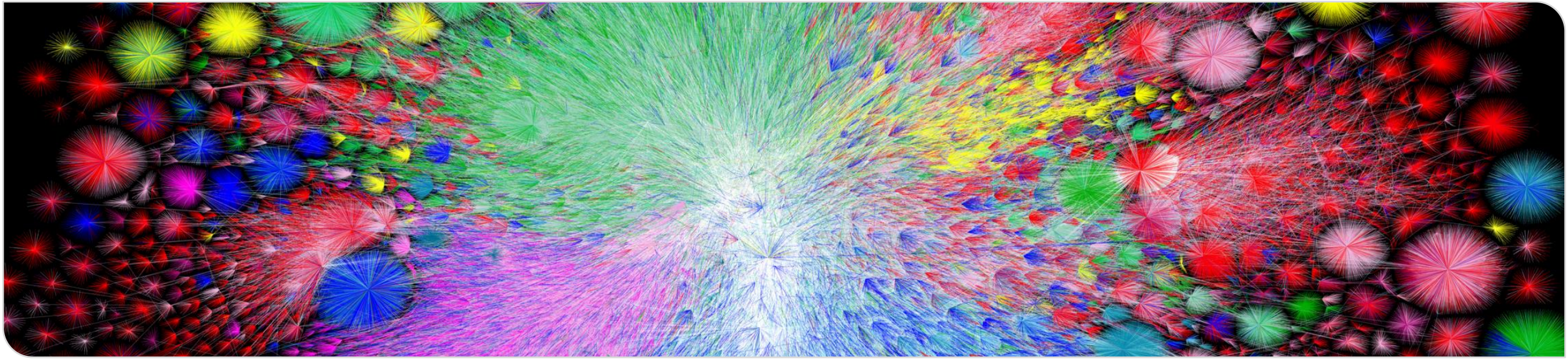
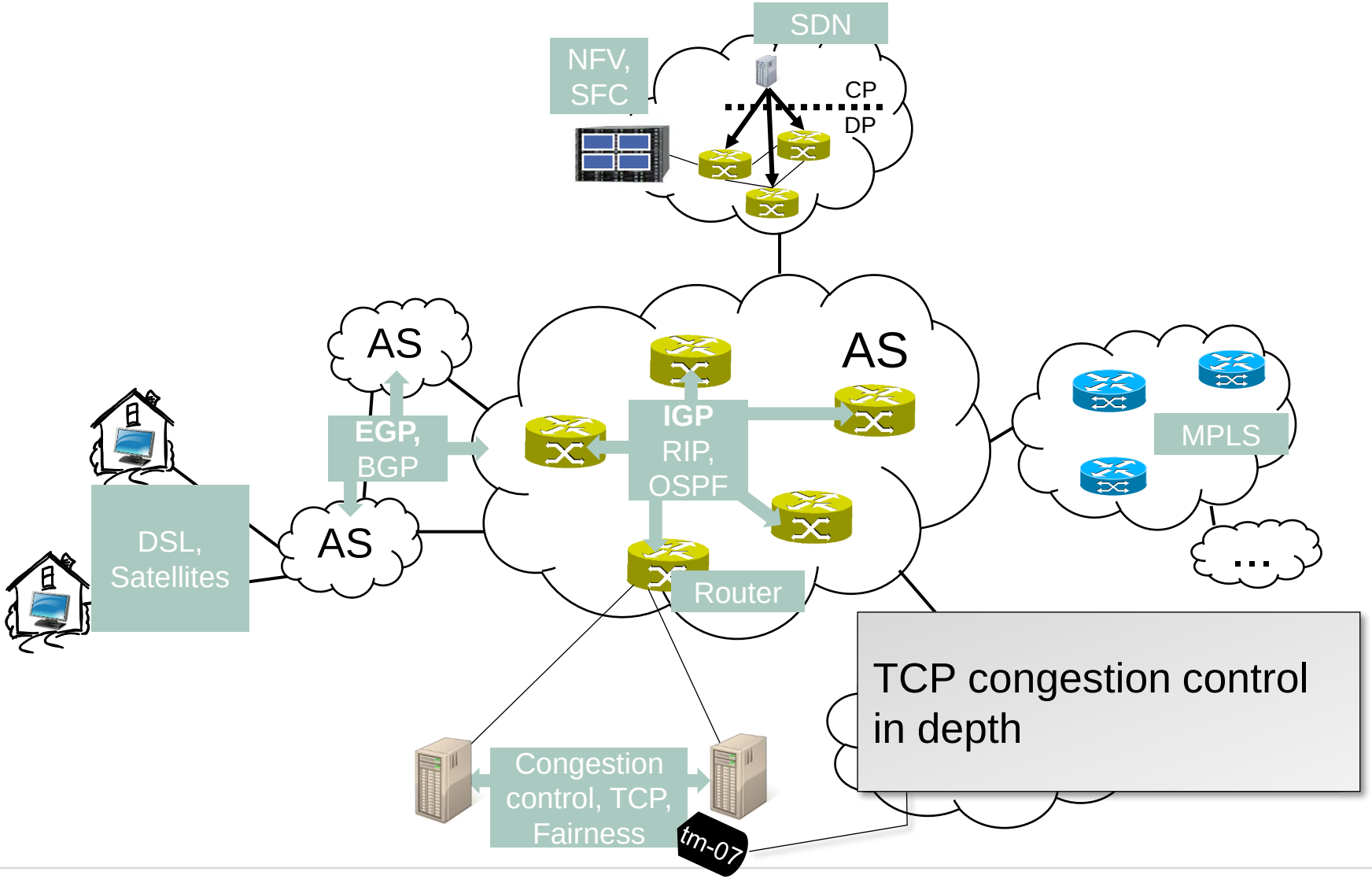
Karlsruher Institut für Technologie

7. Internet Congestion Control

Prof. Dr. Martina Zitterbart
Institute of Telematics




SDN
NFV, SFC
CP
DP
AS
AS
AS
EGP, BGP
IGP
RIP, OSPF
MPLS
...
DSL, Satellites
Router
Congestion control, TCP, Fairness
tm-07
TCP congestion control in depth

7 Internet Congestion Control

7.1 Basics

7.2 TCP Tahoe

7.3 Self Clocking

7.4 Conservation of Packets

7.5 TCP Reno

7.6 Fairness

7.7 Periodic Model and „TCP-Formula“

7.8 Active Queue Management

7.1 Basics

7.1.1 Shared (Network) Resources

Shared (Network) Resources & Congestion Control

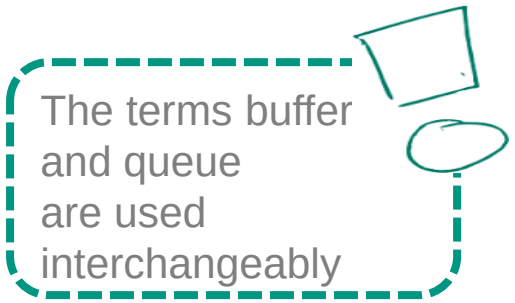
- General problem
  - Multiple users use same resource
  - E.g., multiple video streams use same network link
- High level objective with respect to networks
  - Provide good utilization of network resources
  - Provide acceptable performance for users
  - Provide fairness among users / data streams
- Mechanisms that deal with shared resources
  - Scheduling
  - Medium access control
  - Congestion control
  - …

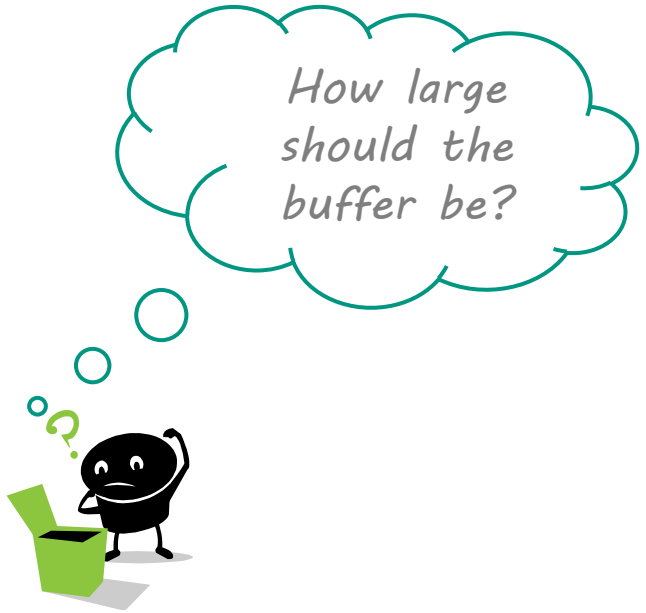
- Congestion Control
  - Shared resource: network link
    - Buffer associated with that link
  - Adjusts load introduced to shared resource in order to avoid overload situations
    - Utilizes feedback information
      - Implicit or explicit feedback possible
      - Feedback from receiver and/or network

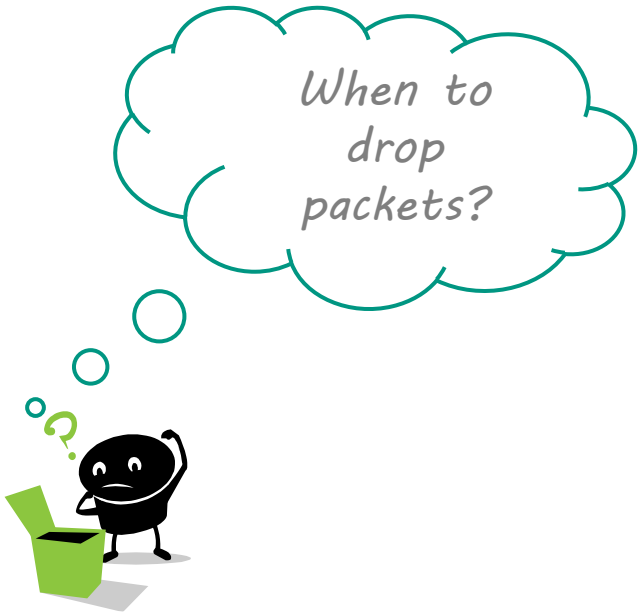
Buffer

- Routers need buffers (queues) to cope with temporary traffic bursts

- Packets that can not be transmitted immediately are placed in the buffer
- If buffer is filled up / fills up, packets need to be dropped





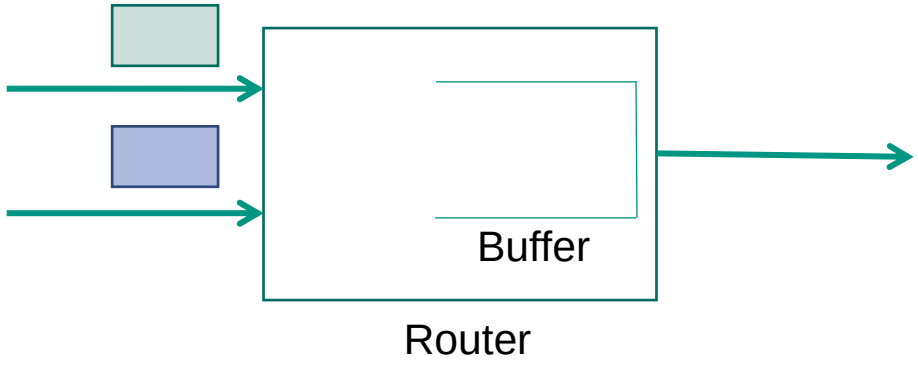


“Critical” Situations

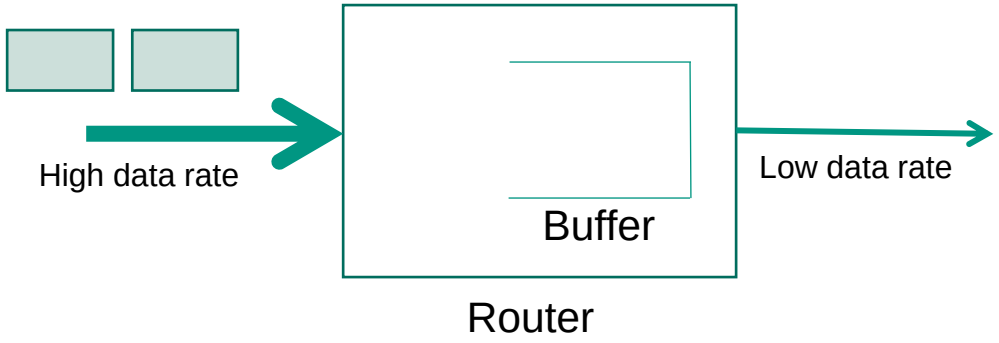
- Example 1
  - Router concurrently receives two packets from different input interfaces which are directed towards same output interface

    → only one of these packets can be sent at a time

  - What to do with the other packet?
    - Buffer or
    - Drop

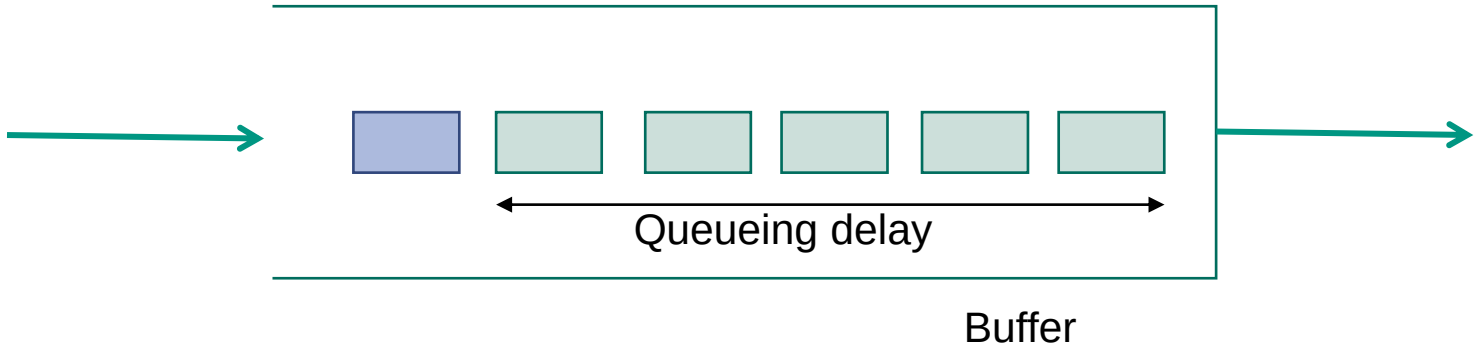


- Example 2
  - Router interfaces with different data rates
    - Input interface has high data rate
    - Output interface has low data rate
  - Two successive packets of a same or different senders arrive at input interface
  - What to do with the second packet? The output interface is still busy sending the first packet while the second arrives.
    - Buffer or
    - Drop



Buffer

- Buffers add latency
  - Typically implemented as FIFO queues
  - Router can only start sending a queued packet after all packets in front of it have been sent
    - Example
      - Five green packets introduce queueing delay for blue packet

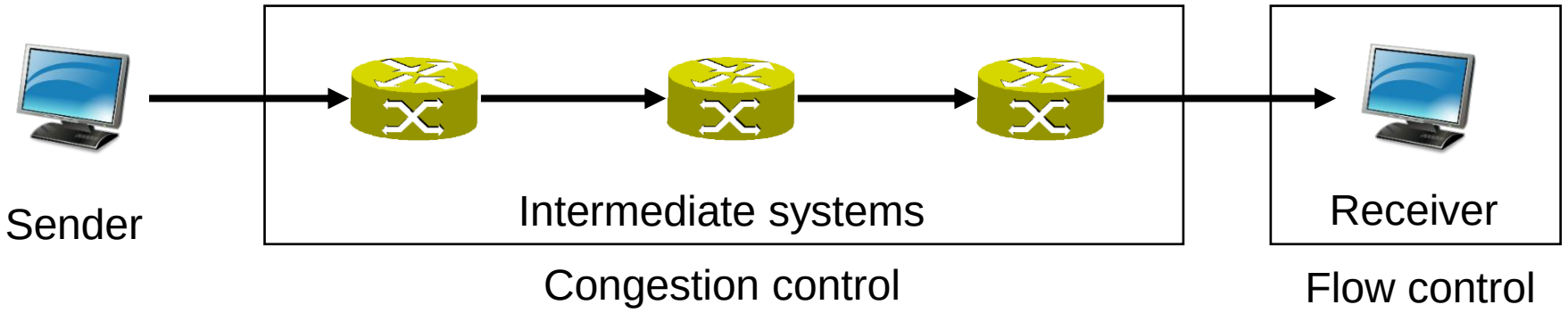


  - End-to-end latency of a packet includes
    - Propagation delay,
    - Transmission delay, and
    - Queueing delay

General Problem

- Sender wants to send data through the network to the receiver
- On every network path, the link with the lowest available data rate limits the maximum data rate that can be achieved end-to-end
  - This link is called bottleneck link
  - The maximum data rate of a link is called link capacity
- Problem: sender can send more data than bottleneck link can handle
  - Sender can overload bottleneck link
  - Sender has to adjust its sending rate
  - Question: How to find the “optimal” sending rate?

Congestion Control vs. Flow Control



- Flow control
  - Bottleneck is located at receiver side
  - Receiver can not cope with desired data rate of sender
  - Discussed in “Einführung in Rechnernetze” EiR
- Congestion control
  - Bottleneck is located in the network
  - Bottleneck link does not provide sufficient available data rate
    - Leads to congested router / intermediate system
  - Main topic of this lecture

7.1.2 Congestion Collapse

Congested Internet

- Observation
  - Drastic performance reduction of TCP (October 1986)
    - Series of congestion collapses
  - Goodput reduced about several orders of magnitude
    - Example: connection between Lawrence Berkeley Laboratory and UC Berkeley
      - Distance between locations: 370 Meter, 2 routers
      - Goodput dropped from 32 kbit/s to 40 bit/s
        - Reduction factor: about 1000
  - Routers dropped up to 10% of the packets

[JaKa88]

Throughput vs. Goodput

- Throughput
  - Amount of network layer data delivered in a time interval
    - E.g., 1 Gbit/s
    - Counts everything including retransmissions

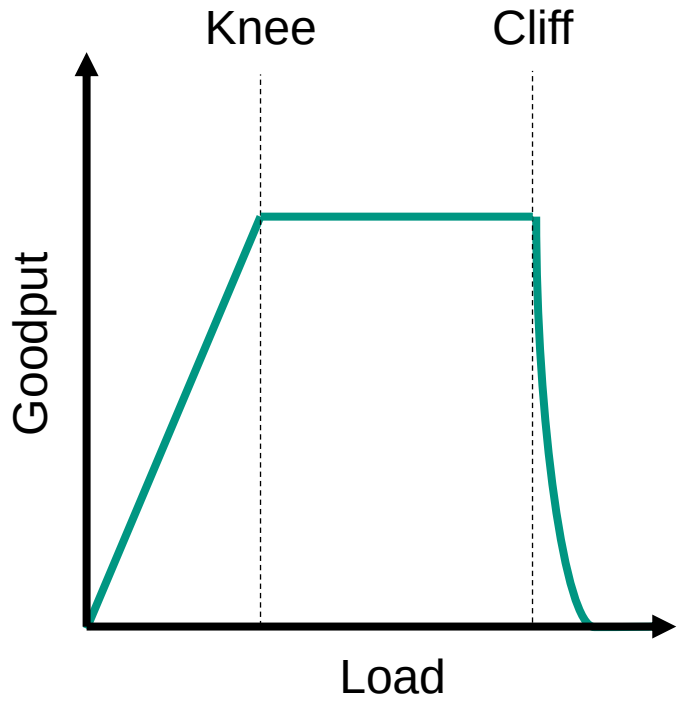
… this is the aggregated amount of data that flows through the router/link

- Goodput
  - „Application-level“ throughput
  - In our case
    - Amount of application data delivered in a time interval
    - Retransmissions at the transport layer do not count
    - Packets dropped in transmission do not count

… this is what the applications receive, traffic at the network level might be higher

What was observed

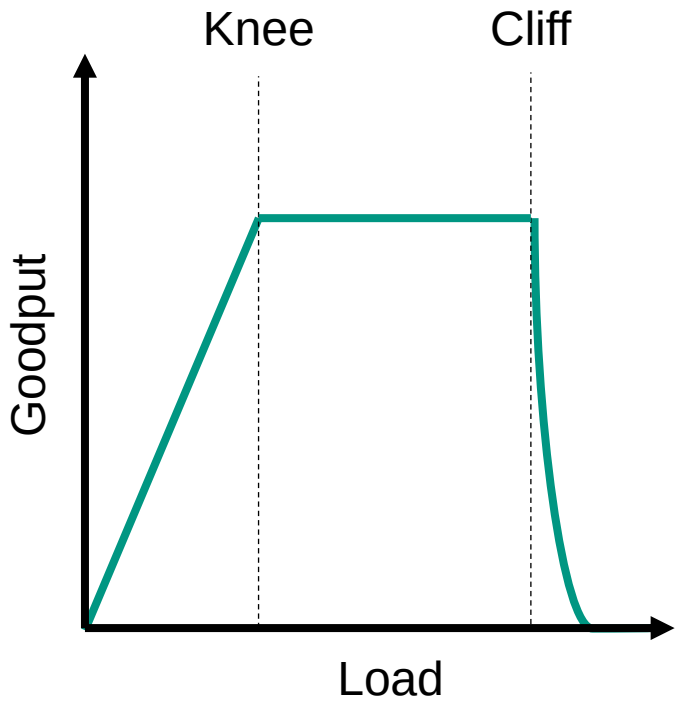
- Load is small (below capacity of bottleneck link)
  - → network keeps up with load
- Load reaches bottleneck link capacity (knee)
  - → Goodput stops increasing, buffers build up, end-to-end latency increases
  - → Network is **congested**
- Load increases beyond cliff
  - → Packets start to be dropped, goodput drastically decreases
  - → **Congestion collapse**



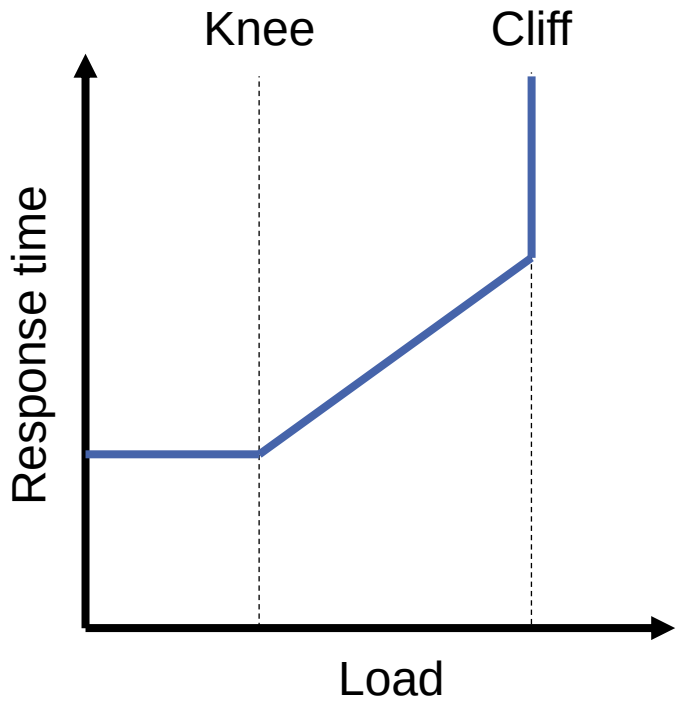
Load refers to aggregated network layer traffic that is introduced by all active data streams. This includes TCP retransmissions.

Goodput and Response Time vs. Load

- Goodput



- Response time



[ChJa89]

How Could Congestion Collapse Happen?

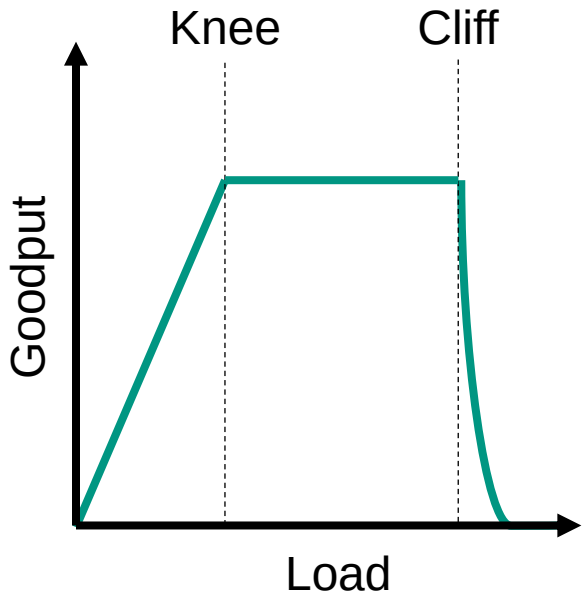
- Initial TCP standard
  - Lacks means to adjust sending rate of TCP connection to bottleneck link capacity
    - Flow control only adjusts sending rate to receiver´s capacity
  - TCP instance does not know the state of the network
    - Data are transmitted as quickly as possible

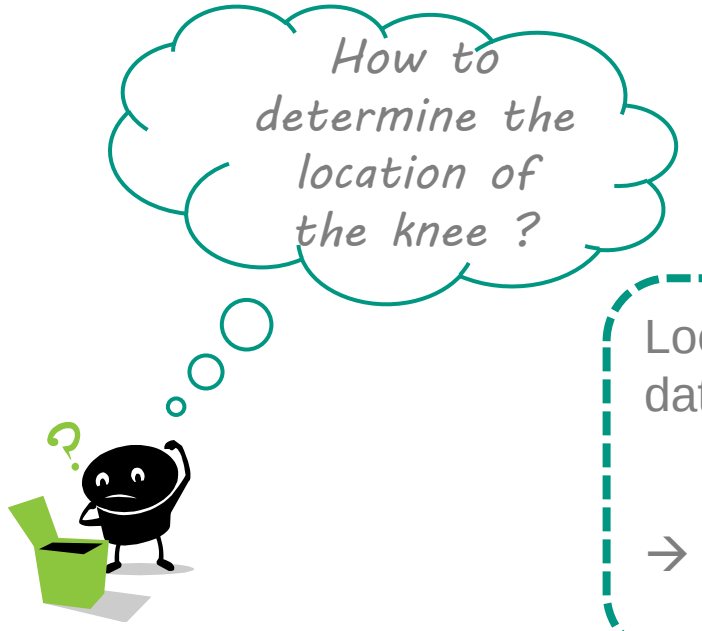
- Congestion due to
  - Single TCP connection
    - Exceeds available capacity at bottleneck link
  - Multiple TCP connections
    - Aggregated load exceeds available capacity
    - Single TCP connection has no knowledge about other TCP connections

  - Prerequisite: flow control window is large enough

Knee and Cliff

- Keep traffic load around knee
  - Good utilization of bottleneck link capacity
  - Low latencies
  - Stable goodput
- Prevent traffic from going over the cliff
  - High latencies
  - High packet losses
  - Highly decreased goodput





Location of knee changes (from a data stream's point of view) with
- Total network load
- Traffic pattern

→ Distributed optimization problem

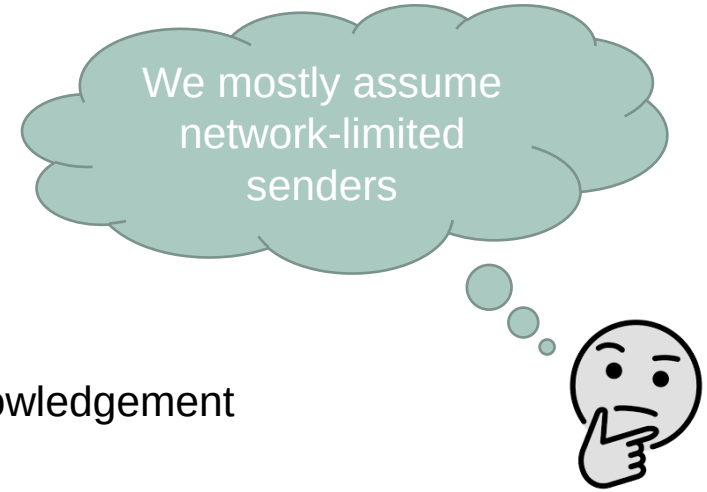
Challenge of Congestion Control

- Challenge
  - Find “optimal” sending rate
- Usually, sender has no global view of the network
  - No knowledge of
    - Network topology
    - Location of bottleneck links
    - Information about competing senders in the network
      - Number of senders, their sending rate
  - Intermediate systems (at least originally) do not tell sender that they are congested
- No trivial answer
  - There are lots of algorithms for congestion control
    - Active research topic
  - In this lecture, we will present you a selection of important algorithms

7.1.3 Network-limited Sender

Network-limited / Application-limited Sender

- Network-limited sender
  - Sender always has data to send and data are sent as quickly as sending interface allows
    - No pacing, i.e., short pauses between subsequent packets
  - Congestion control may limit sending by controlling
    - Congestion window
      - Sender can issue only a full window of data without receiveing an acknowledgement
      - The corresponding packets are sent back-to-back
    - Sending rate
      - Congestion control limits sending rate to available capacity at bottleneck link (see self clocking)



- Application-limited sender
  - Data rate of sender is limited by the application and not by the network
    - Sender does not completely utilize the available bottleneck link capacity

7.2 TCP Tahoe

7.2.1 Recap: TCP

TCP à la RFC 793

- Connection establishment
  - 3 way handshake
    - Full duplex connection
- Connection termination
  - Separately for each direction of transmission
    - Half duplex connections possible
  - 4 way handshake
- Data transfer
  - Byte-oriented sequence numbers
  - Go-back-N
    - Positive cumulative acknowledgements
    - Timeout
  - Flow control (sliding window)

[RFC793]

Pingo 07-01

- Server S sends data to client C
  - Segment with sequence number seq is sent
- Does the next segment that is sent by S always have the sequence number $seq + 1$?
  - Yes
  - No

https://pingo.coactum.de/005694

Pingo 07-02

- Server S sends data to client C
  - Segment with sequence number $seq = 127$ and 10 bytes of payload is sent
- Does this segment has to include the acknowledgement number $ack = 137$?
  - Yes
  - No

https://pingo.coactum.de/005694

Pingo 07-03

- Server S sends data (25 byte each) to client C in following sequence
  - Segment with sequence number $seq = 130$ … is received
  - Segment with sequence number $seq = 155$ … is lost
  - Segment with sequence number $seq = 170$ … is received
- Which acknowledgement number is included in the next TCP segment sent from C to S?
  - 130
  - 131
  - 155
  - 156
  - 170
  - 171

https://pingo.coactum.de/005694

7.2.2 TCP Tahoe in a Nutshell

TCP Tahoe

- Mechanisms used for congestion control
  - Slow start
  - Timeout
  - Congestion avoidance
  - Fast retransmit

- Congestion signals
  - Retransmission timeout (major congestion signal) or
  - Receipt of duplicate acknowledgements ($dupACK$, minor congestion signal)

  → In case of congestion signal: enter slow start

- Reminder
  - The following must always be valid

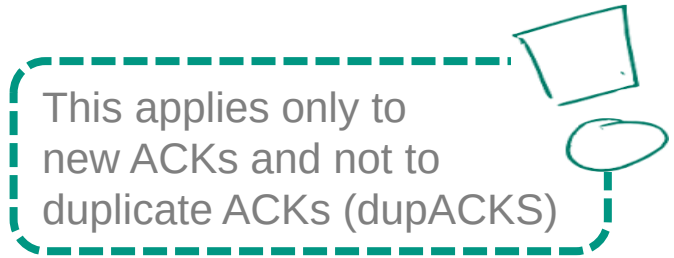
$$LastByteSent - LastByteAcked \leq \min\{CWnd, RcvWindow\}$$

TCP Tahoe

- Variables
  - Congestion window $CWnd$
  - Slow start threshold $SSThresh$
    - Value of $CWnd$ at which TCP instance switches from slow start to congestion avoidance
- Basic approach
  - AIMD (additive increase, multiplicative decrease)
    - Additive increase of $CWnd$ after receipt of an acknowledgement
    - Multiplicative decrease of $CWnd$ if packet loss is assumed (congestion signal)
- Initial values
  - Initially: $CWnd = 1\ MSS$
    - MSS : maximum segment size (maximum payload in a TCP segment)
    - since RFC 2581: initial window $IW \leq 2\ MSS$ and $CWnd = IW$
    - … in newer variants even larger (see chapter „Data Transport Evolution“)
  - $SSThresh$ initially set to “infinite”
  - Number of duplicate ACKs (congestion signal): 3

Algorithm

- $CWnd < SSThresh$ and new ACKs are received: slow start
  - Exponential increase of congestion window
    - Upon receipt of an ACK: $CWnd \mathrel{+}= 1$

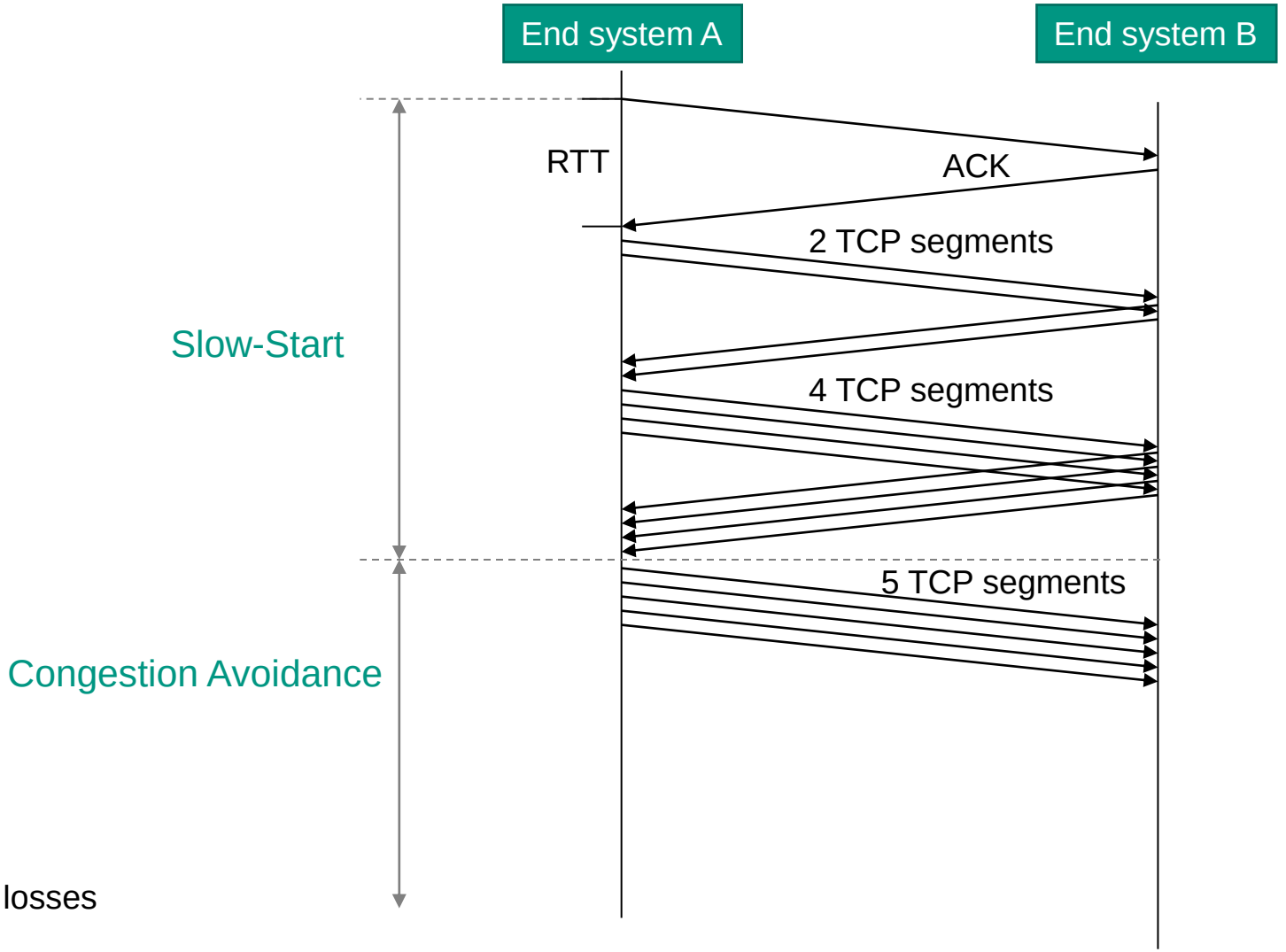


- $CWnd \geq SSThresh$ and new ACKs are received: congestion avoidance
  - Linear increase of congestion window
    - Upon receipt of an ACK : $CWnd \mathrel{+}= 1/CWnd$

- Congestion signal: timeout or 3 duplicate acknowledgements: slow start
  - Congestion is assumed
  - $SSThresh = \max(FlightSize/2,\ 2MSS)$
    - Flight size: amount of data that has been sent but not yet acknowledged
      - This amount is currently in transit
      - Might also be limited due to flow control
  - $CWnd = 1\ MSS$ or $CWnd = IW$
  - On 3 duplicate ACKs: retransmission of potentially lost TCP segment

Time Sequence Diagram: Example

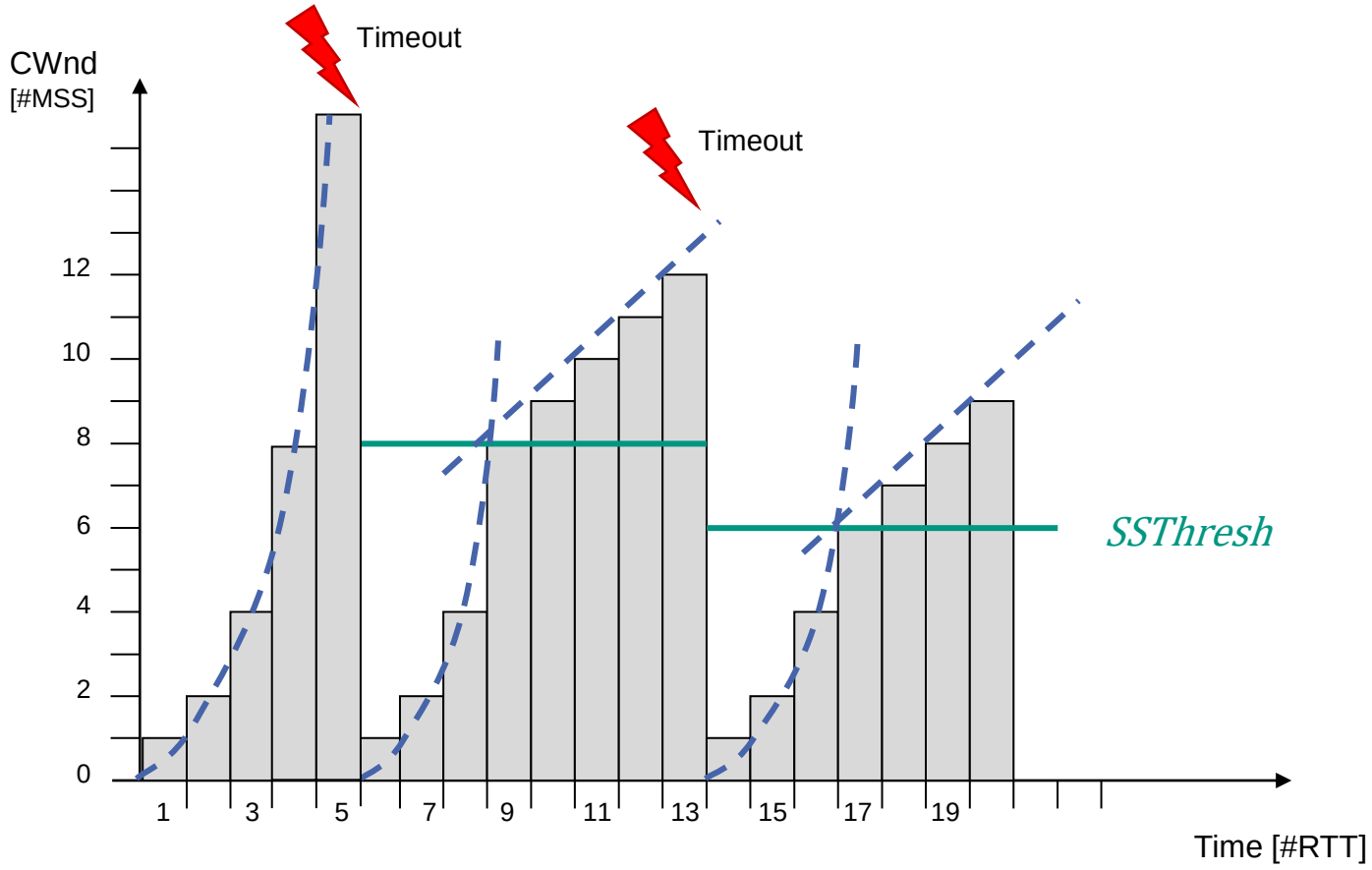
- Value of SSThresh: 4 MSS



- Assumptions
  - No transmission errors, no packet losses

Evolution of Congestion Window

- Coarse-grained abstraction



Coarse-grained Abstraction

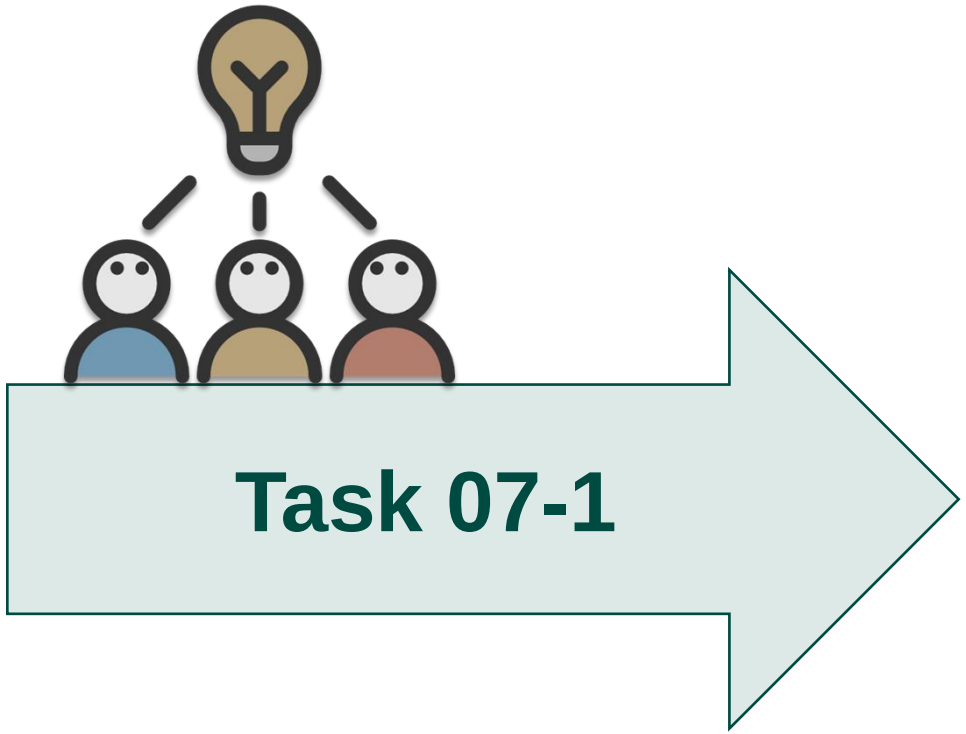
- Conceptual behavior of CWnd
  - Focus is on CWnd, packet-level details are abstracted away

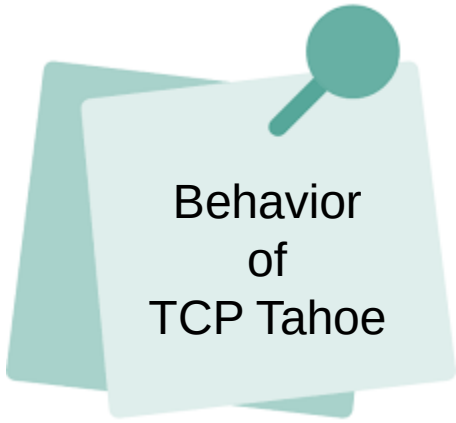
- Assumptions / simplifications
  - Senders always have data to send
    - Transmission is network-limited not application-limited
  - Packet losses occur only due to congestion
    - Transmission errors and the like are not considered
  - RcWnd >= CWnd
    - Flow control does not limit transmission

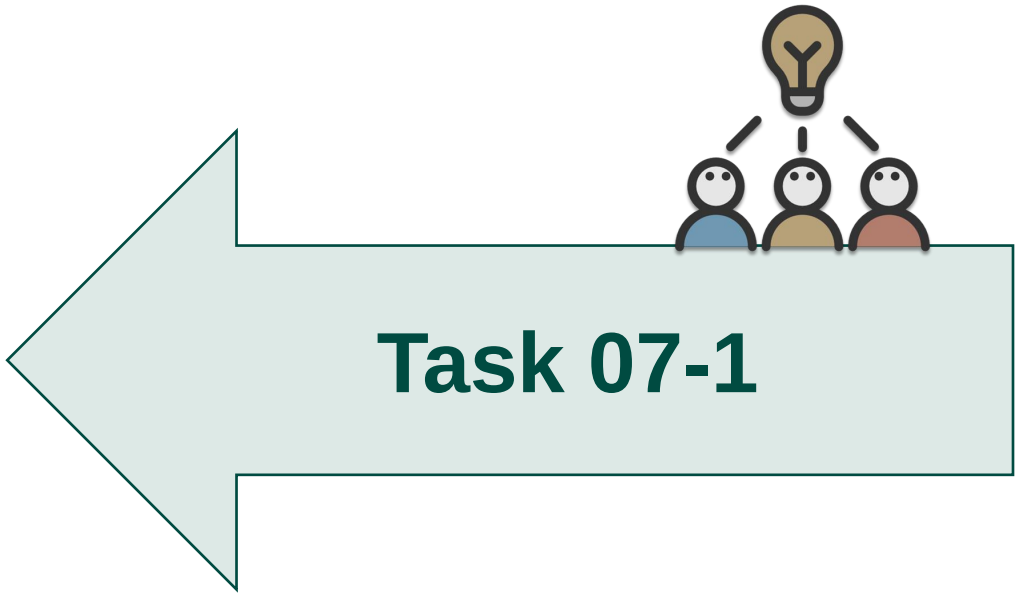
  - All actions related to a CWnd happen within one RTT
    - Processing times are neglected
    - Acknowledgements arrive instantaneously
    - Congestion situation is recognized immediately, reactions happen instantaneously at end of this RTT
      - “Logical order” is maintained
    - Size of the CWnd is always rounded down at the end of the RTT

Basic goal is to first understand the basic concepts of congestion control

Later we will inspect more implementation-related aspects of congestion control


Task 07-1


Behavior
of
TCP Tahoe


Task 07-1

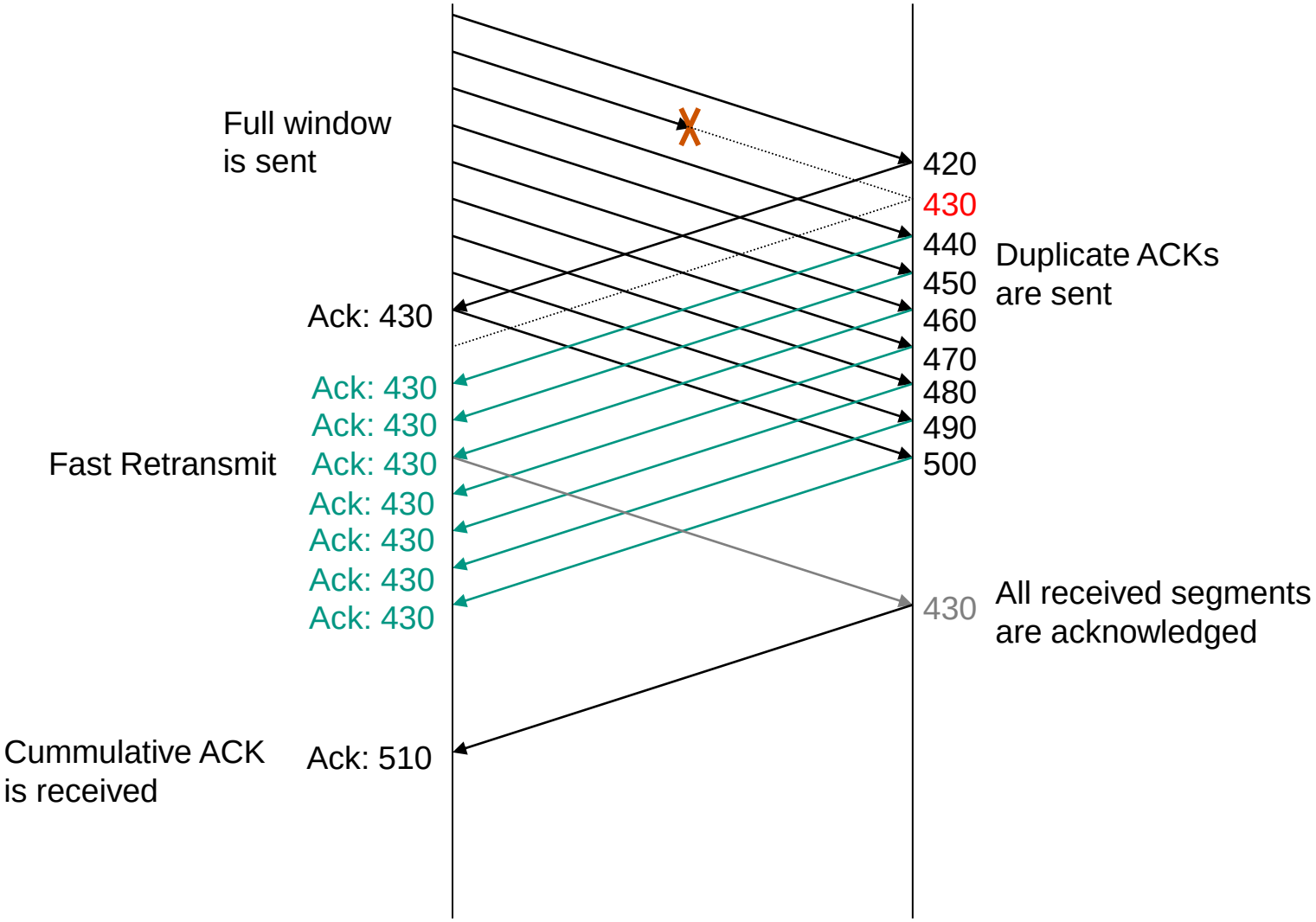
Fast Retransmit

- Observation
  - Transmitted segments

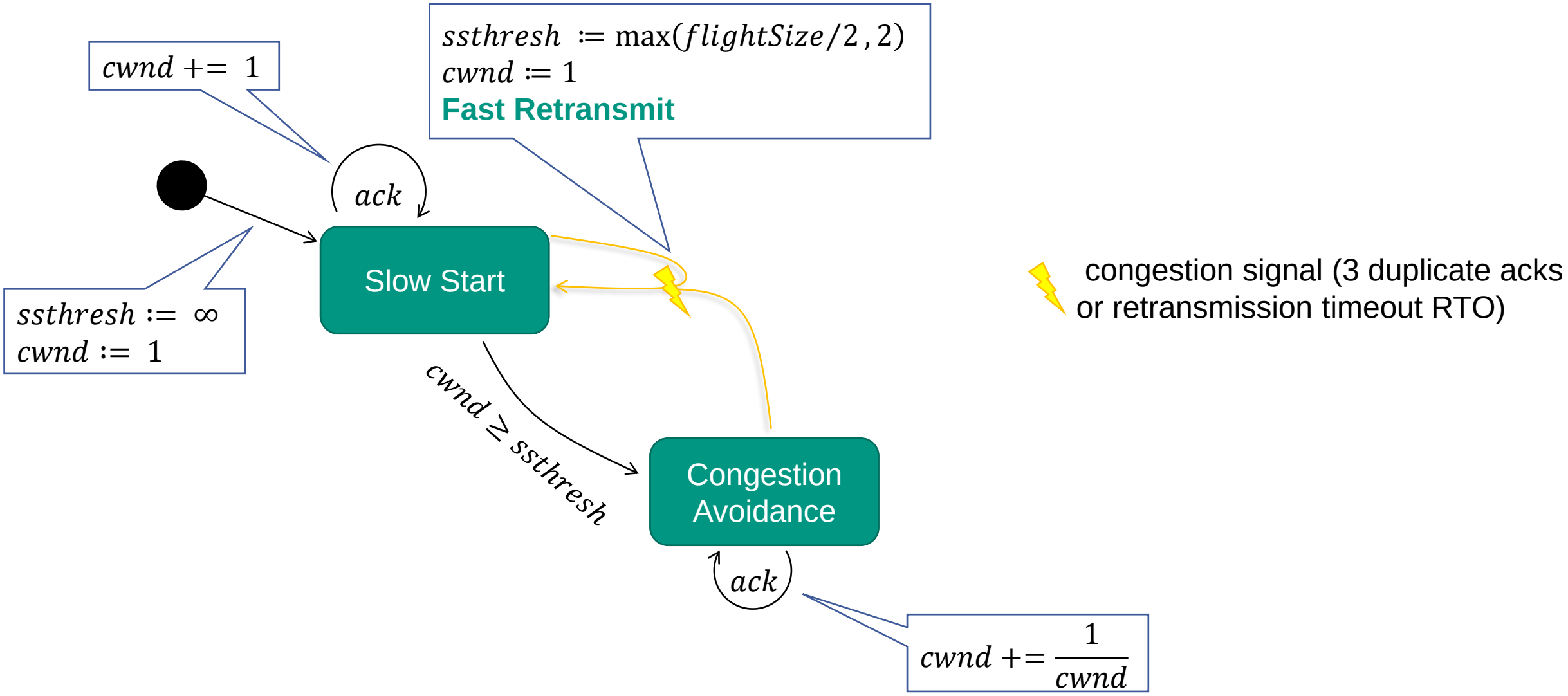
  - Received segments

    - Segment “300” is received out of order due to loss of segment 200
  - → Not every segment that is received out of order indicates congestion
    - E.g., only one segment is dropped, otherwise data transfer is ok
- What would happen?
  - Wait until retransmission timer expires, then retransmission
    - Waiting time is longer than a round trip time (RTT)
- Goal: faster reaction
  - Retransmission after receipt of a pre-defined number of duplicate ACKs
  - → Much faster than waiting for expiration of retransmission timer

Duplicate ACKs



TCP Tahoe – State Machine



Retransmission Timeout (RTO)

- To which value should retransmission timeout be set?

- Estimate round trip time
  - Apply exponential weighted moving average (EWMA)
    - Influence of each value becomes gradually less as it ages
    - Unbiased estimator for average value

$$EstimatedRTT = (1-\alpha) \cdot EstimatedRTT + \alpha \cdot SampleRTT$$

- Observation
  - Variation of RTT can greatly increase in higher loaded networks
    - Introduce “safety margin”
    - Estimation error: difference between measured and estimated RTT

$$Deviation = (1-\gamma) \cdot Deviation + \gamma \cdot |SampleRTT - EstimatedRTT|$$

$$RTO = EstimatedRTT + \beta \cdot Deviation$$

- Recommended values
  - $\alpha = 0{,}125$; $\beta = 4$ and $\gamma = 0{,}25$
  - Initial value of RTO: 1s, minimal value: 1s

Multiple Retransmissions

- Problem 1
  - How large should the time interval be between two subsequent retransmissions of the same segment?

- Approach
  - Exponential backoff *
    - After each new retransmission RTO doubles: $RTO = 2 \cdot RTO$
      - Maximal value should be applied. It should be ≥ 60 seconds

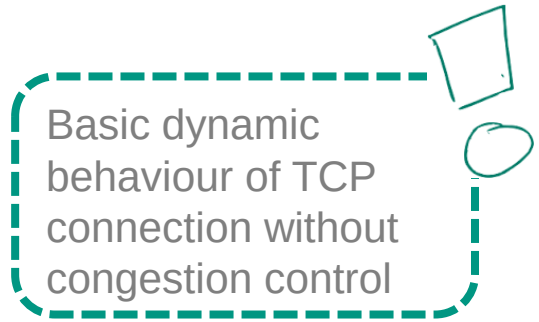
** … same principle as the exponential backoff for subsequent transmission attempts in Ethernet*

Multiple Retransmissions

- Problem 2
  - To which segment does the received ACK belong – to the original segment or to the retransmission?

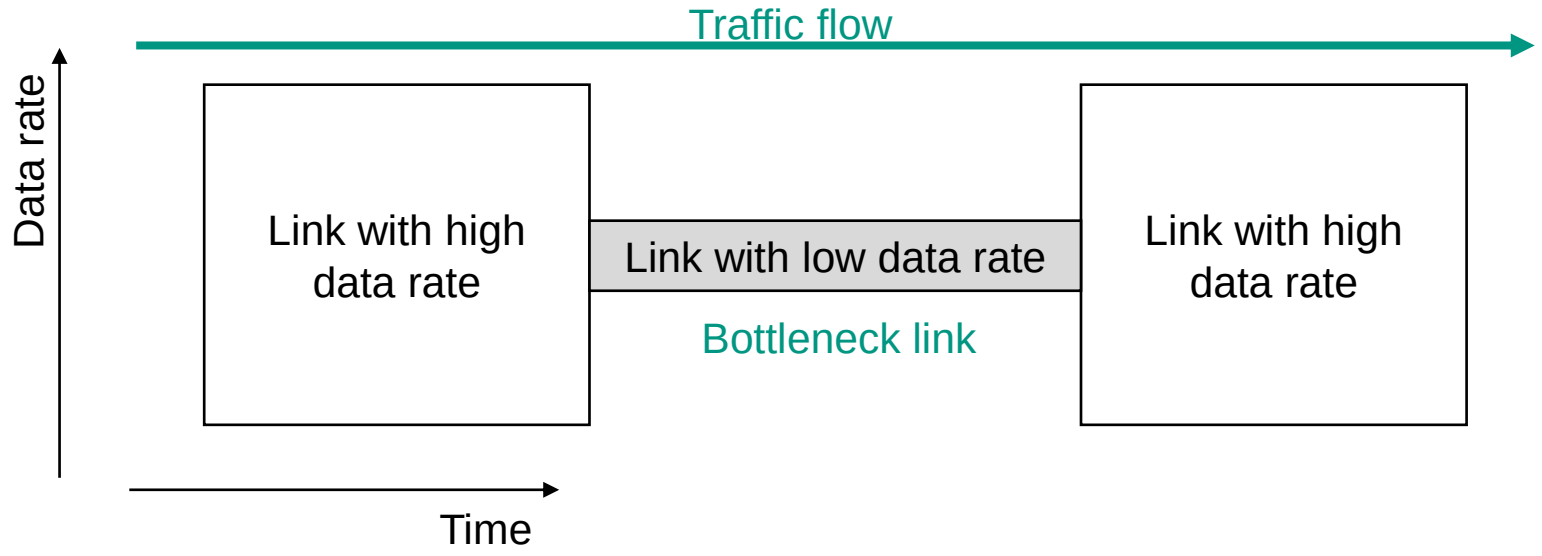
- Approach
  - Karn‘s Algorithm
    - ACKs for retransmitted segments are not included into the calculation of $\mathrm{EstimatedRTT}$ and $\mathrm{Deviation}$
    - Backoff is calculated as before
    - Timeout value is set to the value calculated by backoff algorithm until an ACK to a non-retransmitted segment is received
    - Then original algorithm is reactivated

7.3 Self Clocking



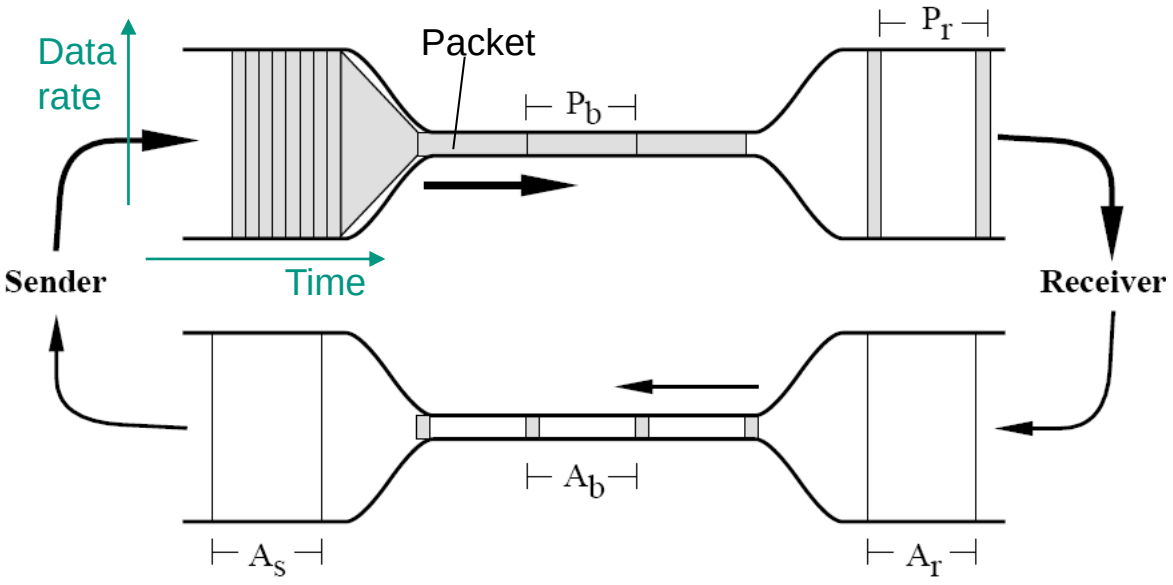
Self Clocking

- Observation is valid for any window-based system (in general)
  - Recap: TCP uses window-based flow control
- Basic assumption
  - Complete flow control window in transit
    - In TCP: receive window $RcvWindow$
  - Bottleneck link with low data rate on the path to the receiver
- Basic scenario



Self Clocking Illustration

- Basic scenario



Area corresponds to size of packet
Area remains the same on both fast and slow links

P: inter-arrival interval between two packets
A: inter-arrival interval between two ACKs
Index b: slow link (wide area network)
Index r: fast local area network (receiver)
Index s: fast local area network (sender)

[JaKa88]

Self Clocking Illustration

- Sender and receiver are connected to fast LANs
- LANs are interconnected by slow wide-area network (bottleneck)

- Sender sends all segments in current receive window as fast as its LAN allows
  - All segments (full receive window) are in transit
  - Sender always has data available to be sent (network-limited, not application-limited)

- Segments arrive at receiver with the inter-arrival time introduced at the bottleneck link
  - Segments are “stretched” at the bottleneck link
- Receiver generates ACKs with same frequency as it receives segments

- Sender needs to receive a new ACK in order to send a new segment.
  New segments have same inter-arrival time as ACKs

→ Capacity of slowest link (bottleneck) determines sending rate of the sender

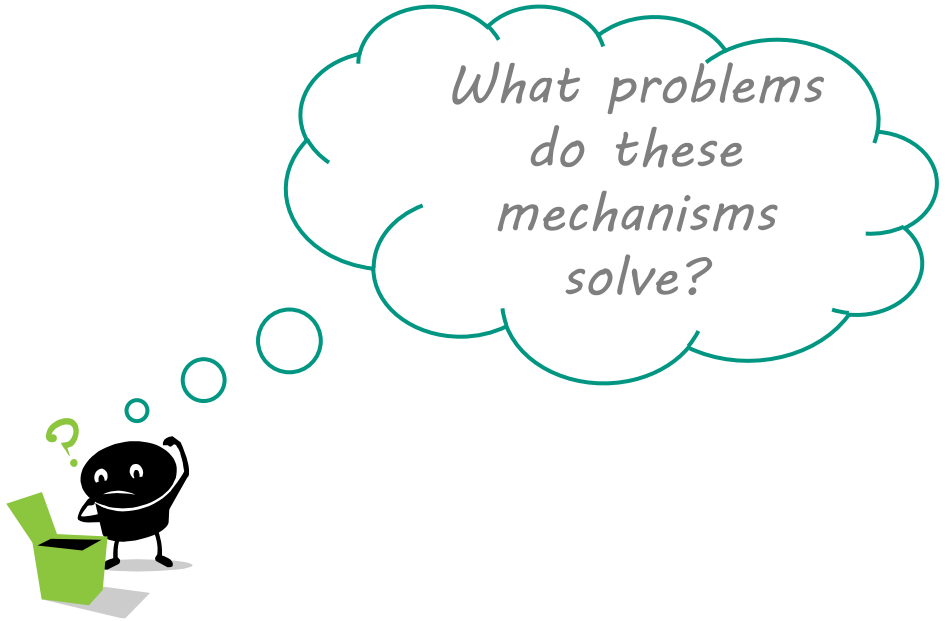
7.4 Conservation of Packets

Conservation of Packets

- Goal: get TCP connection in equilibrium
  - Full window of data in transit
  - “Conservative”
    - No new segment is injected into the network before an old segment leaves the network

  → A system with this property should be robust in the face of congestion

- Basic goal behind slow start and congestion avoidance
  - Enforce packet conservation in order to achieve network stability



Note

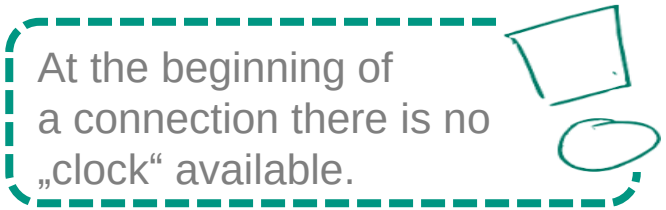
- The following slides are largely based on the seminal paper „Congestion Avoidance and Control“, V. Jacobson, M.J. Karels
- They systematically investigate different improvements
- The qualitative analysis is important and still holds, although todays networks are quite faster

[JaKa88]

7.4.1 Slow Start

Slow Start

- Goal: bring TCP connection into equilibrium
  - Connection has just started or
  - Restart after assumption of (major) congestion
- Problem: get the „clock“ started

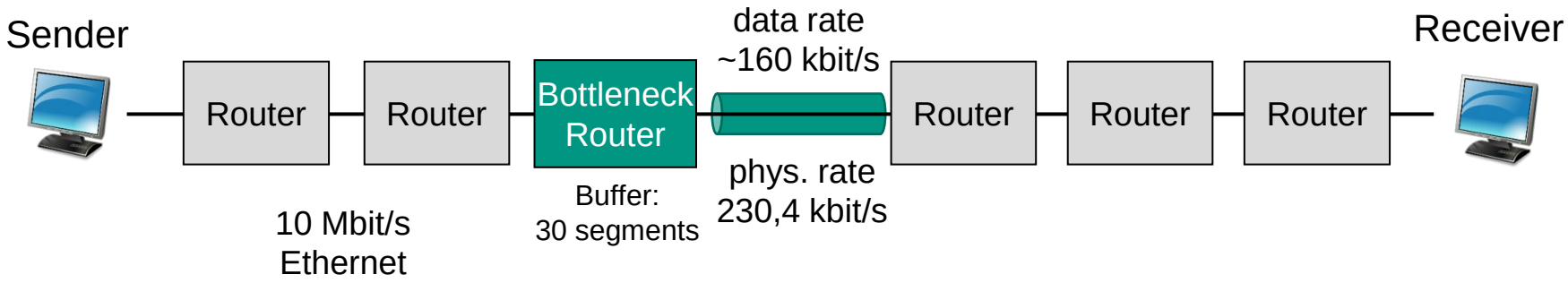


Startup Behavior of TCP Connection

- Goal of the experiment
  - Observation of startup behavior of a single TCP connection with and without slow start
  - Simple experiment is sufficient to show fundamental behavior

Startup Behavior of TCP Connection

- Experiment setup
  - Sender and receiver in different networks with 10 Mbit/s Ethernet



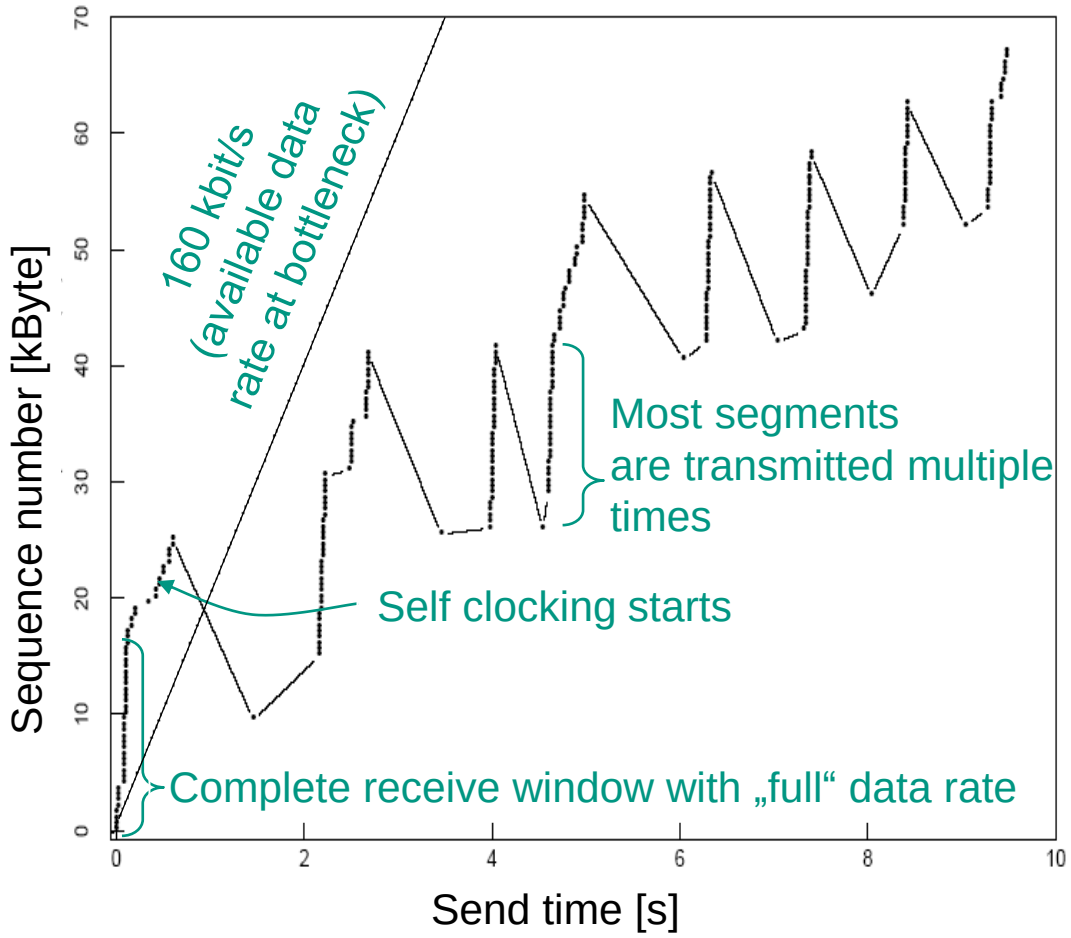
  - Buffer in bottleneck router can keep 30 segments
  - Size of receive window: 16 KByte (== 32 segments à 512 Byte)
    → Buffer in router has similar size as receive window

  - Path consists of 6 routers
    → Capacity of the path is large enough to fit the complete receive window

[JaKa88]

Startup Behavior without Slow Start

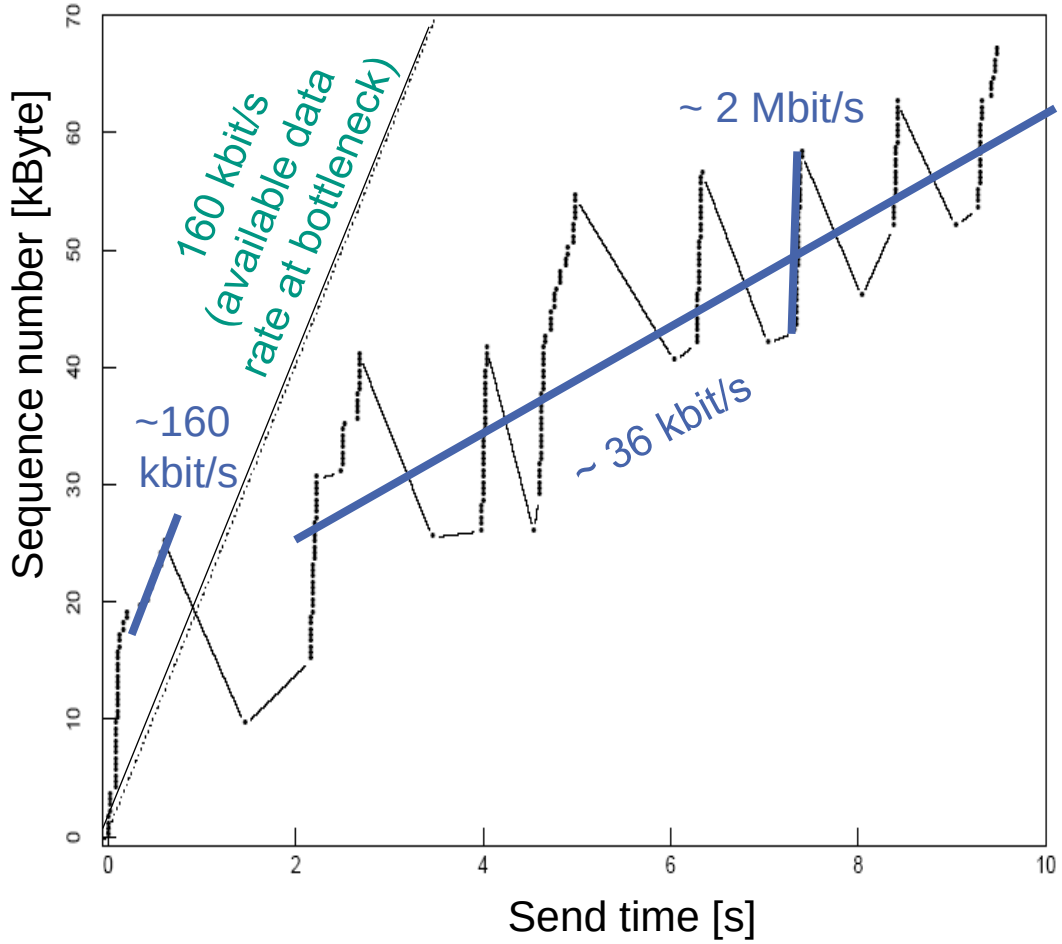
- Diagram
  - Each point corresponds to a segment of 512 bytes
  - X-axis: point in time at which segment was sent
  - Y-axis: sequence number in header of the segment
- Observations
  - Retransmissions
    - Two points with same Y-value and different X-values
    - → Sending with 10 Mbit/s causes packet losses at bottleneck
  - Go-back-N
    - Retransmission starts with first unacknowledged segment



[JaKa88]

Startup Behavior without Slow Start

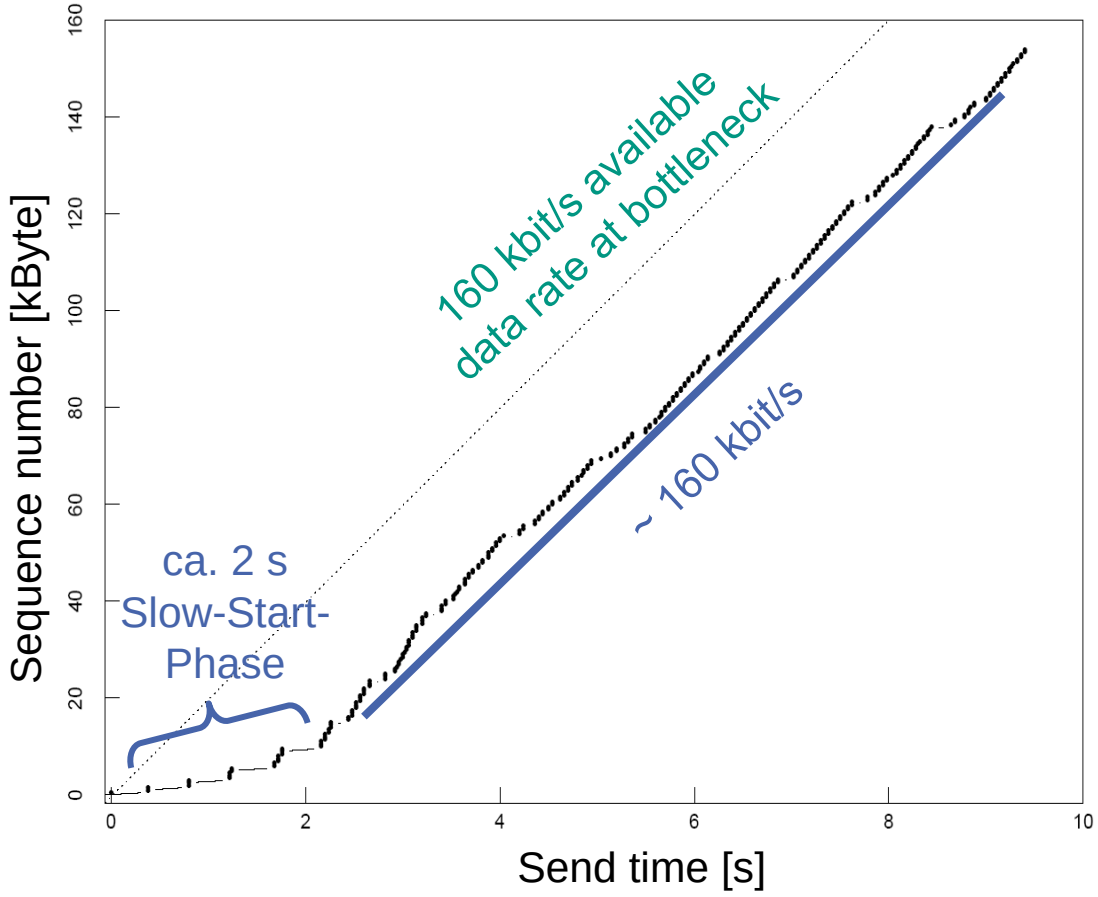
- Desired behavior
  - TCP should send with the bottleneck data rate
- Observation
  - After first loss of a segment, TCP sends in bursts
  - Many segments are sent multiple times
  - Achieved goodput of only 35%



[JaKa88]

Startup Behavior with Slow Start

- Observation
  - Slow start phase of around 2 seconds
    - Size of receive window is reached
    - Receive window “fits” on the path
  - Sender sends roughly with the data rate at the bottleneck
  - No retransmissions are observed
  - Efficiency depends on duration of the data transfer
    - Slow start can be neglected in case of long data transfers



[JaKa88]

7.4.2 Congestion Avoidance

Adapting to Available Bottleneck Link Capacity

- The previous experiment examined the behavior of a single TCP connection
- Now: consider multiple concurrent TCP connections

- Assumption: TCP connection operates in equilibrium
  - Packet loss is with a high probability caused by a newly started TCP connection
    - New connection requires resources on bottleneck link

  → Load of already existing TCP connection(s) needs to be reduced

Congestion Avoidance

- Two basic components
  - Implicit congestion signals
    - Retransmission timeout
    - Duplicate acknowledgements
  - Strategy to adjust traffic load: AIMD (Additive Increase, Multiplicative Decrease)
    - Multiplicatively decrease load in case a congestion signal was experienced
      - On retransmission timeout

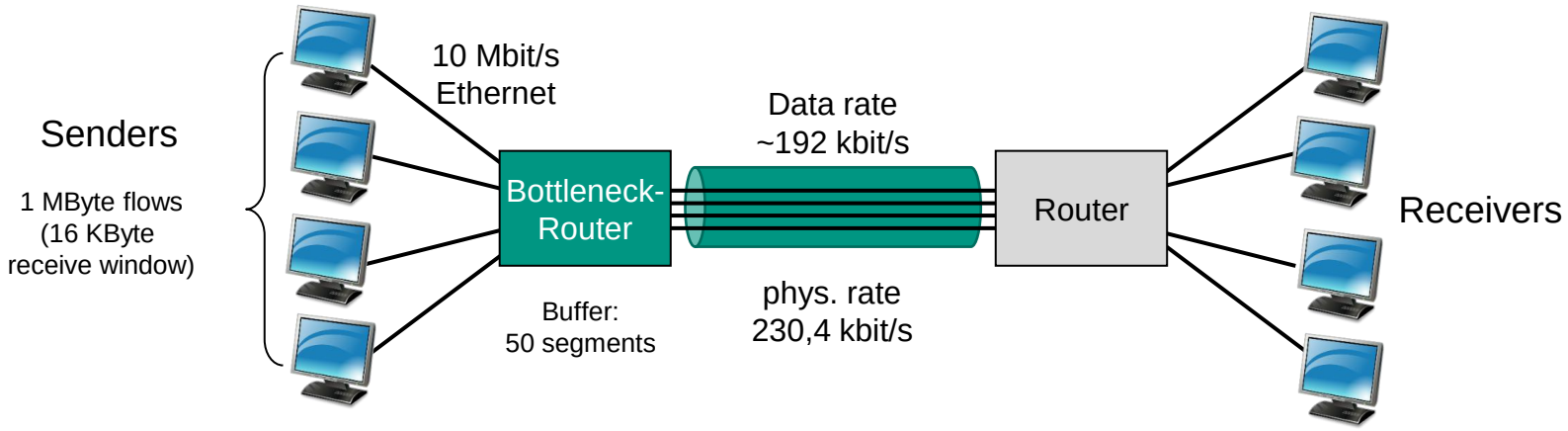
        $$\text{CWnd} = y * CWnd, \qquad with\ 0 < y < 1$$

      - In case of TCP Tahoe: $y = 1/2$
    - Additively increase load if no congestion signal is experienced
      - On acknowledgement received: $CWnd \mathrel{+}= 1/CWnd$

Congestion Avoidance

- Setup of experiment
  - Several TCP connections (max. 4) share a bottleneck link
    - Bottleneck router can buffer max. 50 segments
  - Every 3s 1 MByte of data is transferred on each connection (2048 segments à 512 Bytes)
    - Receive window: 16 KByte (32 segments à 512 Bytes)

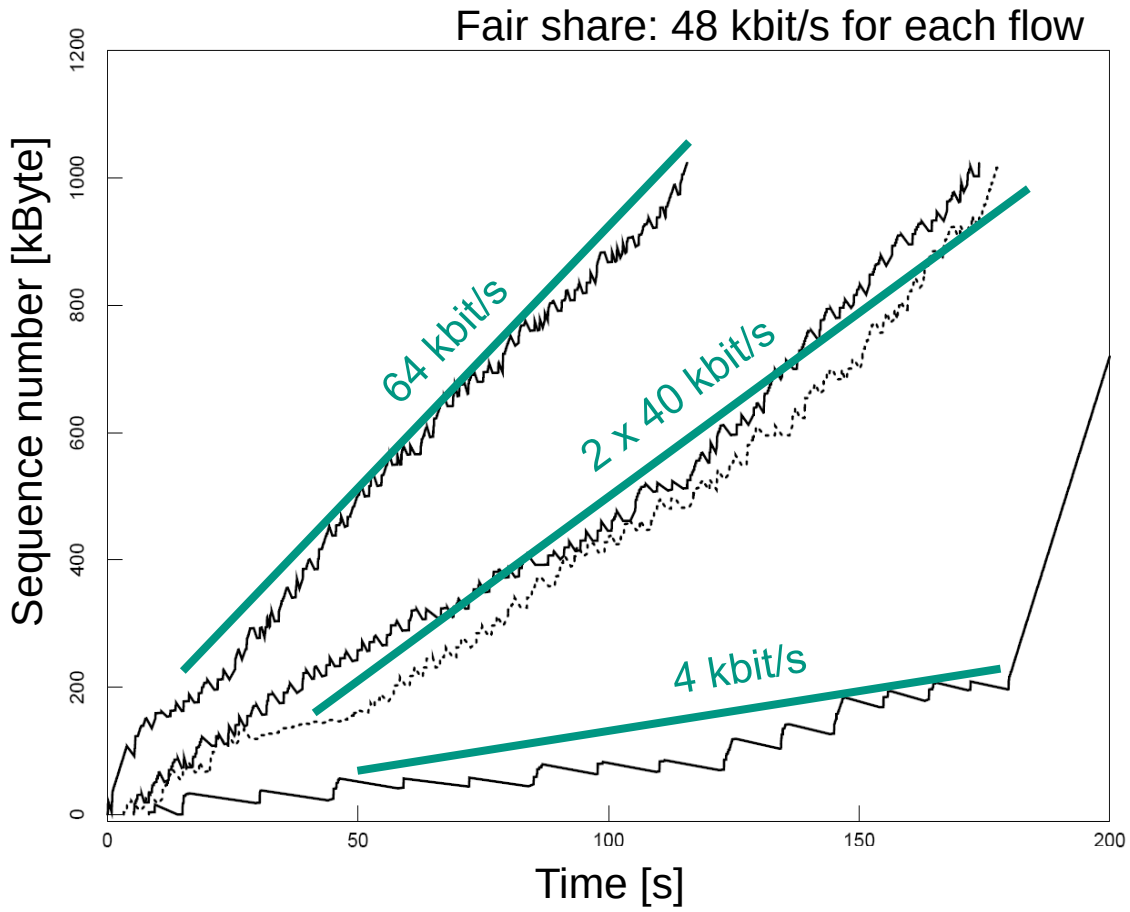
  - Two flows are sufficient to cause a buffer overflow in bottleneck router
    - All four flows exceed available buffer size by 160%



[JaKa88]

Four Streams without Congestion Avoidance

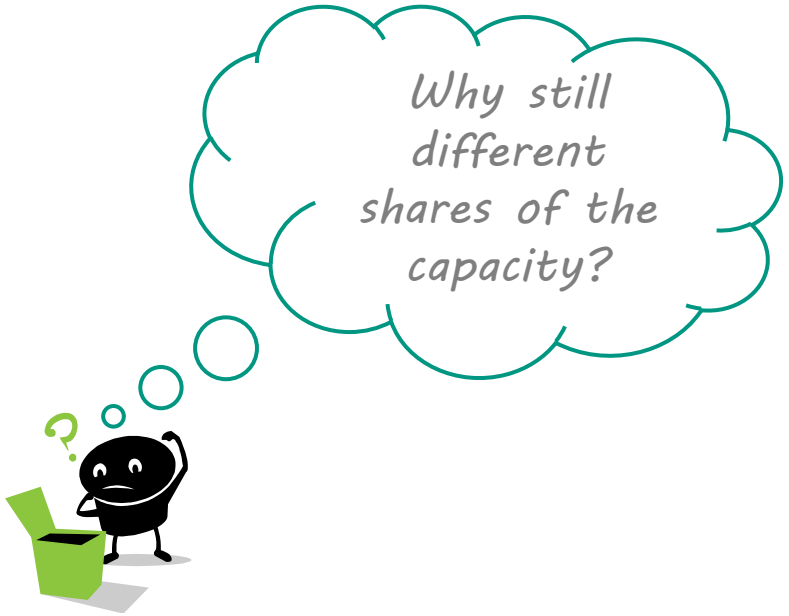
- Observation
  - 4000 of 11.000 sent segments are retransmissions!
  - Each stream should receive a 48 kbit/s share
    - But: very unfair allocation
  - Together the streams achieve a goodput of $(64 + 80 + 4)\ kbit/s = 148\ kbit/s$
    - The rest of the link capacity is wasted for unnecessary retransmissions

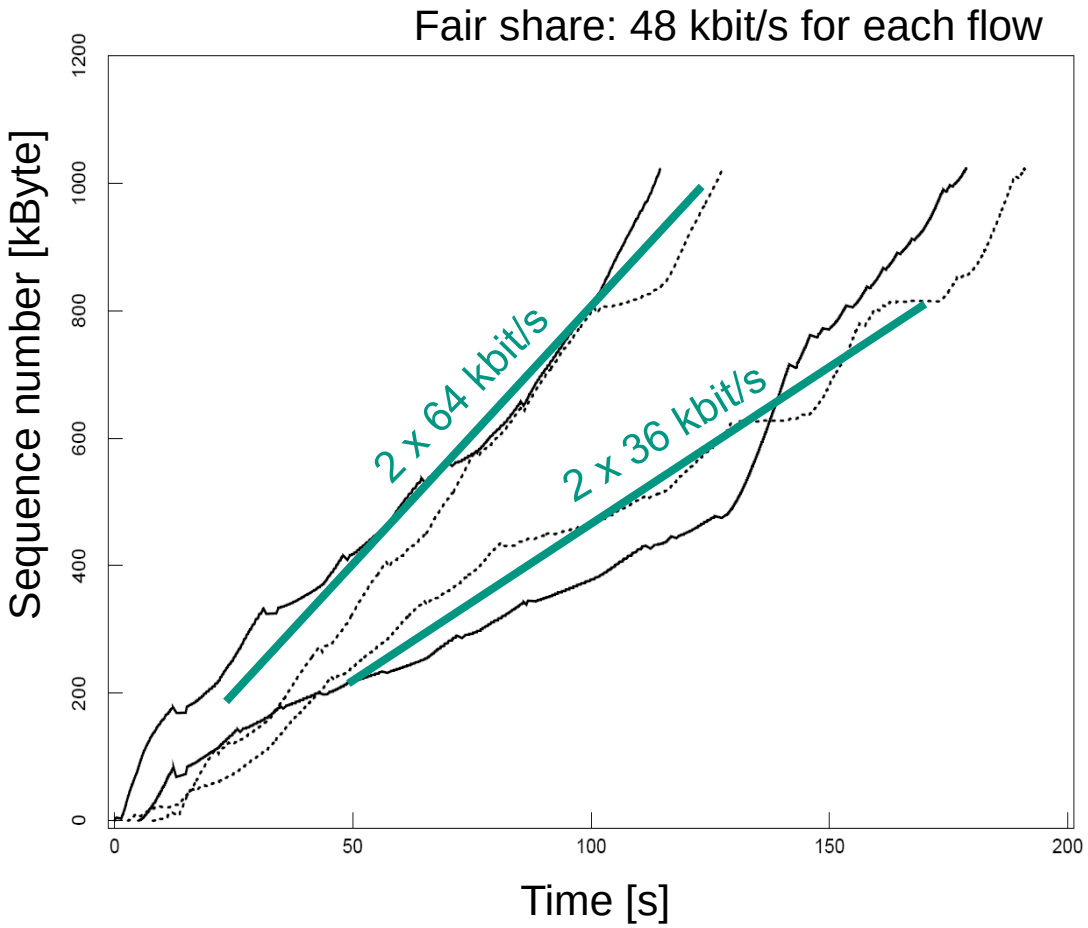


[JaKa88]

Four Streams with Congestion Avoidance

- Observation
  - Only 89 of 8281 segments are retransmissions
    - Roughly 1%
  - Capacity is not wasted with unnecessary retransmissions

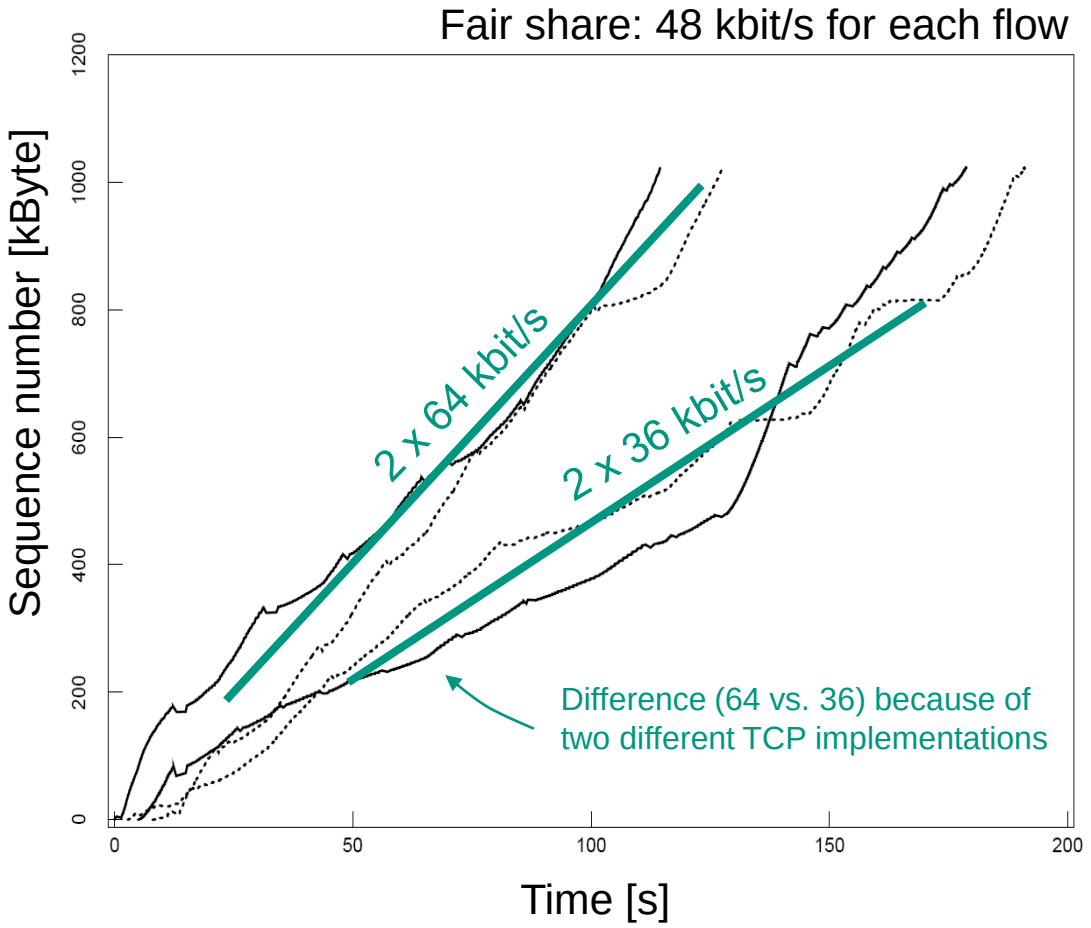




[JaKa88]

Four Streams with Congestion Avoidance

- Observation
  - Receivers use different acknowledgement policies
    - Delayed ACKs till 35% of receive window is filled or 200ms have passed
      - 36 kbit/s
      - Bursts with 5-7 segments per ACK. Loss rate of 1,8%
    - Delayed ACKs for at most one segment
      - 64 kbit/s
      - Max. 2 segments per burst. Loss rate of 0,5%



[JaKa88]

7.5 TCP Reno

TCP Reno

- Changes compared to TCP
  - Different reaction to minor congestion signal
  - Introduction of fast recovery

- In case of minor congestion signal
  - No reset to slow start
    - Receipt of duplicate ACK implies successful delivery of new segments, i.e., packets have left the network
    - New packets can also be injected in the network (→ conservation of packets)
  - Fast recovery
    - Controls sending of new segments until receipt of a non-duplicate ACK (i.e., a new ACK)
      - Retains self clocking

- In case of a major congestion signal
  - Reset to slow start as in TCP Tahoe

Fast Recovery

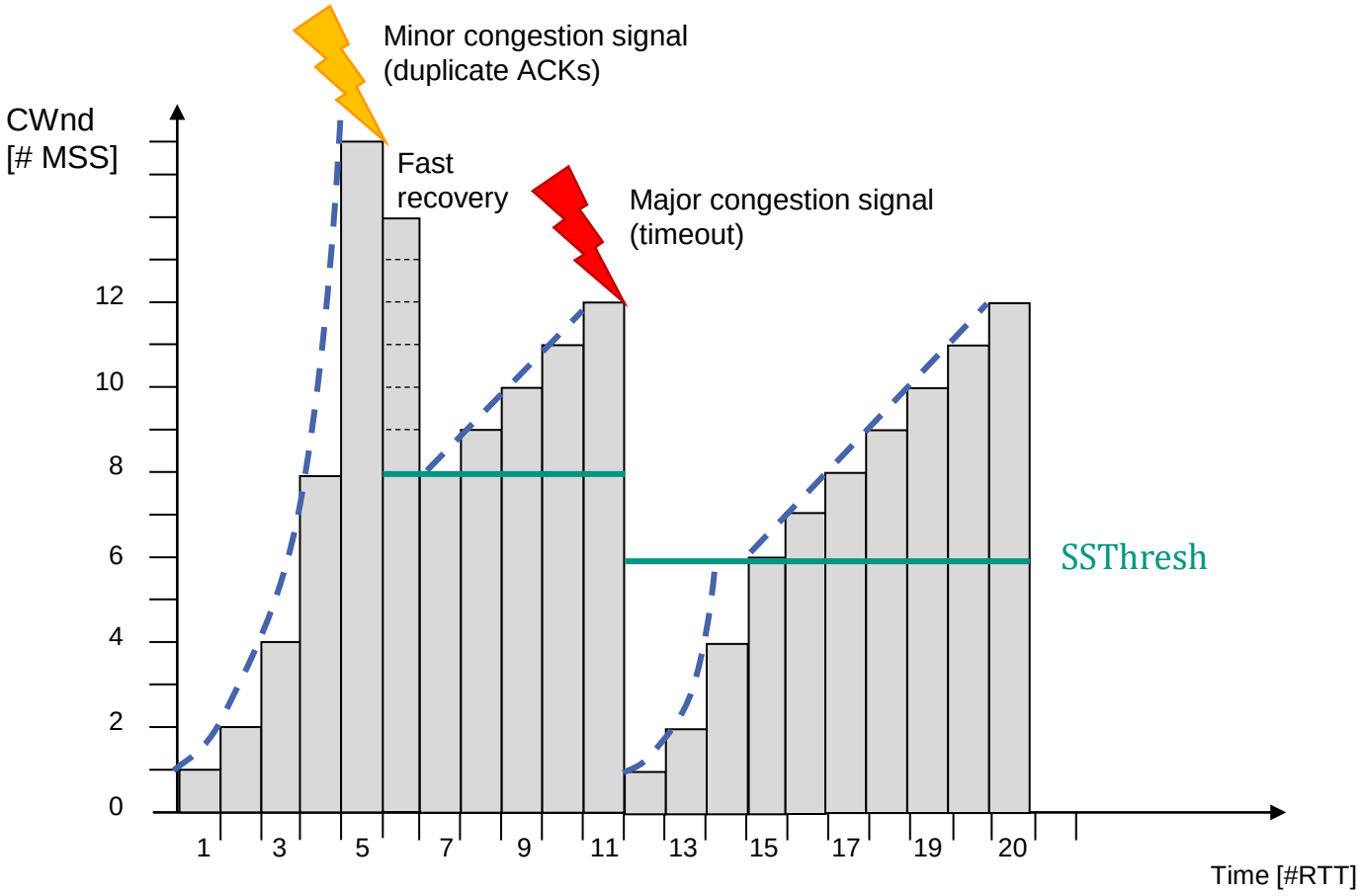
- Starting condition
  - Receipt of a specified number of duplicate ACKs
    - Usually set to 3 duplicate ACKs
- Idea
  - New segments should continue to be sent, even if packet loss is not yet recovered
    - Self clocking continuous
- Reaction
  - Reduce network load by halving the congestion window
    - Retransmit first missing segment (fast retransmit)
  - Consider continuous activity, i.e., further received segments while no *new* data is acknowledged
    - Increase congestion window by number of duplicate ACKs (usually 3)
    - Further increase after receipt of each additional duplicate ACK
  - Receipt of new ACK (new data is acknowledged)
    - Set congestion window to its value at the beginning of fast recovery

… in congestion avoidance

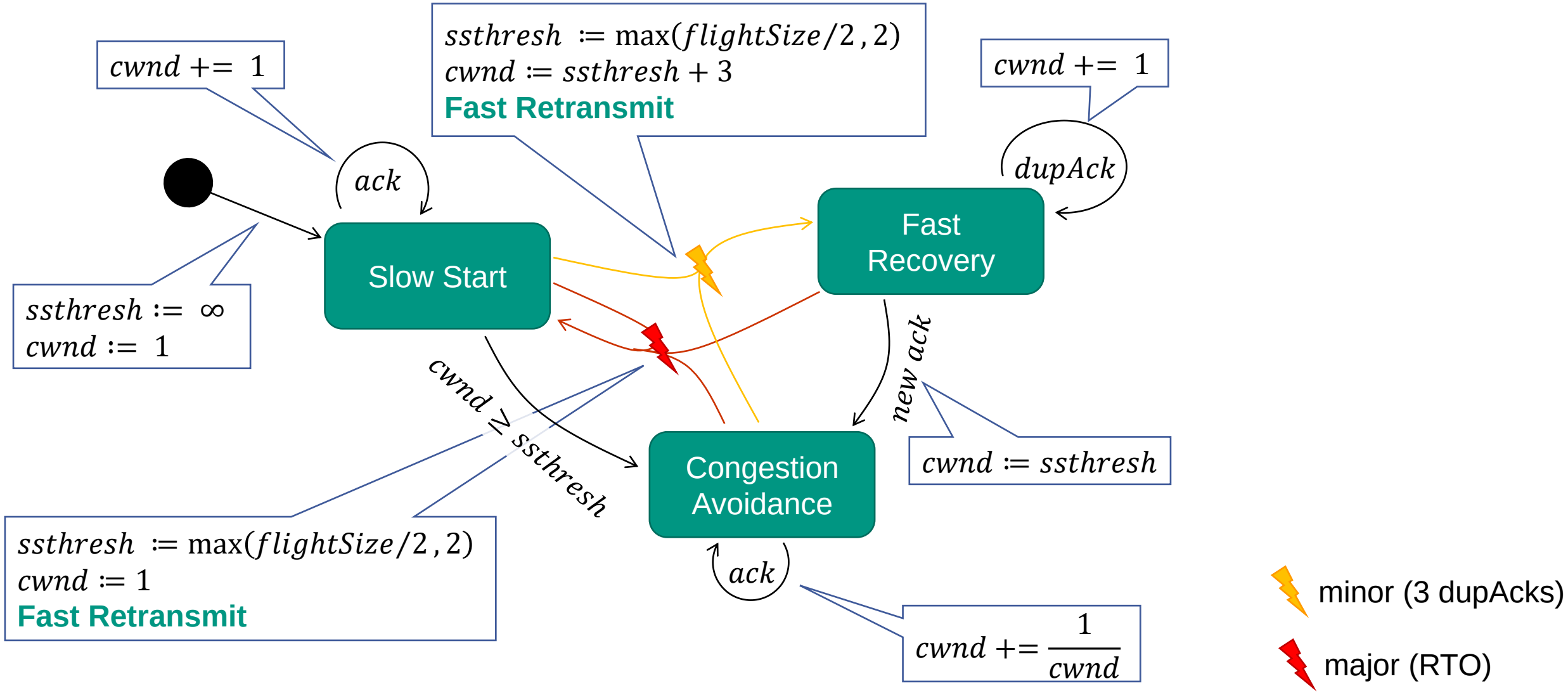
- Timeout: slow start
  - $SSThresh = \max(FlightSize/2,\ 2MSS)$
  - $CWnd = 1$

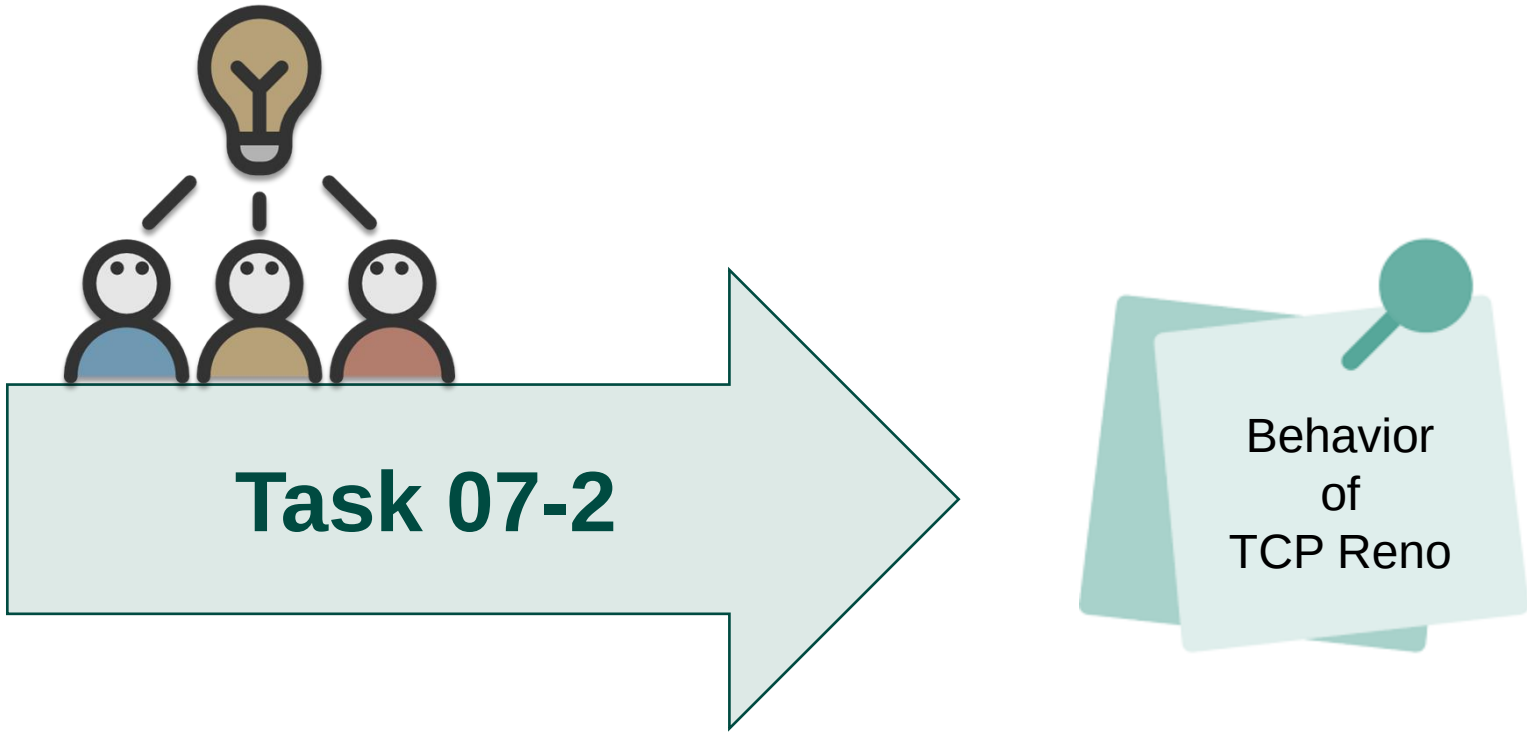
- 3 duplicate ACKs: fast recovery
  - Retransmission of oldest unacknowledged segment (fast retransmit)
  - $SSThresh = \max(FlightSize/2,\ 2MSS)$
  - $CWnd = SSThresh + 3MSS$
  - Receipt of additional duplicate ACK
    - $CWnd \mathrel{+}= 1$
    - Send new, i.e., not yet sent segments (if available)
  - Receipt of a “new” ACK: congestion avoidance
    - $CWnd = SSThresh$

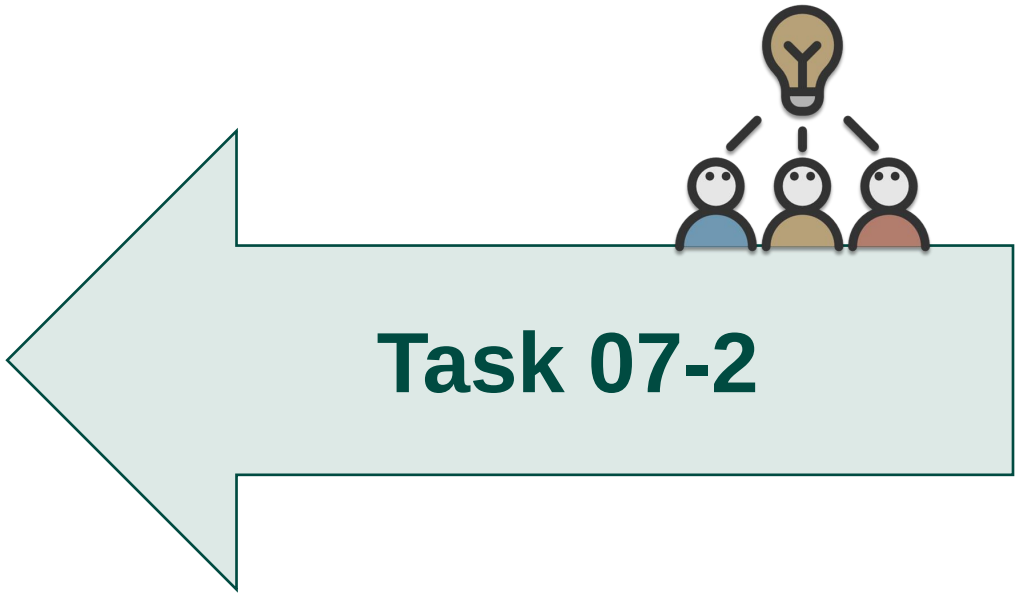
Evolution of Congestion Window with TCP Reno



TCP Reno - State Machine




Task 07-2
Behavior
of
TCP Reno


Task 07-2

Homework

TM-HW 07-01

TCP Reno

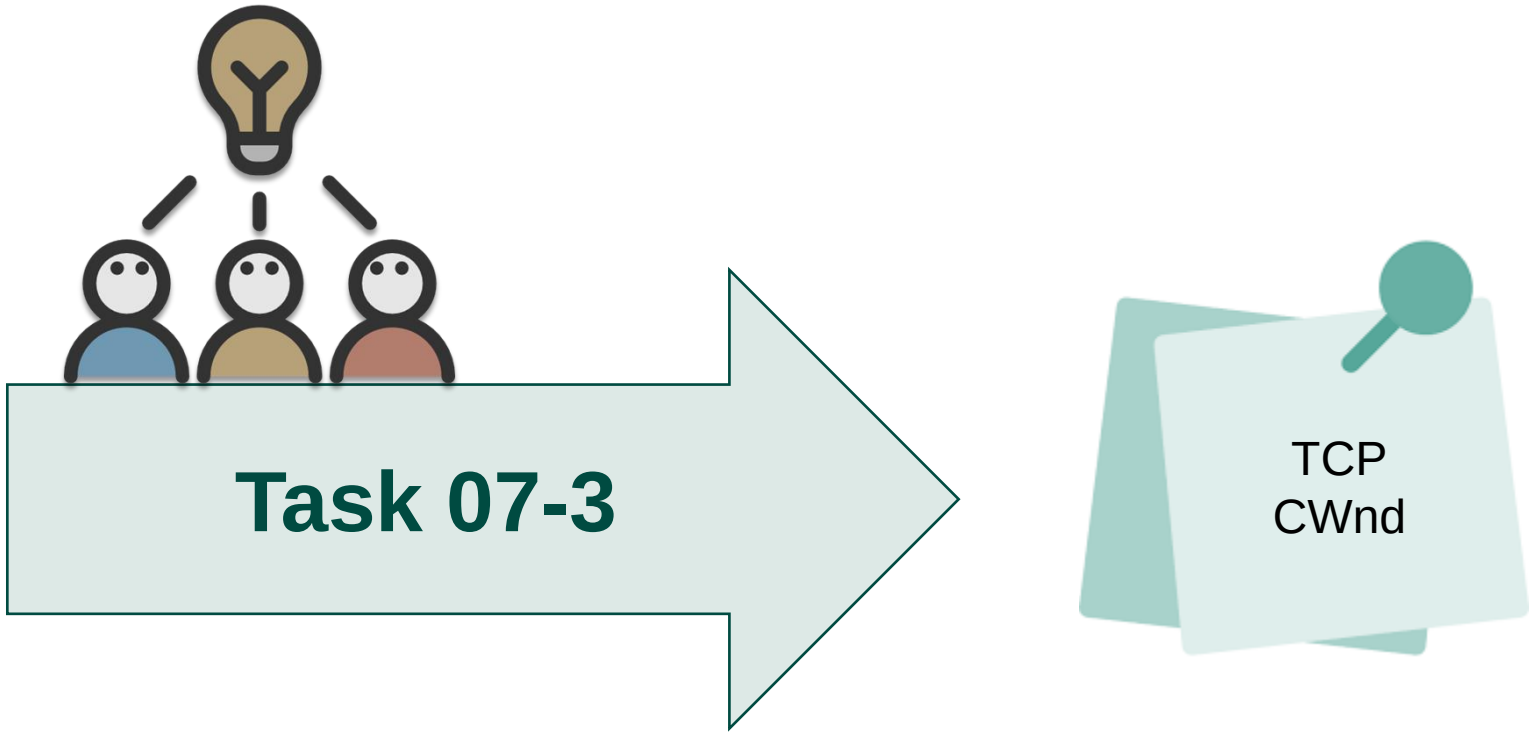
- Determine connection states
- Extend time sequence diagram
- Size of congestion window

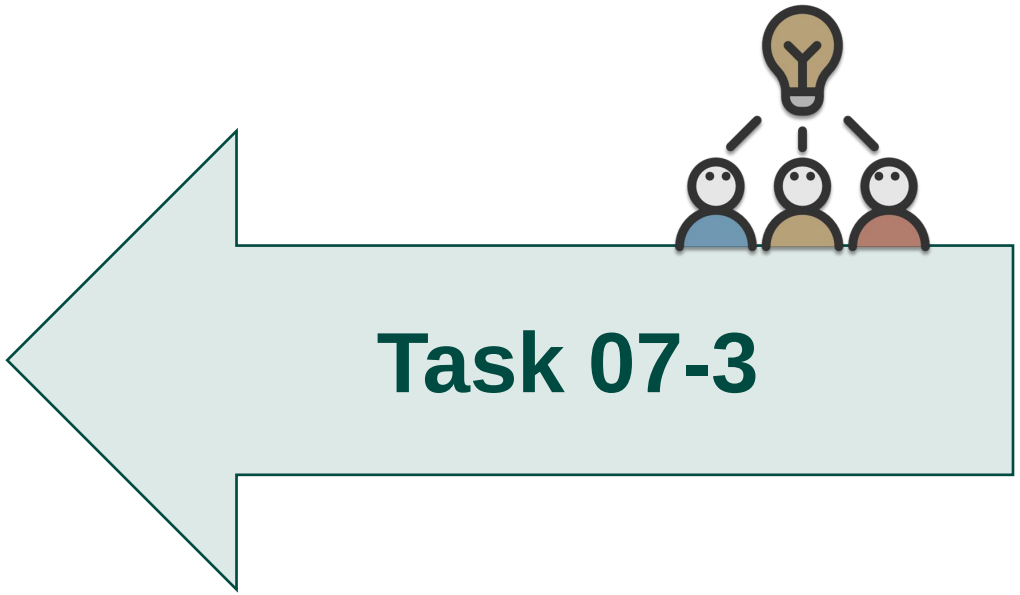
Homework

TM-HW 07-02

TCP part 2

- Conservation of packets


Task 07-3
TCP
CWnd

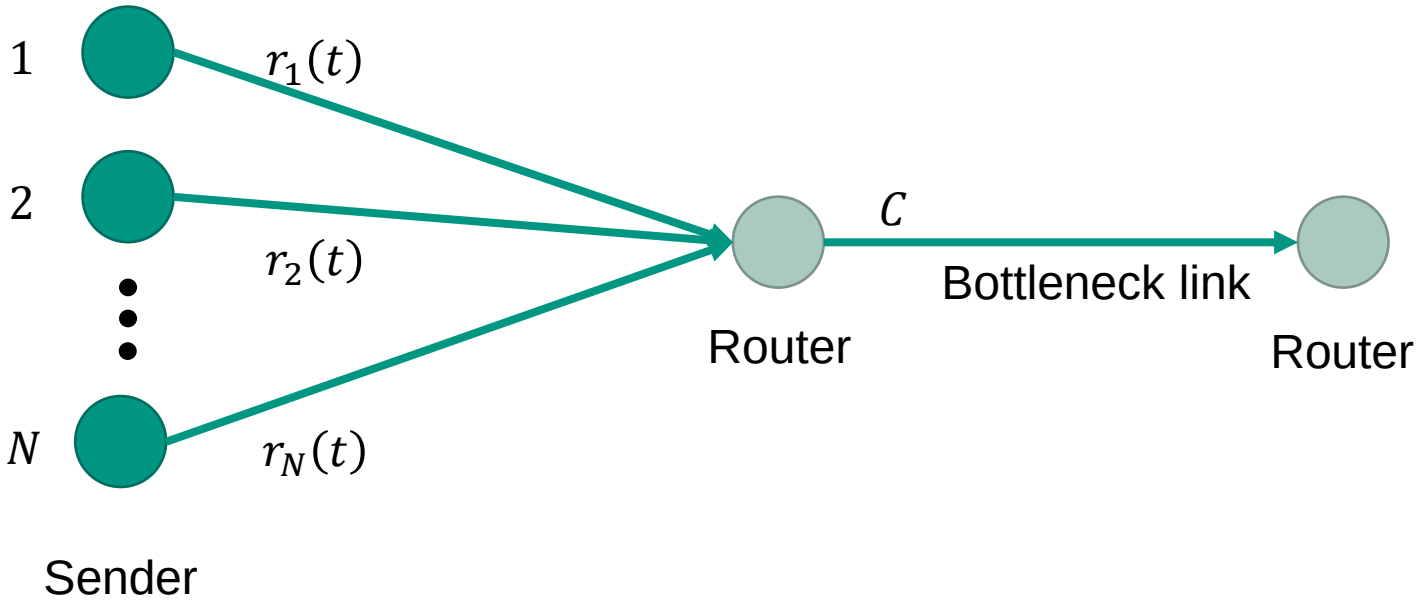

Task 07-3

7.6 Fairness

Basic Scenario

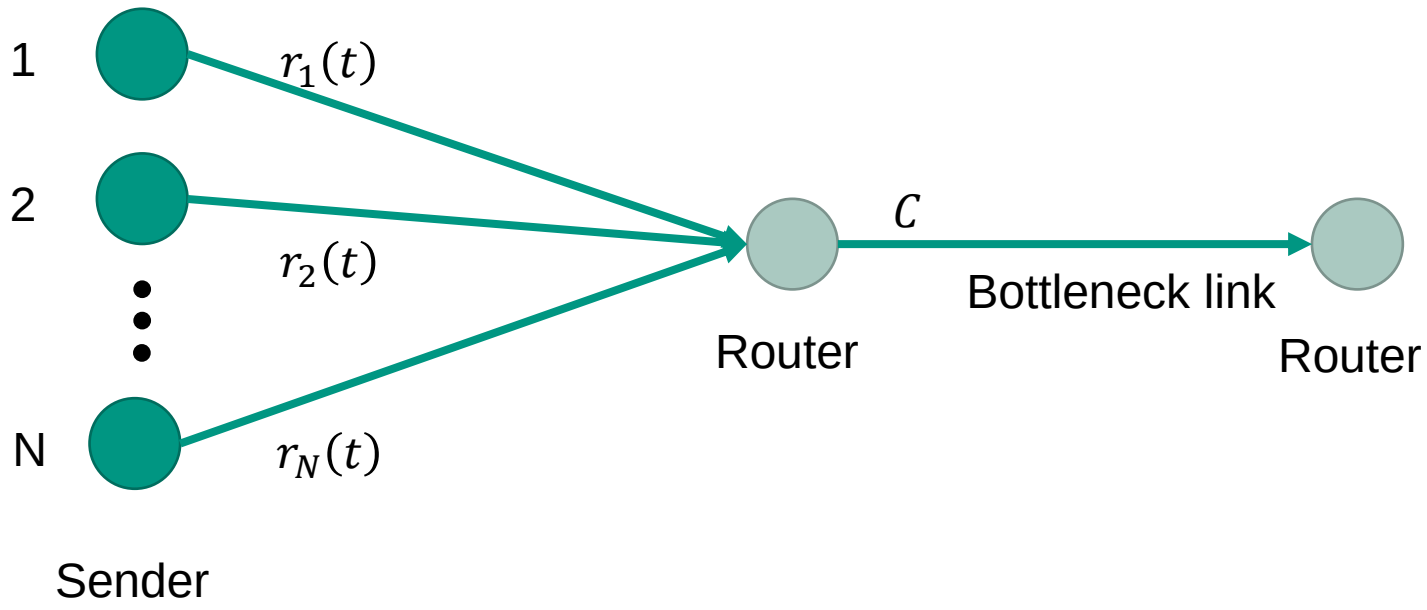
- N sender use same bottleneck link
  - Data rate of sender i: $r_i(t)$
  - Capacity of bottleneck link: C

- Bottleneck link
  - Link with lowest available data rate on the path to the receiver



Efficiency

- Closeness of the total load on the bottleneck link to its link capacity
  - $\sum_{i=1}^{N} r_i(t)$ should be as close to C as possible, i.e., close to the knee
  - Overload and underload are not desirable

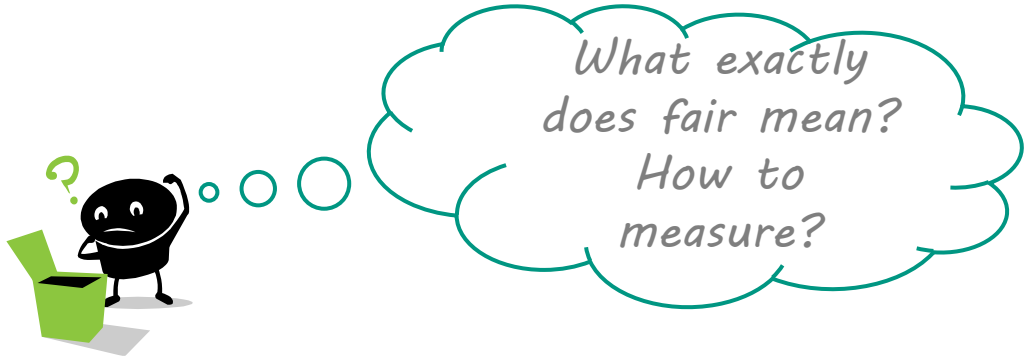


Does not make assumption on how the data rates are distributed among senders

7.6.1 Fairness Measures

Fairness

- All senders that share a bottleneck link get a fair allocation of this bottleneck link capacity



- Examples
  - Jain´s fairness index
  - Max-min fairness
  - … others do exist

Jain´s Fairness Index

- Quantify „amount“ of unfairness

$$F(r_i, \ldots, r_N) = \frac{(\sum r_i)^2}{N(\sum r_i^2)}$$

- Fairness index is bounded between 0 and 1
  - Totally fair allocation has fairness index of 1
    - All r_i are equal
  - Totally unfair allocation has fairness index of $1/N$
    - One user gets entire capacity

Max-min Fairness

- Situation
  - Each sender has equal right to the resource
  - But: some senders intrinsically demand fewer resources than others
    - E.g., in case of application-limited senders

- Intuitive allocation of fair share
  - Allocate senders with “small” demand what they want
  - Equally distribute unused resources to “big” senders

- Max-min fair allocation
  - Resources are allocated in order of increasing demand
  - No sender gets a resource share larger than its demand
  - Senders with unsatisfied demands get an equal share of the resource

Max-min Fairness

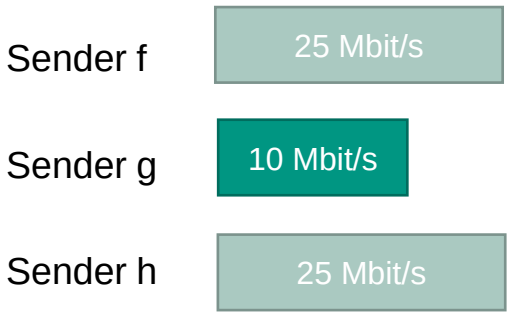
- Result of max-min fair allocation
  - Maximizes the minimum share of a sender whose demand is not fully satisfied
- Max-min fairness
  - is achieved if a sender *f* cannot increase its sending rate without decreasing the sending rate of another sender g, where $r_g < r_f$
- Implementation
  - Set of senders $1, 2, \ldots N$ with demanded sending rates $s_1, s_2 \ldots s_N$
    - Without loss of generality: $s_1 \leq s_2 \leq \cdots \leq s_N$
  - Link capacity: C
  - Give share of C/N to sender with smallest demand
    - If more than demanded, then $(C/N) - s_1$ is still available to others
  - Distribute $(C/N) - s_1$ equally to others
    - For each other sender: $(C/N) + \big((C/N) - s_1\big)/(N-1)$
  - … continue until sender gets no more than it asks for

[Kesh97]

Example

- Capacity of bottleneck link: 60 Mbit/s
- 3 senders (*f, g, h*), where *g* is application-limited to 10 Mbit/s

- Max-min fair allocation
  - Sender *g*
    - First step: allocate ${}^{60\ Mbit/s}/_{3} = 20\ Mbit/s$ to sender *g*
  - Distribute $20\frac{Mbit}{s} - 10\frac{Mbit}{s} = 10\frac{Mbit}{s}$ to others (i.e., *f* and *h*)
  - → senders *f* and *h*: 25 Mbit/s



[Kesh97]

7.6.2 Additive Increase Multiplicative Decrease

Additive Increase Multiplicative Decrease (AIMD)

- General feedback control algorithm
- Applied to congestion control
  - Additive increase of data rate until congestion
  - Multiplicative decrease of data rate in case of congestion signal

$$r_i(t+1) = \begin{cases} r_i(t) + a & if\ no\ congestion\ is\ detected \\ r_i(t) * b & if\ congestion\ is\ detected \end{cases}$$

- Converges to equal share of capacity at bottleneck link

[ChJa89]

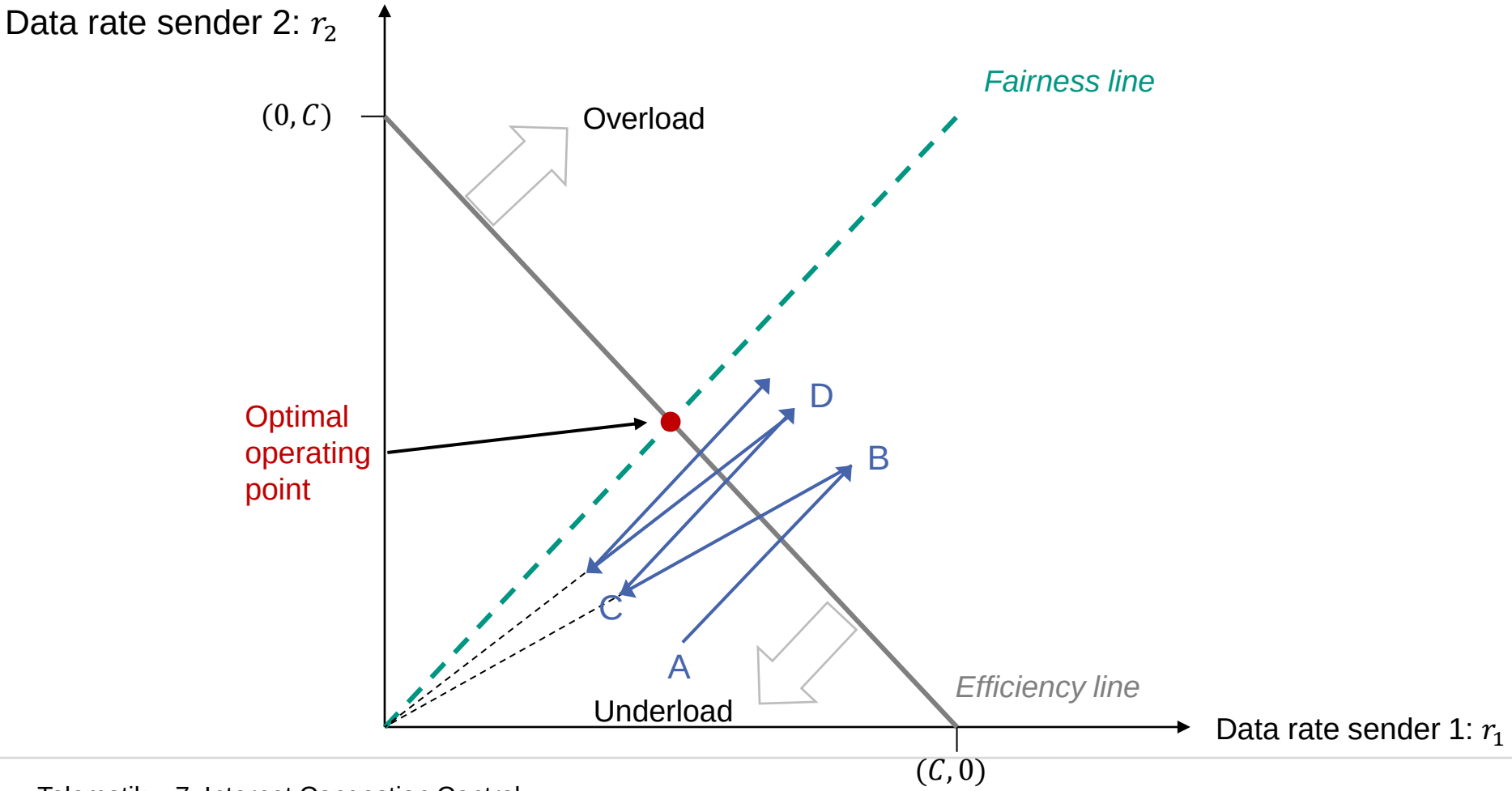
Assumptions

… behind the following considerations

- Explicit network feedback
  - Increase / decrease load
- System operates near the knee
- System operation is synchronous and in discrete time steps
  - Rate at time $t + 1$ is function of rate at time t
  - Network feedback is received at time t
    - Network feedback is received instantaneously without any delay
    - All users at the bottleneck receive same feedback

AIMD: Fairness

- Network with two senders that share a bottleneck link with capacity C
- Goal: bring system close to optimal point $(C/2, C/2)$



AIMD: Fairness

- Efficiency line
  - $r_1 + r_2 = C$ holds for all points on the line
  - Points under the line reflect an underloaded system
    - Control decision: increase rate
  - Points above the line reflect an overloaded system
    - Control decision: decrease rate
- Fairness line
  - All allocations with fair allocation, i.e., $r_1 = r_2$
  - Multiplying with b does not change fair allocation: $br_1 = br_2$

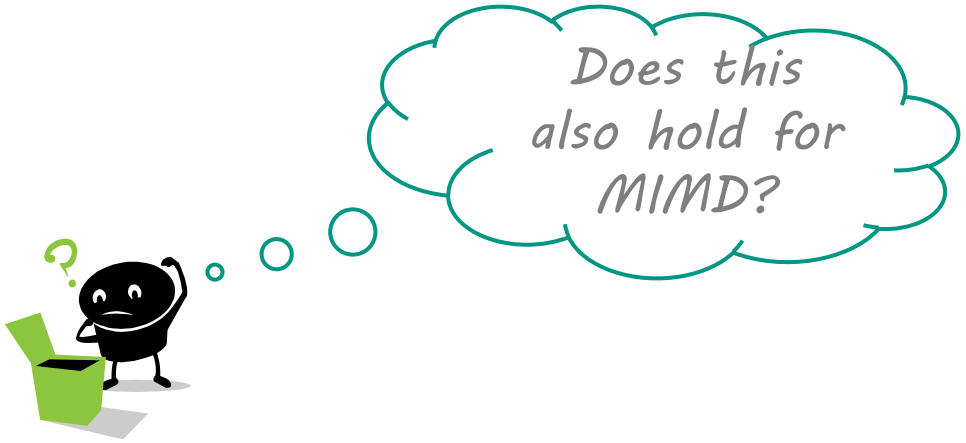
$$F(r_1, r_2) = \frac{(r_1 + r_2)^2}{2({r_1}^2 + {r_2}^2)}$$

- Optimal operating point
  - Intersection of efficiency line and fairness line: point $(C/2, C/2)$

Optimality of AIMD

- Additive increase
  - Resource allocation of both users increased by a
  - In the graph: moving up along a 45-degree line
- Multiplicative decrease
  - Move down along the line that connects to the origin

→ Point of operation iteratively moves closer to optimal operating point



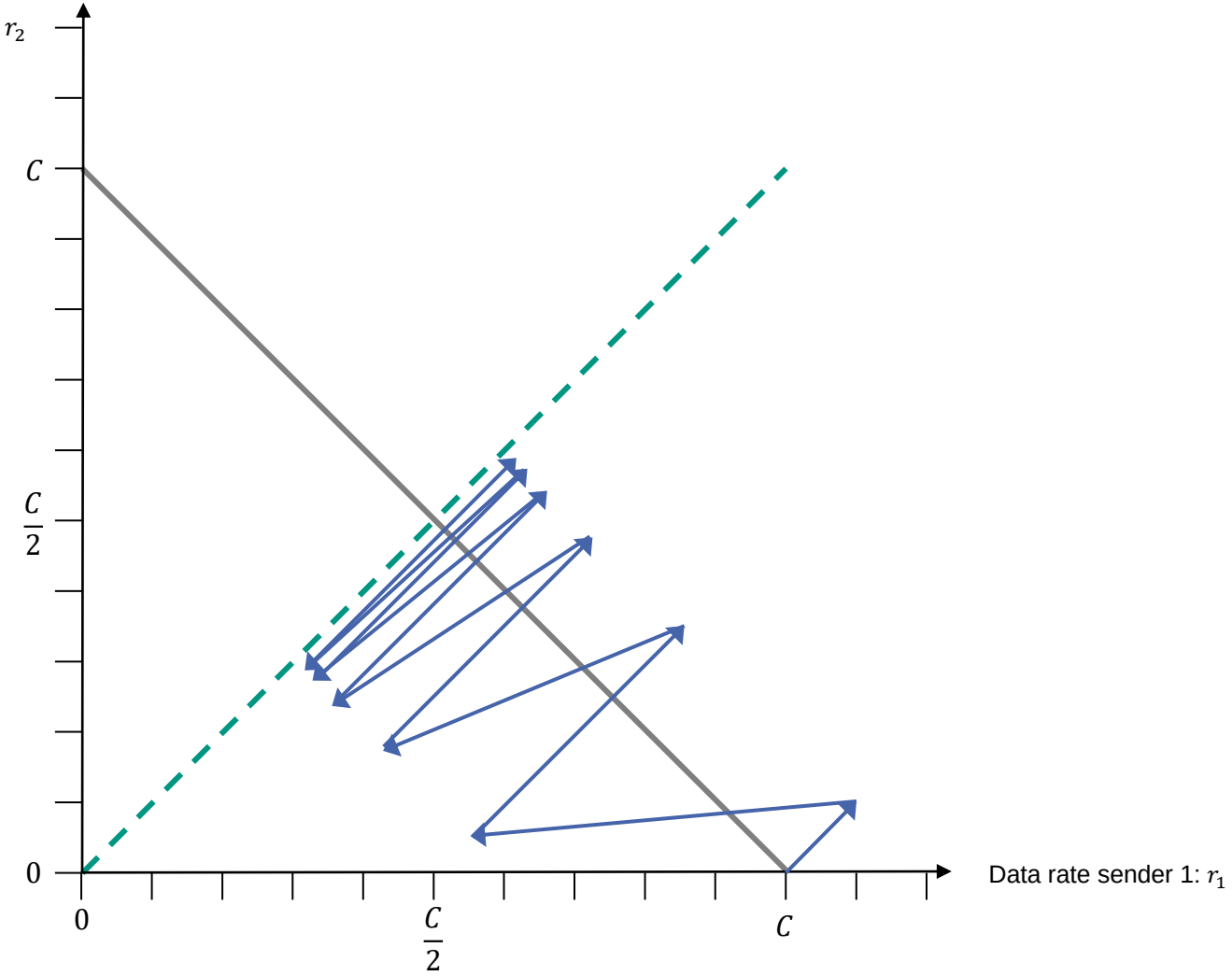
Example: AIMD

- Allocation vector diagram of two TCP connections sharing a common bottleneck link
  - AIMD parameters
    - Increase value $\alpha = 0.1$
    - Decrease factor $\beta = 0.5$
  - Assumptions
    - Connections are not application limited (i.e., always have data to be sent)
    - Connections have same RTT
    - Bottleneck buffer uses FIFO drop tail queue (no AQM)
  - Scenario
    - Sender 1 (x-axis) starts immediately and fully allocates available bottleneck link capacity
    - Sender 2 (y-axis) starts delayed
    - Packet loss at 120 % utilization

AQM: Active Queue Management

Example: AIMD

- Increase value $\alpha = 0.1$
- Decrease factor $\beta = 0.5$
- Sender 1 starts immediately
- Sender 2 starts delayed
- Packet loss at 120 %



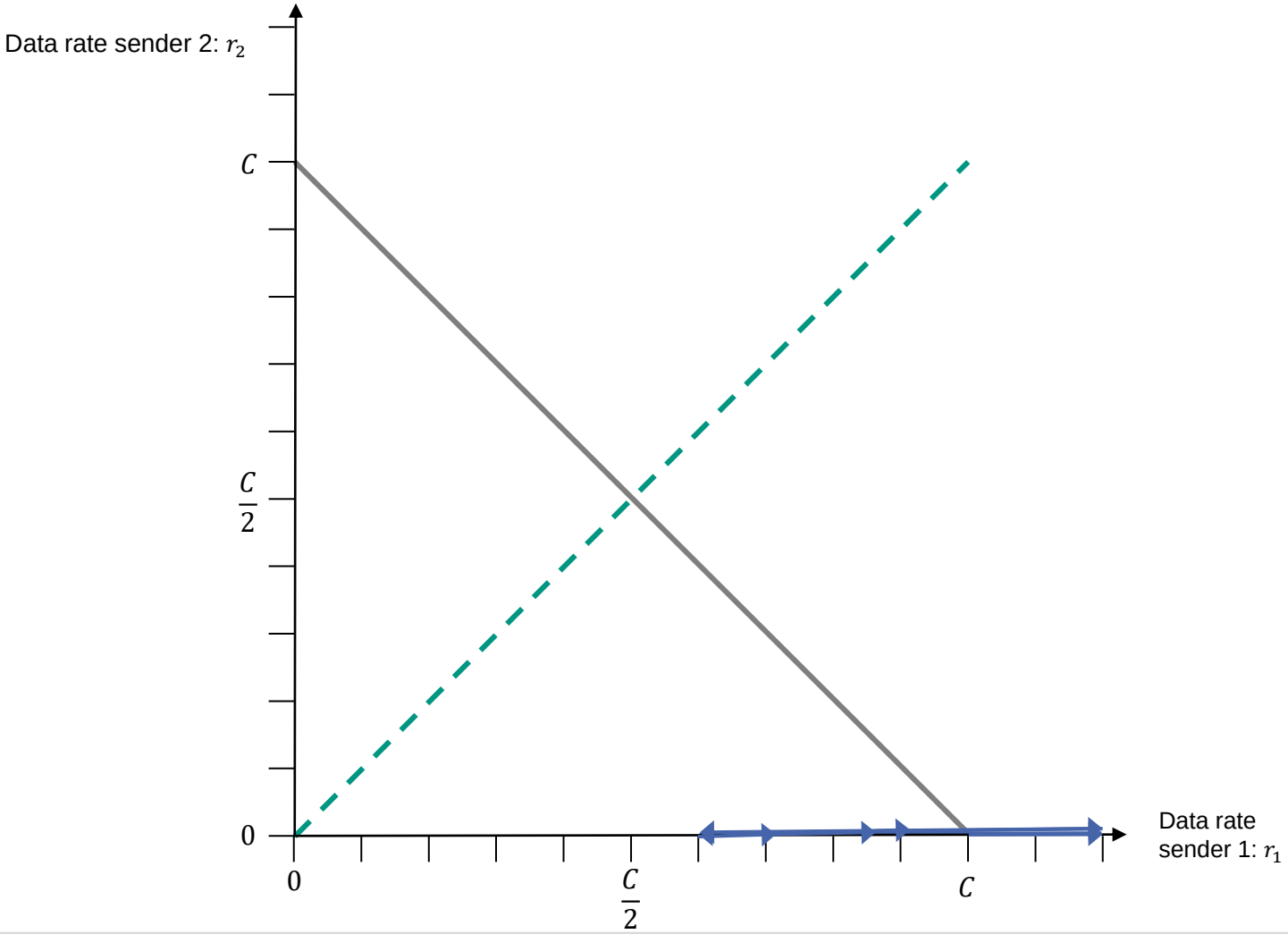
User 1	User 2	Sum
1,0000	0,0000	1,0000
1,1000	0,1000	1,2000
0,5500	0,0500	0,6000
0,6500	0,1500	0,8000
0,7500	0,2500	1,0000
0,8500	0,3500	1,2000
0,4250	0,1750	0,6000
0,5250	0,2750	0,8000
0,6250	0,3750	1,0000
0,7250	0,4750	1,2000
0,3625	0,2375	0,6000
0,4625	0,3375	0,8000
0,5625	0,4375	1,0000
0,6625	0,5375	1,2000
0,3313	0,2688	0,6000
…	…	…

Example: MIMD

- Allocation vector diagram of two TCP connections sharing a common bottleneck link
  - MIMD parameters
    - Increase factor $\alpha = 1.2$
    - Decrease factor $\beta = 0.5$
  - Assumptions
    - Connections are not application limited
    - Connections have same RTT
    - Bottleneck buffer uses a FIFO drop tail queue (no AQM)
  - Scenario
    - Sender 1 (x-axis) starts immediately and fully allocates available bottleneck link capacity
    - Sender 2 (y-axis) starts delayed
    - Packet loss at 120 % utilization

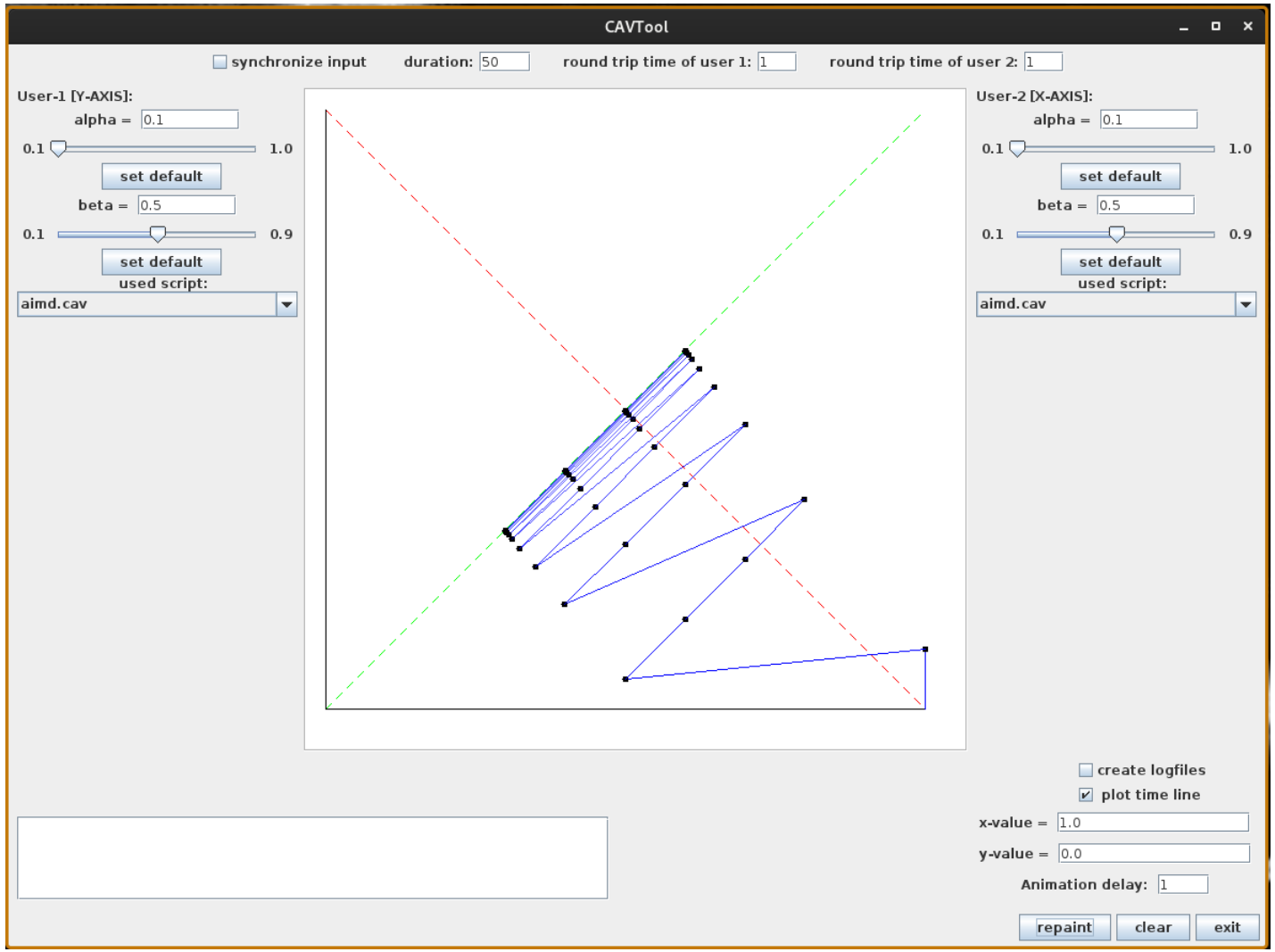
Example: MIMD

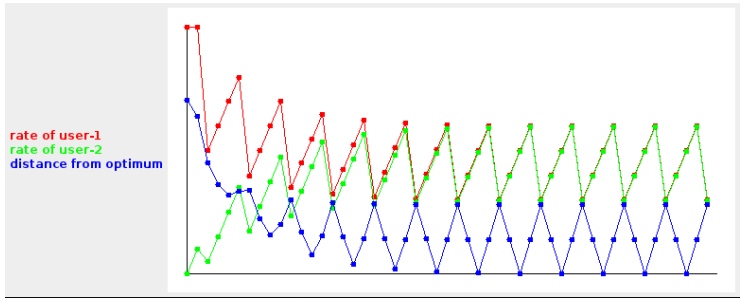
- Increase factor $\alpha = 1.2$
- Decrease factor $\beta = 0.5$
- Sender 1 starts immediately
- Sender 2 starts delayed
- Packet loss at 120 %



User 1	User 2	Sum
1,0000	0,0100	1,0100
1,2000	0,0120	1,2120
0,6000	0,0060	0,6060
0,7200	0,0072	0,7272
0,8640	0,0086	0,8726
1,0368	0,0104	1,0472
1,2442	0,0124	1,2566
0,6221	0,0062	0,6283
0,7465	0,0075	0,7540
0,8958	0,0090	0,9048
1,0750	0,0107	1,0857
1,2899	0,0129	1,3028
0,6450	0,0064	0,6514
0,7740	0,0077	0,7817
0,9288	0,0093	0,9380
1,1145	0,0111	1,1257
1,3374	0,0134	1,3508
0,6687	0,0067	0,6754
…	…	…

Demo: CAVT: Congestion Avoidance Visualization

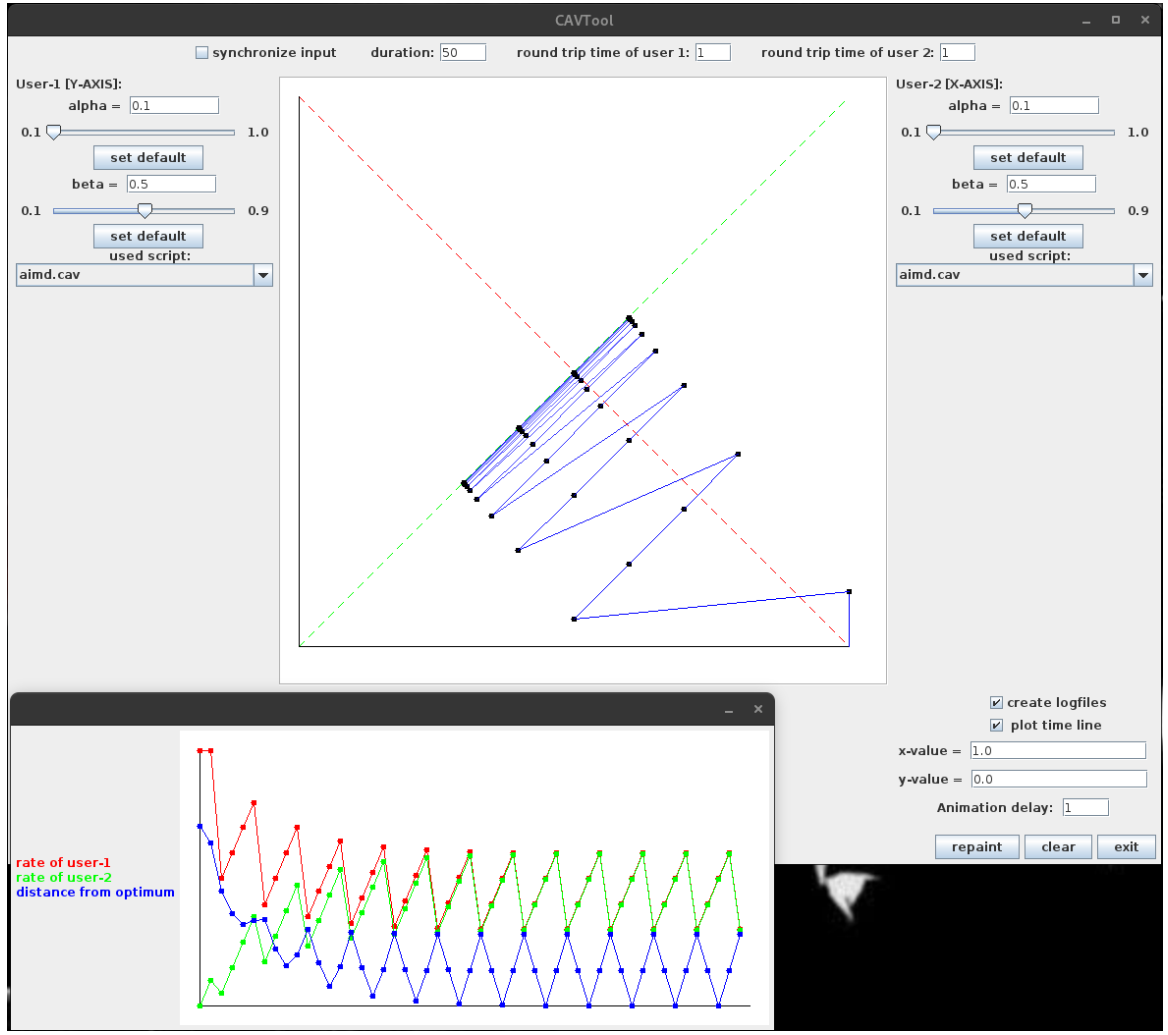




[https://folk.universitetetioslo.no/michawe/research/tools/cavt/index.html]

[WeMu03]

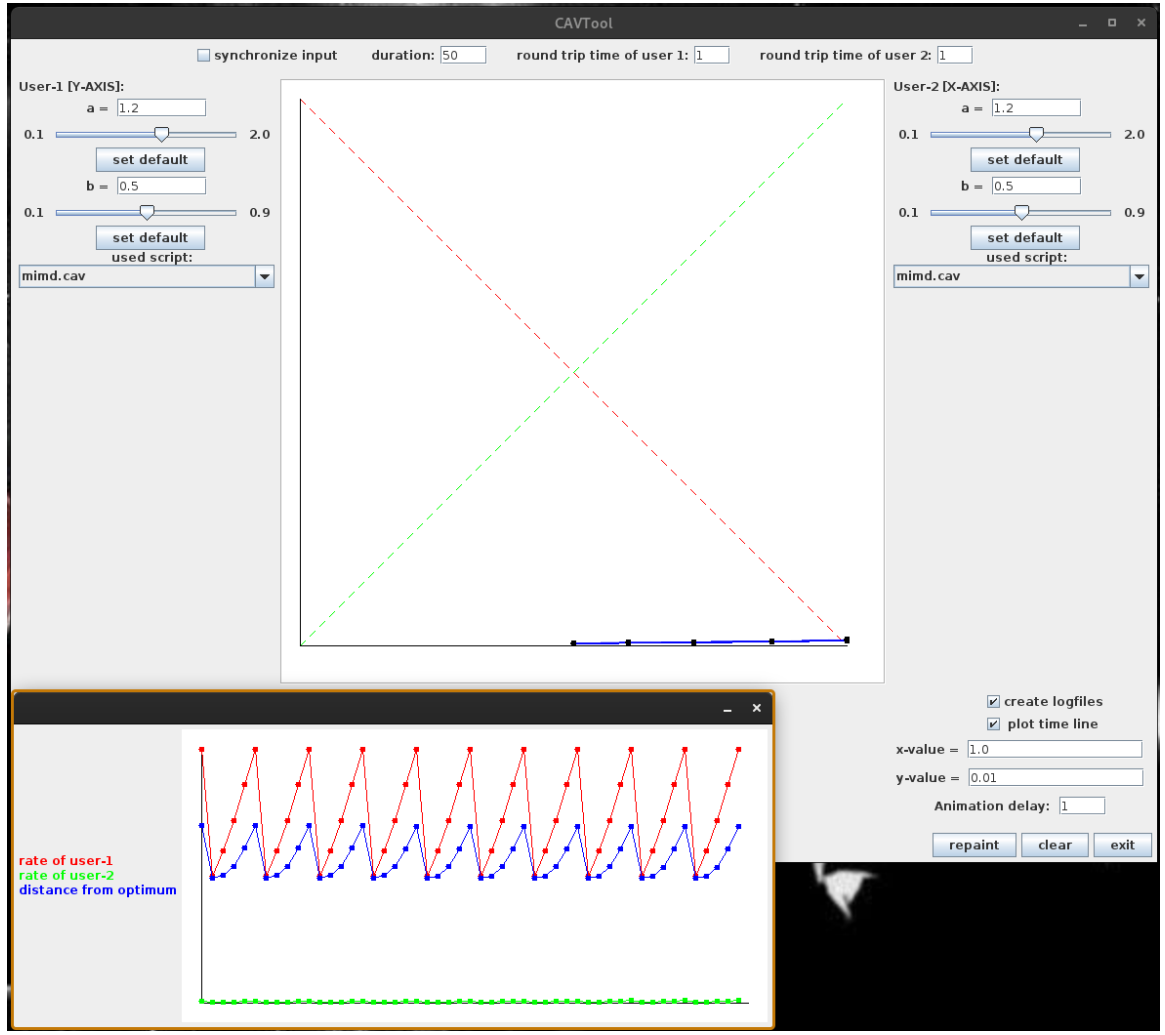
Demo: Two AIMD senders, same RTT



Converges to equal share.

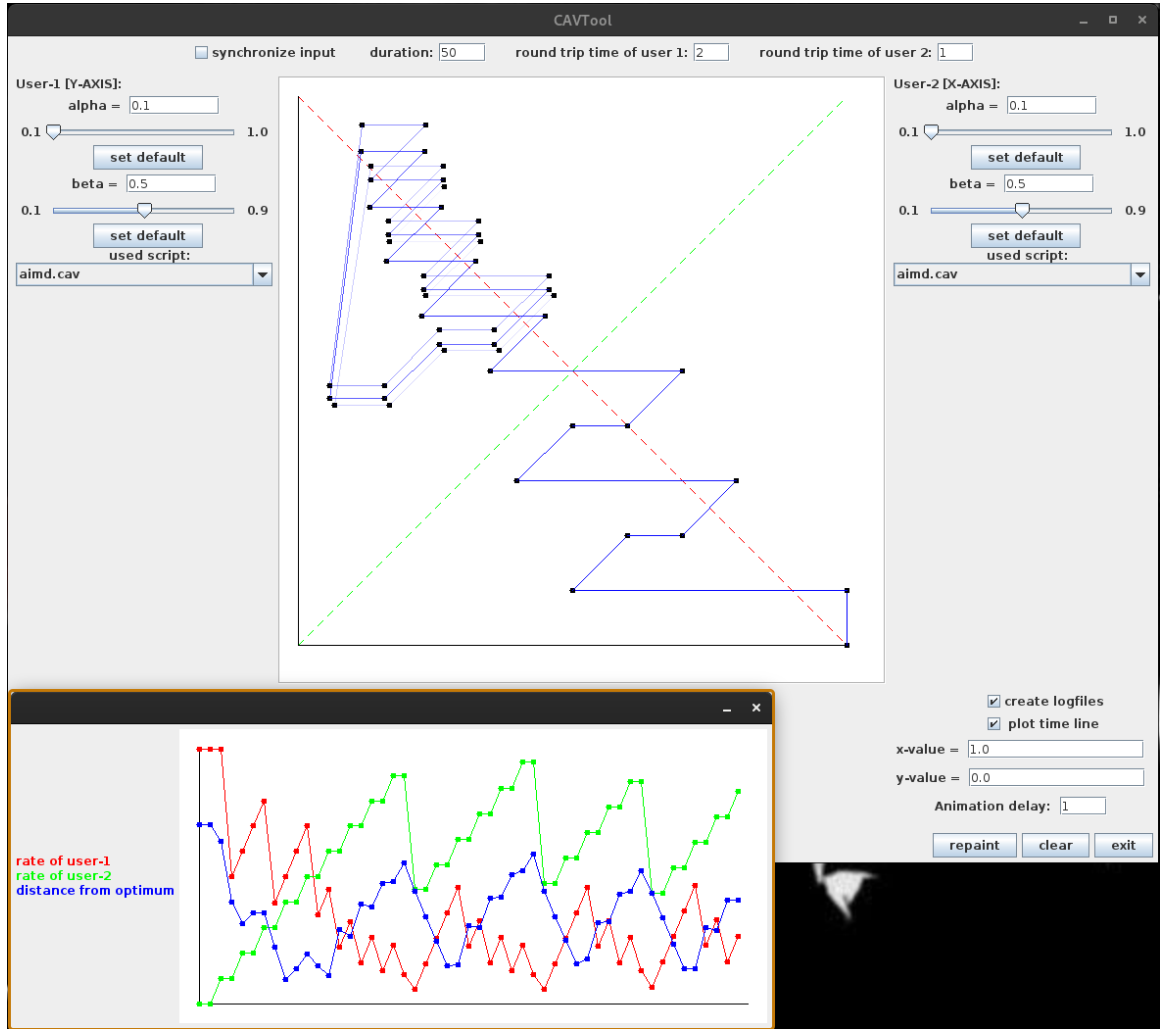
- Parameters used
  - both sender AIMD (default)
  - $\alpha = 0.1$ (default)
  - $\beta = 0.5$ (default)
  - x-value = 1.0
  - y-value = 0.0
  - duration: 50 (default)
  - RTT user 1 = 1 (default)
  - RTT user 2 = 1 (default)
  - Animation delay = 200

Demo: Two MIMD senders, same RTT



- Parameters used
  - both sender MIMD
  - $\alpha = 1.2$ (default)
  - $\beta = 0.5$ (default)
  - x-value = 1.0
  - y-value = 0.01
  - duration: 50 (default)
  - RTT user 1 = 1 (default)
  - RTT user 2 = 1 (default)
  - Animation delay = 200

Demo: Two AIMD senders, different RTT



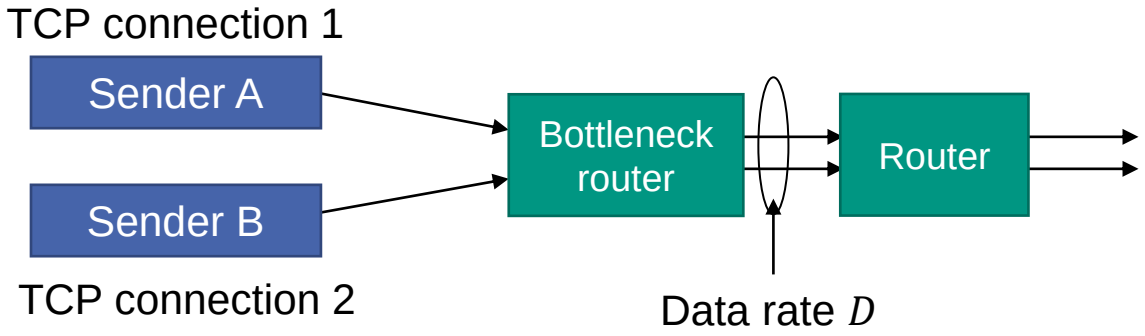
RTT-unfairness

- Parameters used
  - both sender AIMD
  - $\alpha = 0.1$ (default)
  - $\beta = 0.5$ (default)
  - x-value = 1.0
  - y-value = 0.0
  - duration: 50 (default)
  - RTT user 1 = 2
  - RTT user 2 = 1 (default)

7.6.3 TCP Fairness – Game the System

TCP Fairness

- Example



  - Two TCP connections share a bottleneck link with data rate D
    - Goal: each connection gets 50% of link capacity, i.e., can send with data rate $D/2$

- Assumptions
  - Both TCP connections have same round trip time (RTT)
  - Both TCP connections use same MSS
  - Losses of TCP segments are detected instantaneously

TCP Fairness

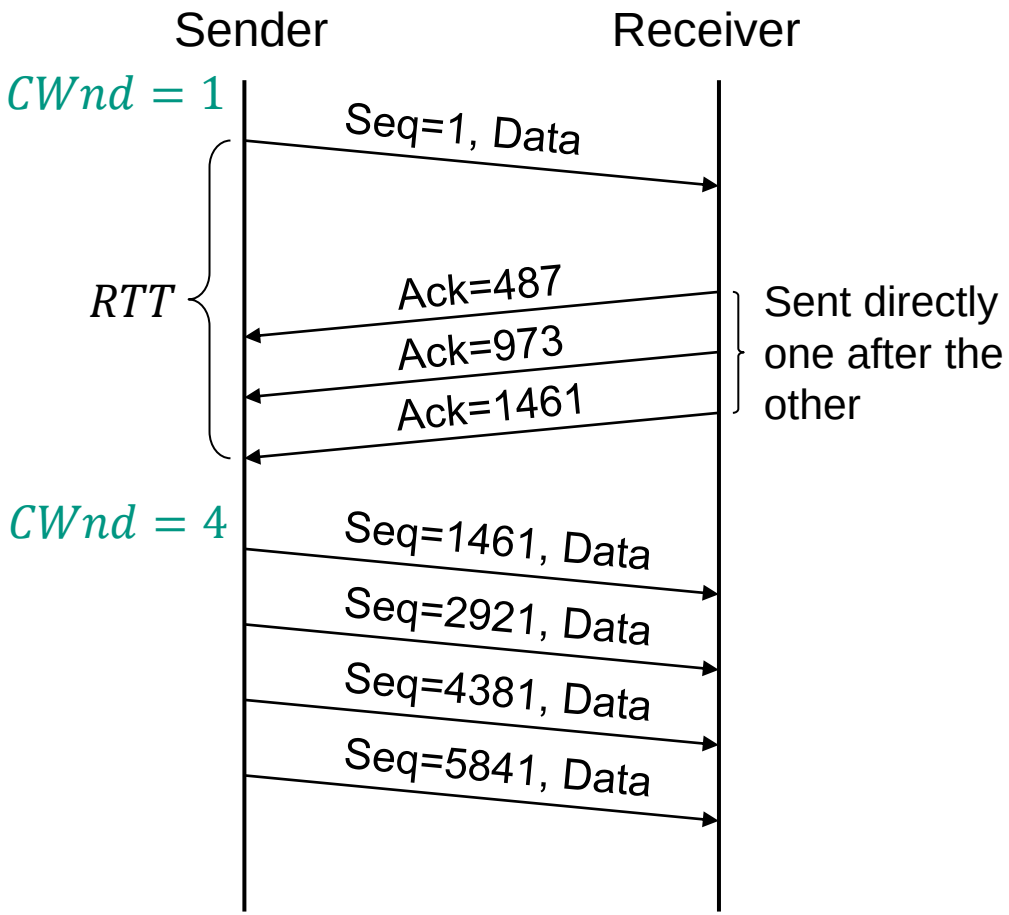
- Observation 1
  - Faire share of resource refers to *TCP connections*
- “Greedy” user
  - … opens multiple TCP connections concurrently
    - Example: multiple TCP connections for downloading a web page
      - Up to 180 concurrent connections observed with Internet Explorer
  - Link with capacity D, two users, one connection per user
    - Each user gets capacity $^{D}/_{2}$
  - Link with capacity D, two users, user 1 with a single connection, user 2 with nine connections
    - User 1 can use $\frac{1}{10}D$, user 2 can use $\frac{9}{10}D$

TCP Fairness

- Observation 2
  - *ACK generation* of receiver can influence resource allocation
- “Greedy” receiver
  1. Can send several ACKs per received segment
  2. Can send ACKs faster than it receives segments
- Goal
  - Open congestion window faster and, consequently, get more resources allocated

TCP Fairness

- Send several ACKs per received segment
- Possible due to inconsistencies of TCP specification
  - TCP sequence number counts bytes
  - Congestion window counts segments
  - Attack would be impossible if congestion window also counted bytes
- Behavior of receiver
  - Acknowledges each segment in several pieces
  - Consequently, congestion window opens faster
    - Increase per received ACK



[Sava99]

Appropriate Byte Counting

- Modification to the way TCP increases congestion window
  - Mitigates impact of delayed ACKs and lost ACKs
  - Prevents attacks from misbehaving receivers

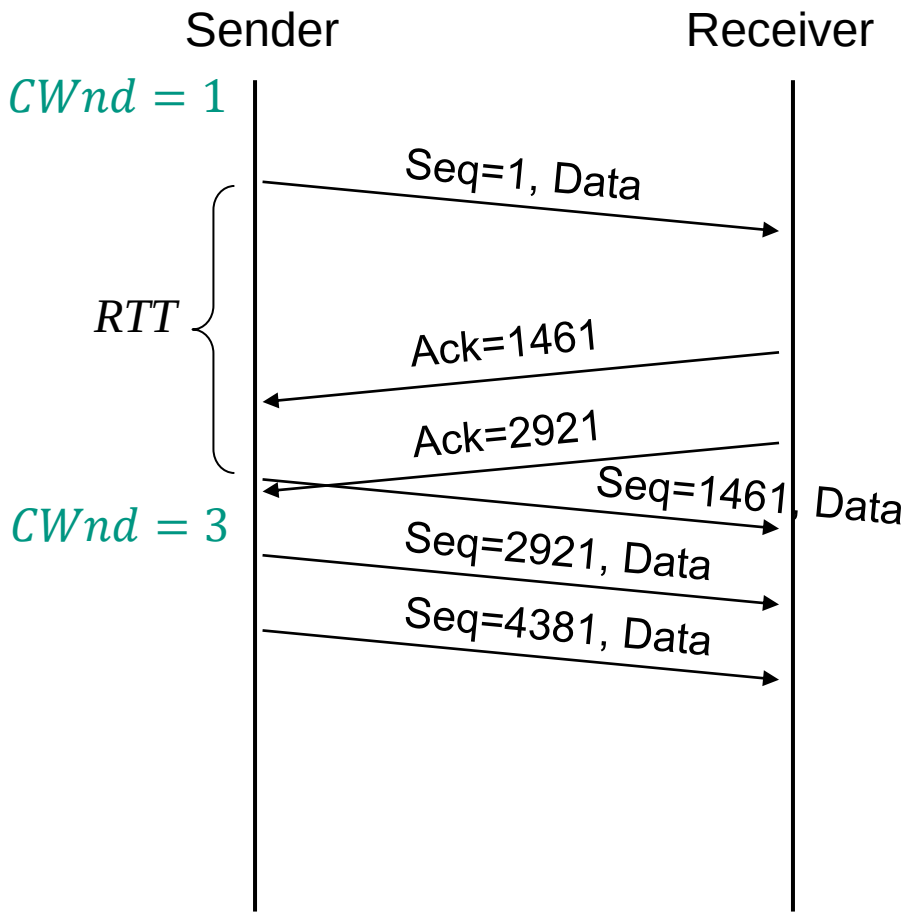
- “Traditional” implementation
  - Receipt of new ACK: $CWnd \mathrel{+}= 1/CWnd$
  - $CWnd$ is increased by constant amount per arriving acknowledgement

- Appropriate byte counting during congestion avoidance
  - $CWnd$ is increased based on the number of bytes acknowledges by received ACK
  - Count number of acknowledged bytes not MSS
    - New variable introduced: *bytes_acked*
    - *bytes_acked* += number of acknowledged bytes in ACK
    - if *bytes_acked* >= CWnd [in Bytes]:
      - *CWnd* += 1 MSS
      - *bytes_acked* -= *CWnd* [in Bytes]

[RFC3465, RFC5681]

TCP Fairness

- Send ACKs faster - before data is received
- Assumption
  - Packet losses are unlikely
  - Receiver anticipates successful reception
- Possible since ACK is not a proof for receipt of corresponding data
  - Receiver can forge ACKs
- Dangerous
  - Attack on end-to-end reliability



[Sava99]

7.7 Periodic Model and „TCP Formula“

Analysis of TCP

- Problem statement
  - Is TCP suitable for my application?
  - Which performance (e.g., data rate, delay), which behavior can I expect?
    → e.g., under certain network conditions

- Possible options
  - Measurements
  - Modelling
    - Simulations
    - Analysis

- Note: congestion control is crucial factor regarding TCP performance
  - In the following focus on TCP Reno

Variables

- For a better overview
  - RTT: Round trip time [seconds]
  - p: Loss probability of a segment
  - MSS: Maximum segment size [bit]
  - W: Value of a congestion window [MSS]
  - D: Data rate measured in bit per second [bit/s]

Periodic Model

- Goals
  - Model long-term steady state behavior of TCP
  - Evaluate achievable throughput of a TCP connection under certain network conditions
- Simple model – strong simplifications
- Non goals
  - Model of short-lived TCP connections
  - Influence of a TCP connection on other connections or the network
  - Model link utilization achieved by one or more TCP connections

[HaJa03, Math97]

Periodic Model

- Basic assumptions
  - Network has constant loss probability *p*
  - Observed TCP connection does not influence *p*

- Further simplification: periodic losses
  - For an individual connection segment losses are equally spaced

  → Link delivers $\mathrm{N} = \frac{1}{p}$ segments followed by a segment loss

[HaJa03, Math97]

Periodic Model

- Additional simplifications / model assumptions
  - Slow start is ignored
  - Congestion window increases linearly (congestion avoidance)
  - RTT is constant
  - Losses are detected using duplicate ACKs
    - No timeouts
  - Retransmissions are not modelled
    - Go-Back-N is not modelled
  - Connection only limited by CWnd
    - Flow control (receive window) is never a limiting factor
  - Always MSS sized segments are sent

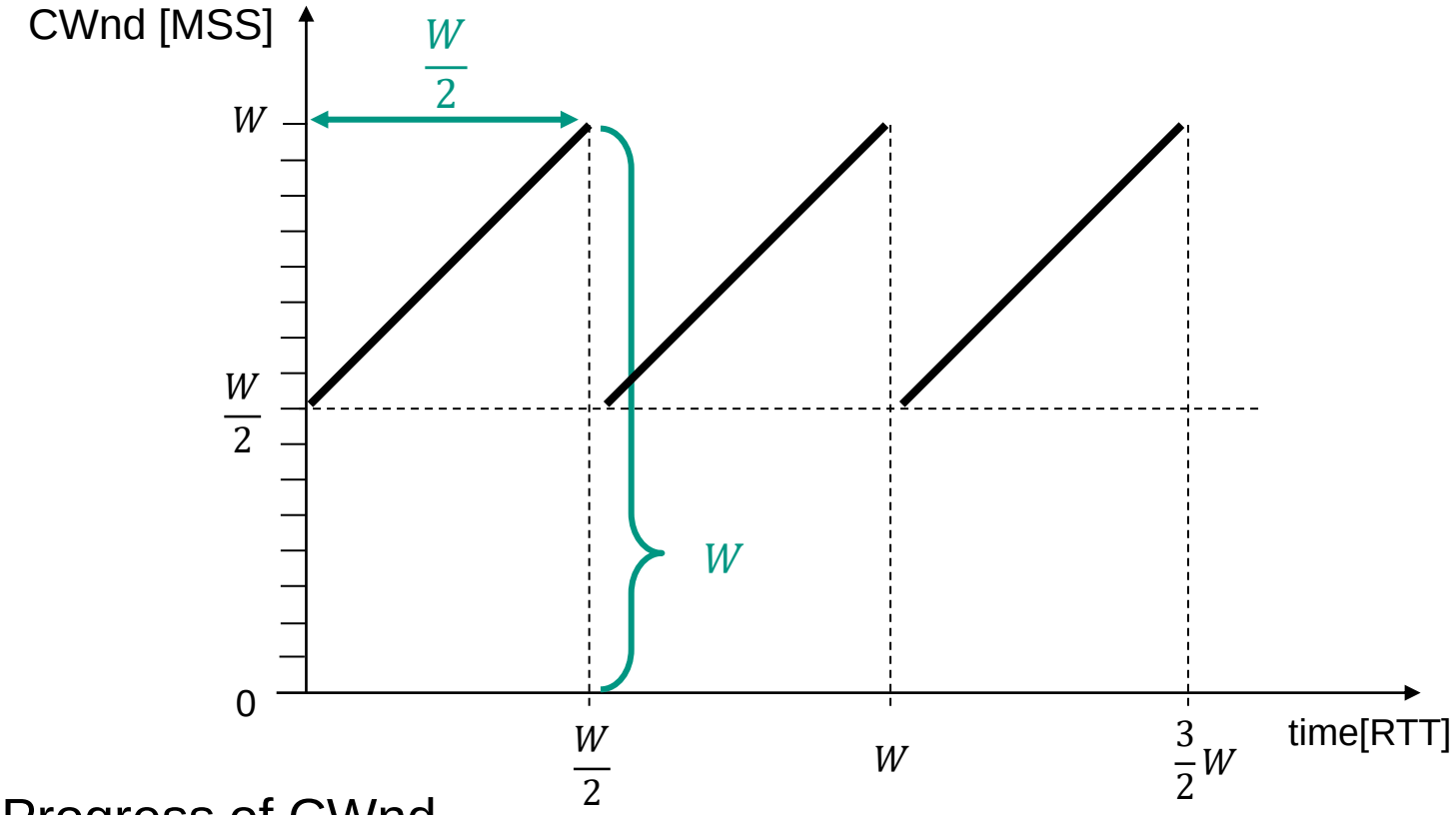
- Question: Which performance do we get?
  - Which parameters does it depend on?

… *validation of a model through simulations or measurements needed*


[HaJa03, Math97]

Periodic Model

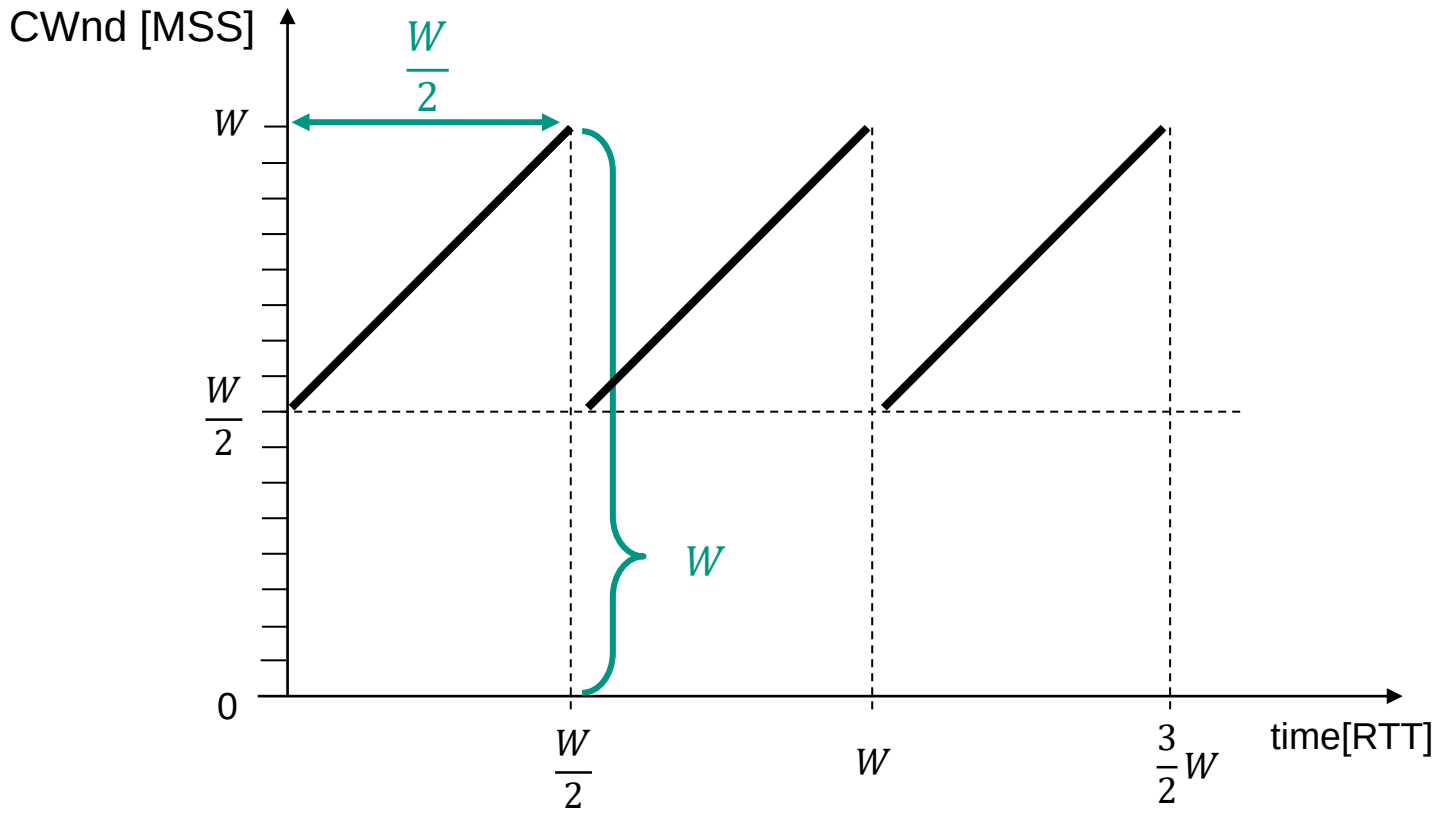
- Under given assumptions



- Progress of CWnd
  - Perfect periodic saw tooth curve
  - $\frac{W}{2} \cdot MSS \leq CWnd \leq W \cdot MSS$

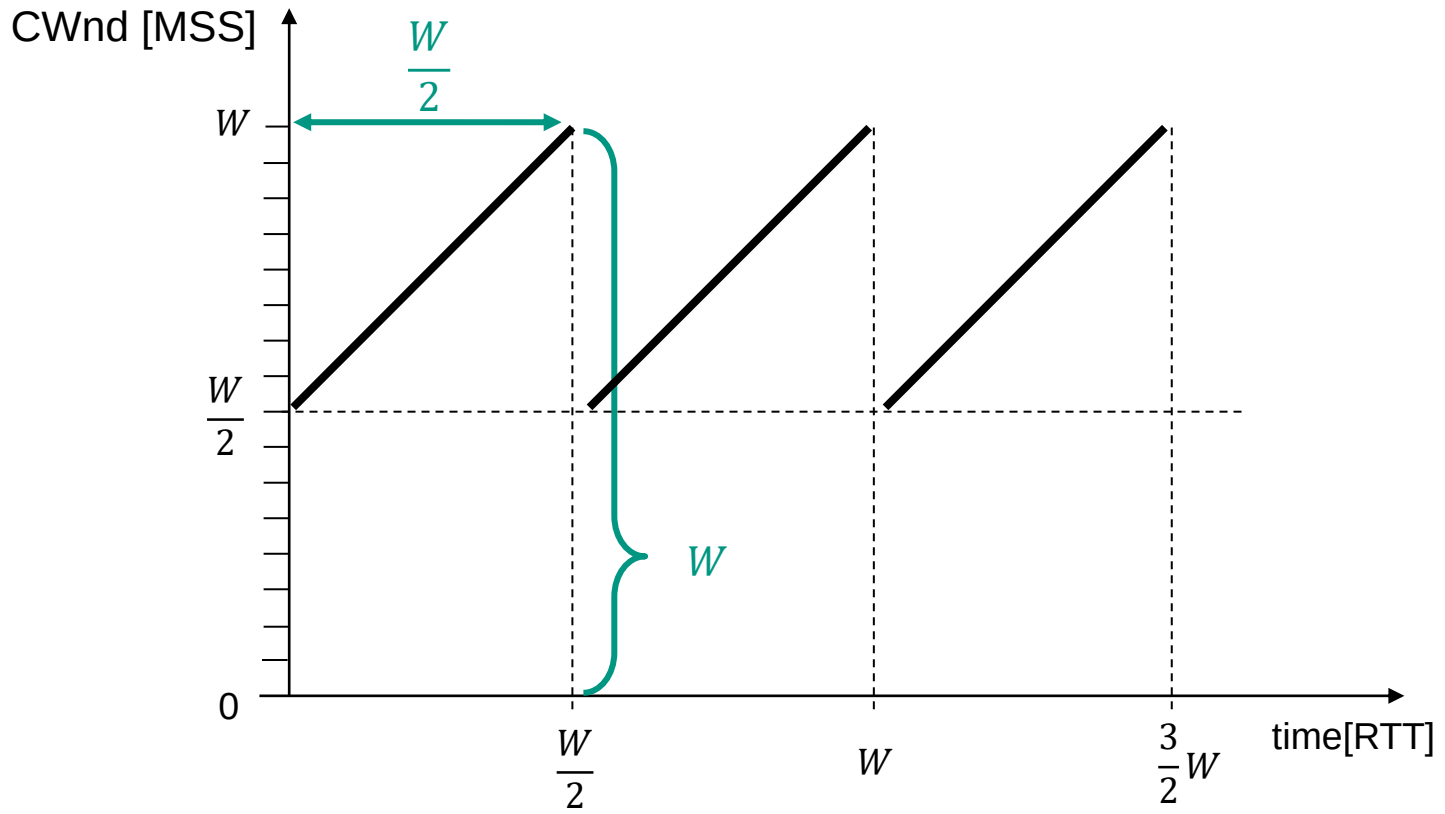
Periodic Model

- Data rate when segment loss occurs?
  - $D = \frac{W \cdot MSS}{RTT}$



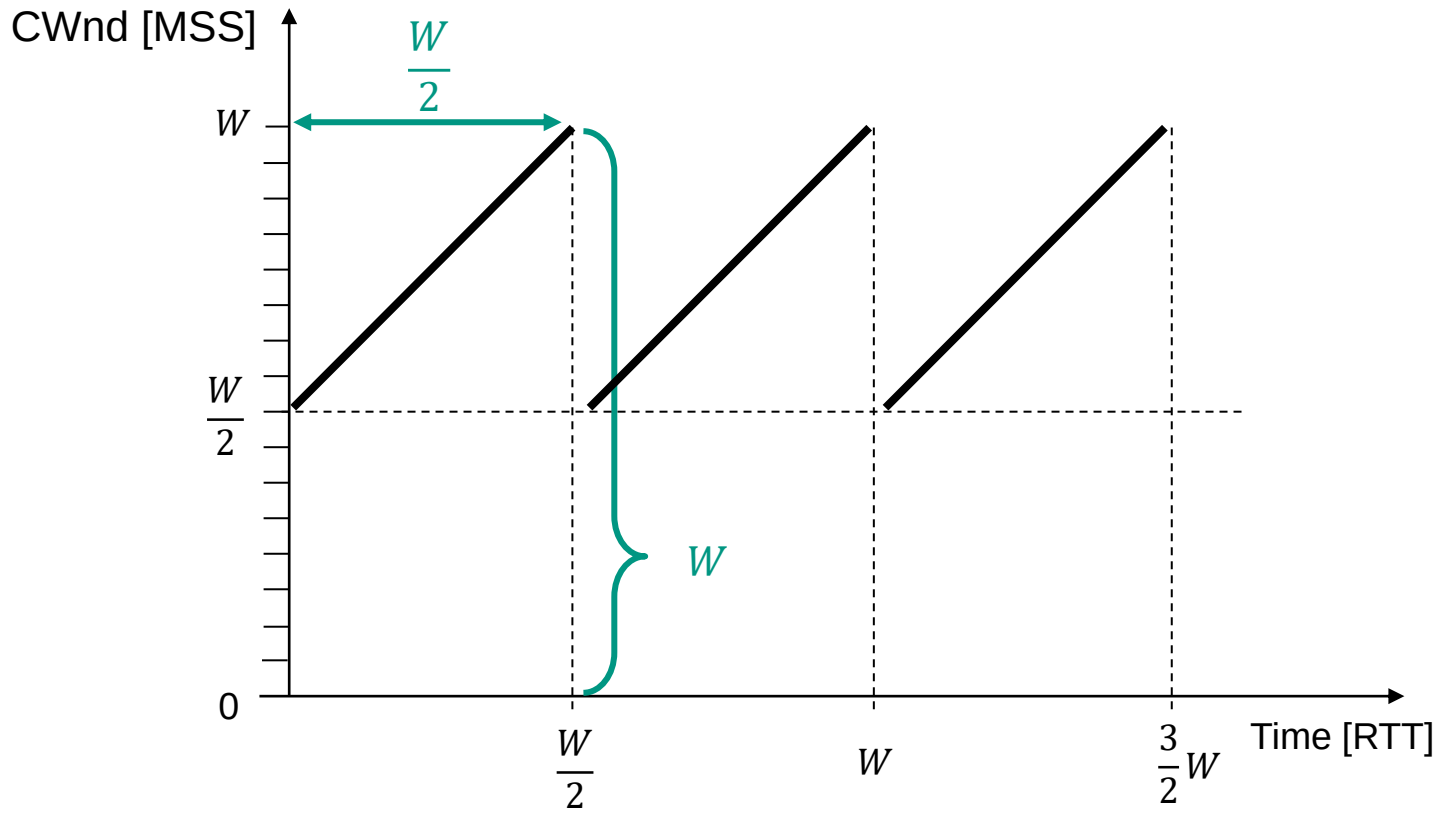
Periodic Model

- How long does it take until congestion window reaches W again?
  - $\frac{W}{2} \cdot RTT$



Periodic Model

- Average data rate of a TCP connection?
  - $D = \frac{0{,}75\, W \cdot MSS}{RTT}$



Periodic Model

- Step 1: Determine W as a function of p

Periodic Model

- Minimal value of congestion window: $\frac{W}{2}$
- Congestion window opens by one segment per RTT
  - Duration of a period: $t = \frac{W}{2} RTT$
- Number of delivered segments within one period
  - Corresponds to the area under the saw tooth curve

$$N = \left(\frac{W}{2}\right)^2 + \frac{1}{2}\left(\frac{W}{2}\right)^2 = \frac{3}{8}W^2$$

  - According to the assumptions

$$N = \frac{1}{p}$$

- Which results in

$$\boldsymbol{W = \sqrt{\frac{8}{3p}}}$$

Periodic Model

- Step 2: Determine data rate D as a function of p

Periodic Model – “TCP Formula”

- Average data rate

$$D = \frac{N \cdot MSS}{t} \qquad \text{with } N = \frac{1}{p}$$

$$D = \frac{1/p \cdot MSS}{RTT \cdot W/2} \qquad \text{with } W = \sqrt{\frac{8}{3p}}$$

- „Inverse Square-Root p Law“

$$\boldsymbol{D = \frac{1}{RTT}\sqrt{\frac{3}{2p}} \cdot MSS = 1{,}22\,\frac{1}{RTT\sqrt{p}} \cdot MSS}$$

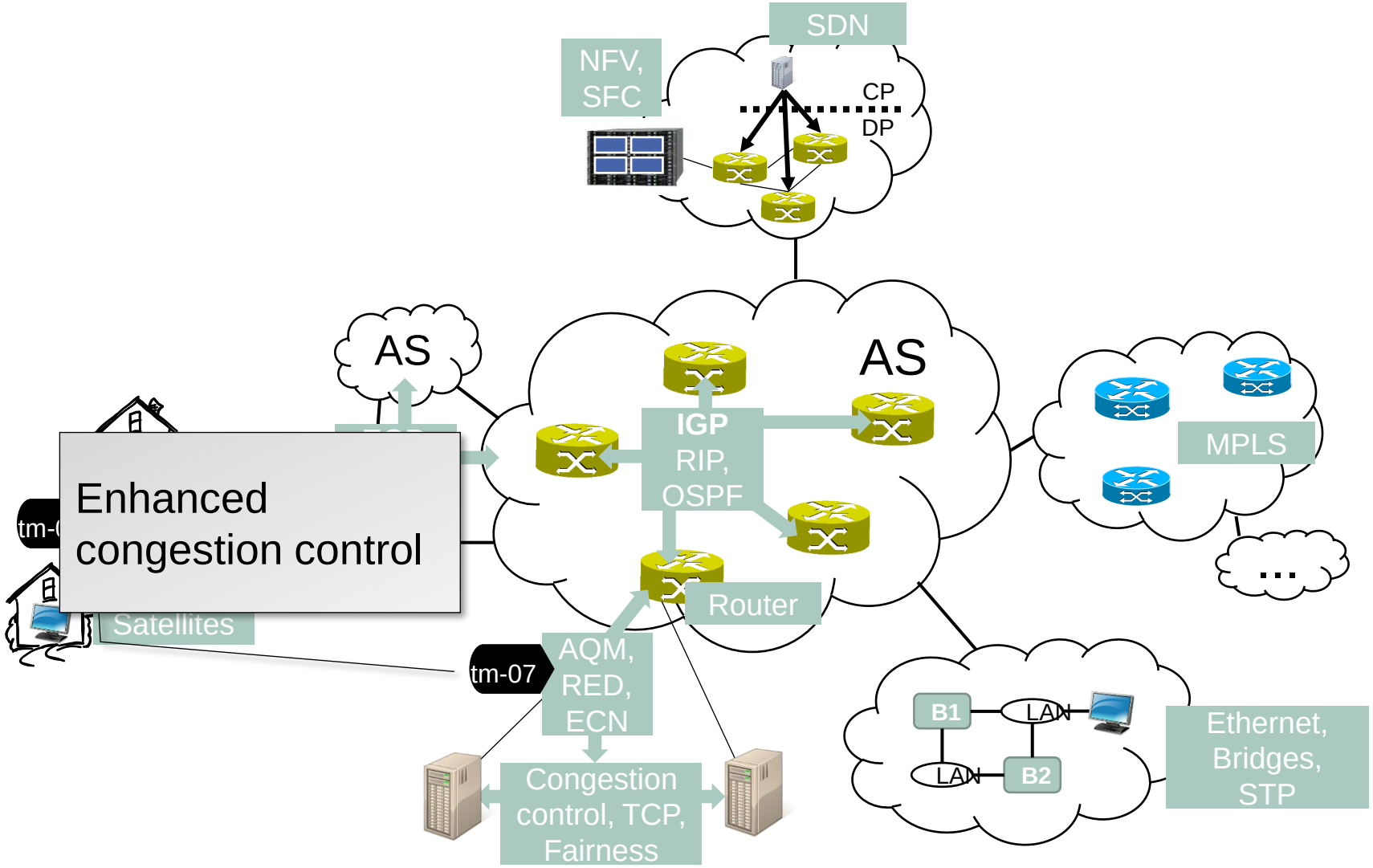
  - Fundamental relationship between
    - Loss probability p of a segment
    - Round trip time RTT, MSS and
    - Data rate D

Periodic Model – “TCP Formula”

$$D = \frac{1}{RTT}\sqrt{\frac{3}{2p}} \cdot MSS$$

- Example: estimation of average data rate
  - Round trip time $RTT = 200\ \mathrm{ms}$
  - Loss probability of each segment $p = 0{,}05$
  - Average data rate according to „Inverse Square Root p Law“

$$D = \frac{1}{0{,}2s}\sqrt{\frac{3}{2 \cdot 0{,}05} \cdot MSS} = 27{,}4\frac{\mathrm{MSS}}{s}$$

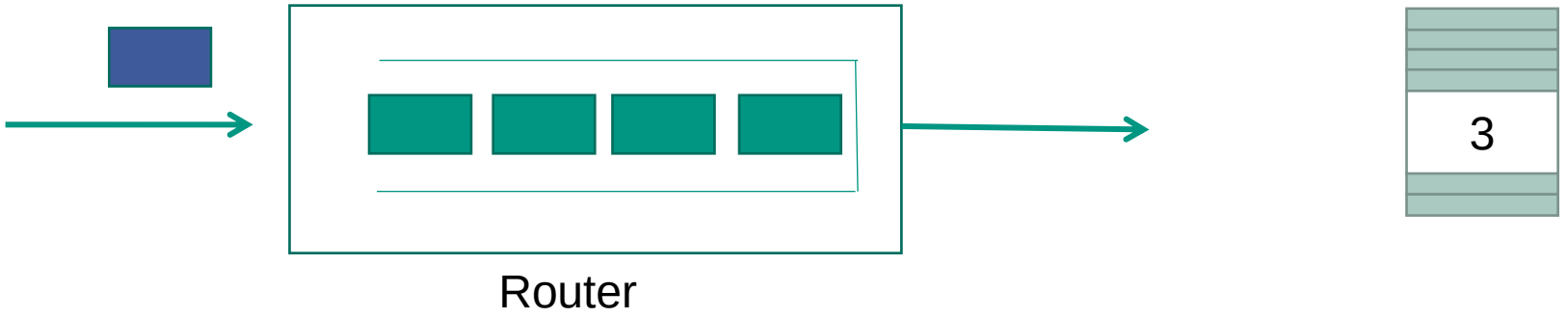


Enhanced congestion control

7.8 Active Queue Management

Simple Queue Management

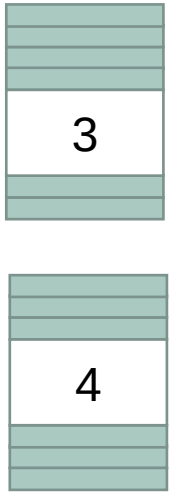
- Buffer in the router is full
  - Next segment must be dropped
    - Tail drop



- TCP detects congestion and backs off
- Problems
  - Synchronization: Segments of several TCP connections are dropped (almost) at the same time
  - Nearly full buffer cannot absorb short bursts

Active Queue Management (AQM)

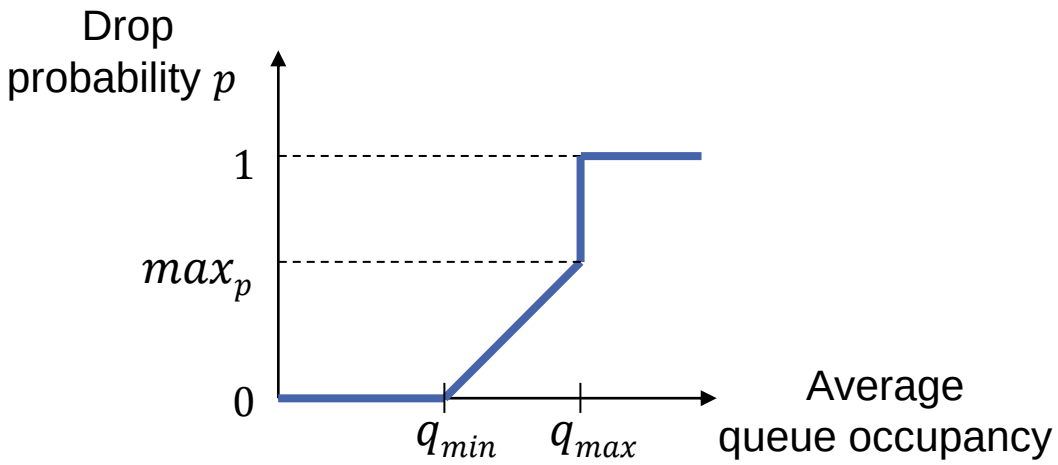
- Basic approach
  - Detect arising congestion within the network
  - Give early feedback to senders
    - Intentionally trigger implicit congestion signal: packet loss
    - Alternative: Send explicit congestion notification (ECN)
  - Routers drop (or mark) segments, before queue completely filled up
    - Randomization: random decision on which segment to be dropped
      - Prevents global synchronization of all senders
      - Ensures more fairness
    - Average queue occupancy decreases
    - Congestion can be detected early
  - Observations at the receiver on layer 4
    - Typically only a single segment is missing
- AQM algorithms
  - Random Early Detection (RED)
  - Newer algorithms: CoDel, FQ-CoDel, PIE …



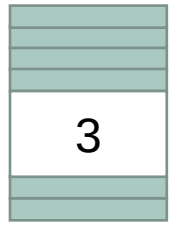
7.8.1 Random Early Detection

Random Early Detection (RED)

- Approach
  - Average queue occupancy $< q_{min}$
    - No drop of segments ($p = 0$)
  - $q_{min} \leq$ average queue occupancy $< q_{max}$
    - Probability of dropping an incoming packet is linearly increased with average queue occupancy
  - Average queue occupancy $\geq q_{max}$
    - Drop all segments ($p = 1$)



- Problem
  - How to set RED parameters: q_{min}, q_{max}, max_p?



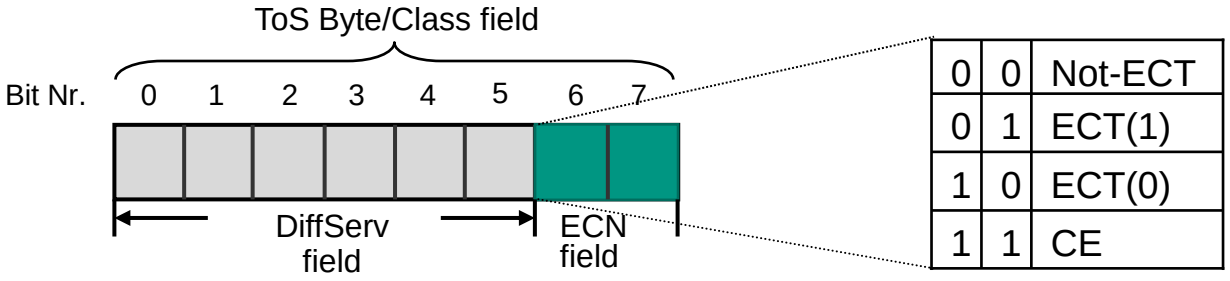
7.8.2 Explicit Congestion Notification

Explicit Congestion Notification (ECN)

- Goal
  - Send explicit congestion signal, avoid unnecessary packet drops
- Approach
  - Enable AQM to explicitly notify about congestion
  - AQM does not have to drop packets to create *implicit* congestion signal
- How to notify?
  - Mark IP datagram, but do not drop it
  - Marked IP datagram is forwarded to receiver
- How to react?
  - Marked IP datagram is delivered to receiver instance of IP
  - Information must be passed to corresponding receiver instance of TCP
    - TCP works on layer 4, not on layer 3 as IP does
  - TCP sender must be notified
    - Adjust congestion control window accordingly

ECN

- Notification in layer 3 (IP)
  - Usage of two bits in the former type-of-service field of the IP datagram



- Classic ECN
  - Sender marks packets with ECT(0)
  - Routers with ECN support:
    - Congested router overwrites ECT(0) with CE instead of dropping packet

We only show classic ECN behavior here

- ECT (1): Experimental Use
  - For different marking behavior → Currently used for L4S Experiments [RFC9331]

[RFC8311]

Classic ECN

- Notification in layer 4 (TCP)
  - New flags in TCP header
    - ECE bit (ECN echo): indicates congestion to TCP sender (carried in ACK)
      - Sent by TCP receiver if an IP datagram with CE was received
      - Sent in all subsequent ACKs until CWR is received
    - CWR bit (Congestion window reduced): indicates reception of ECE to TCP receiver
      - Sent by TCP sender after reception of TCP segment with ECE
        - ECE serves as minor congestion signal
      - Sender reduces CWnd accordingly
        - $SSThresh = \max(FlightSize/2,\ 2 * MSS)$
        - $CWnd = SSThresh$
  - Furthermore, usage of ECN needs to be negotiated during TCP connection setup
    - ECE and CWR bits are set in SYN segment
- Routers without ECN support still may drop packets!
  - Sender must react to both implicit (dropped packet) and explicit signals (marked packet)

[RFC3168]

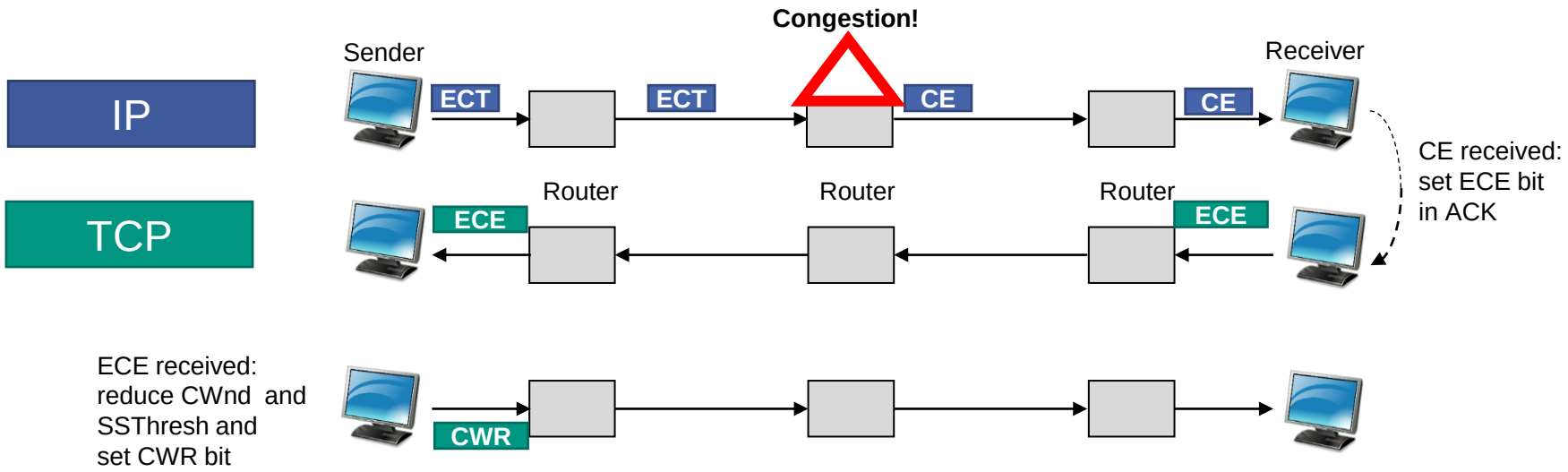
Classic ECN

- When to set ECT(0), i.e. signalling ECN capability?
  - Only set, if loss of that packet would be detected by end system and interpreted as congestion indication
  - → not set in „pure“ ACK packets (that do not carry user data)
    - TCP congestion control does not react on dropped ACK packets
  - → not set in SYN or SYN/ACK packets

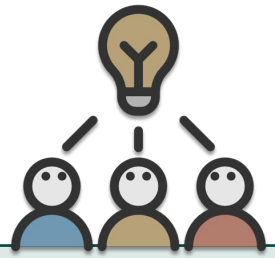
[RFC3168]

Classic ECN

- Illustration




Task 07-4
ECN



Task 07-4

Summary Chapter 7

- Introduced basic TCP congestion controls

TCP Tahoe
historical algorithm,
first congestion control

TCP Reno
Improvement of TCP Tahoe

Introduce general „mechanics“ behind Internet congestion control

  - In [Welz22] described as phase 1: Network Overload (1990´s)
    - Problem: High loss rate → Congestion Collapse
    - Solution: Introduction of first Congestion Control Algorithms
      - TCP Tahoe & TCP Reno
      - Window-based algorithms
    - More on phases 2 and 3 in later chapters

  - Additional aspects discussed
    - Self clocking, conservation of packets, fairness, active queue management, periodic model and “TCP formula”

Rather classical view on congestion control.

Challenges in todays networks are discussed in the following sections.

[Welz22]

Congestion Control Evolution

- Phase 1: Network Overload (1990´s) — tm-07
  - Problem: High loss rate → Congestion Collapse
  - Solution: Introduction of first Congestion Control Algorithms
  - TCP Tahoe & TCP Reno
- Phase 2: Inefficient Bandwidth Usage (2000´s) — tm-09
  - Increasing link capacity
  - Problem: Tahoe & Reno approach link capacity rather slowly
    → Network underloaded
  - New: TCP CUBIC & BBR
- Phase 3: Reducing Response Time (since 2010´s) — tm-09
  - Web-Browsing
    - Typically many small transmissions, interactive application
  - Flow completion time
    - Highly influences perceived quality of experience (QoE) by users
  - Problem
    - Connection finishes before actual data rate reaches link capacity
  - New: TCP Fast Open, QUIC, …

[Welz22]

PROBLEMS

1) Why are buffers needed in routers and switches?
2) Do large buffers have any disadvantages?
3) What is the difference between “goodput” and “throughput”?
4) Can it happen that throughput is high but goodput is low? Explanation!
5) Explain the tasks of flow control and congestion control.
6) What does conservation of packets mean?
7) What is the difference between “congestion control” and “congestion avoidance”?
8) How can unfairness be quantified? Why is this important?
9) Which kind of congestion control is used in the Internet?
10) Which problems did TCP Tahoe solve?
11) Which problems did TCP Reno solve?
12) Explain the periodic model? What are the basic assumptions that are applied?
13) In ECN, what is the difference between “CE” and “ECE”?
14) Describe a scenario that leads to duplicate acknowledgements.
15) Explain, how AIMD converges.
16) True or false? After a timeout of the retransmission timer, the SSThresh value is always smaller as its previous value?

LITERATURE

References

[ChJa89] D.-M. Chiu, R. Jain; Analysis of the increase and decrease algorithms for congestion avoidance in computer networks; Computer Networks and ISDN Systems, Bd. 17, Nr. 1, pp. 1–14, Juni 1989.

[HaJa03] M. Hassan, R. Jain; High Performance TCP/IP Networking – Concepts, Issues and Solutions; Prentice Hall, 2003

[JaKa88] V. Jacobson, M. J. Karels; Congestion Avoidance and Control; Proceedings of SIGCOMM'88, Stanford, CA, August 1988, pp. 314-329

[Kesh97] S. Keshav; An Engineering Approach to Computer Networking; Addison-Wesley, 1997

[Math97] M. Mathis, J. Semke, J. Mahdavi, T. Ott; The Macroscopic Behaviour of the TCP Congestion Avoidance Algorithm; ACM CCR, Vol. 27, Issue 3, July 1997, pp. 67-82

[RFC793] J. Postel (Ed.); Transmission Control Protocol; RFC 793, September 1981

References

[RFC3168] K. Ramakrishnan, S. Floyd, D. Black; The Addition of Explicit Congestion Notification (ECN) to IP; RFC 3168, September 2001

[RFC3465] M. Allman; TCP Congestion Control with Appropriate Byte Counting (ABC); RFC 3465, February 2003

[RFC5681] M. Allman, V. Paxson, E. Blanton; TCP Congestion Control; RFC 5681, September 2009

[Sava99] S. Savage, N. Cardwell, D. Wetherall, T. Anderson; TCP Congestion Control with a Misbehaving Receiver; ACM Computer Communications Review, Vol. 29, Issue 5, October 1999, pp. 71-78

[Welz22] M. Welzl, P. Teymoori, S. Islam, D. Hutchison, and S. Gjessing; Future Internet Congestion Control: The Diminishing Feedback Problem; IEEE Communications Magazine, Vol. 60, Issue 9, Sep. 2022, pp. 87-92

[WeMu03] M. Welzl, M. Mühlhäuser; CAVT: a congestion avoidance visualization tool; ACM SIGCOMM Computer Communication Review 33.3 (2003): 95-101