EEE4121F-B

Congestion in Networks

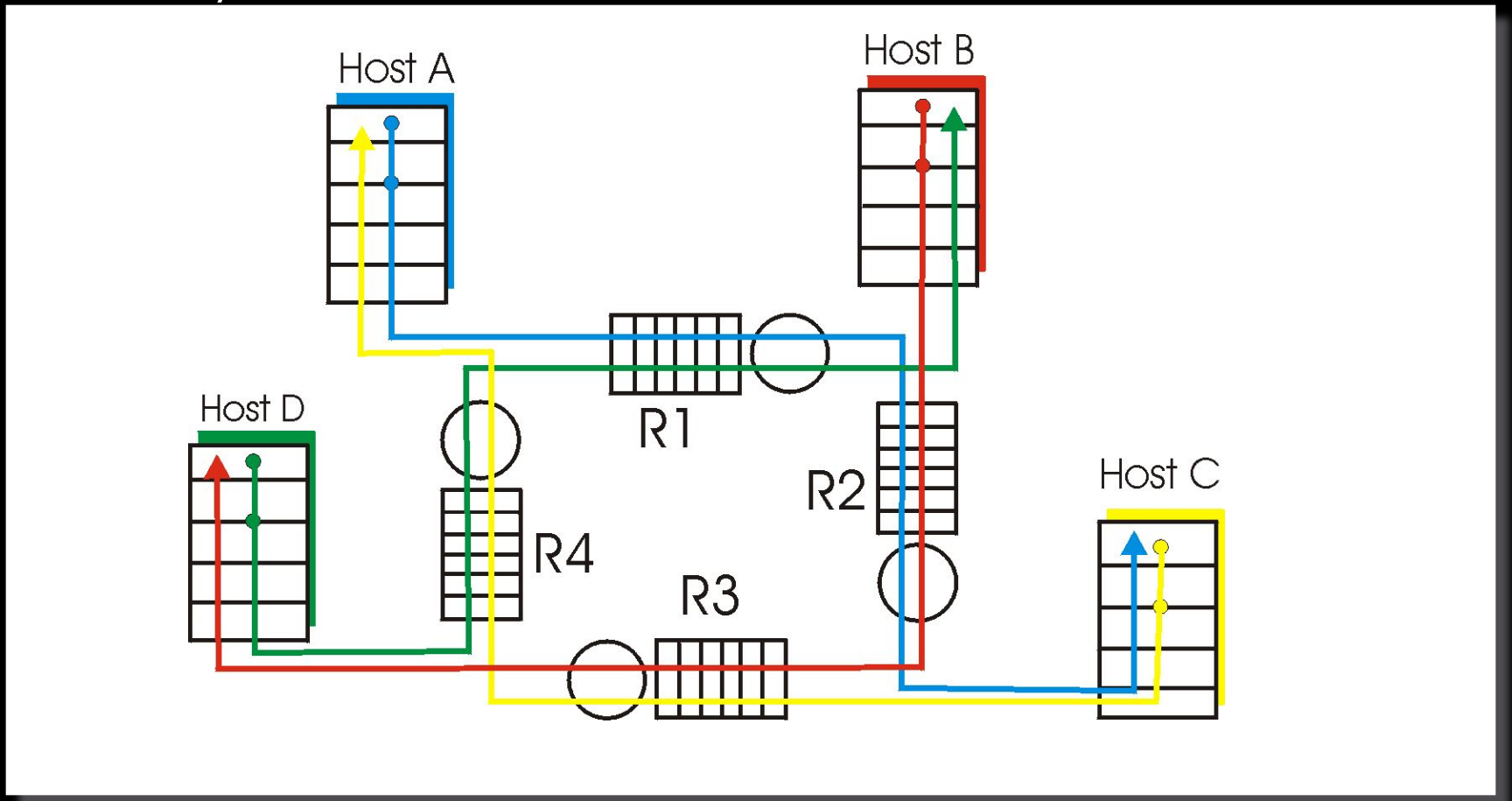
Internet Pipes?

- How should you control the faucet?
  - Too fast – sink overflows
  - Too slow – wasted resources
- Goals
  - Fill the bucket as quickly as possible
  - Avoid overflowing the sink
- Solution – watch the sink

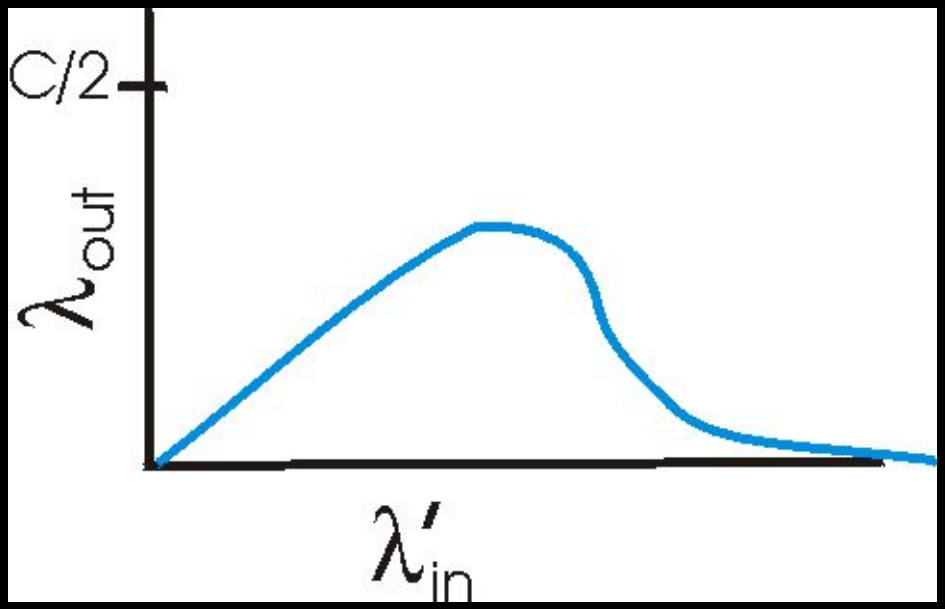
Causes & Costs of Congestion

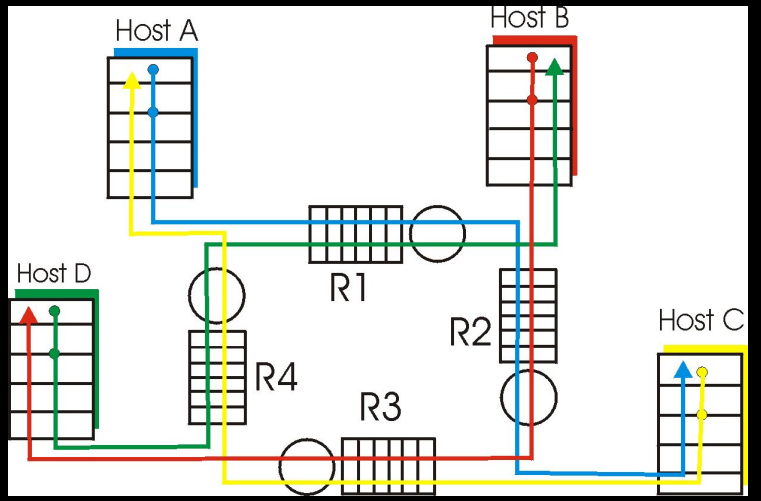
- Four senders – multihop paths
- Timeout/retransmit

Q: What happens as rate increases?



Causes & Costs of Congestion





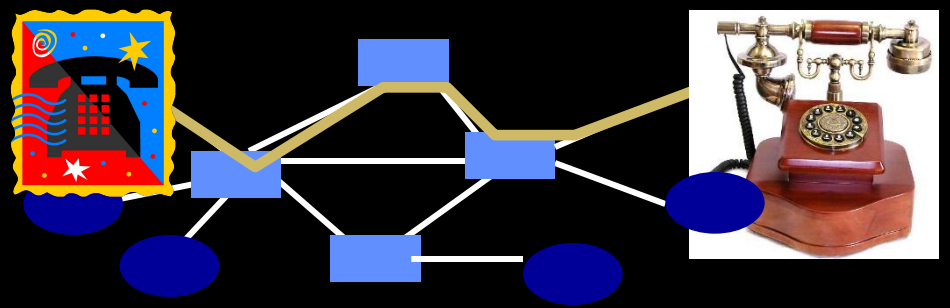
- When packet dropped, any "upstream transmission capacity used for that packet was wasted!

Congestion Collapse

- Definition: *Increase in network load results in decrease of useful work done*
- Many possible causes
  - Spurious retransmissions of packets still in flight
    - Classical congestion collapse
    - How can this happen with packet conservation? RTT increases!
    - Solution: better timers and TCP congestion control
  - Undelivered packets
    - Packets consume resources and are dropped elsewhere in network
    - Solution: congestion control for ALL traffic

No Problem Under Circuit Switching

- Source establishes connection to destination
  - Nodes reserve resources for the connection
  - Circuit rejected if the resources aren't available
  - Cannot have more than the network can handle



Congestion Control and Avoidance

- A mechanism that:
  - Uses network resources efficiently
  - Preserves fair network resource allocation
  - Prevents or avoids collapse
- Congestion collapse is not just a theory
  - Has been frequently observed in many networks

Congestion Control Approaches

- Two broad approaches

- End-end congestion control:
  - No explicit feedback from network
  - Congestion inferred from end-system observed loss, delay
  - Approach taken by TCP

- Network-assisted congestion control:
- Routers provide feedback to end systems
- Single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
- Explicit rate sender should send at
- Problem: makes routers complicated

Congestion Case Study: TCP/IP vs ATM

TCP Mechanism

- Review TCP concepts
  - 5.2 and 6.3 in Computer Networks: A systems approach or
  - 3.5, 3.6 and 3.7 in Computer Networking: A top down approach

How ATM Works?

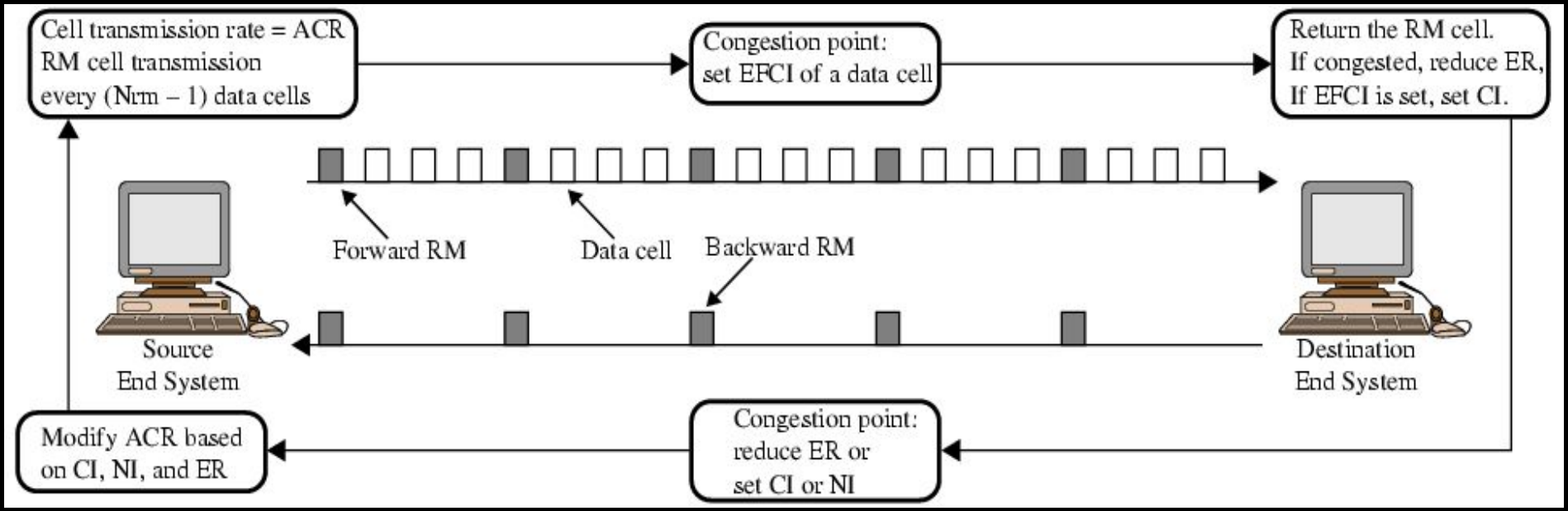
- ATM is connection-oriented -- an end-to-end connection must be established and routing tables setup prior to cell transmission
- Once a connection is established, the ATM network will provide end-to-end Quality of Service (QoS) to the end users
- All traffic, whether voice, video, image, or data is divided into 53-byte cells and routed in sequence across the ATM network
- Routing information is carried in the header of each cell
- Routing decisions and switching are performed by hardware in ATM switches
- Cells are reassembled into voice, video, image, or data at the destination

ATM SERVICES

Service: transport cells across ATM network
- analogous to IP network layer
- very different services than IP network layer

Network Architecture	Service Model	Guarantees ?				Congestion feedback
		Bandwidth	Loss	Order	Timing	
Internet	best effort	none	no	no	no	no (inferred via loss)
ATM	CBR	constant rate	yes	yes	yes	no congestion
ATM	VBR	guaranteed rate	yes	yes	yes	no congestion
ATM	ABR	guaranteed minimum	no	yes	no	yes
ATM	UBR	none	no	yes	no	no

Cell flow on ABR connection (Cont'd)



ABR Feedback v TCP ACK

- ABR feedback controls rate of transmission
  - Rate control
- TCP feedback controls window size
  - Credit control
- ABR feedback from switches or destination
- TCP feedback from destination only

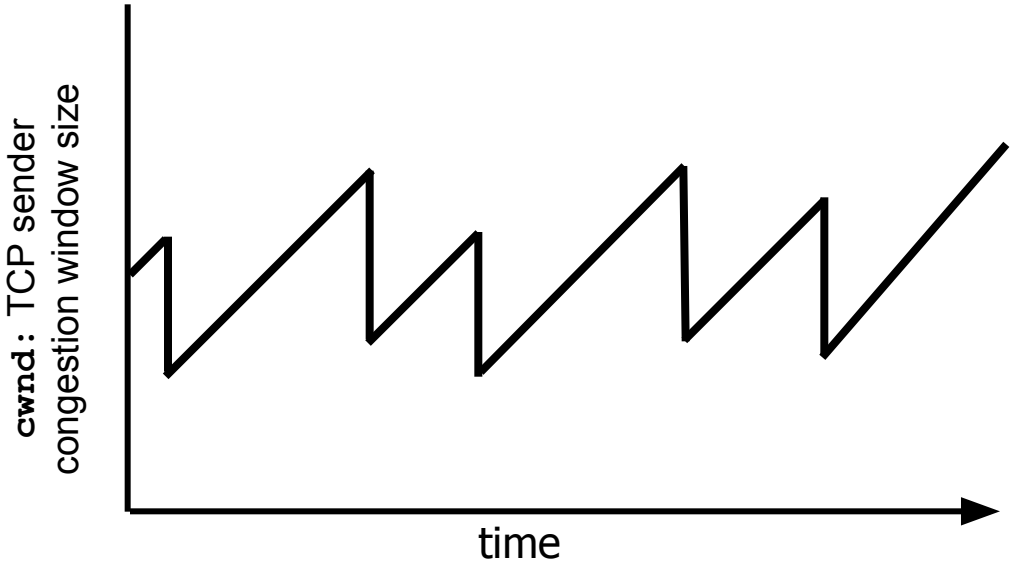
Congestion control in TCP/IP is difficult

- IP is connectionless and stateless, with no provision for detecting or controlling congestion
- TCP only provides end-to-end flow control and can only deduce presence of congestion by indirect means
- No cooperative, distributed algorithm to bind together various TCP entities

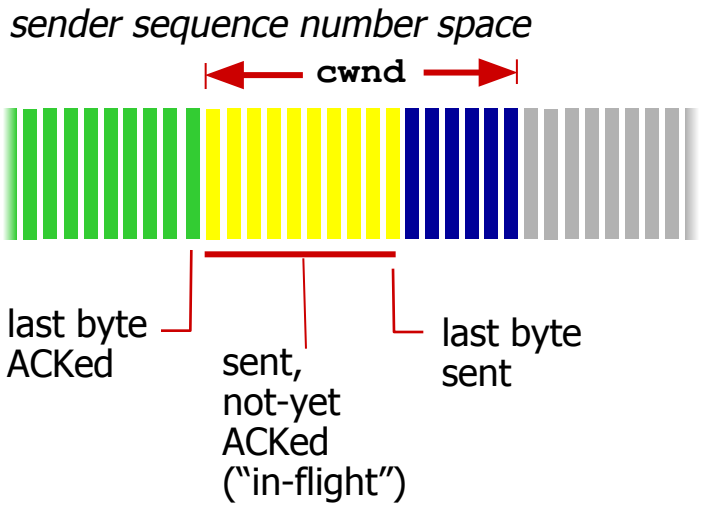
TCP congestion control: additive increase multiplicative decrease

- *approach:* sender increases transmission rate (window size), probing for usable bandwidth, until loss occurs
  - *additive increase:* increase **`cwnd`** by 1 MSS every RTT until loss detected
  - *multiplicative decrease*: cut **`cwnd`** in half after loss

AIMD saw tooth behavior: probing for bandwidth



TCP Congestion Control: details



- sender limits transmission:

$$\texttt{LastByteSent-LastByteAcked} \leq \texttt{cwnd}$$

- **cwnd** is dynamic, function of perceived network congestion

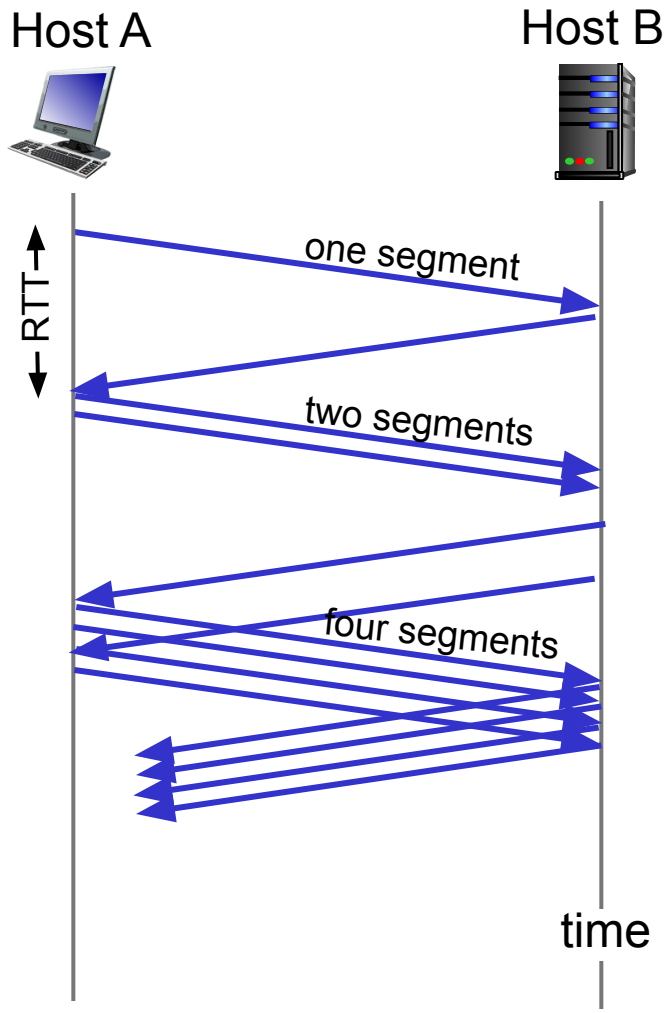
TCP sending rate:

- *roughly:* send cwnd bytes, wait RTT for ACKS, then send more bytes

$$\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

TCP Slow Start

- when connection begins, increase rate exponentially until first loss event:
  - initially **`cwnd`** = 1 MSS
  - double **`cwnd`** every RTT
  - done by incrementing **`cwnd`** for every ACK received
- *summary:* initial rate is slow but ramps up exponentially fast



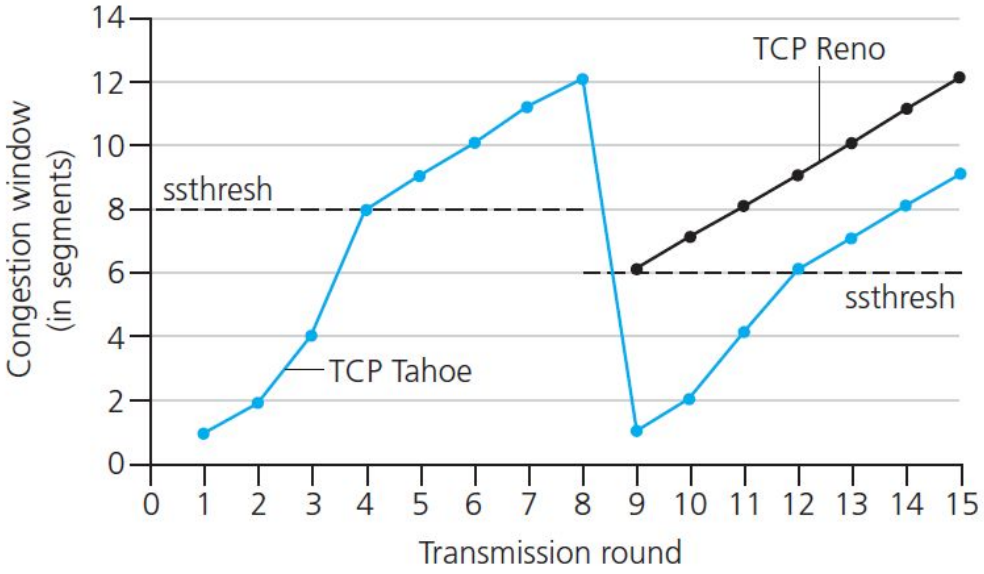
TCP: detecting, reacting to loss

- loss indicated by timeout:
  - **`cwnd`** set to 1 MSS;
  - window then grows exponentially (as in slow start) to threshold, then grows linearly
- loss indicated by 3 duplicate ACKs: TCP RENO
  - dup ACKs indicate network capable of delivering some segments
  - **`cwnd`** is cut in half window then grows linearly
- TCP Tahoe always sets **`cwnd`** to 1 (timeout or 3 duplicate acks)

TCP: switching from slow start to CA

Q: when should the exponential increase switch to linear?

A: when **`cwnd`** gets to 1/2 of its value before timeout.



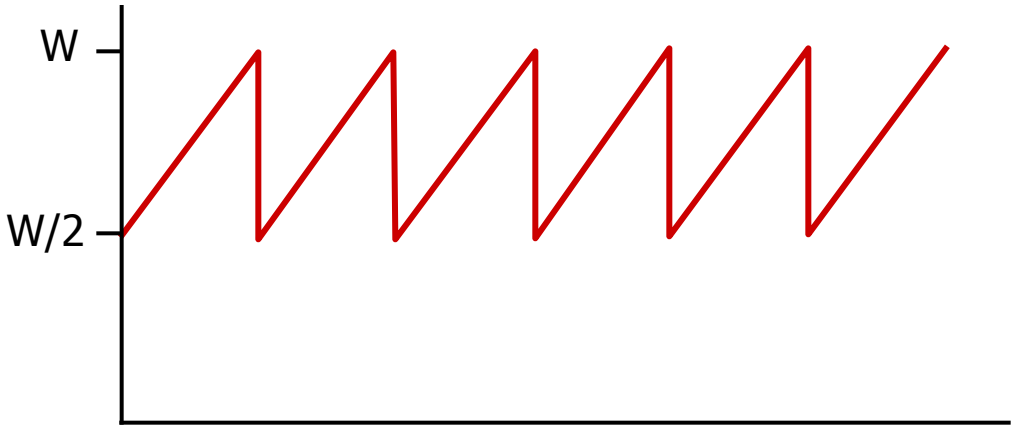
Implementation:

- variable **`ssthresh`**
- on loss event, **`ssthresh`** is set to 1/2 of **`cwnd`** just before loss event

TCP throughput

- avg. TCP thruput as function of window size, RTT?
  - ignore slow start, assume always data to send
- W: window size (measured in bytes) where loss occurs
  - avg. window size (# in-flight bytes) is ¾ W
  - avg. thruput is 3/4W per RTT

$$\text{avg TCP thruput} = \frac{3}{4}\frac{W}{RTT} \text{ bytes/sec}$$



TCP Futures: TCP over "long, fat pipes"

- example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput
- requires W = 83,333 in-flight segments
- throughput in terms of segment loss probability, L [Mathis 1997]:

$$\text{TCP throughput} = \frac{1.22 \cdot \text{MSS}}{\text{RTT}\sqrt{L}}$$

➜ to achieve 10 Gbps throughput, need a loss rate of L = $2 \cdot 10^{-10}$ *– a very small loss rate!*

- new versions of TCP for high-speed NEEDED

New TCP Versions

- Old versions of TCP have poor performance in high speed, high RTT networks (“long, fat networks”)
- These networks are characterized by large bandwidth and delay product (BDP) which represents the total number of packets needed in flight while keeping the bandwidth fully utilized, in other words, the size of the congestion window.

Recap on BDP

Satellite network: 512 kbit/s, 900 RTT

- BDP = 512×10^3 b/s × 900×10^{-3} s = 56.25 KiB

Residential DSL: 2 Mbit/s, 50 ms RTT

- BDP = 2×10^6 b/s × 50×10^{-3} s = 12.5 kB

3G (HSDPA): 6 Mbit/s, 100 ms RTT

- BDP = 6×10^6 b/s × 10^{-1} s = 75 kB

High-speed terrestrial network: 1 Gbit/s, 1 ms RTT

- BDP = 10^9 b/s × 10^{-3} s = 125 kB

International research & education network: 100 Gbit/s, 200 ms RTT

- BDP = 10^{11} b/s × 0.2 s = 2.5 GB

New Versions of TCP

- In standard TCP like TCP-Reno, TCPNewReno and TCP-SACK, TCP grows its window one per round trip time (RTT).
- This makes the data transport speed of TCP used in all major operating systems including Windows and Linux rather sluggish, extremely under-utilizing the networks especially if the length of flows is much shorter than the time TCP grows its windows to the full size of the BDP of a path.

Example

For instance, if the bandwidth of a network path is 10 Gbps and the RTT is 100 ms, with packets of 1250 bytes, the BDP of the path is around 100,000 packets.

For TCP to grow its window from the midpoint 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.

Improvements to TCP

BIC & CUBIC

Binary Increase Congestion Control (BIC TCP)

- Uses a binary search algorithm where the window grows to the mid-point between the last window size (i.e., cwnd before loss) and the new window size (i.e., cwnd set after loss).
- This "search" into the mid-point intuitively makes sense because the capacity of the current path must be somewhere between the two min and max window sizes if the network conditions do not quickly change since the last loss event
- After the window grows to the mid-point, if the network does not have packet losses, then it means that the network can handle more traffic and thus BIC-TCP sets the mid-point to be the new min and performs another "binary-search" with the min and max windows.

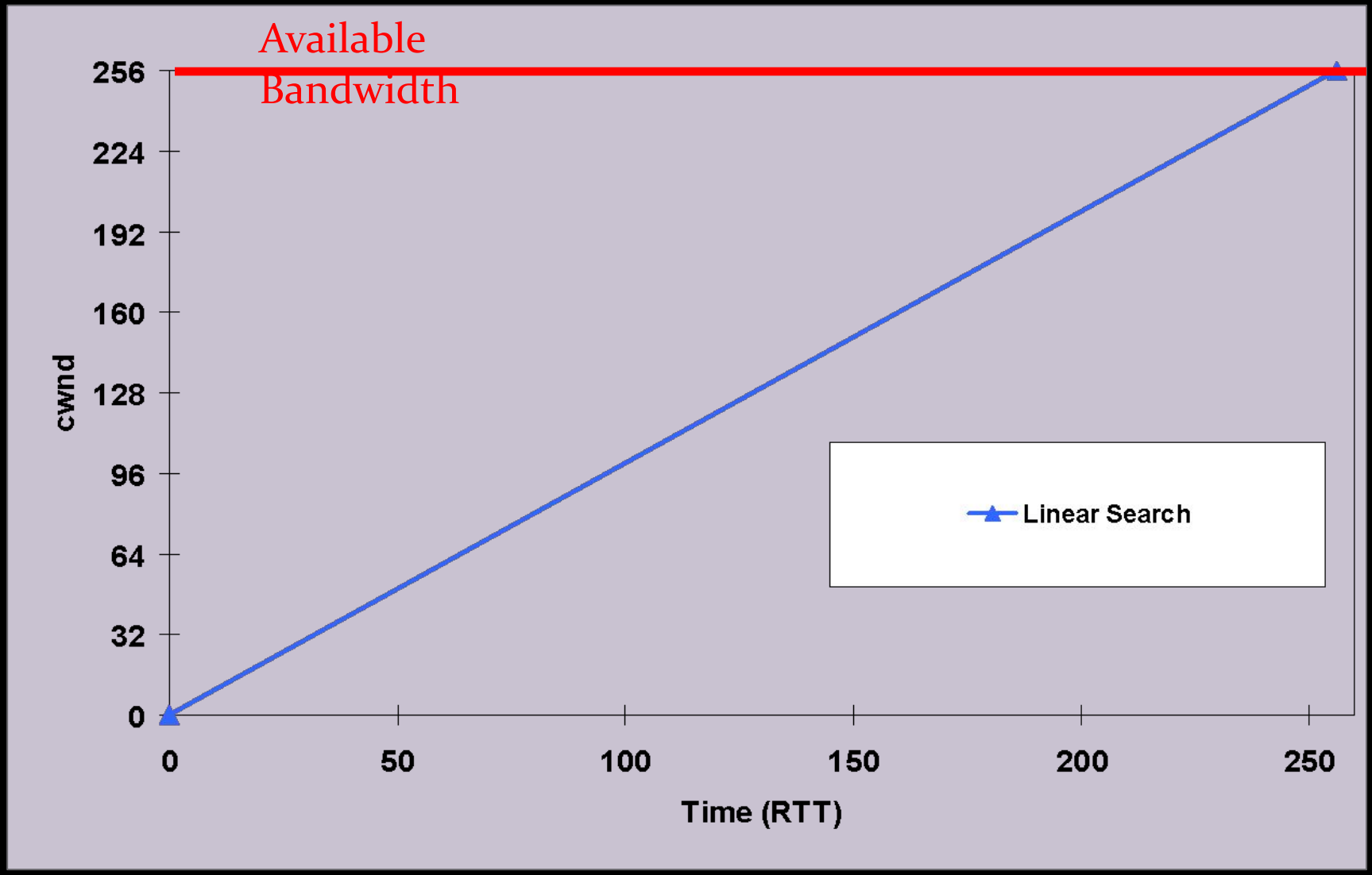
A Search Problem

- A Search Problem
  - BIC consider the increase part of congestion avoidance as a search problem, in which a connection looks for the available bandwidth by comparing its current throughput with the available bandwidth, and adjusting cwnd accordingly.

- How does TCP normally find the available bandwidth?
- Linear search

```
while (no packet loss){
    cwnd++;
}
```

Linear Search



BIC TCP

- When loss event occurs, window W is reduced by a multiplicative factor β
- The window size (cwnd) just before the loss event is given to W_{max}
- The window size after the reduction (by multiplicative factor β) is set to W_{min}
- BIC TCP performs Binary Search to find the midpoint between W_{max} and W_{min}
- The reasoning is that since the loss event occurred at W_{max} then the window size that the network can handle is somewhere in the middle of these two points

BIC TCP

Some caveats

- Jumping to the midpoint could be too much in one RTT, so if the distance between the current midpoint and W_{min} is too large, only a constant increase will be allowed
- This increase is called S_{max}
- So before binary search can kick in the midpoint jump from W_{min} needs to be smaller than S_{max}
- This region is called the linear increase region

BIC TCP

Some caveats

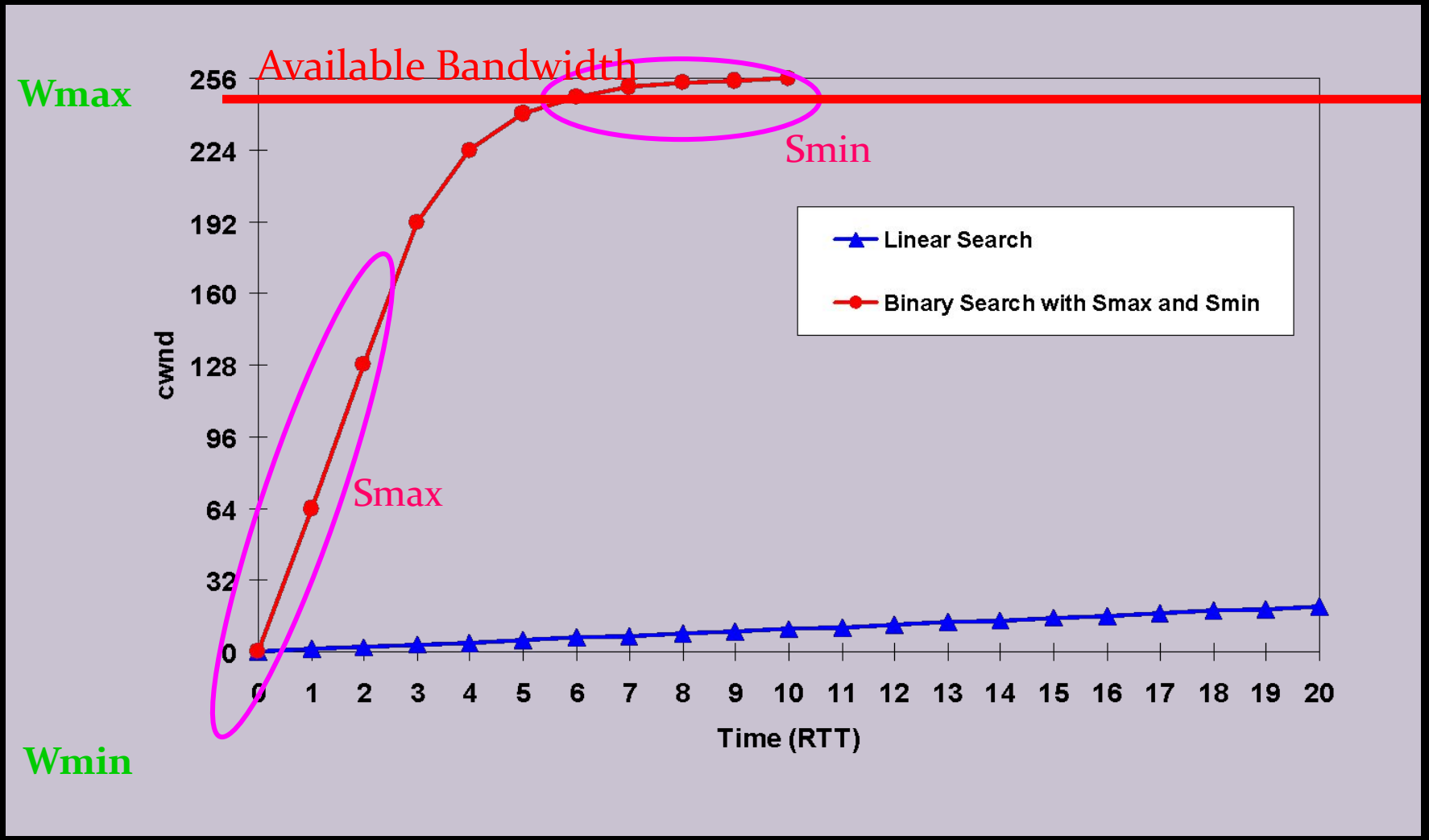
- Once the jump to midpoint distance is smaller than S_{max} we enter binary search region
- In binary search each jumps gets smaller and smaller until such a point when the increases are not meaningful (and we will never reach W_{max})
- We then stop binary search and increment the window by the smallest allowed window increase called S_{min}

BIC: Binary Search with Smax and Smin

```
while (Wmin <= Wmax){
    inc = (Wmin+Wmax)/2 - cwnd;
    if (inc > Smax)
            inc = Smax;
    else if (inc < Smin)
            inc = Smin;
    cwnd = cwnd + inc;
    if (no packet losses)
            Wmin = cwnd;
    else
            break;
}
```

- Wmax: Max Window
- Wmin: Min Window
- Smax: Max Increment
- Smin: Min Increment

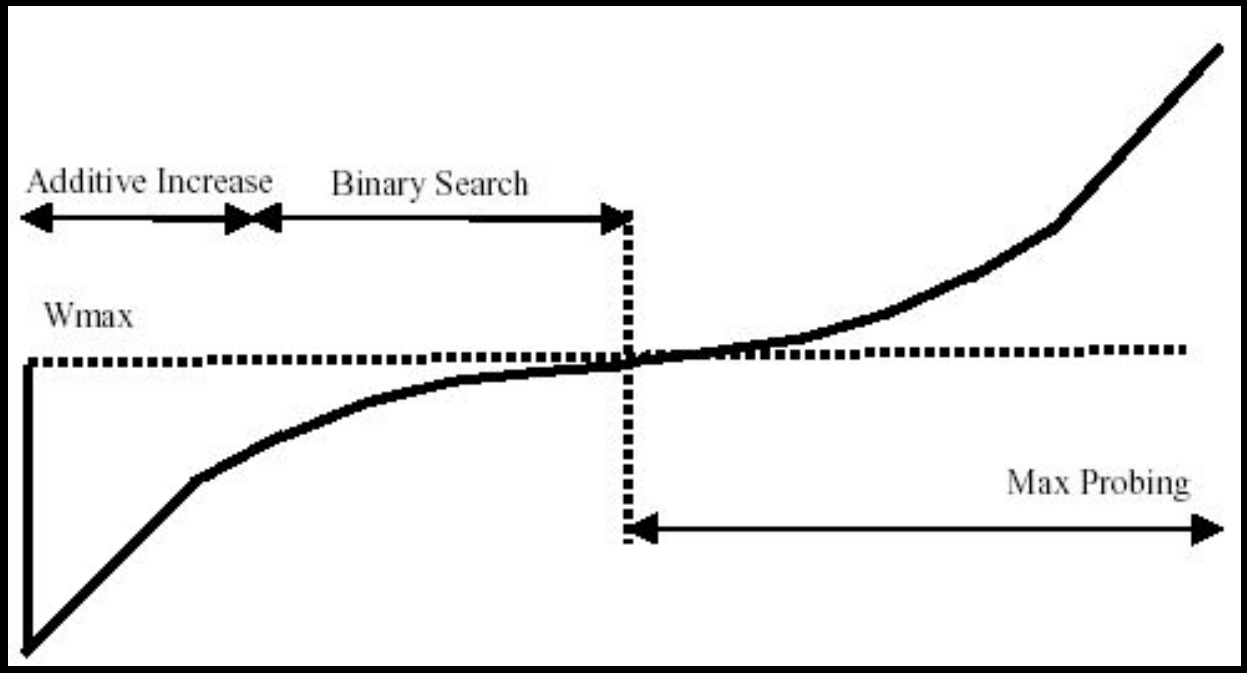
Binary Search with Smax and Smin



BIC TCP

- If we reach W_{max} with no losses it means that network can handle a new W_{max} value. Now we must find it and fast
- BIC-TCP enters a new phase called “max probing”. Max probing uses a window growth function exactly symmetric to those used in additive increase and binary search (which is logarithmic; its reciprocal will be exponential) and then additive increase.

In Summary BIC function



• BIC overall performs very well in evaluation of advanced TCP stacks on fast long-distance production.

In Summary BIC function

- BIC (also HSTCP & STCP) growth function can be still aggressive for TCP especially under short RTTs or low speed networks.
- There are several phases (binary increase, max probing, S_{min}/S_{max} bounded growths) make it very complicated to implement in OS kernel
- TCP Cubic tries to solve some of these issues

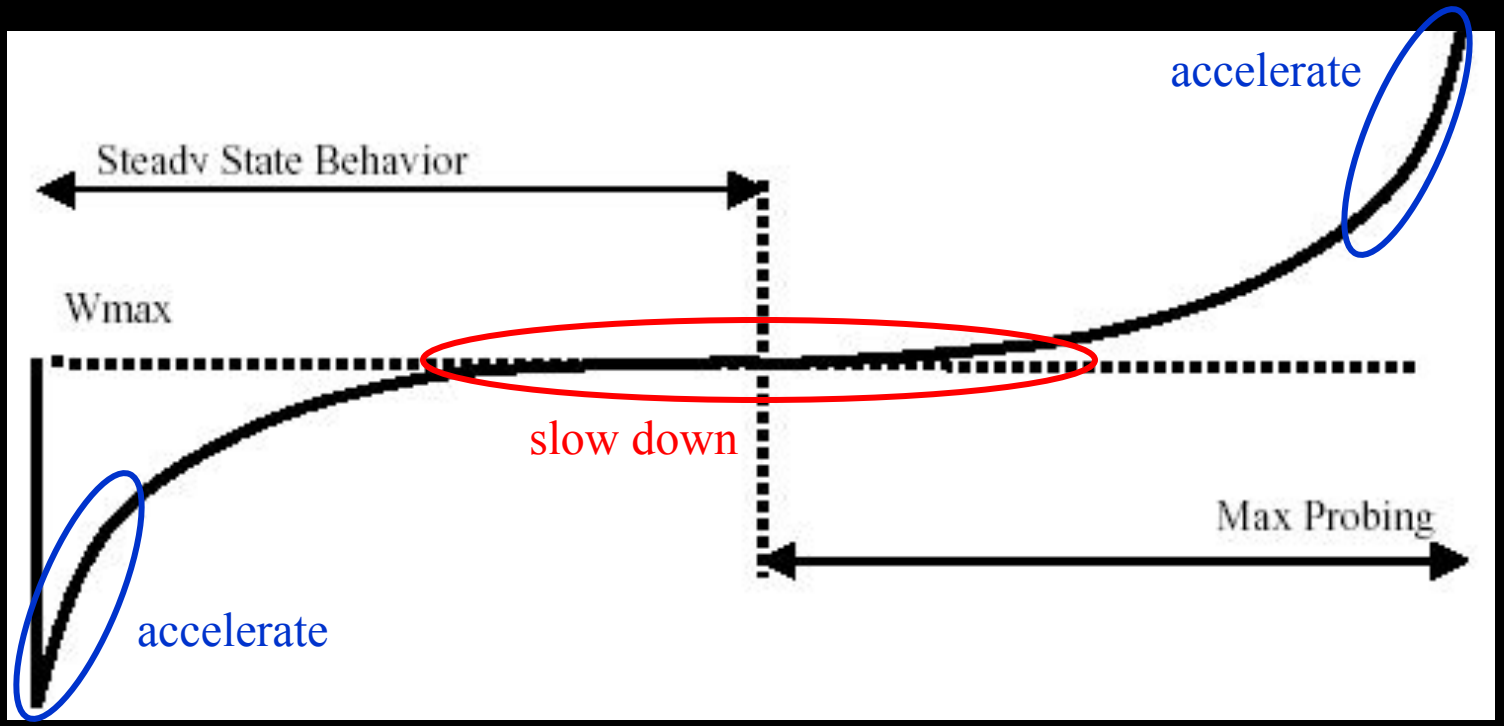
TCP CUBIC

- Growth function very similar to BIC TCP
- Uses a cubic function of the elapsed real-time since the last loss event.
- Has a concave and convex profile
  - a cubic function is chosen out of simplicity
  - any odd order function would achieve the same purpose

TCP CUBIC

- After a loss event W_{max} set to be the window size when loss event happened and window W (cwnd) is reduced by a multiplicative factor β
- cwnd starts to increase using the concave profile of the cubic function. The cubic function is set to plateau at W_{max}
- After the cwnd reaches W_{max} the growth enter the convex profile and convex growth begins
- cwnd remains constant around W_{max} forming a plateau and average W for a TCP CUBIC connection is W_{max}.

The CUBIC function



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

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

where ***C*** is a scaling factor, ***t*** is the elapsed time from the last window reduction, and ***β*** is a constant multiplication decrease factor

CUBIC Algorithm

- If (received ACK && state == cong avoid)
  - Compute $W_{cubic}(t+RTT)$.
  - If cwnd < W_{TCP}
    - CUBIC in TCP mode
  - If (cwnd > W_{TCP}) && (cwnd < W_{max})
    - CUBIC in concave region
  - If cwnd > W_{max}
    - CUBIC in convex region

TCP Mode

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

- Detailed derivation of this equation is in Section 3.3 in CUBIC paper

Concave Region

$$cwnd_{new} = cwnd + \frac{W_{cubic}(t + RTT) - cwnd}{cwnd}$$

- Detailed derivation of this equation is in Section 3.4 in CUBIC paper

Convex Region

- cwnd > Wmax
- New bandwidth might be available
- **Use the same window growth function.**

$$cwnd_{new} = cwnd + \frac{W_{cubic}(t + RTT) - cwnd}{cwnd}$$

- Detailed derivation of this equation is in Section 3.5 in CUBIC paper

Can we do better?

One last TCP version … BBR TCP

Let's rethink congestion

Why do we consider loss of a packet as congestion?

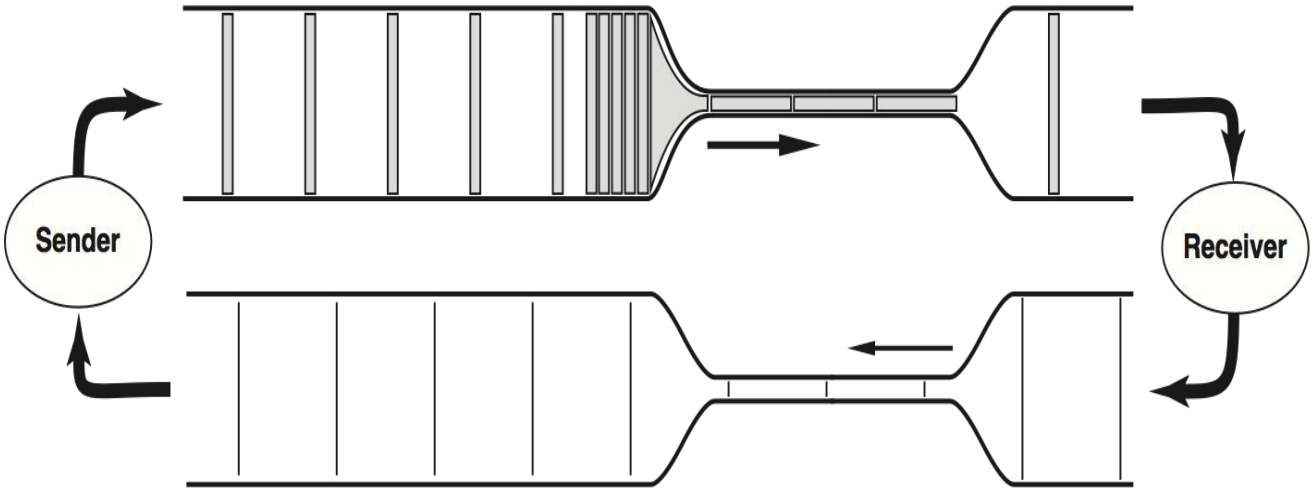
- In the 1980 when networks were slow, router buffers small, it made more sense
- Today speeds are typically in hundreds of Gbps and router buffers in GBs
- Even TCP CUBIC uses this assumption

Let's rethink congestion

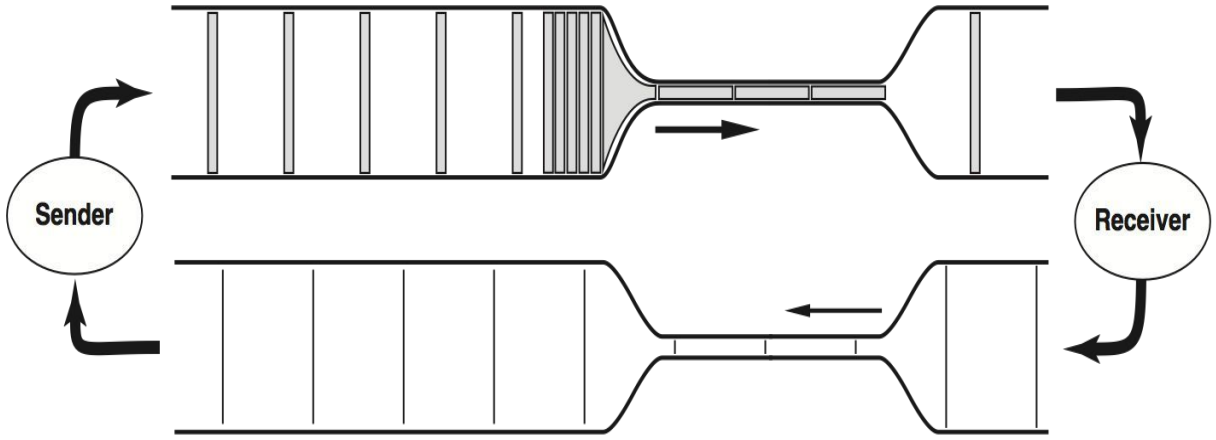
Why do we consider loss of a packet as congestion?

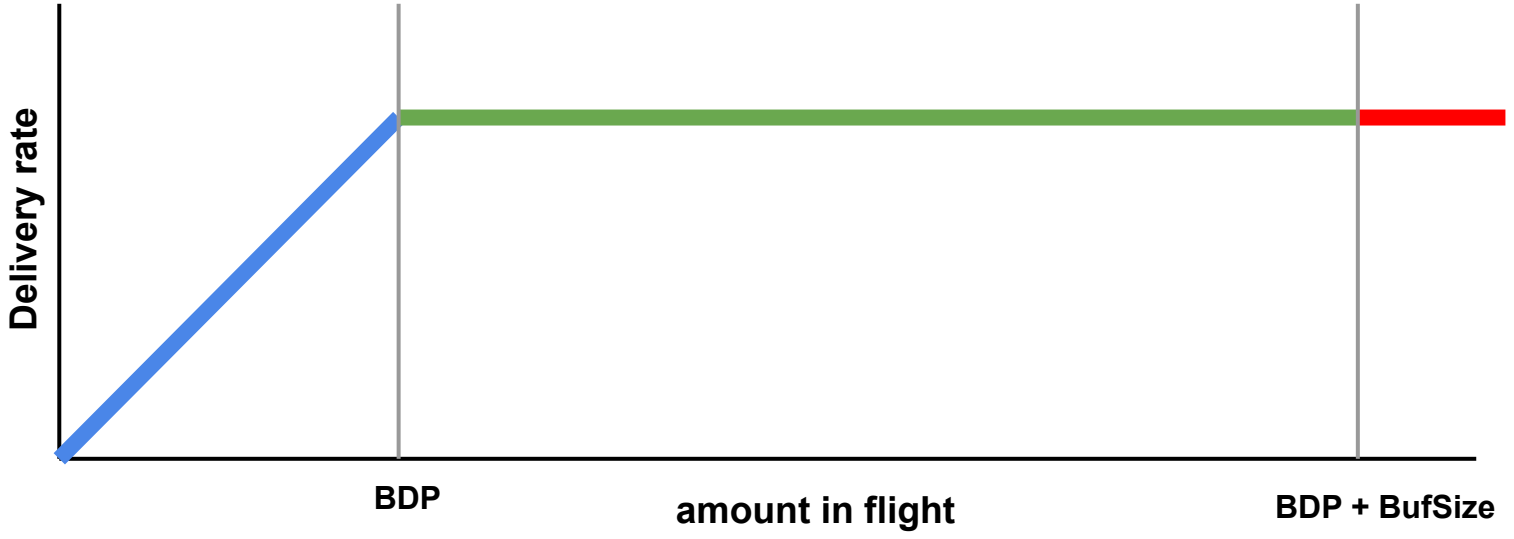
- When bottleneck buffers are large, RTTs tend to increase as packets spend longer times waiting for service (Bufferbloat)
- When bottleneck buffers are small, links tend to be under utilised
- We need an alternative to loss based congestion control

Let’s rethink congestion



Let's rethink congestion

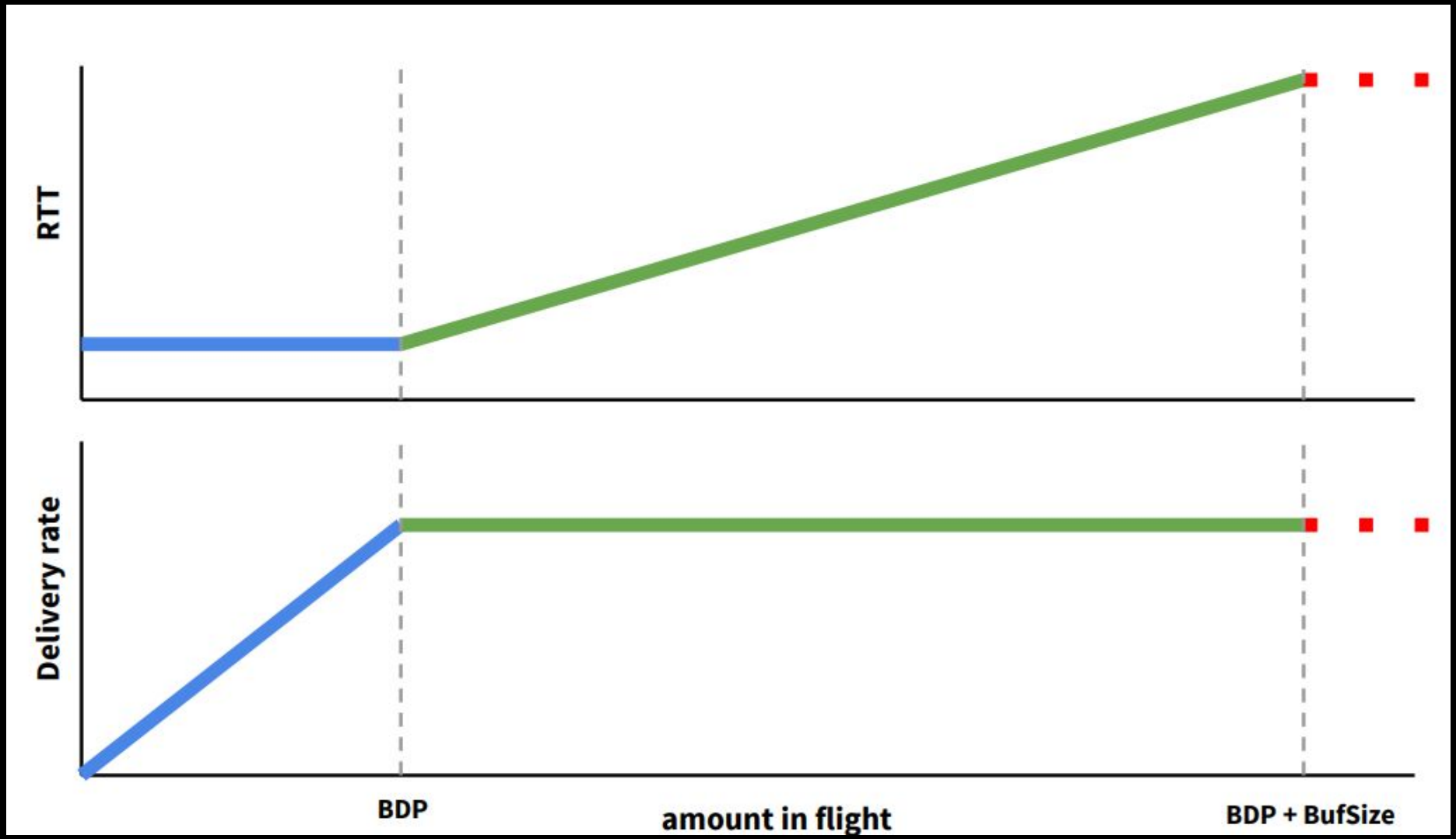




Let's rethink congestion

- Green line represents the bottleneck bandwidth (BW) (the slowest link speed of the TCP connection) and essentially the delivery rate
- The blue line represents the round trip propagation delay (RTT)
- BDP is the product of BW and RTT
- Once the sender starts sending above BDP, the excess will end up in the bottleneck buffer
- If sender sends above BDP + buffersize, packets start getting dropped

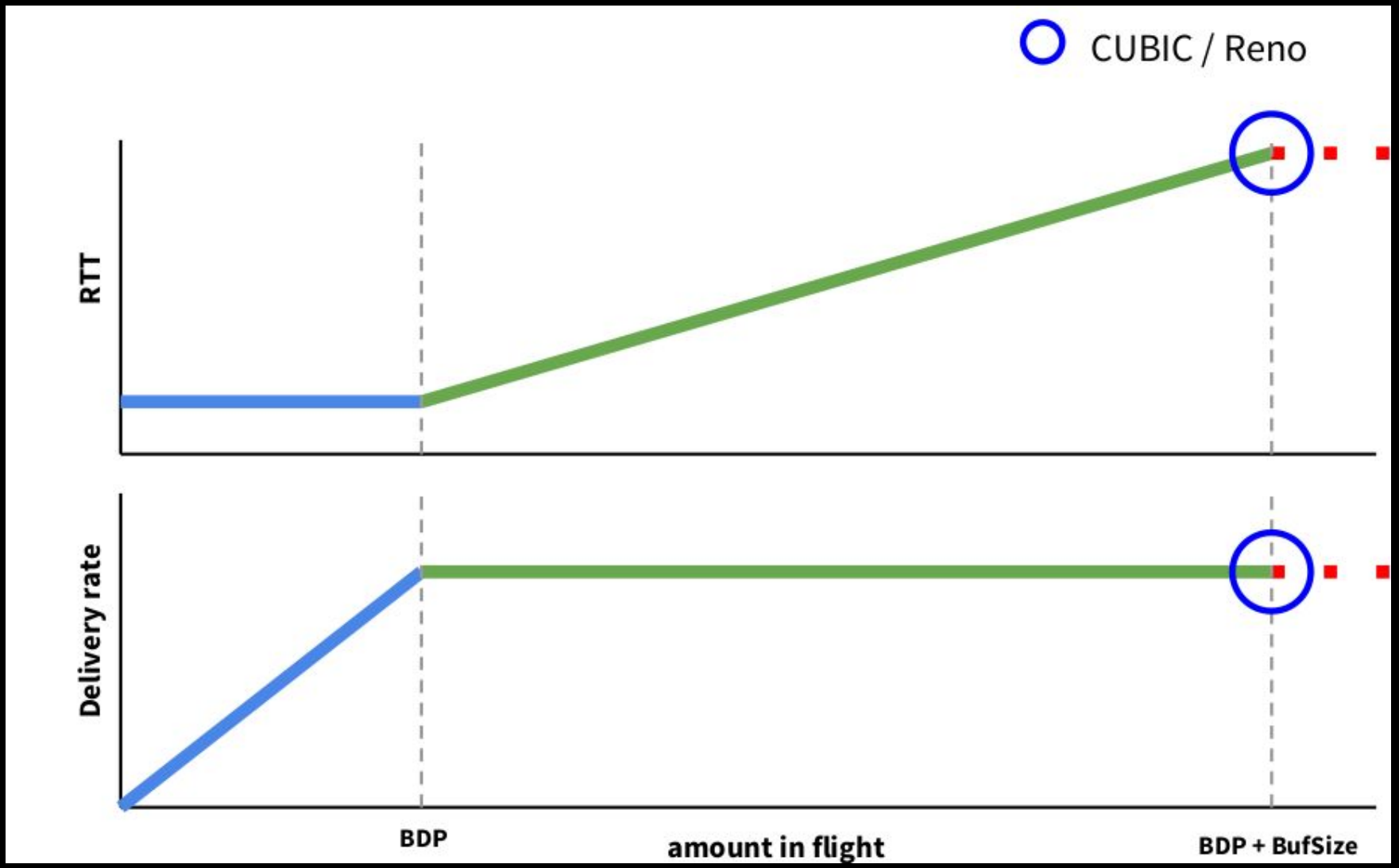
Let’s rethink congestion



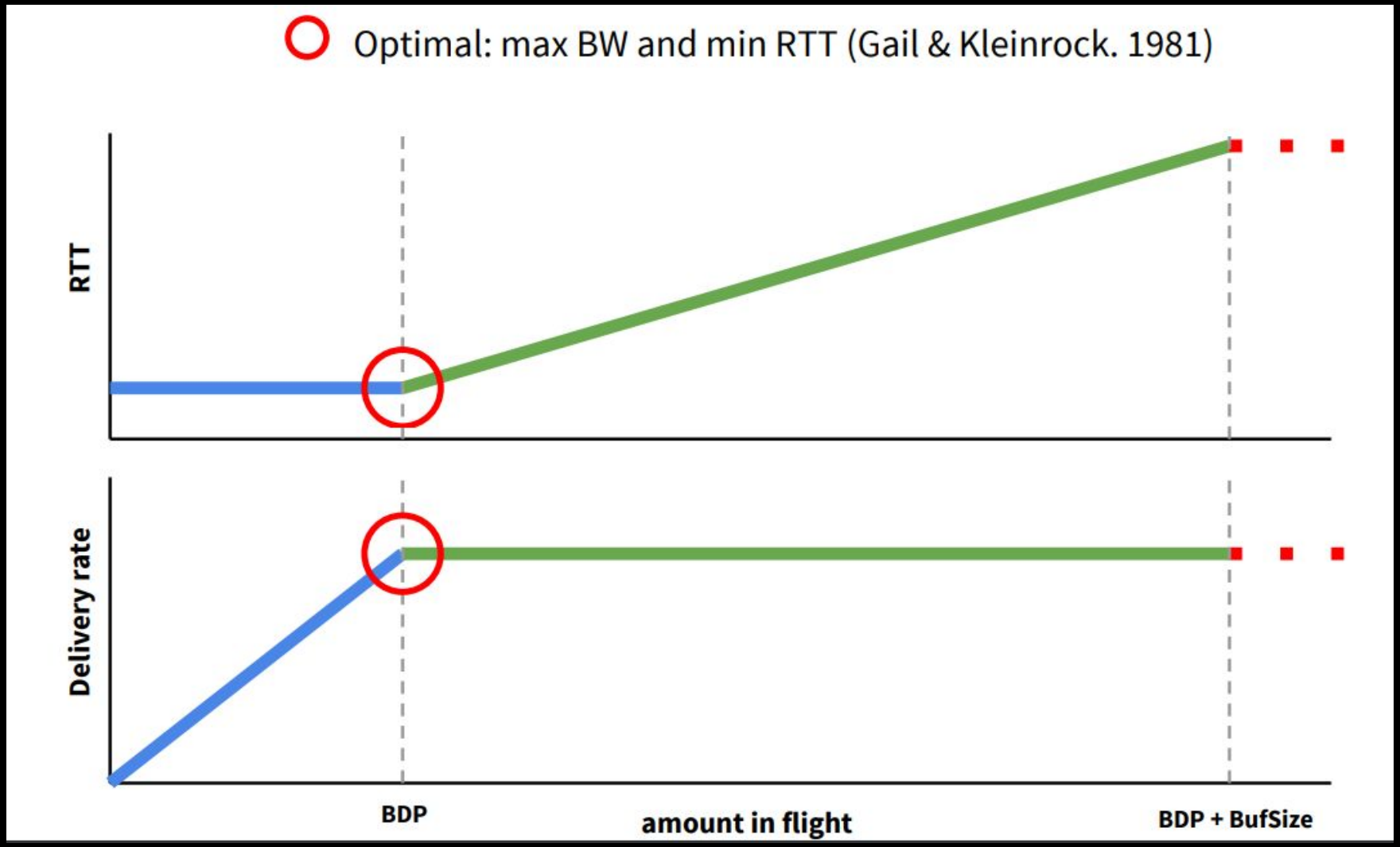
Let's rethink congestion

- To the right of the BDP edge is loss based congestion
- This is where all traditional TCP congestion control is operating
- Typically with very large buffers, RTTs will be very large
- At the same time TCP will be over utilizing router buffers

Let’s rethink congestion



Let’s rethink congestion



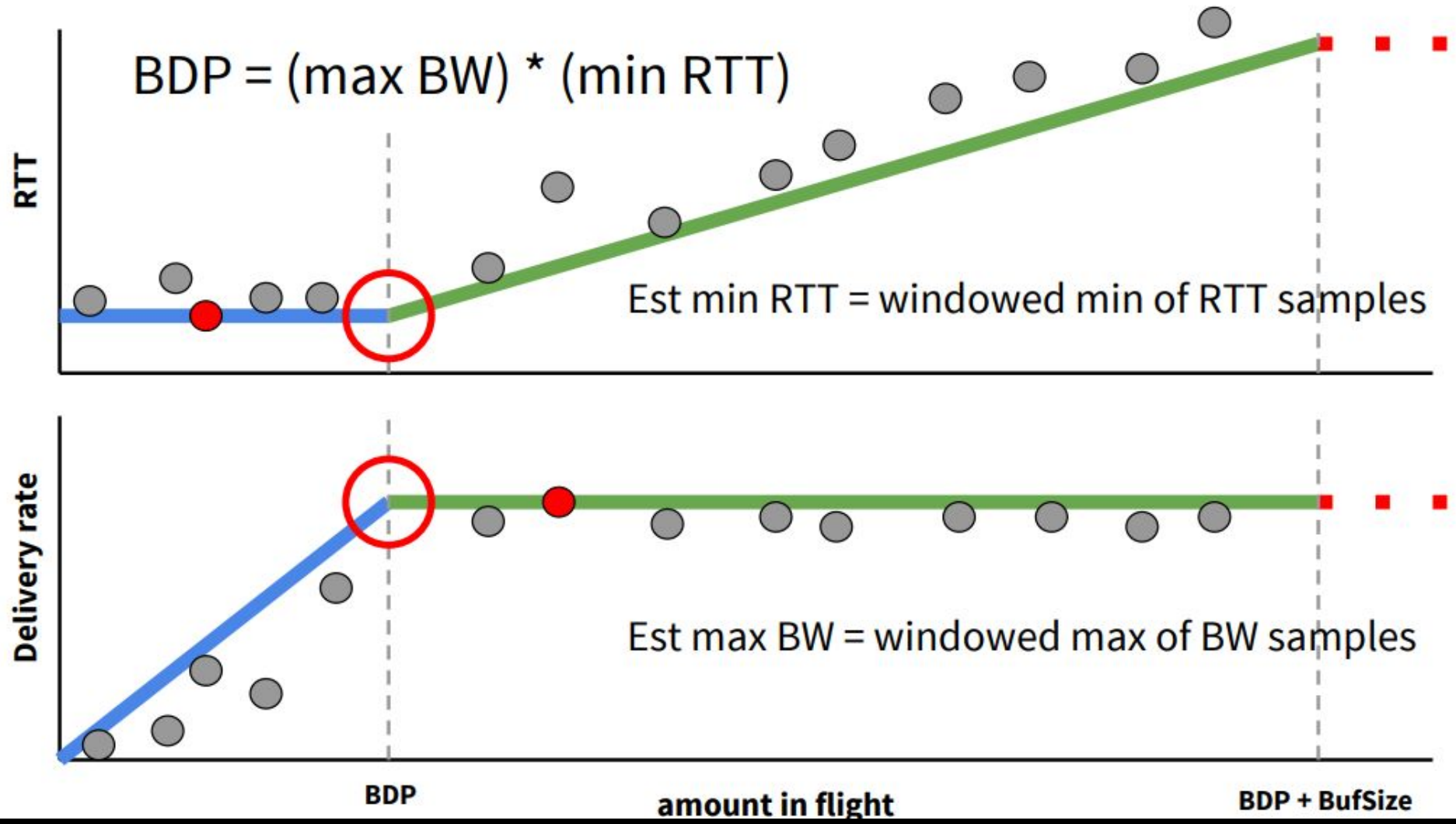
Let’s rethink congestion

- Lets operate close to the BDP line!
- We can maximize delivery rate while minimizing delay (as proven by Kleinrock)
- But! it's impossible to create a distributed algorithm that converges at this point
- Unless there is a way to measure the bottleneck bandwidth and round trip propagation

BBR TCP

- Combining these measurements with a robust servo loop using recent control systems advances could result in a distributed congestion-control protocol that reacts to actual congestion, not packet loss or transient queue delay, and converges with high probability to Kleinrock's optimal operating point.
- Hence a congestion control TCP based on measuring the two parameters that characterize a path: bottleneck bandwidth and round-trip propagation time, or BBR was developed

BBR TCP


BDP = (max BW) * (min RTT)
RTT
Est min RTT = windowed min of RTT samples
Delivery rate
Est max BW = windowed max of BW samples
BDP
amount in flight
BDP + BufSize

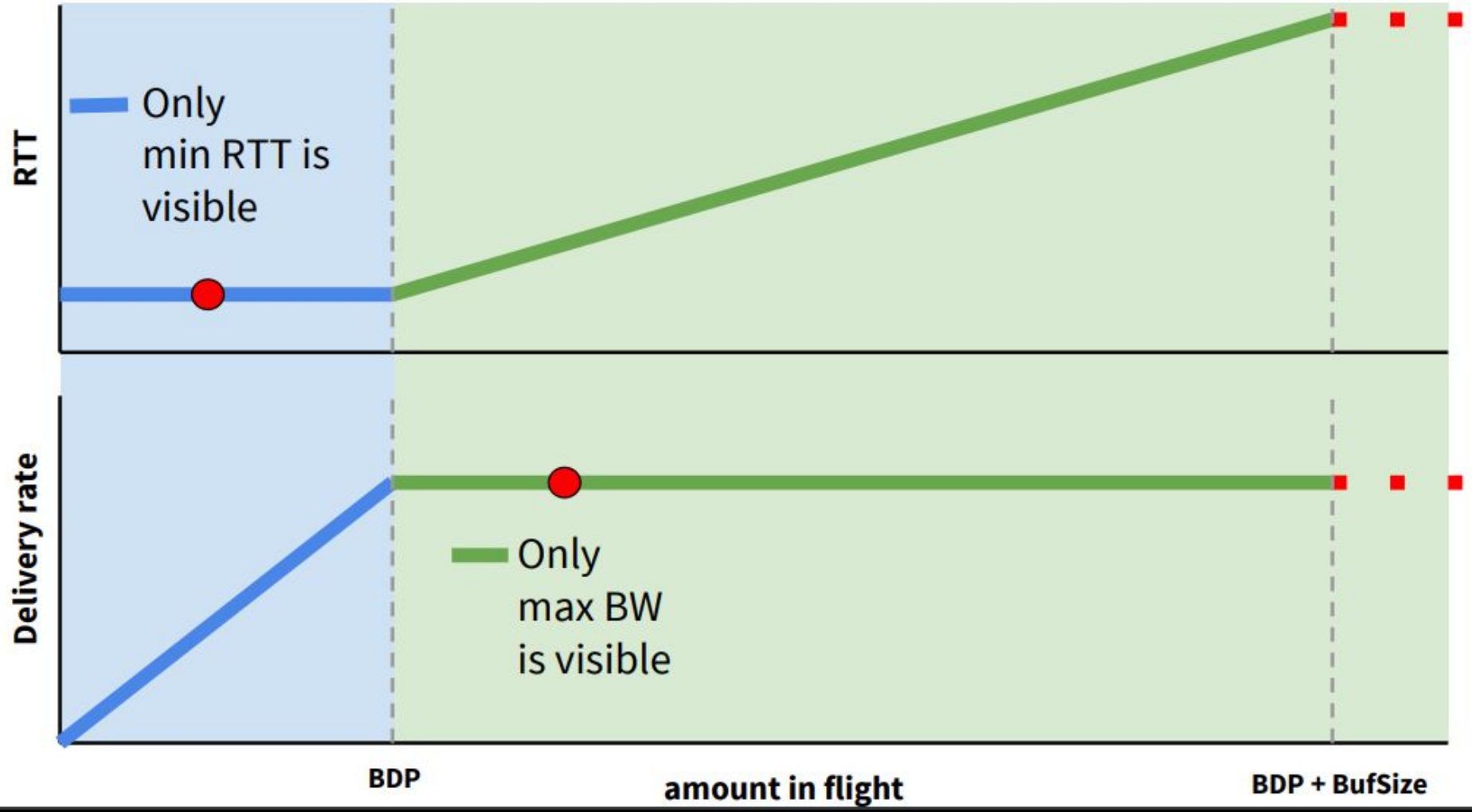
BBR TCP

- We can estimate min RTT
  - this will typically be the measured RTT samples at the beginning of the TCP connection as they represent no packets in buffers (yet)
  - larger RTT will be affected by time packets spend in buffers
- We can estimate max BW rate
  - this will typically be the send/delivery rate after BDP is reached, rate will stay constant
  - smaller BW rates will typically represent rates when connection was app limited or “pipe not full”

BBR TCP

- We notice an uncertainty problem
- min RTT only knowable before rate is larger than BDP
- max BW only knowable after rate is larger than BDP
- We cannot measure both at the same time

BBR TCP

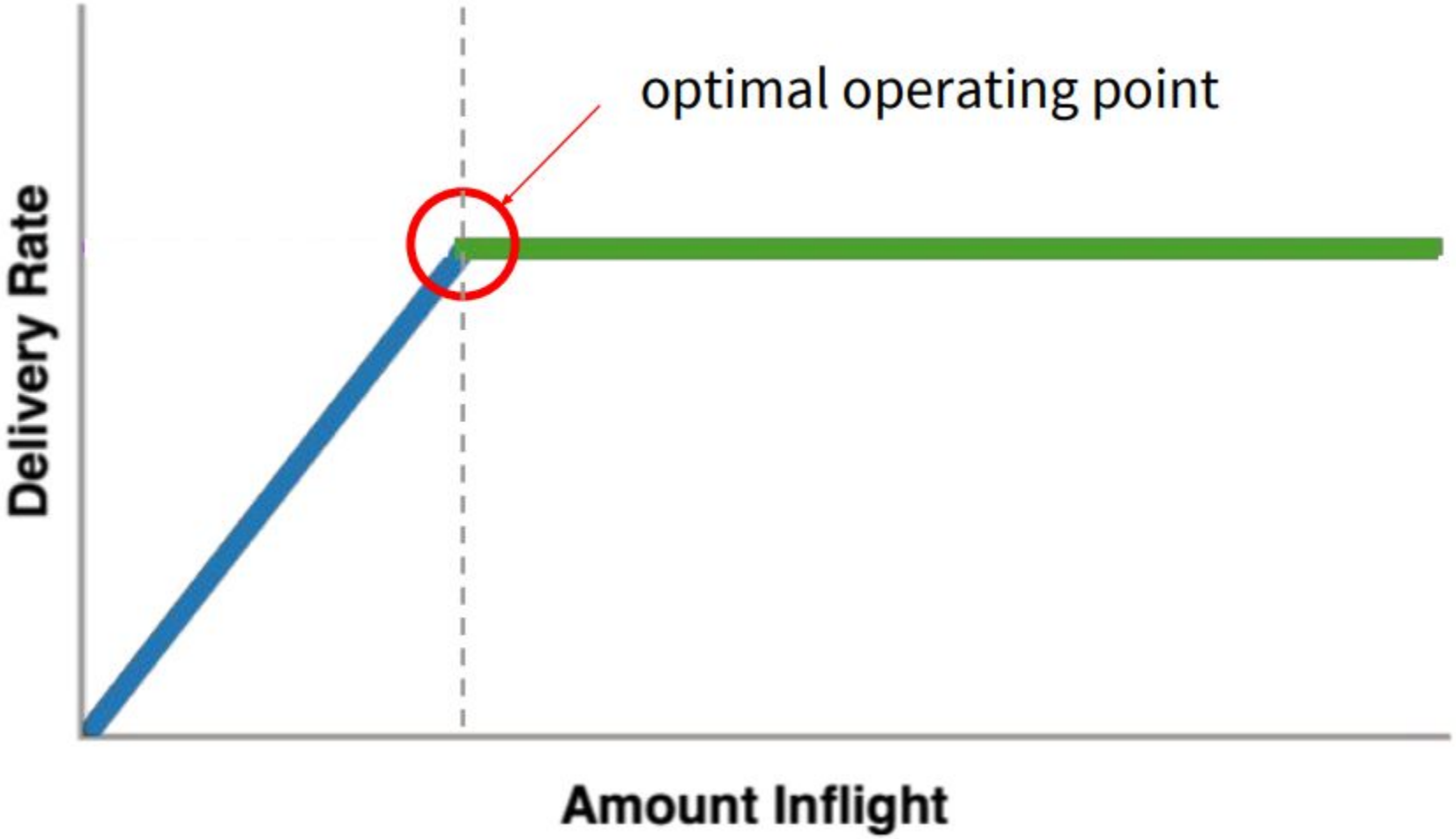

RTT
Only
min RTT is
visible
Delivery rate
Only
max BW
is visible
BDP
amount in flight
BDP + BufSize

BBR TCP

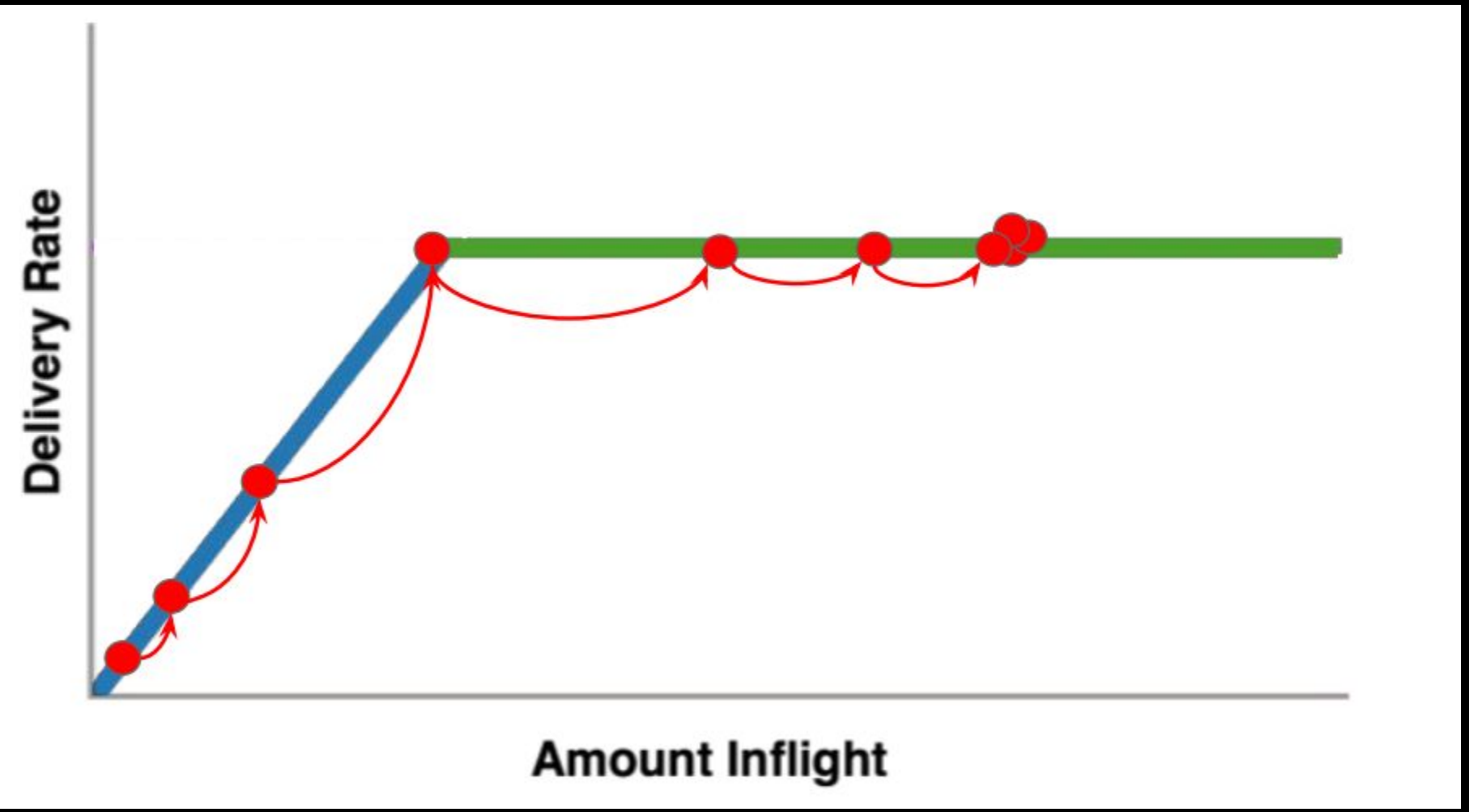
Stay near (max BW, min RTT) point:

- **Model network**, update max BW and min RTT estimates on each ACK
- **Control sending based on the model**, to…
  - Probe both max BW and min RTT, to feed the model samples
  - Pace near estimated BW, to reduce queues and loss
  - Vary pacing rate to keep inflight near BDP (for full pipe but small queue))
- BBR = Bottleneck Bandwidth and Round-trip propagation time

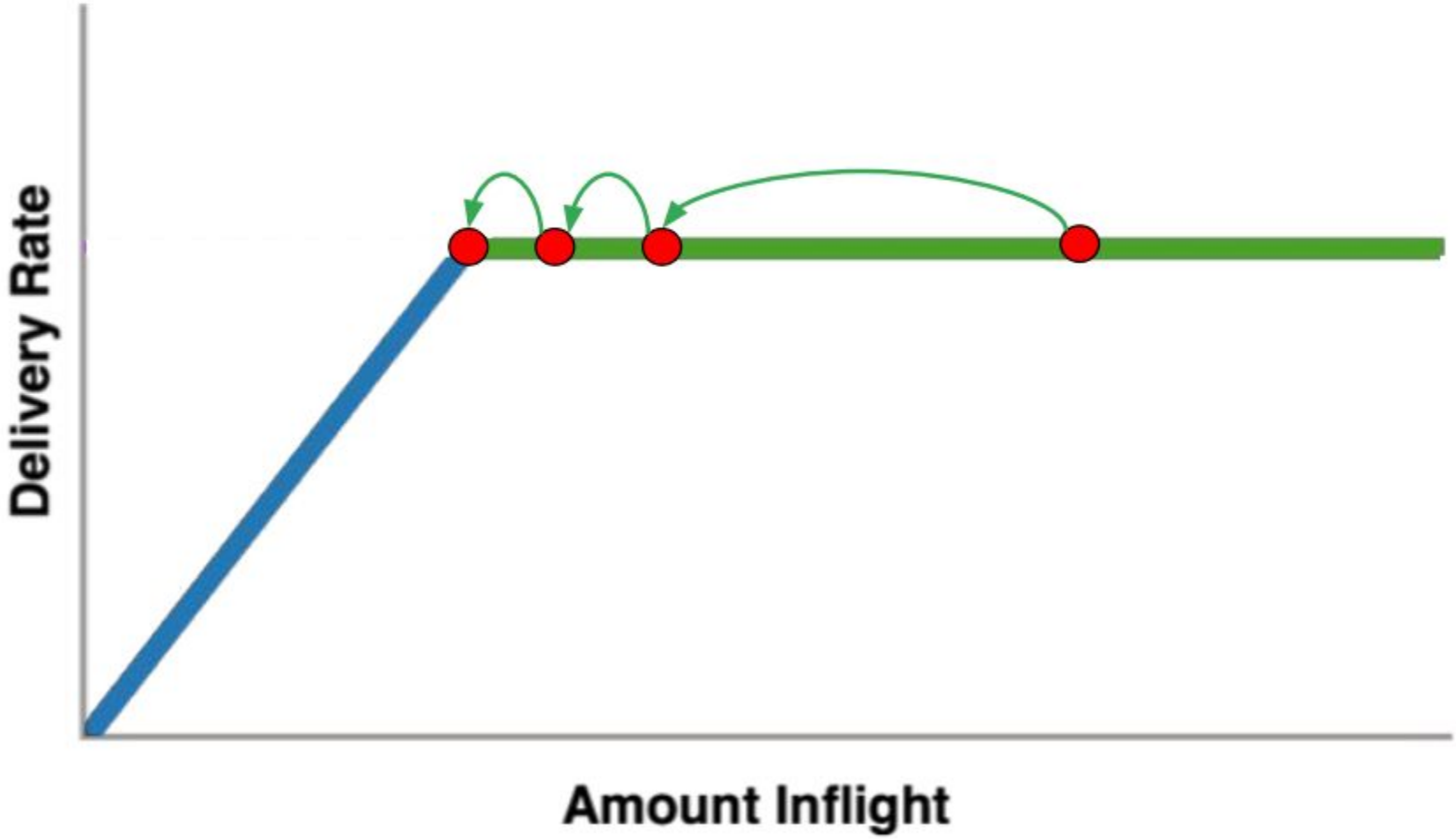
BBR: model-based walk toward max BW, min RTT



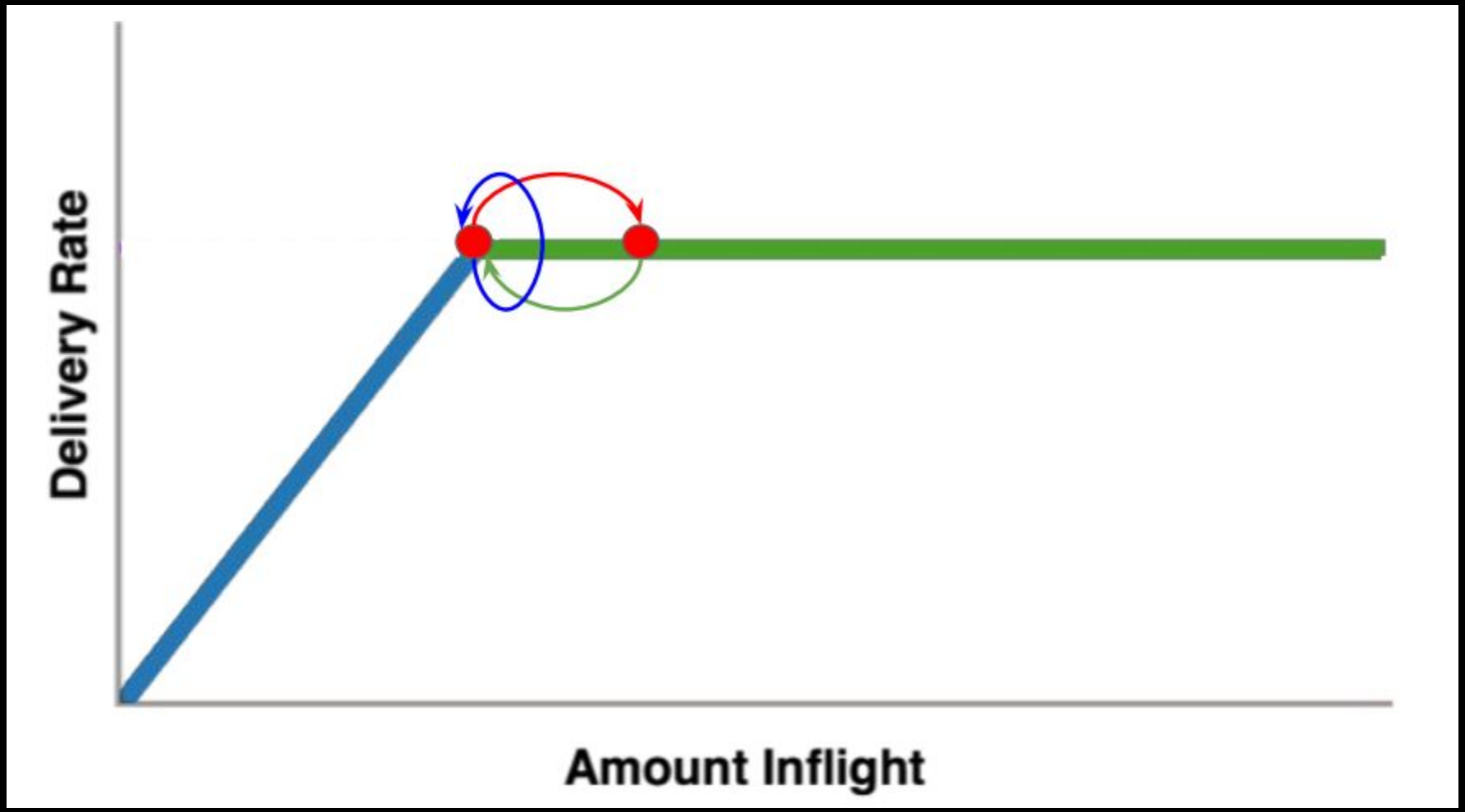
STARTUP: exponential BW search



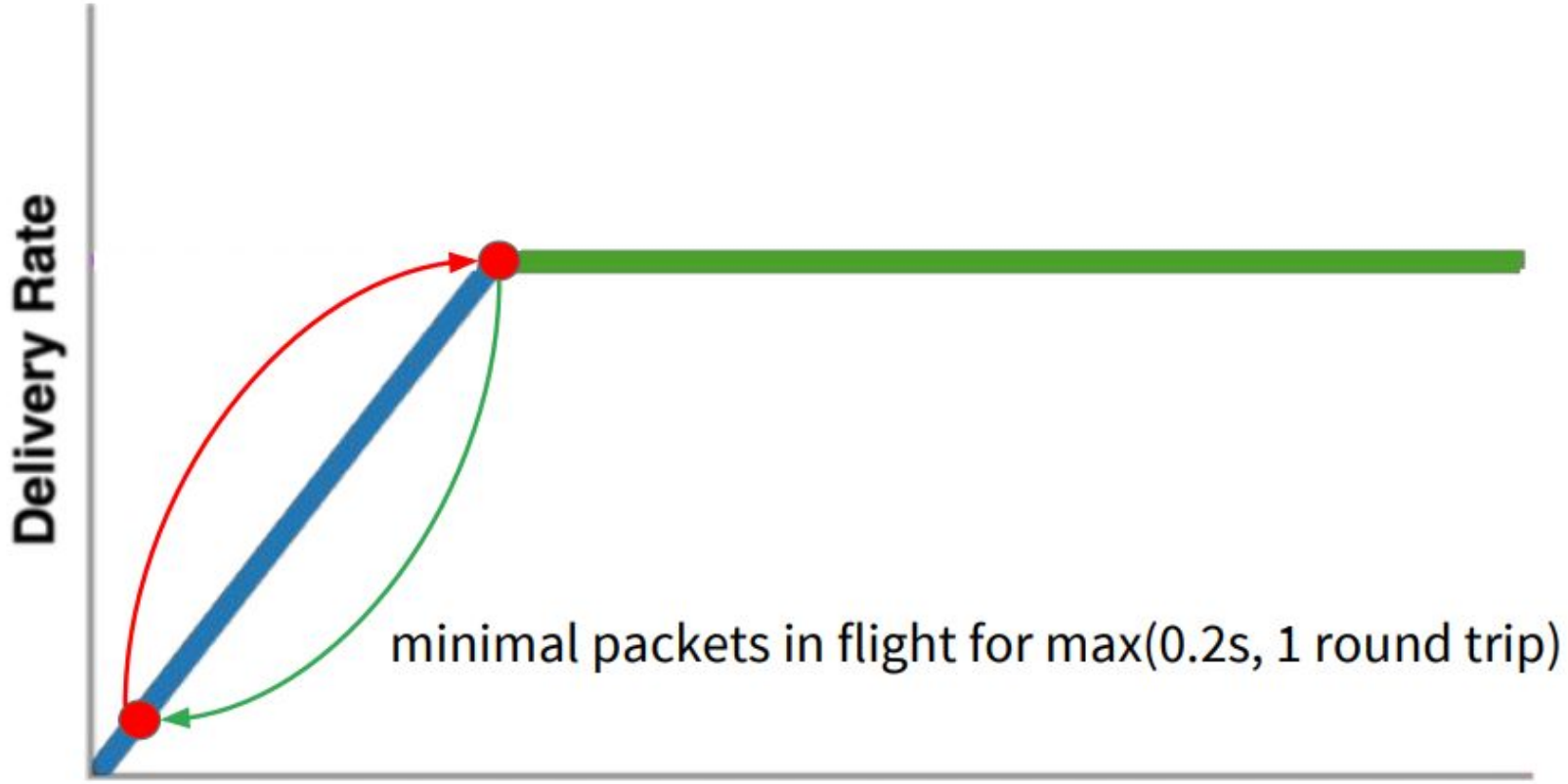
DRAIN: drain the queue created during startup



PROBE_BW: explore max BW, drain queue, cruise



PROBE_RTT briefly if min RTT filter expires (=10s)*



[*] if continuously sending

BBR TCP Algorithm

```
function onAck(packet)
  rtt = now - packet.sendtime
  update_min_filter(RTpropFilter, rtt)
  delivered += packet.size
  delivered_time = now
  deliveryRate = (delivered - packet.delivered)
                 /(now - packet.delivered_time)
  if (deliveryRate > BtlBwFilter.currentMax
      || ! packet.app_limited)
      update_max_filter(BtlBwFilter,
                        deliveryRate)
  if (app_limited_until > 0)
      app_limited_until - = packet.size
```

BBR TCP Summary

- BBR only needs to be implemented in one side of the TCP connection for it to work
- BBR achieved TCP fairness when all the flows are BBR
- BBR is slightly starved out of resources when sharing links with other TCP like Cubic/Reno (this is still being researched)
- You can change your TCP variant yourself

Interesting BBR Video

https://www.youtube.com/watch?v=4CIcNyWnxqQ

Conclusions

- Homework: read the Cubic and BBR papers
- Next week we will cover Routing and Network management

References

1. Computer Networking - A top down approach
2. Computer Networks - A systems approach
3. CUBIC: a new TCP-friendly high-speed TCP variant
   https://dl.acm.org/citation.cfm?id=1400105
4. BBR: Congestion-Based Congestion Control
   https://queue.acm.org/detail.cfm?id=3022184