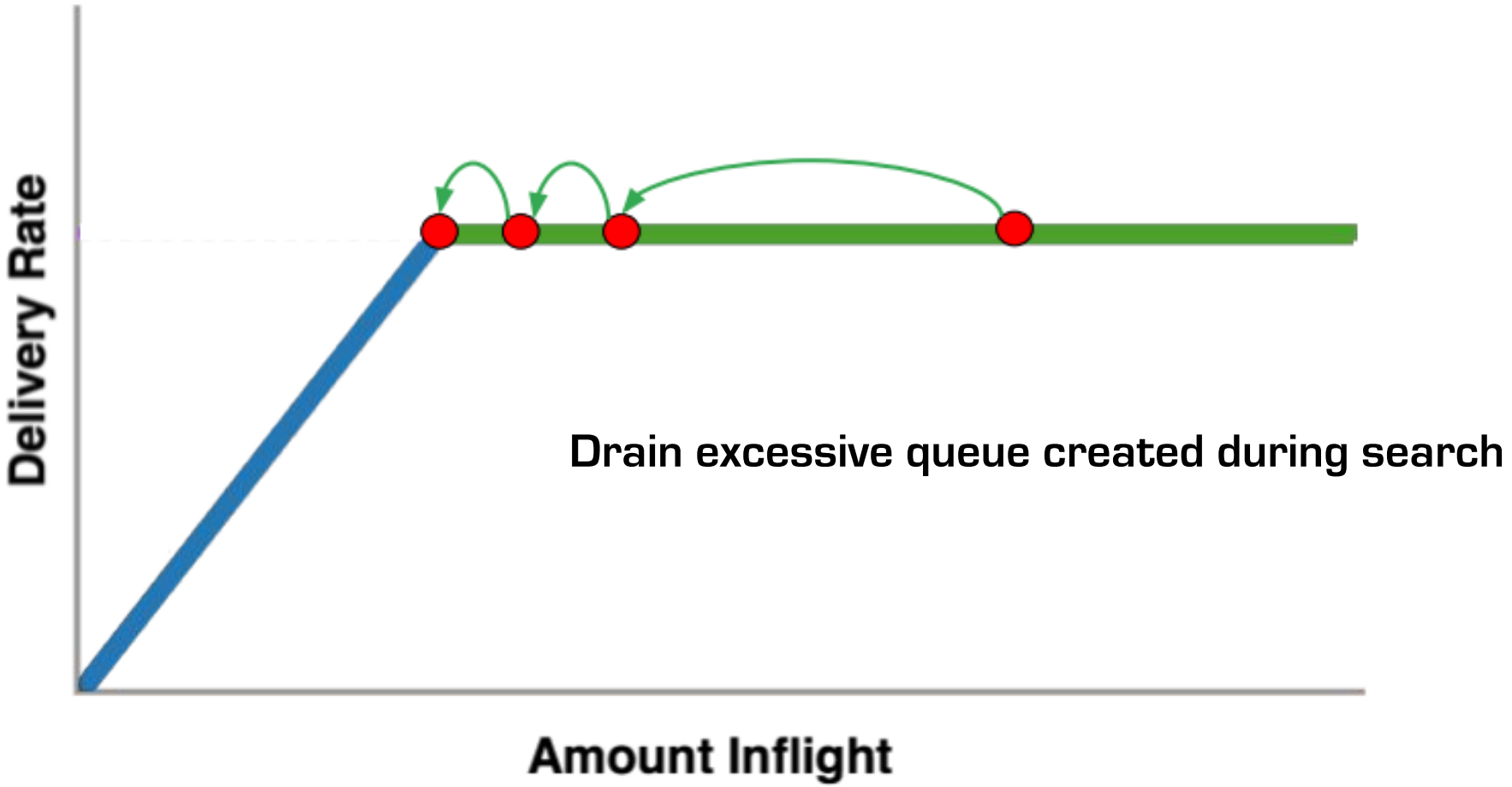
QUIC's congestion control: BBR

Searching for the ideal bandwidth



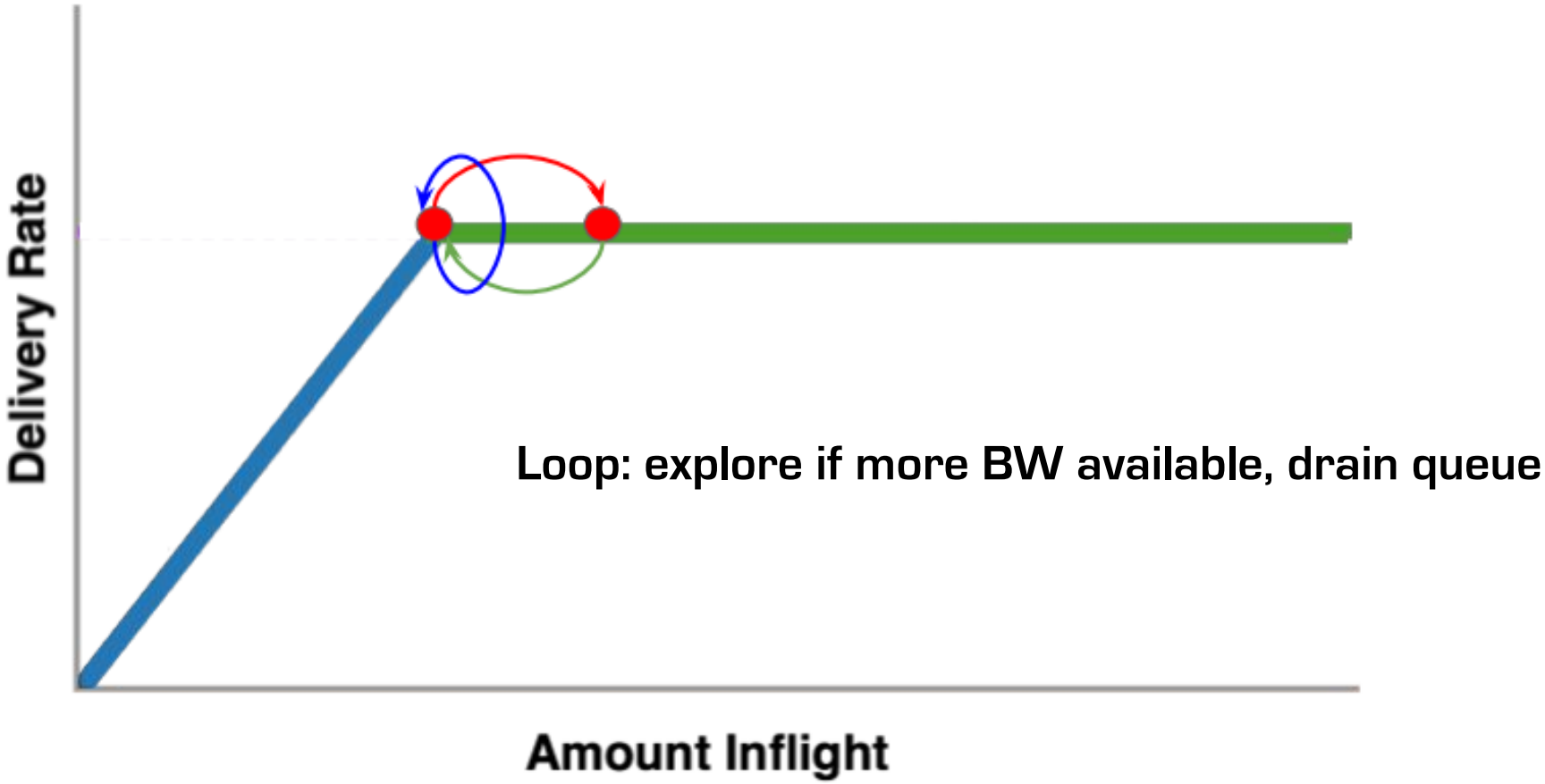
QUIC's congestion control: BBR

Searching for the ideal bandwidth



QUIC's congestion control: BBR

Searching for the ideal bandwidth



QUIC's congestion control: BBR

Searching for the ideal bandwidth

Send really little for a short time
Give other flows a chance

