Congestion Control

Future Internet Communication Technologies

Prof. Dr. Panagiotis Papadimitriou

Institut für Kommunikations-Technik

Leibniz Universität Hannover

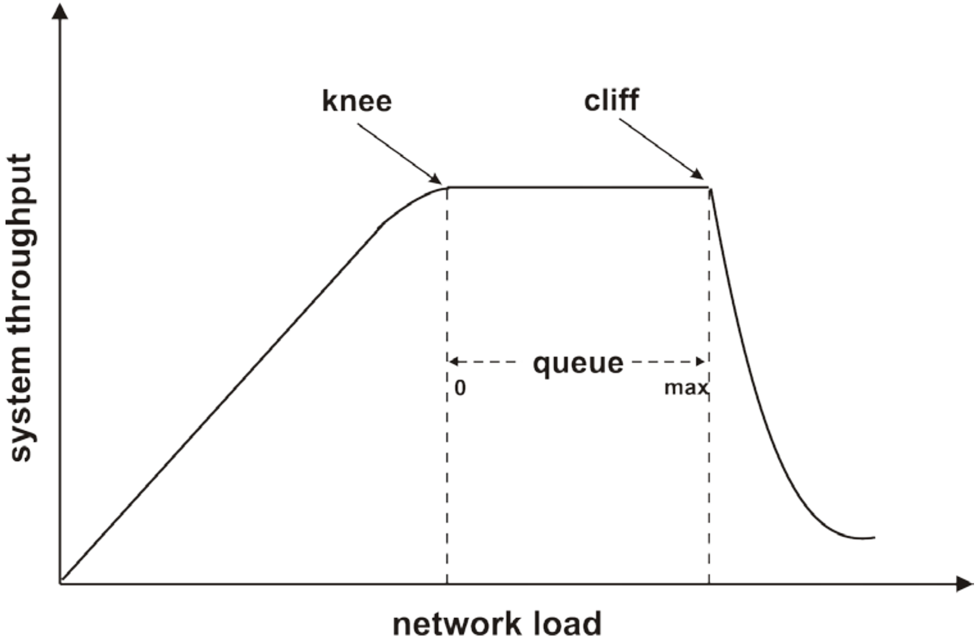
Outline

- TCP Congestion Control
- Adaptive AIMD Congestion Control
- Congestion Control for Large BDP
- Delay-based Congestion Control
- Congestion Control without Reliability
- Multi-Path Congestion Control

TCP Congestion Control

Network Congestion

- Congestion occurs when packets are dropped due to buffer overflow within any router across an Internet path



- Packet loss can also occur due to:
  - buffer overflow in the receiver
  - link errors (due to fading, interference, etc.), especially across wireless channels

Congestion Control

- Congestion control aims to achieve:
  - Prevention of congestion collapse
  - Efficient bandwidth utilization
  - Fair bandwidth sharing among competing flows

- Congestion control:
  - is carried out in the transport layer (end-to-end)
  - might be augmented by mechanisms in the network (Active Queue Management, e.g. ECN) or link layer (e.g., Automatic Repeat Request – ARQ)

Sliding Window

- Sliding window with size of 8 packets:

1	2	3	4	5	6	7	8	9	10	11	12	13

  - Packets 1-10 have been transmitted and packets 1-2 have been acknowledged

1	2	3	4	5	6	7	8	9	10	11	12	13

  - When packet 3 is acknowledged, the window slides so that packet 11 can be transmitted

TCP Sending Window

Institut für Kommunikations-Technik

- TCP uses a sending window that gives the maximum number of packets per transmission, as follows:

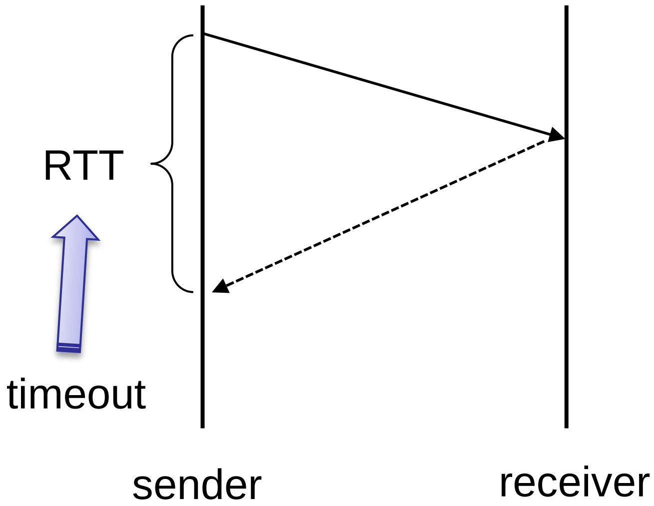
```
SendingWindow = min(CongestionWindow,
                    AdvertisedWindow)
```

`CongestionWindow`: number of packets in-flight (sent but not yet acknowledged)

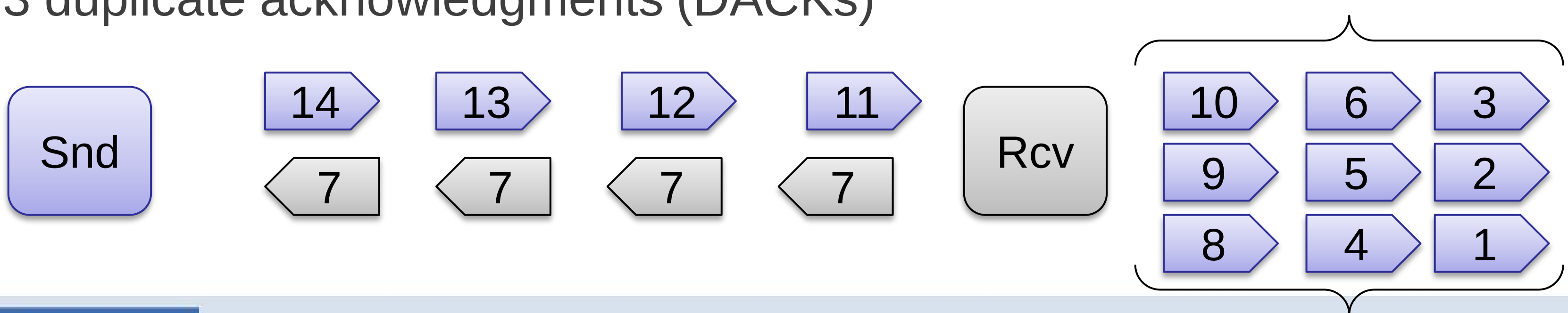
`AdvertisedWindow`: free space at receiver's buffer

Packet Loss Detection

- Packet loss is typically detected by:
  - timeout (estimated RTT, Karn’s algorithm)



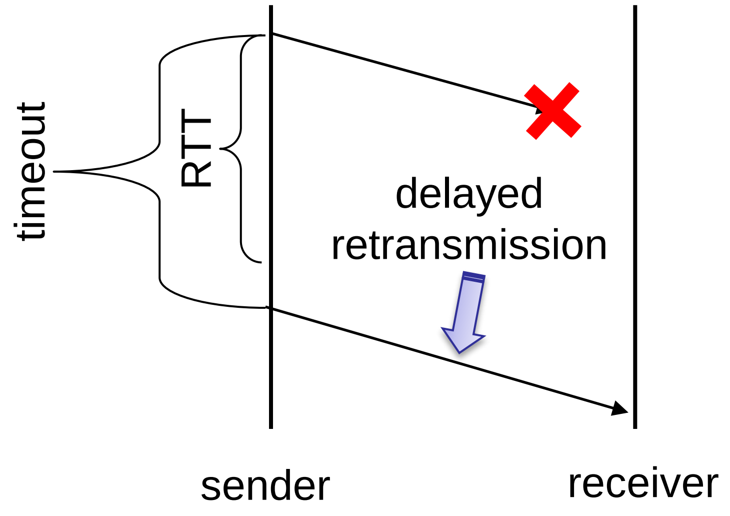
  - 3 duplicate acknowledgments (DACKs)

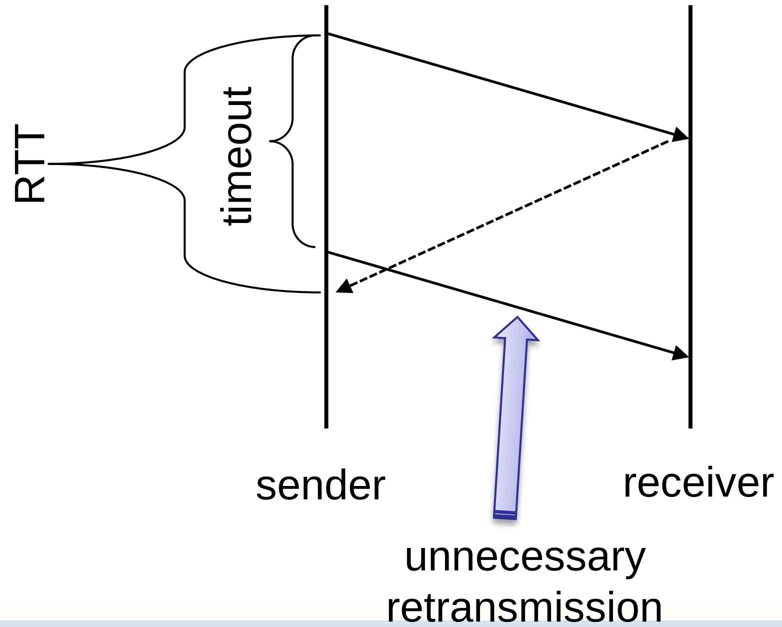


TCP Timeout

- TCP timeout requires accurate RTT estimation:
  - Not easy, since RTT varies

- Effects of inaccurate timeout adjustment:
  - Longer than RTT:
    - Slow reaction to packet loss
  - Shorter than RTT:
    - Unnecessary retransmissions





RTT Estimation

- TCP estimates RTT, as:
  - `EstimatedRTT=(1-α)*EstimatedRTT+α*SampleRTT`
  - $0.1 \leq \alpha \leq 0.2$ (typically, $\alpha = 0.125$)

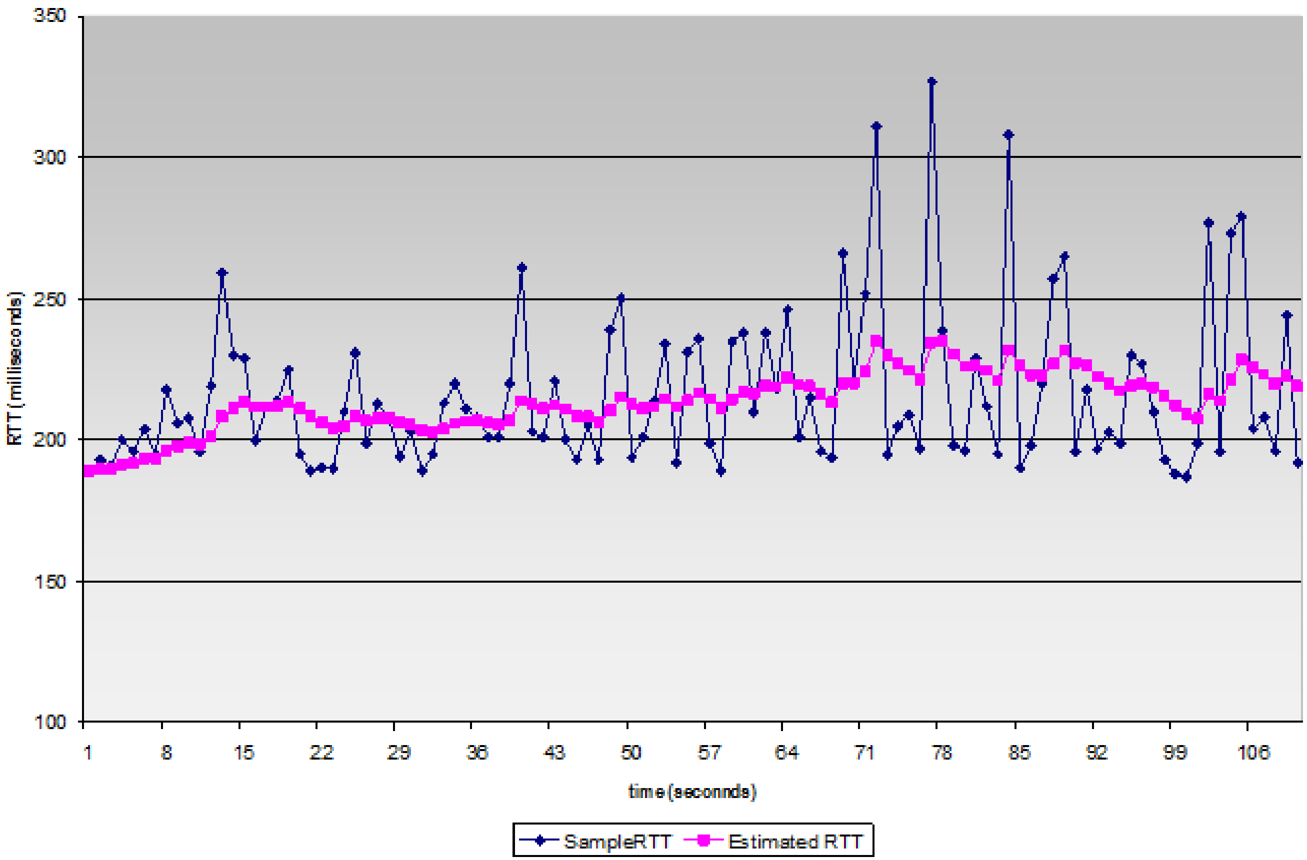


Figure from Computer Networking, A Top-Down Approach

Timeout Adjustment

- Retransmission timeout (RTO) is adjusted, as:
  - `RTO = EstimatedRTT + 4 * DevRTT`
- `DevRTT` represents how much `SampleRTT` deviates from `EstimatedRTT`:
  - `DevRTT = (1-β) * DevRTT + β * |SampleRTT-EstimatedRTT|`
  - typically, `β = 0.25`

TCP Congestion Control

Institut für Kommunikations-Technik

- TCP provides the following mechanisms for congestion control:
  - Slow Start
  - Congestion Avoidance (Additive Increase Multiplicative Decrease - AIMD)
  - Fast Recovery
  - Fast Retransmit

	Old Tahoe	Tahoe	Reno/NewReno
Cong. Avoidance	X	X	X
Slow Start	X	X	X
Fast Recovery		X	X
Fast Retransmit			X

Congestion Avoidance with AIMD

- TCP carries out congestion control based on the AIMD (Additive Increase Multiplicative Decrease) algorithm.

- AIMD (*α*, *β*) adjusts the congestion window *w* as follows:

when a new ACK is received: $w \leftarrow w + \dfrac{a}{w}$

when packet loss is detected:

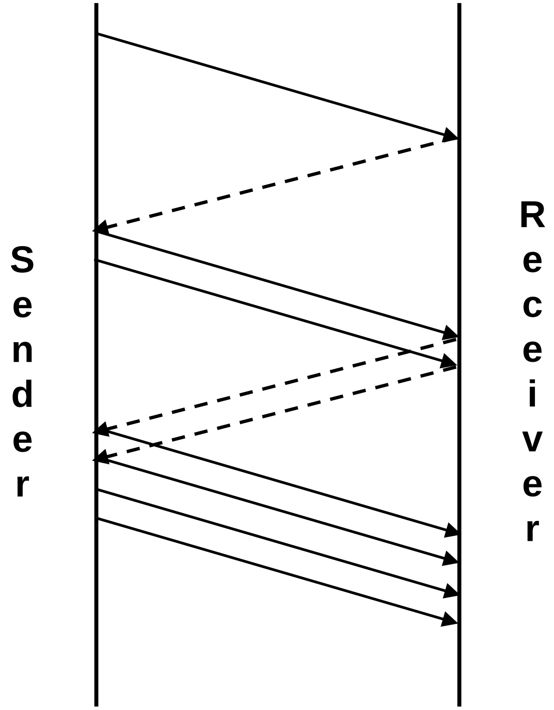
  if timeout: $w \leftarrow 1$ // Tahoe, Reno, NewReno

  if 3 DACKS: $w \leftarrow w - \beta w$ // Reno, NewReno (Fast Recovery)

- TCP uses AIMD (1, 0.5)

Slow Start

- Additive increase is very slow in exploring the available bandwidth
- Slow Start increases the congestion window exponentially till a threshold (*ssthresh*) has been reached:



Slow Start and Congestion Avoidance

- Slow Start is followed by the Congestion Avoidance phase:

when `a new ACK is received:`

if `(w < ssthresh):` $w \leftarrow w + a$ // slow-start

else`:` $w \leftarrow w + \frac{a}{w}$ // congestion avoidance

when `packet loss is detected:`

$ssthresh \leftarrow w - \beta w$

if `timeout:` $w \leftarrow 1$

if `3 DACKS:` $w \leftarrow w - \beta w$ // Fast Recovery

Initial Congestion Control Parameters

- TCP typically uses the following initial congestion window and *ssthresh* adjustments (they may vary depending on TCP version/implementation):

$$\texttt{congestion\_window} = 2\ \text{MSS}$$

$$\texttt{ssthresh} = 64\ \text{KB}$$

MSS: Maximum Segment Size

TCP Throughput Limitation

- TCP throughput can be expressed in terms of packet loss rate (p), MSS and RTT as:

$$T = \frac{1.22 \times MSS}{RTT\sqrt{p}}$$

- For example: MSS: 1500 bytes
  RTT: 100 ms
  Link capacity: 10 Gbps

  To saturate the link, packet loss should be 2×10^{-10}.

TCP Performance Metrics

- Throughput, defined as:

$$Throughput = \frac{Data}{Connection\_Time}$$

- Goodput, defined as:

$$Goodput = \frac{Original\_Data}{Connection\_Time}$$

where `Original_Data` is the number of bytes delivered to the receiver excluding retransmitted packets and overhead

TCP Fairness Index

- Fairness between TCP flows is given by the Fairness Index, which is defined as:

$$Fairness\_Index = \frac{\left(\sum_{i=1}^{n} Throughput_i\right)^2}{n\left(\sum_{i=1}^{n} Throughput_i^{\,2}\right)} \in (0,1]$$

where:

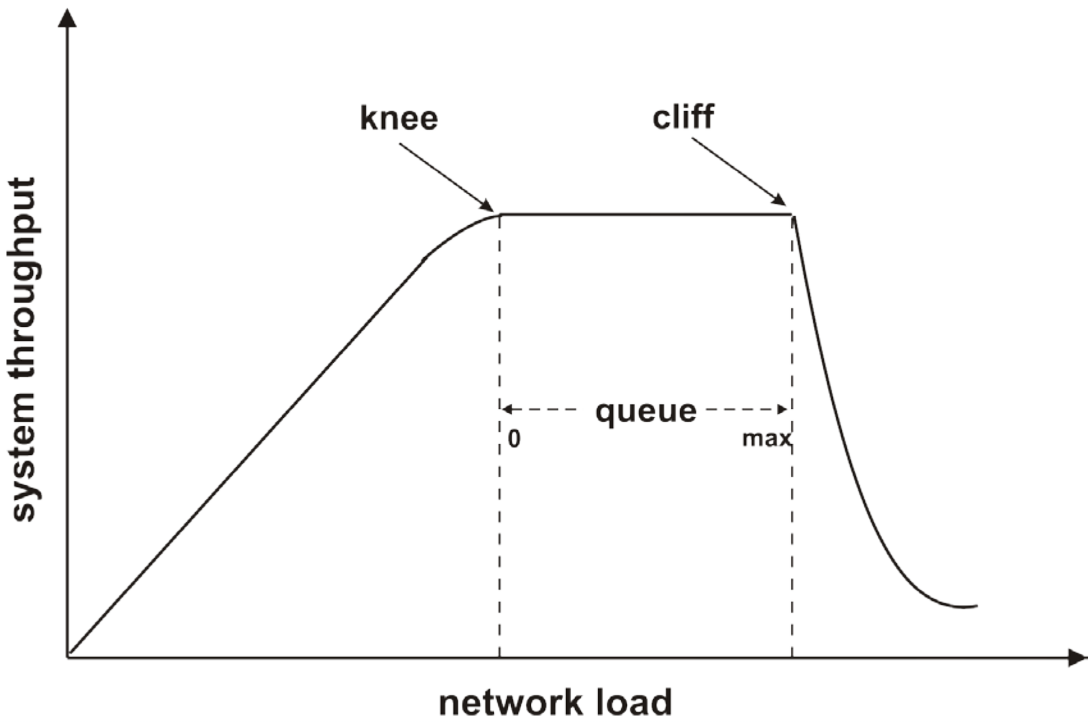
- `Throughput`$_i$: throughput of the i^{th} flow
- n: number of flows

Adaptive AIMD Congestion Control

Optimal System Operation

- When the system operates between the knee and the cliff:
  - Bandwidth is fully utilized
  - System and flow throughput is stable

- Requirements for AIMD protocols (e.g., TCP):
  - Dynamic adjustment of parameter β (from cliff to knee)
  - Proper adjustment of parameter α to maintain TCP friendliness



Optimal Adjustment of Parameter β

- For a downward adjustment from cliff to knee:
  - *β* should be adjusted as:

$$\beta = \frac{1}{1+\frac{\text{bufSize}}{d \times B}} = \frac{1}{1+\frac{\text{bufSize}}{\text{BDP}}}$$

- bufSize = BDP ⇨ β = 0.5 (typical adjustment)

  ⮡ bandwidth-delay product

- For large BDP: bufSize << BDP ⇨ β > 0.5

B: bandwidth

d: delay

bufSize: buffer size

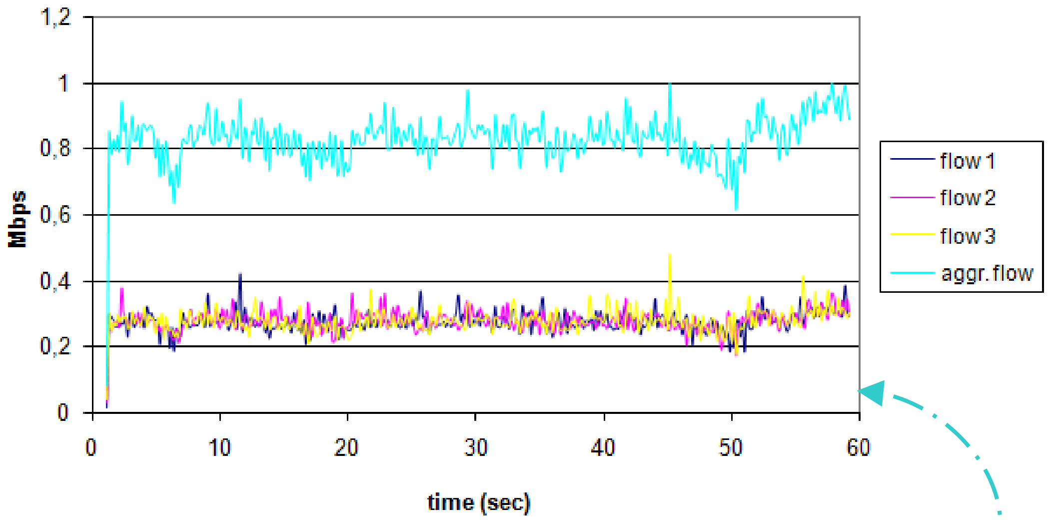
Adjusting β Using RTT

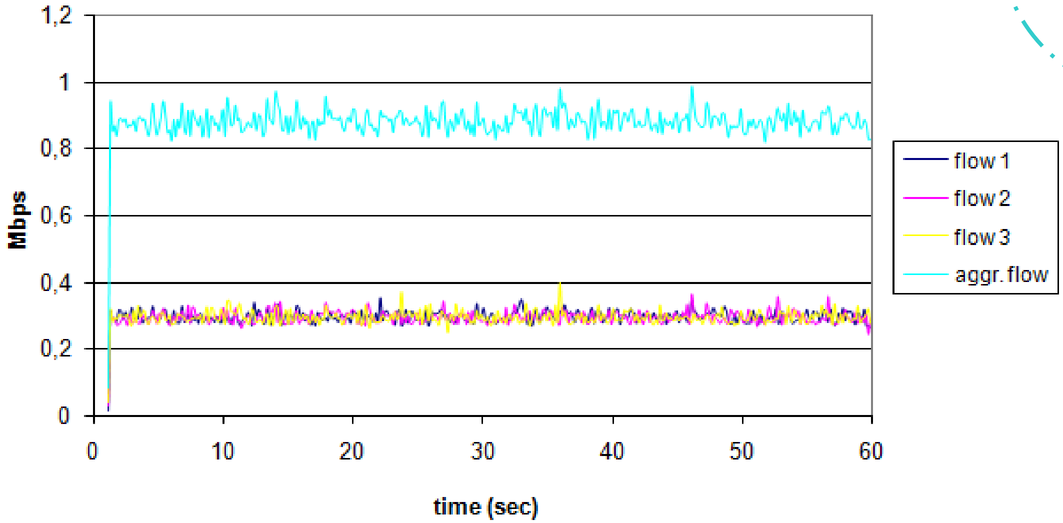
- AIMD protocol needs to monitor RTT and RTT_{min}
- For a downward adjustment from cliff to knee, *β* should be adjusted as:

$$\left.\begin{aligned}\beta &= \frac{RTT_{min}}{RTT} \\ \alpha &= \frac{4(1-\beta^2)}{3}\end{aligned}\right\} \text{AIMD}(\alpha, \beta)$$

TCP-friendliness equation

Performance Gains with Adaptive AIMD





AIMD Throughput

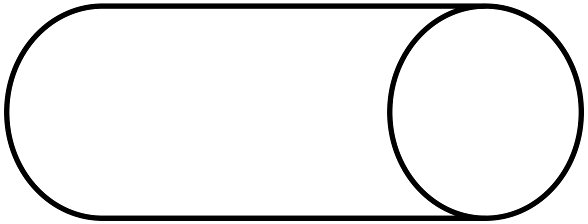
Adaptive AIMD Throughput

3 flows, B = 1 Mbps

Congestion Control
for Large Bandwidth-Delay Products

Bandwidth Delay Product

- Bandwidth Delay Product (BDP) is the product of link capacity and end-to-end delay.

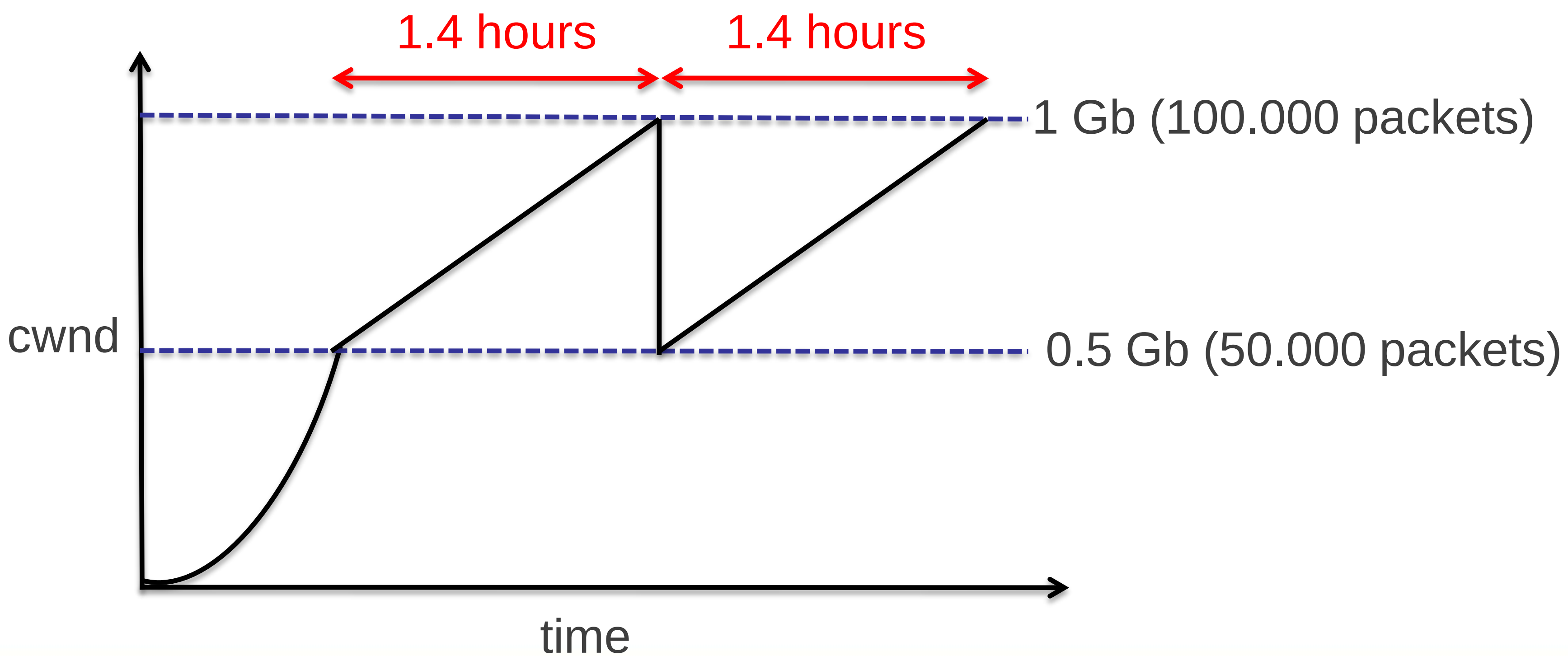


BDP = 10 Gbps x 100 ms = 1 Gb

TCP and Large Bandwidth Delay Products

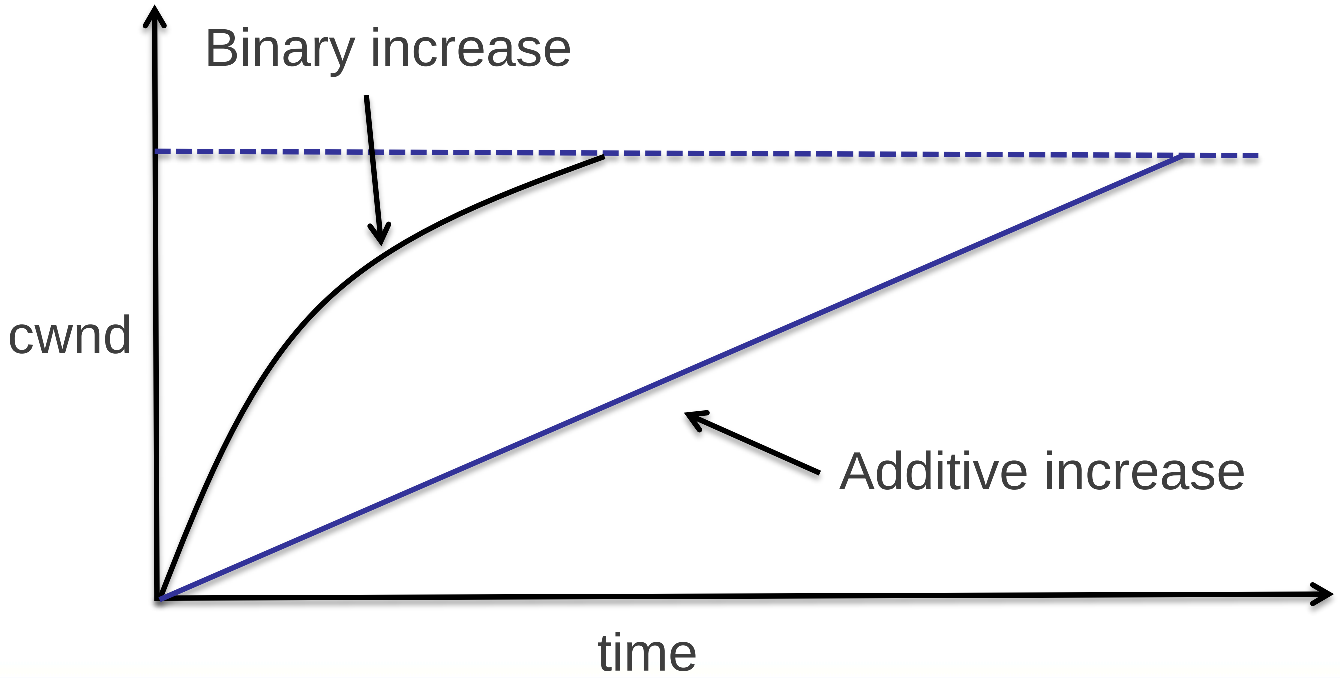
- Additive increase is very slow for large BDPs
- Halving *cwnd* is very aggressive for large BDPs

- Example: BDP = 1 Gb, packet size = 1250 Bytes



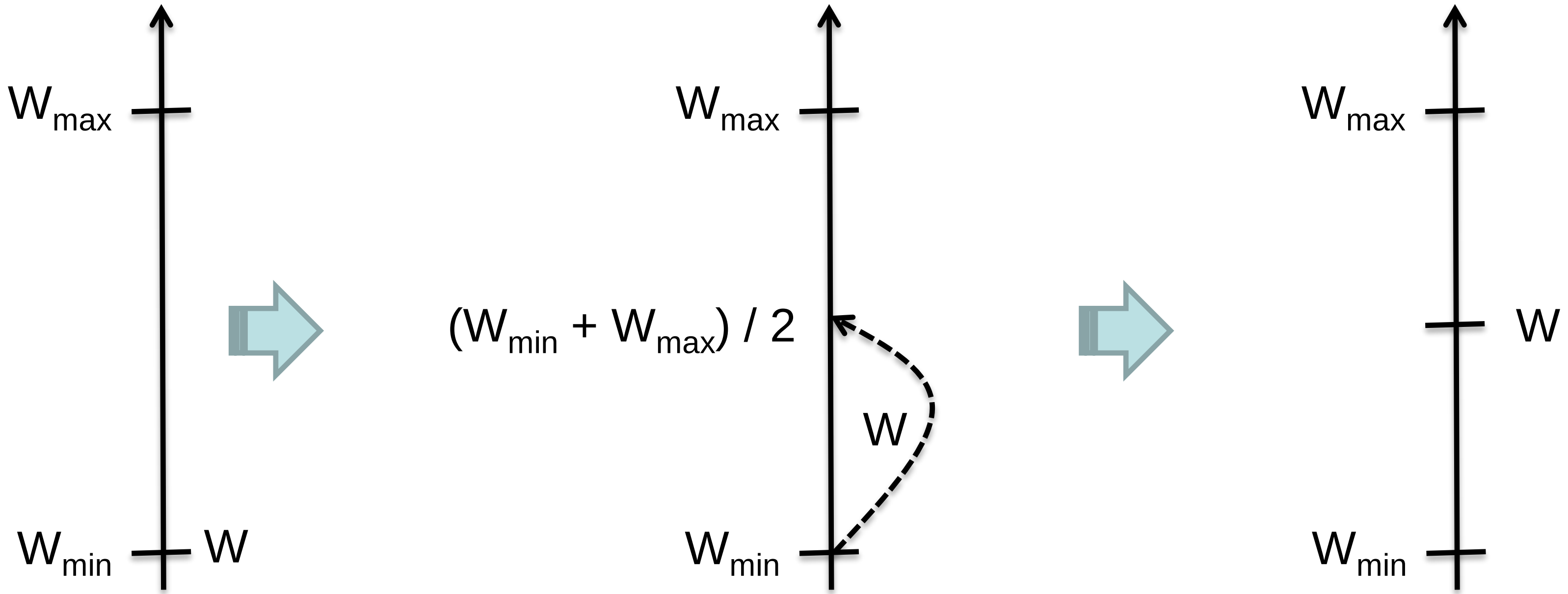
Binary Increase Congestion Control (BIC TCP)

- Adaptive *cwnd* increase
  - Binary increase (logarithmic)
- Gentle *cwnd* decrease: $w \leftarrow w - \frac{1}{8}w$



Binary Increase

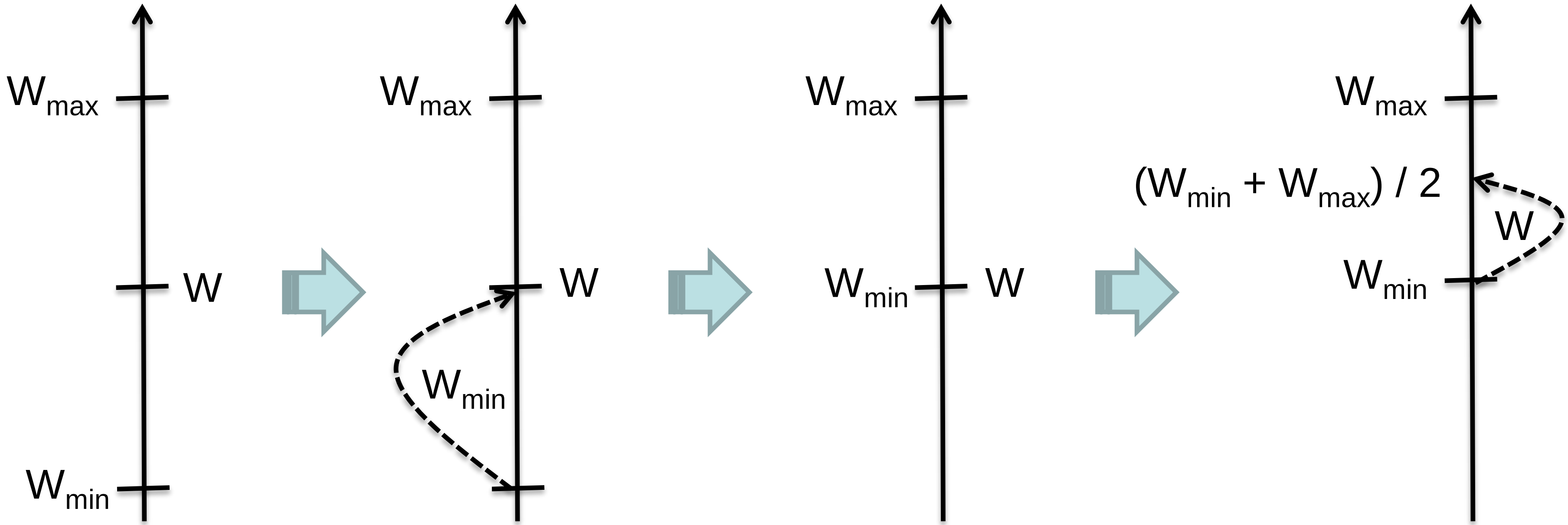
- W_{min} : last window size without packet loss
- W_{max} : window size with the most recent packet loss



Binary Increase – No Packet Loss

Institut für Kommunikations-Technik

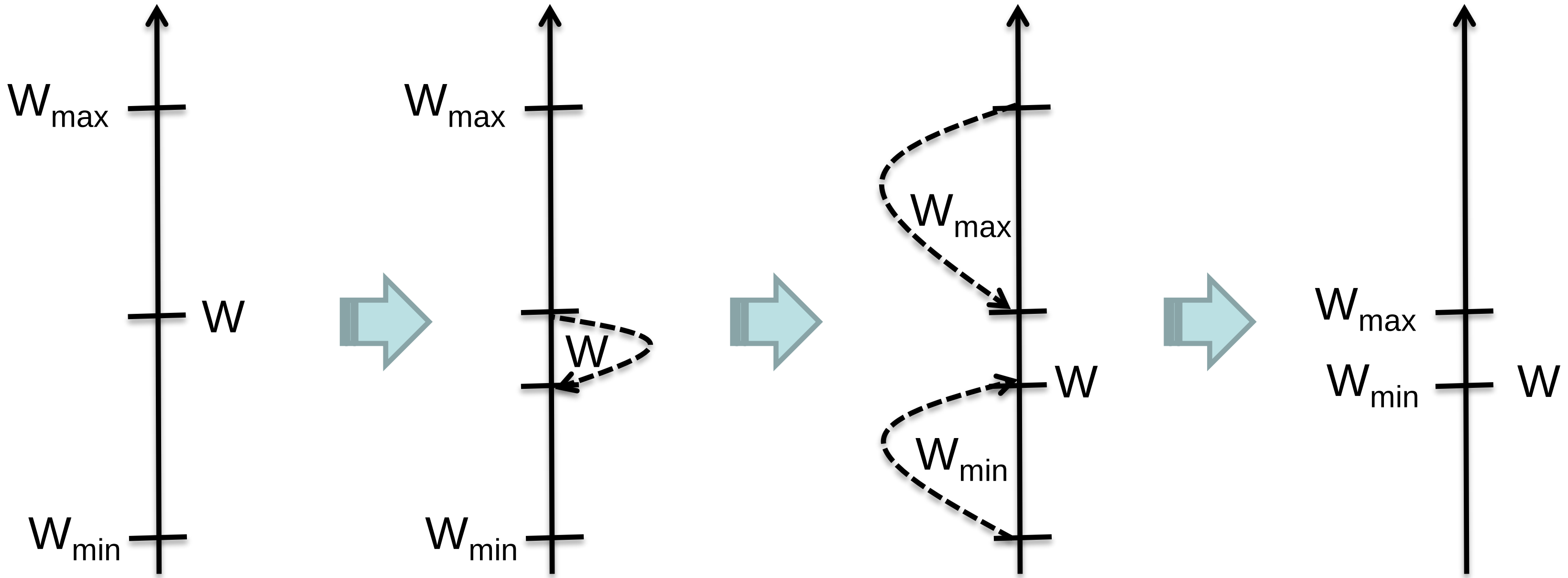
- W_{min} : last window size without packet loss
- W_{max} : window size with the most recent packet loss



Binary Increase – Packet Loss

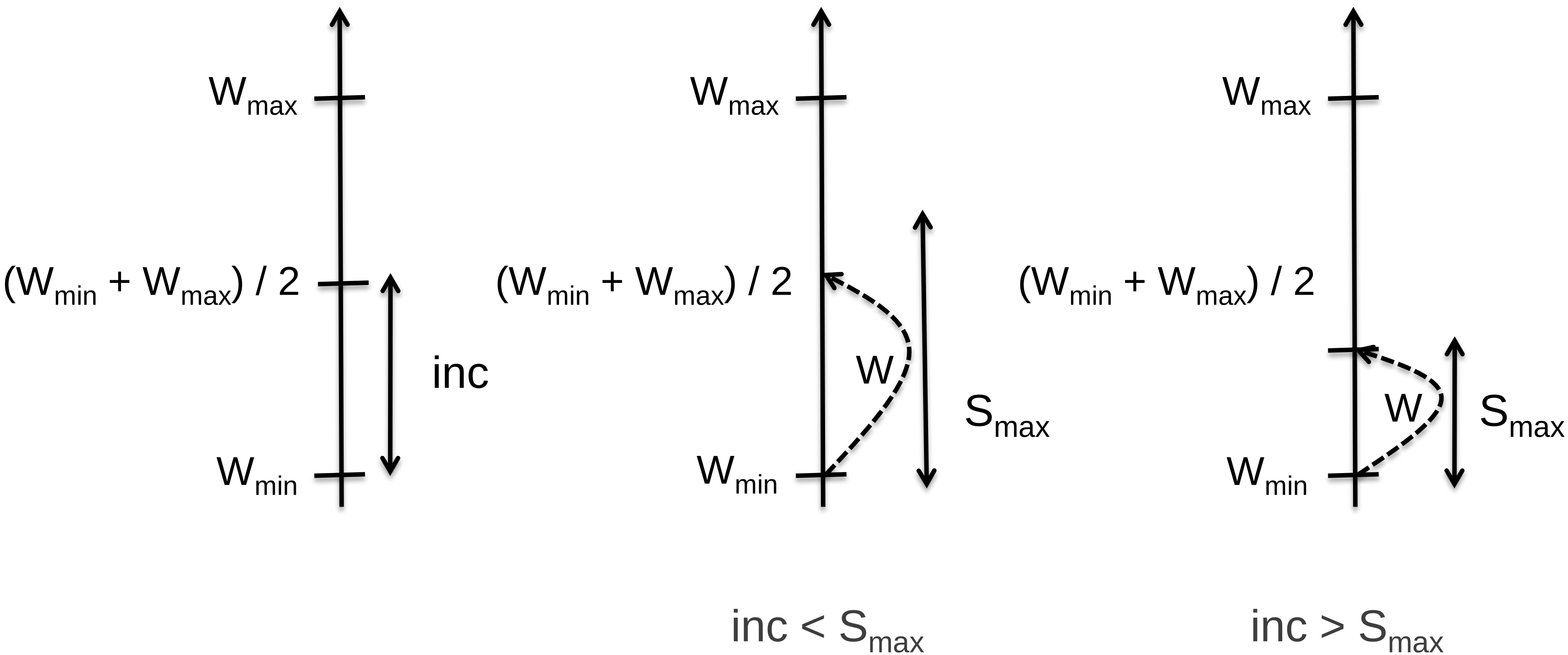
Institut für Kommunikations-Technik

- W_{min} : last window size without packet loss
- W_{max} : window size with the most recent packet loss



Binary Increase – Maximum Increment

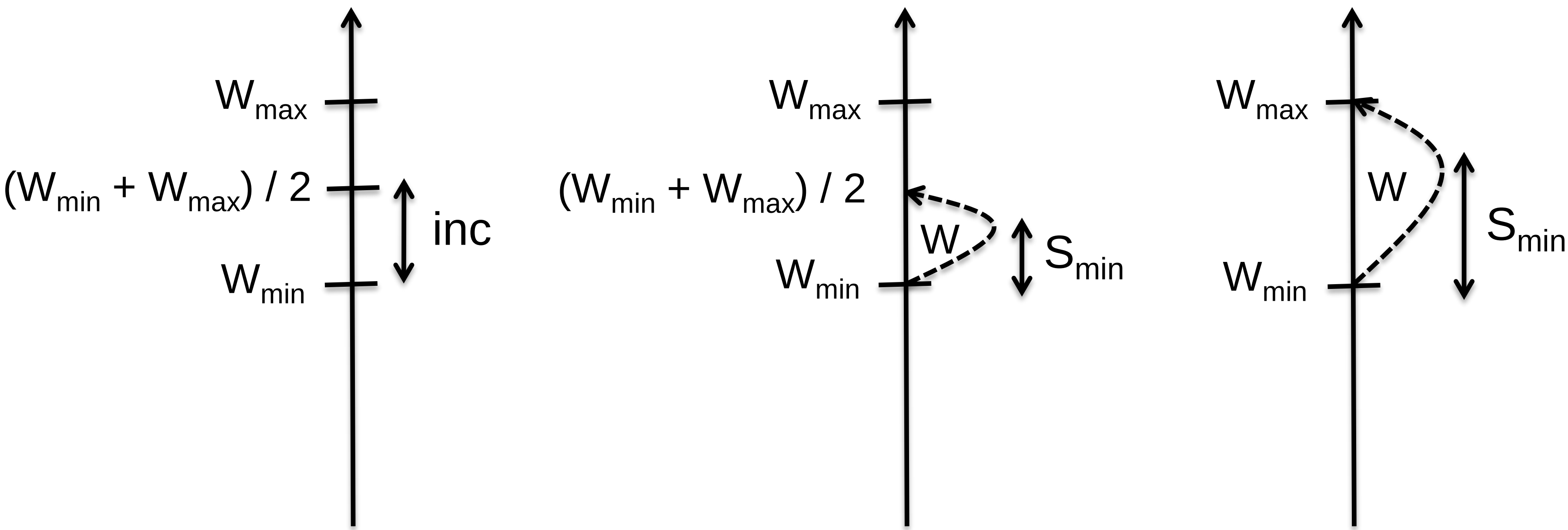
- W_{min} : last window size without packet loss
- W_{max} : window size with the most recent packet loss
- S_{max} : maximum increment



Binary Increase – Minimum Increment

Institut für Kommunikations-Technik

- W_{min} : last window size without packet loss
- W_{max} : window size with the most recent packet loss
- S_{min} : minimum increment



inc > S_{min}

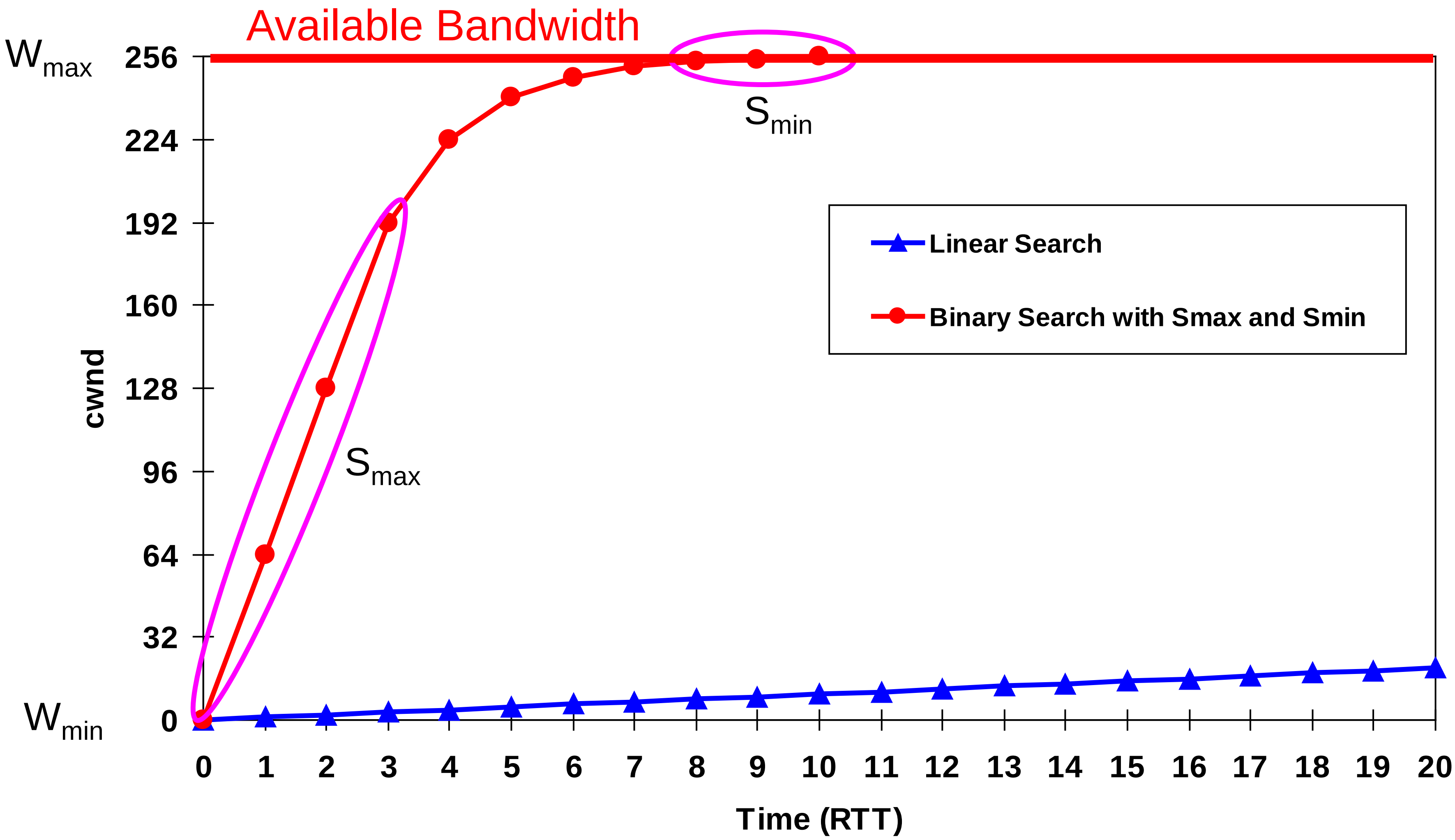
inc ≤ S_{min}

BIC TCP Window Adjustment Algorithm

- S_{min} : minimum increment
- S_{max} : maximum increment

```
while (inc > S_min):
    inc = (W_max - W_min) / 2
    if (inc > S_max): inc = S_max

    cwnd = cwnd + inc

    if there is no packet loss: W_min = cwnd

    else: W_max = cwnd

          cwnd = cwnd – cnwd / 8

          W_min = cwnd
```

BIC TCP Window Growth



CUBIC TCP

- CUBIC is an enhancement of BIC TCP:
  - Simplified window control
    - Window growth based on a cubic function
    - Window reduction by a constant factor β
  - Window growth function is independent of RTT
    - Window growth function is based on the elapsed time since the last loss event
  - Improved friendliness with TCP
    - TCP window growth emulation for short RTTs

CUBIC Window Growth Function

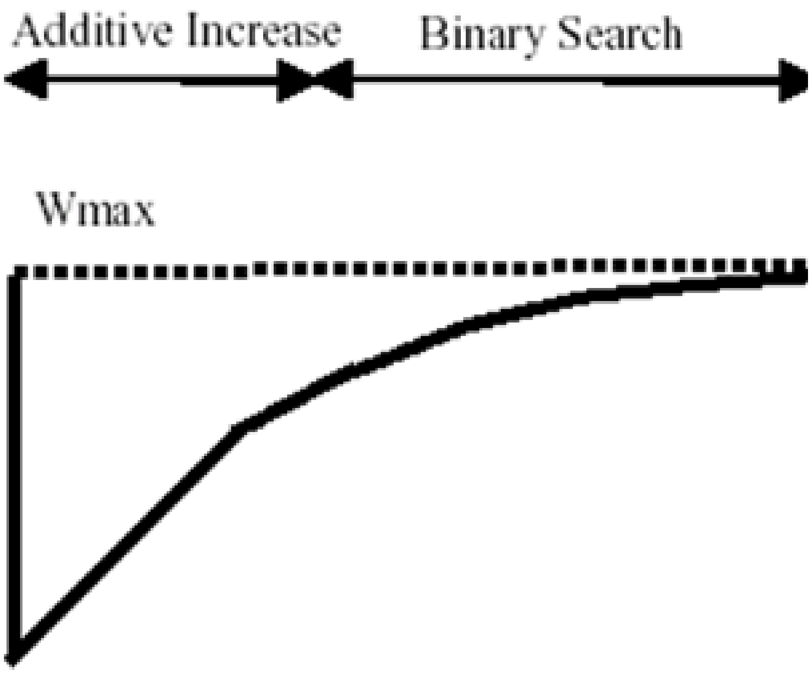
- CUBIC increases the window based on the function:

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

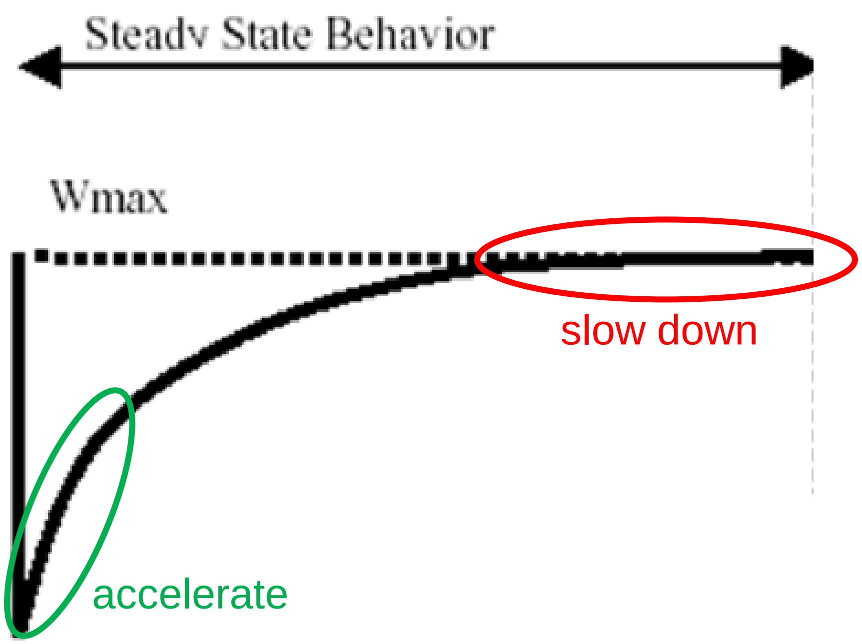
where:

- C: scaling factor (set to 0.4)
- t: elapsed time since the last loss event
- K: $K = \sqrt[3]{W_{max}\beta / C}$ (time period for increasing W to W_{max})
- β: window decrease factor (set to 0.2)

Window Growth: BIC vs. CUBIC



BIC



CUBIC

Improved TCP-Friendliness

- The CUBIC window growth function is independent of RTT and results in slower window increase than TCP for short RTTs

- CUBIC increases the window according to standard TCP when the cubic window growth function gives a smaller window size
  - Friendliness with TCP is improved for short RTTs

- Dual-mode window growth:
  - **if** $W_{TCP} > W_{CUBIC}$ **then** set window to W_{TCP}
    else set window to W_{CUBIC}

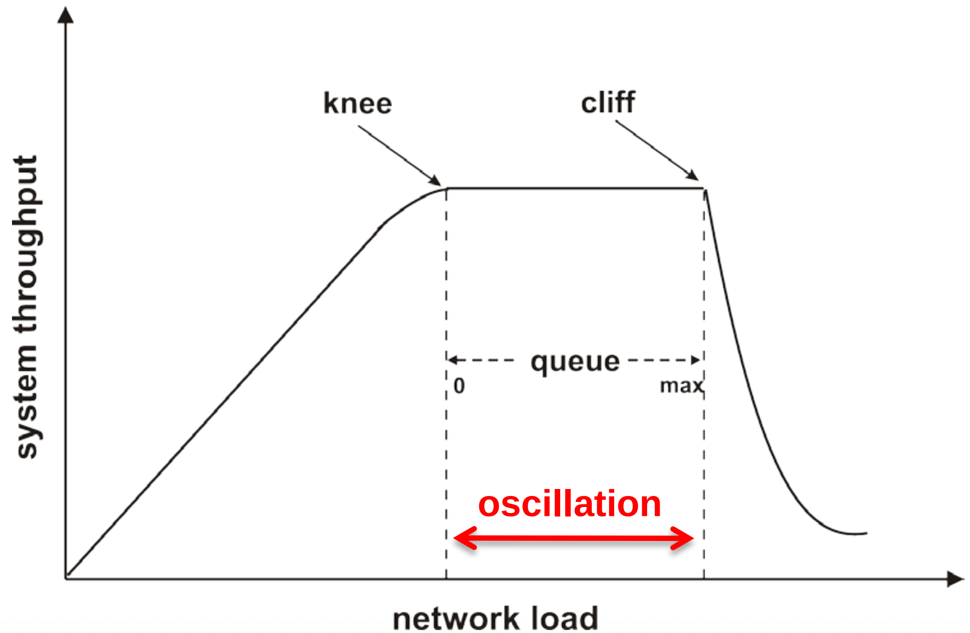
where: $$W_{TCP} = W_{\max}(1-\beta) + 3\frac{\beta}{2-\beta}\frac{t}{RTT}$$

Delay-based Congestion Control

Congestion Control based on Packet Loss

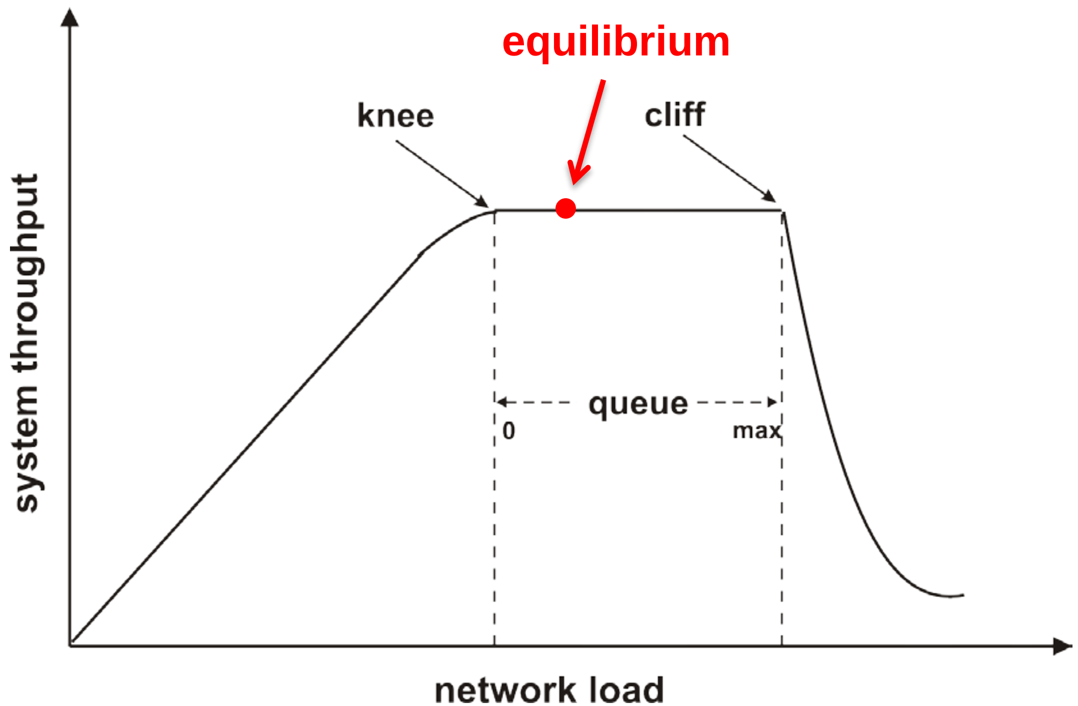
- Most congestion control mechanisms react to packet loss :
  - *cwnd* is increased additively (TCP) or adaptively (BIC TCP), when no packet loss is detected
  - *cwnd* is decreased multiplicatively upon packet loss

- Significant magnitude of oscillation:



Delay-based Congestion Control

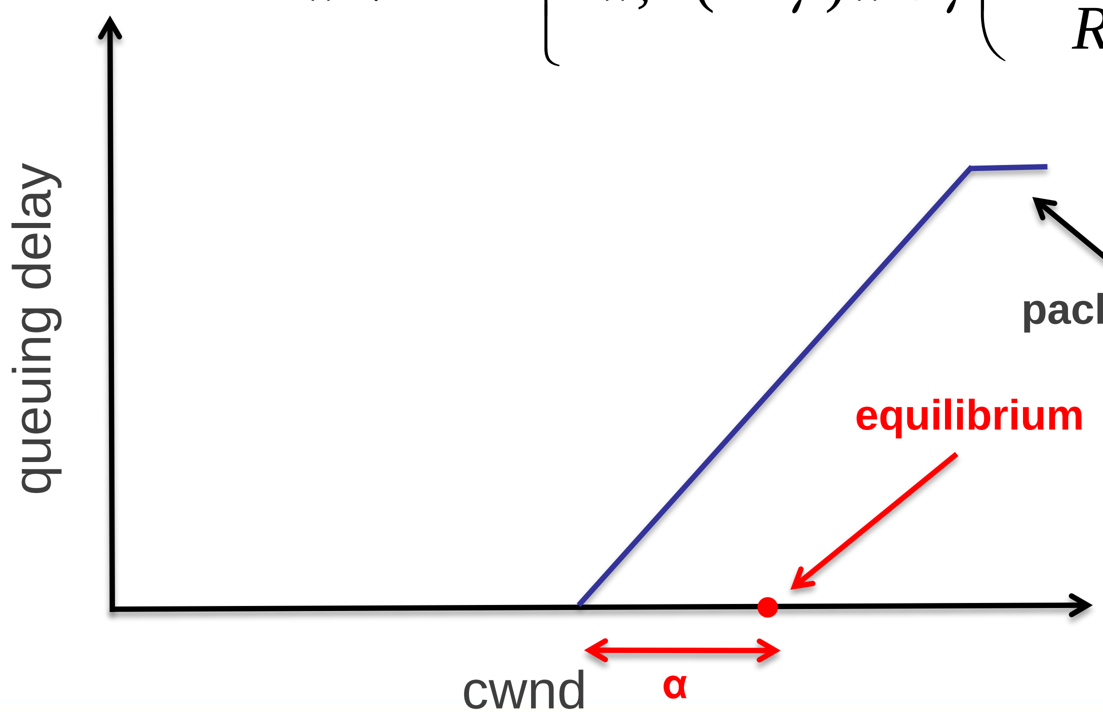
- Delay-based congestion control (Vegas, FAST):
  - adjusts *cwnd* based on measured queuing delay
  - aims to stabilize throughput while maintaining a small queue (equilibrium) and thus preventing packet loss



FAST TCP Window Adjustment

- baseRTT: minimum RTT (round-trip propagation delay)
- *α*: number of packets queued in the routers across the path in equilibrium (protocol parameter)
- $\gamma \in (0, 1]$

$$w \leftarrow \min\left\{2w,\ \ (1-\gamma)w + \gamma\left(\frac{baseRTT}{RTT}w + a\right)\right\}$$



FAST TCP Issues

- *baseRTT* estimation:
  - *baseRTT* can be overestimated when a new flow joins the network while there is queuing delay
  - if the route is changed to a longer path, the increase in the propagation delay will be perceived as congestion
- Tuning parameter α:
  - Improper adjustment of α in conjunction with small router buffers can prevent a FAST TCP flow from reaching its equilibrium
    - Packet loss can occur
    - FAST TCP is therefore designed to react to packet loss (similar to AIMD)
- RTT estimation:
  - RTT can be overestimated when there is queuing delay in the reverse path

Congestion Control without Reliability

Congestion Control for Delay-Sensitive Applications

- Delay-sensitive (e.g. multimedia) applications bear packets with limited useful lifetime:
  - Retransmissions usually deliver useless packets
  - Timeliness is more important than reliability
  - TCP’s in-order delivery can introduce arbitrary delays

- Congestion control is yet essential:
  - Free-transmitting protocols can cause long delays when congestion occurs
  - The lack of congestion control would destabilize the Internet

Congestion Control without Reliability

- Essential features for delay-sensitive applications:
  - Packet delivery without reliability
  - Out-of-order delivery
  - Congestion control with smooth transmission patterns

UDP

- Approaches:
  - Use UDP and let the applications control the congestion themselves
  - Design and deploy a new protocol that carries out congestion control on top of UDP

Datagram Congestion Control Protocol (DCCP)

- DCCP provides congestion control on top of UDP:
  - Data delivery without reliability
  - Out-of-order delivery
  - Congestion control mechanism choice via Congestion Control IDs (CCIDs)
- Available CCIDs:
  - CCID 2: TCP-like Congestion Control
  - CCID 3: TFRC Congestion Control

CCID 2: TCP-like Congestion Control

- CCID 2 corresponds to TCP-like congestion control for applications that are not harmed by abrupt rate changes.

- CCID 2 vs. TCP:

	TCP	DCCP: CCID 2
Cong. Avoidance	X	X
Slow Start	X	X
SACK	X	X
Reliable Delivery	X	
Reverse Path Congestion Control		X

CCID 3: TFRC (TCP-Friendly Rate Control)

Institut für Kommunikations-Technik

- CCID 3 corresponds to TFRC congestion control for applications that require smooth transmission patterns.

- TFRC adjusts its transmission rate in terms of packet loss (*p*), RTT and retransmission timeout (*RTO*):

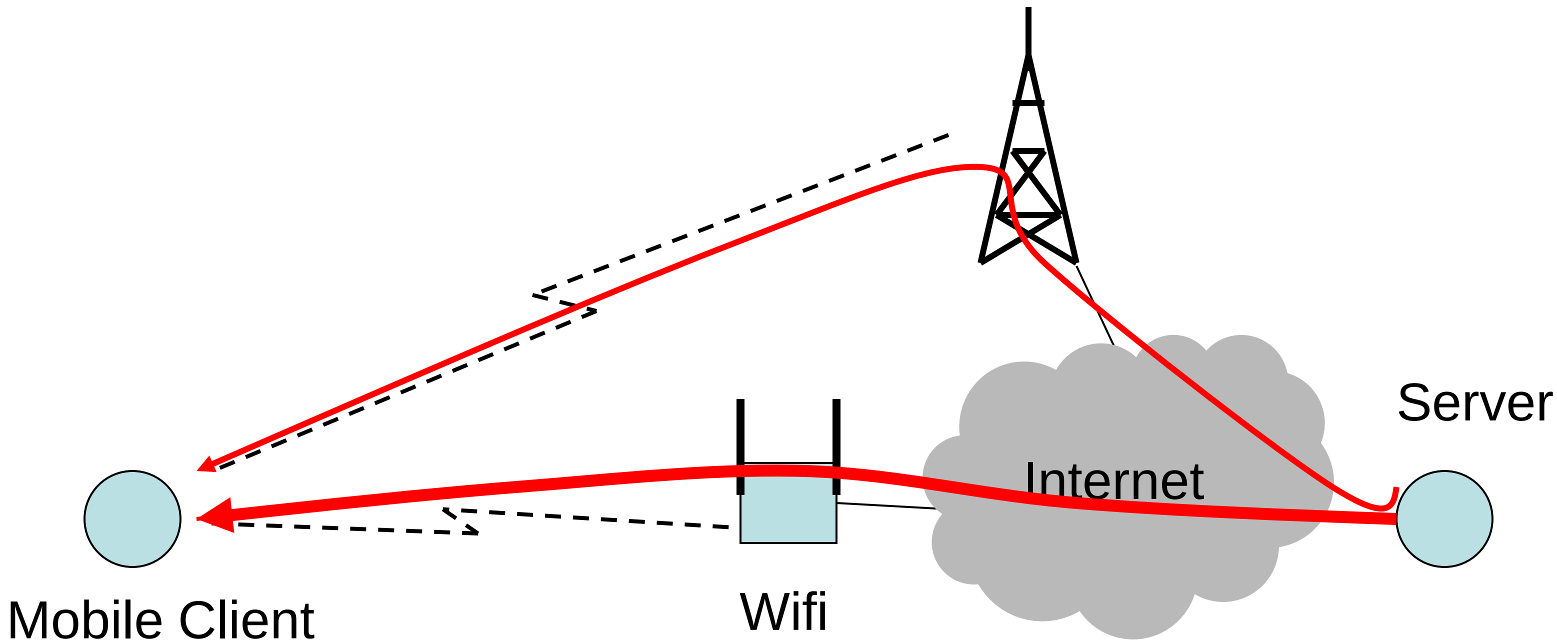
$$T(p, RTT, RTO) = \frac{1}{RTT\sqrt{\frac{2p}{3}} + RTO(3\sqrt{\frac{3p}{8}})p(1+32p^2)}$$

- TFRC achieves smooth rate adjustments:
  - Maximum rate increase is 0.14 packets per RTT
  - 5 RTTs are required for halving the transmission rate

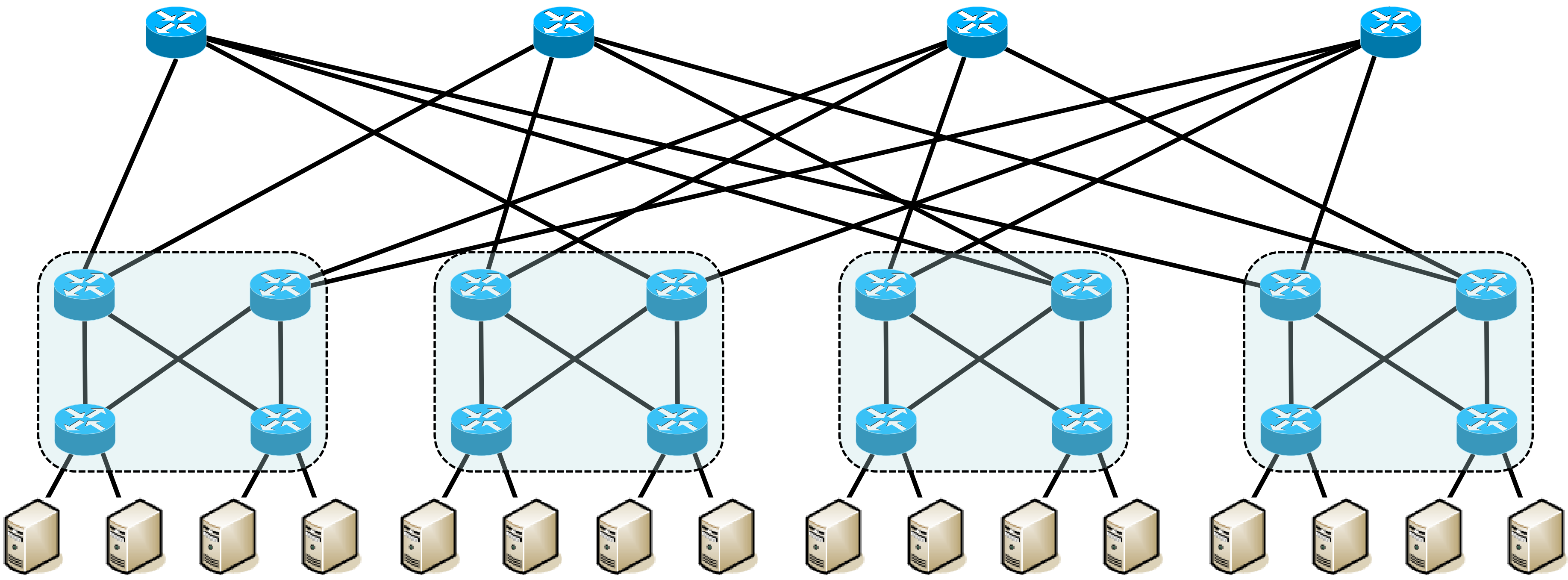
Multi-Path Congestion Control

Some slides/figures from M. Handley's MPTCP Presentation

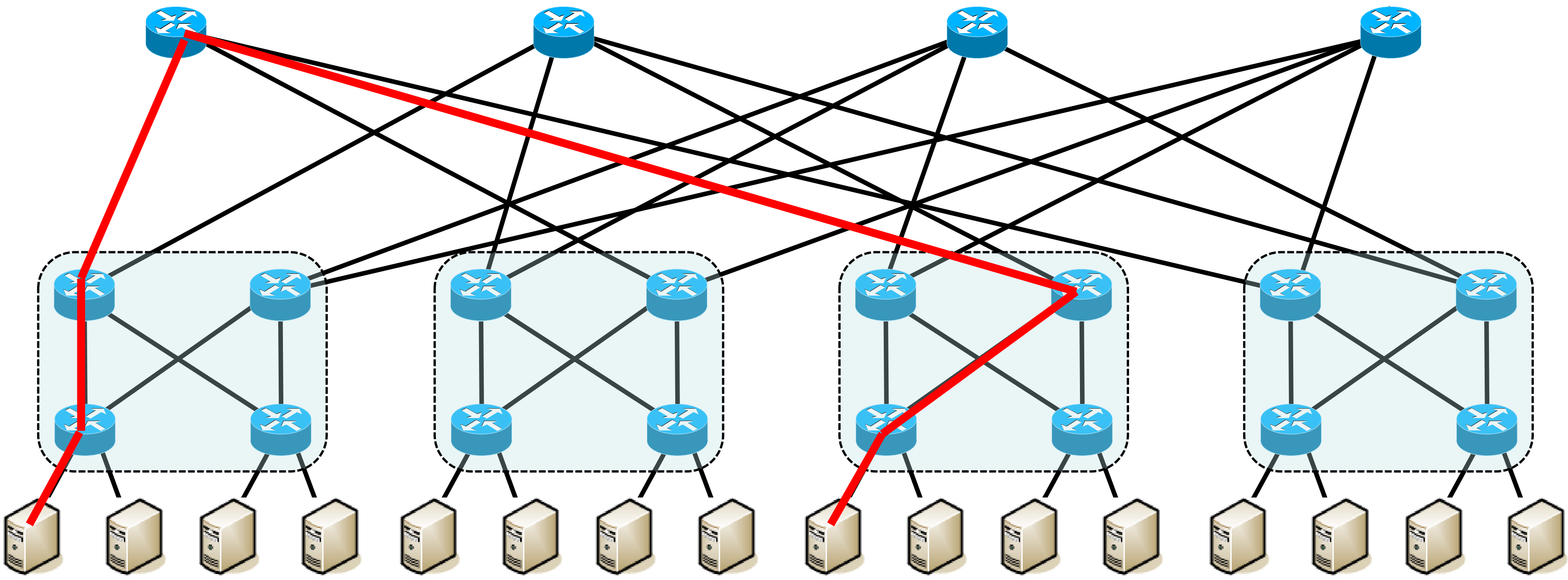
Multi-Path Data Transport



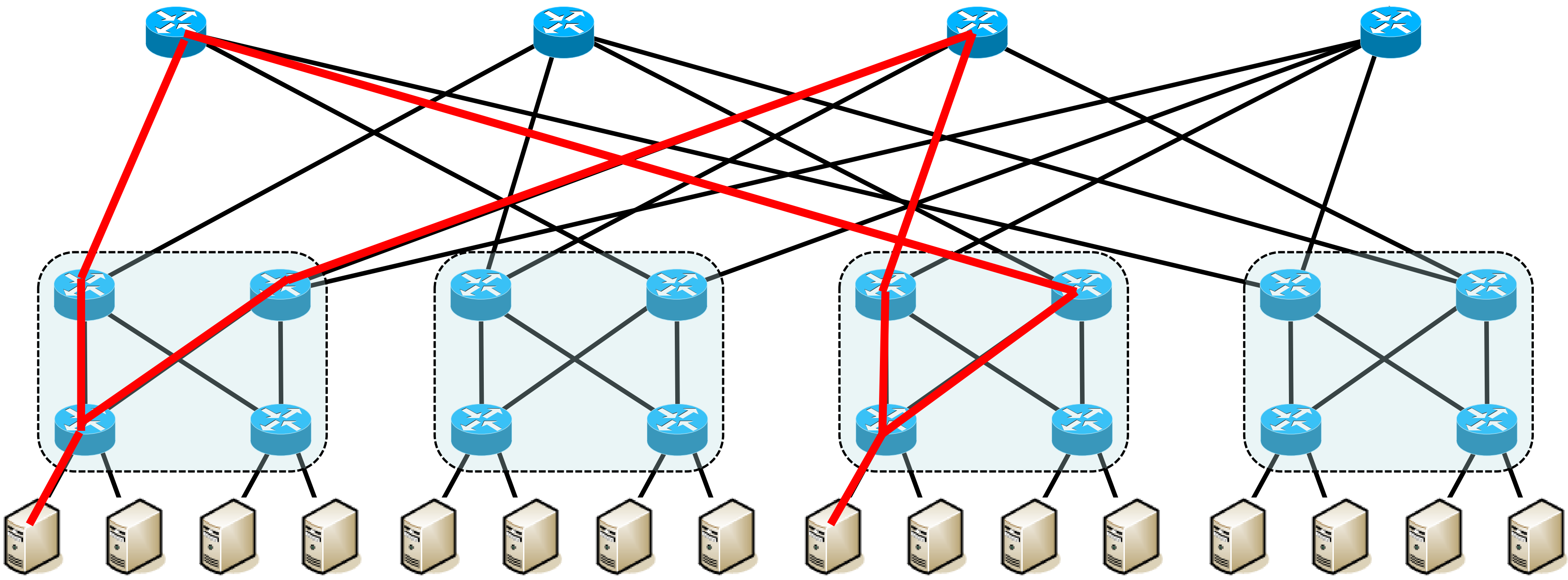
Multi-Path Data Transport in Data-Centers



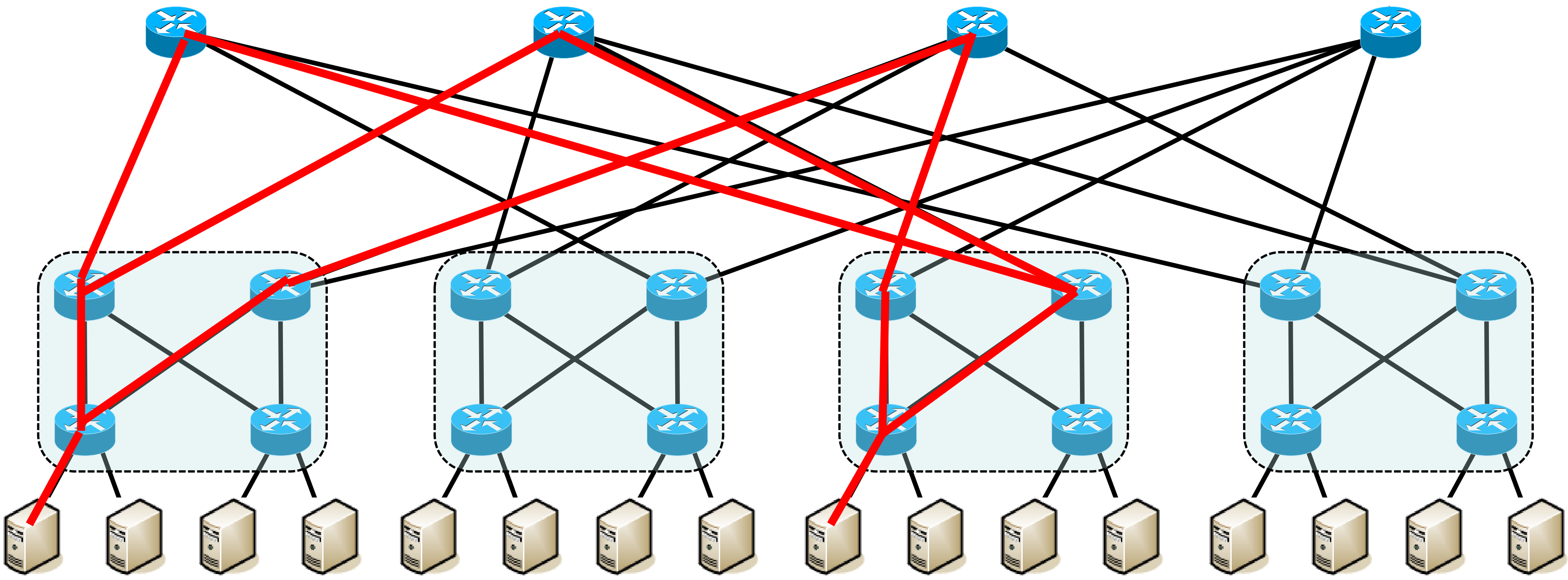
Multi-Path Data Transport in Data-Centers



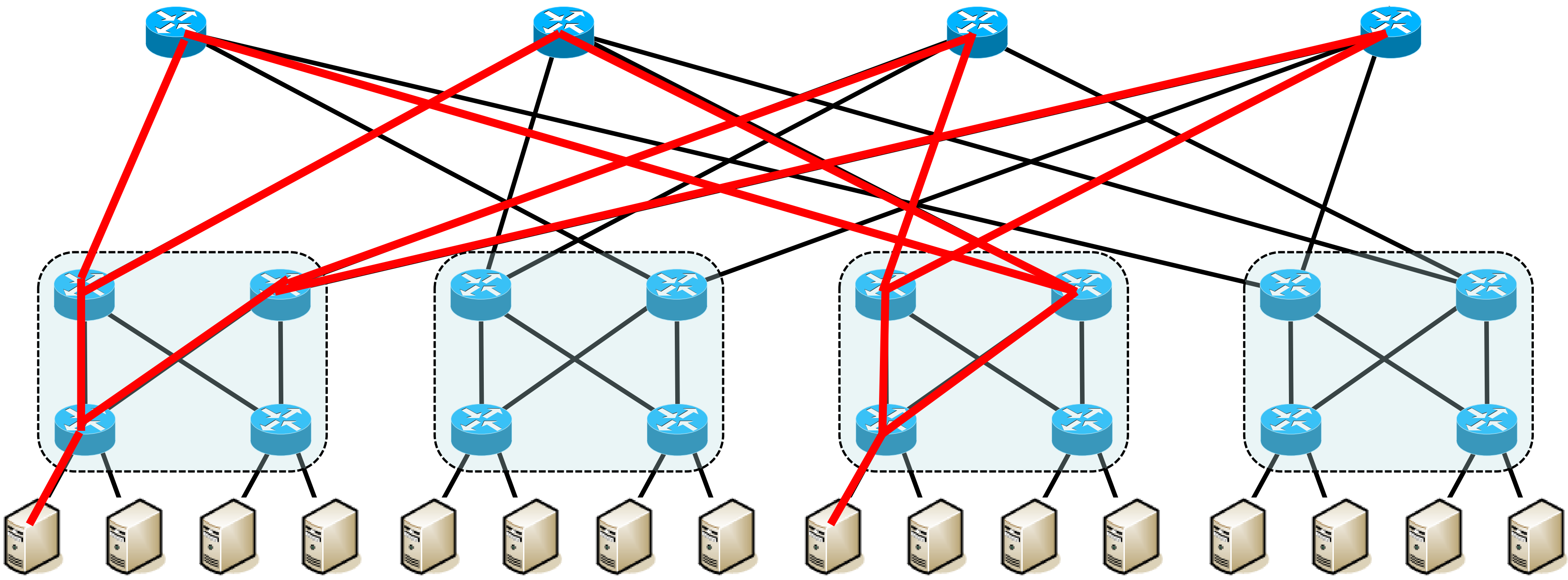
Multi-Path Data Transport in Data-Centers



Multi-Path Data Transport in Data-Centers



Multi-Path Data Transport in Data-Centers

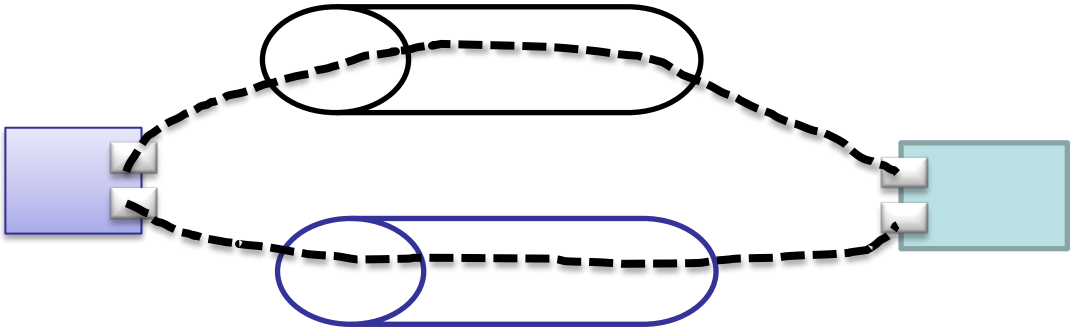


Multi-Path Congestion Control

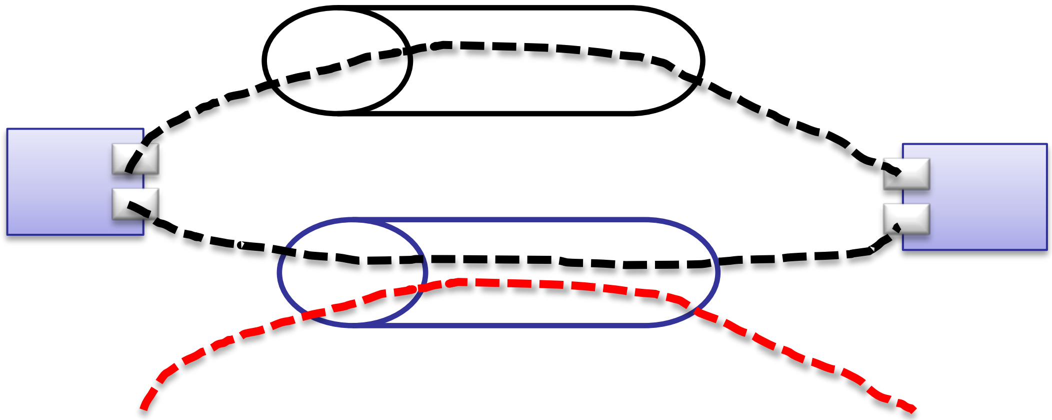
- TCP cannot move away traffic from a congested path:
  - TCP merely adjusts the transmission rate
  - Network resources can be available from other paths
- Pooling resources from other paths:
  - Routing is too slow and not is aware of congestion
  - Congestion control across multiple paths
- Multi-path congestion control:
  - moves away traffic from a congested path
  - can balance load across multiple paths
  - achieves better resource utilization
  - increases throughput

Multi-Homing and Multi-path Congestion Control

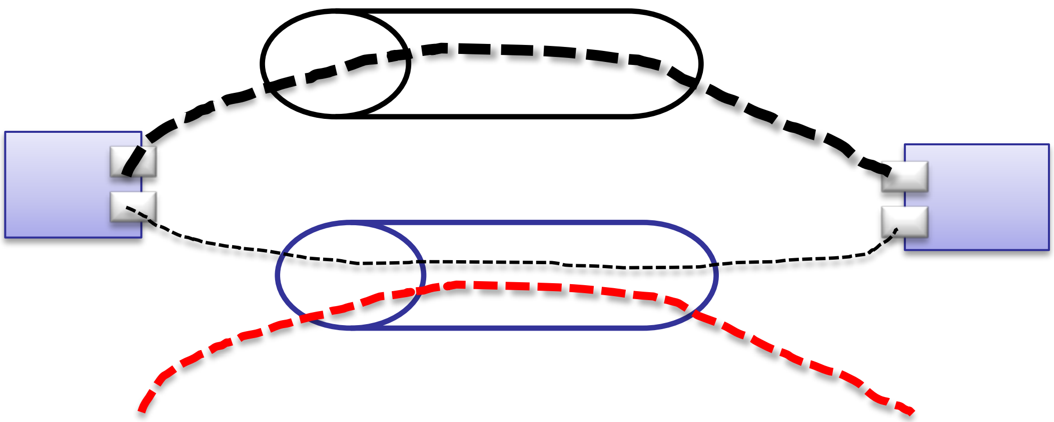
- End-nodes can have multiple interfaces:
  - Any interface can be the end-point of a sub-flow
  - Different sub-flows can use different paths
  - Traffic is split among multiple sub-flows



Multi-path Congestion Control Example



Congestion at 2nd path



Traffic is moved away from 2nd to 1st path

Multi-Path TCP (MPTCP)

- MPTCP features:
  - Window-based mechanism
  - Resource pooling
  - Load balancing across multiple paths
  - TCP-friendliness
    - An MPTCP sub-flow in a given path should not achieve higher throughput than a single TCP flow

“Fully-Linked” Window Adjustment Algorithm

- Congestion window w_i per sub-flow i is adjusted as:

$$\texttt{\textbf{after} each RTT: } w_i \leftarrow w_i + \frac{w_i}{w_{total}}$$

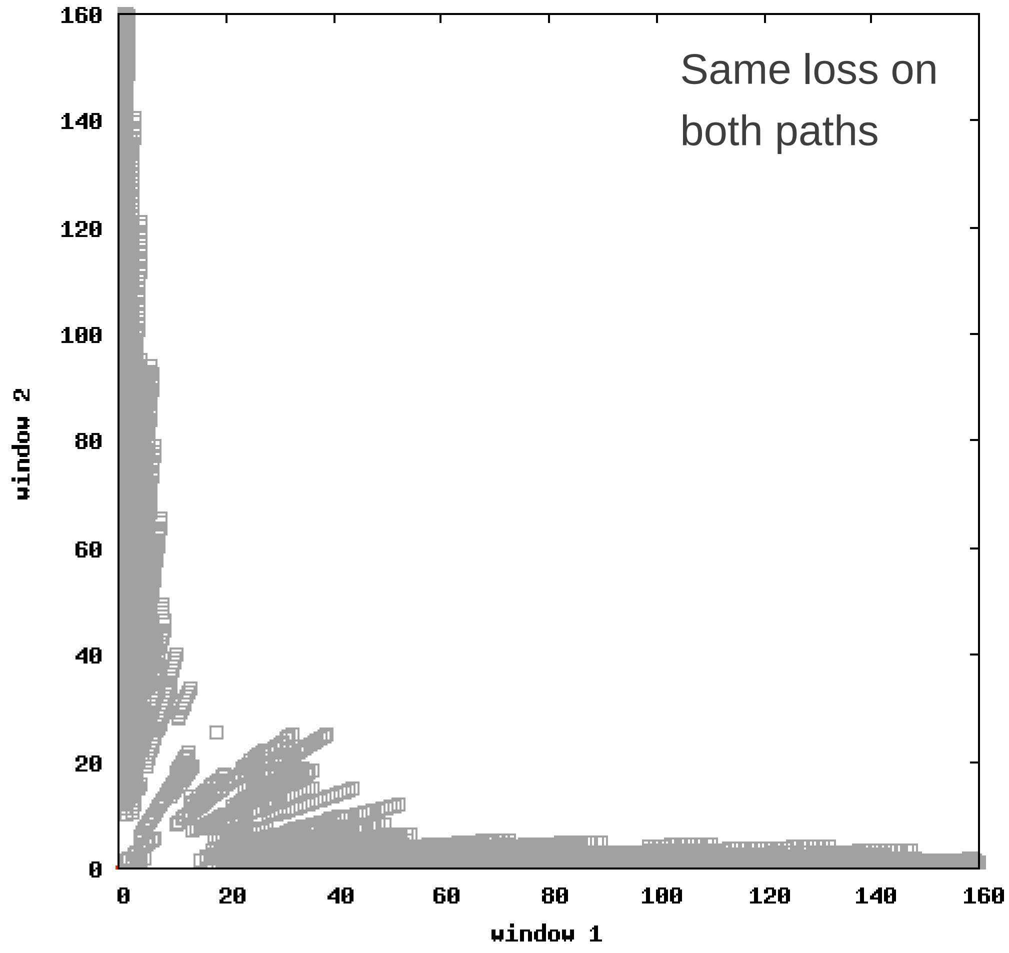
$$\texttt{\textbf{when} packet loss is detected: } w_i \leftarrow w_i - \frac{w_{total}}{2}, w_i > 0$$

- Similar reactions to TCP (TCP-friendly):
  - Window size is increased by 1 $(\sum_i \frac{w_i}{w_{total}} = 1)$
  - Window size is halved $(\frac{w_{total}}{2})$ upon congestion

“Fully-Linked” Window Adjustment Problem

Institut für Kommunikations-Technik

- When paths are equally congested, linking causes the traffic to flap between them:
  - Coupling moves traffic away from the more congested path until loss rates equalize
  - But losses are never exactly equal, so traffic flaps between paths randomly



"Linked Increases" Window Adjustment Algorithm

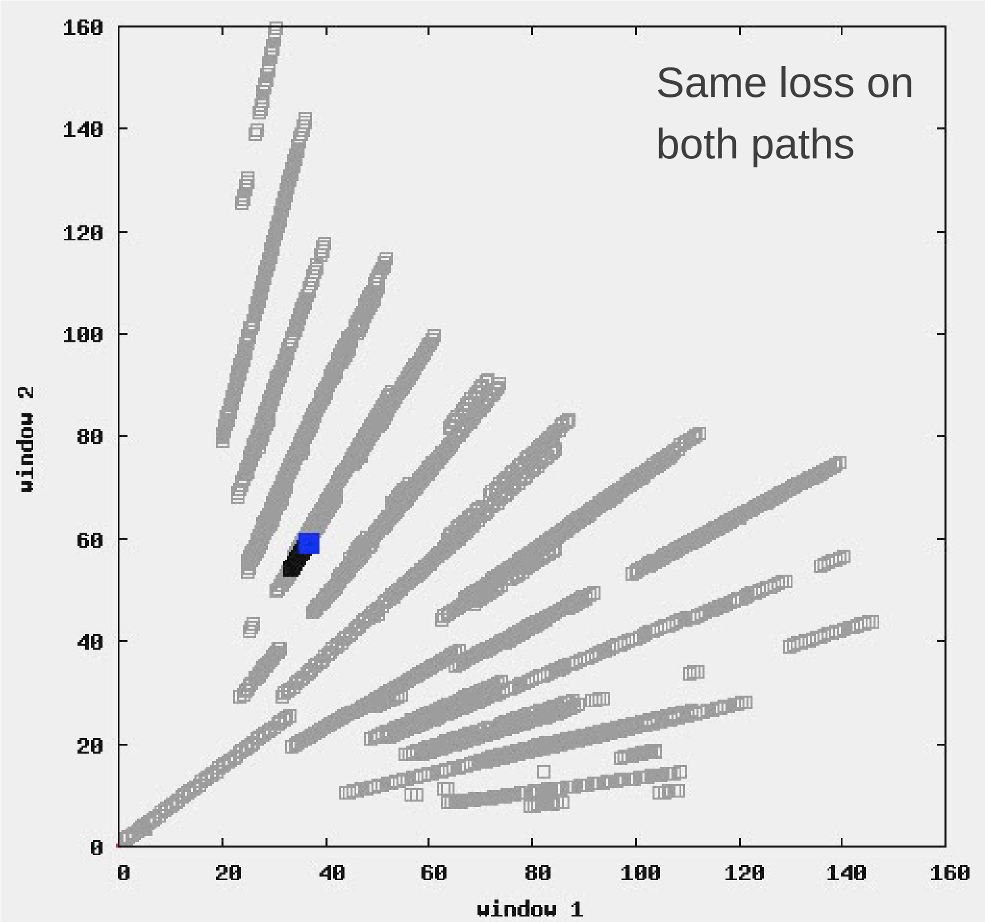
- Linking only the window increases can mitigate traffic flapping
- Congestion window w_i per sub-flow *i* is adjusted as:

after each RTT: $w_i \leftarrow w_i + \frac{w_i}{w_{total}}$

when packet loss is detected: $w_i \leftarrow w_i - \frac{w_i}{2}$

"Linked Increases" Window Behavior

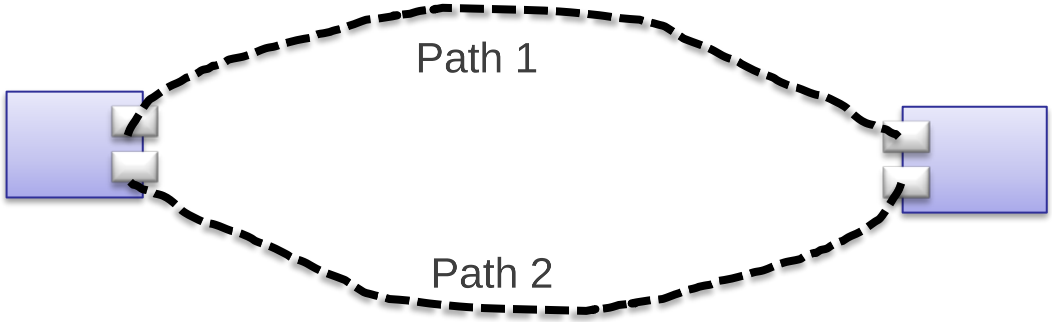
- Traffic does not flap with "linked increases" algorithm under equal loss rates on both paths



Performance with Equal RTTs

- High throughput when RTTs among paths are equal

w_1 = 10 packets, RTT_1 = 10 ms ⇒ 1000 packets/sec



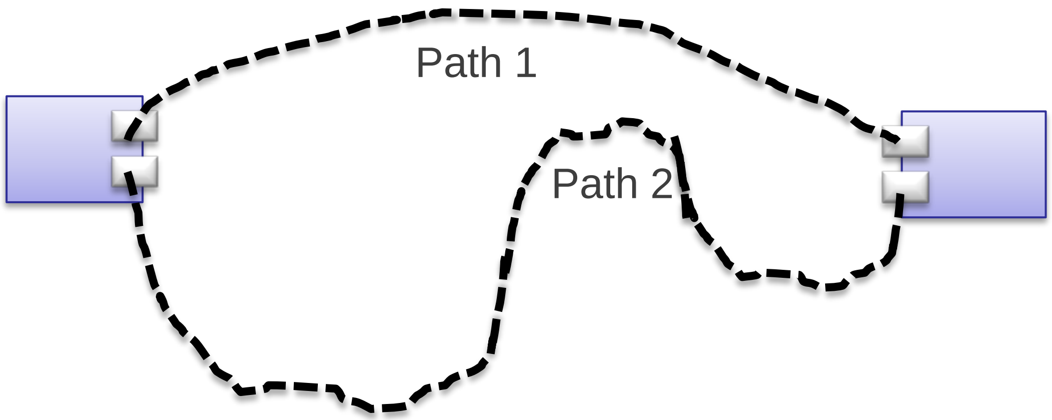
w_2 = 10 packets, RTT_2 = 10 ms ⇒ 1000 packets/sec

- Total throughput is 2000 packets/sec
- Same throughput with single-path TCP

Performance with Different RTTs

- Low throughput when RTTs among paths are different

$w_1 = 10$ packets, $RTT_1 = 10$ ms ⇨ 1000 packets/sec



$w_2 = 10$ packets, $RTT_2 = 100$ ms ⇨ 100 packets/sec

- Total throughput is 1100 packets/sec
- Single-path TCP would have achieved 2000 packets/sec on Path 1

MPTCP Window Adjustment Algorithm

- MPTCP adjusts the congestion window w_i per sub-flow *i*:

```
when a new ACK is received on path i:
```

$$w_i \leftarrow w_i + \min_{S \subseteq R : i \in S} \frac{\max_{s \in S} \frac{w_s}{RTT_s^{\,2}}}{\left(\sum_{s \in S} \frac{w_s}{RTT_s} \right)^2}$$

```
when packet loss is detected:
```

$$w_i \leftarrow w_i - \frac{w_i}{2}, w_i > 0$$

References

- D. Chiu and R. Jain, **Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks**, Journal of Computer Networks and ISDN, 1989
- P. Papadimitriou and V. Tsaoussidis, **Optimization of AIMD Congestion Control for Media-Streaming Applications**, Journal of Internet Engineering, 2007
- L. Xu, et al., **Binary Increase Congestion Control for Fast, Long Distance Networks**, IEEE INFOCOM 2004
- S. Ha, et al., **CUBIC: A New TCP-Friendly High-Speed TCP Variant**, ACM SIGOPS Operating Systems Review, 2008
- D. Wei, et al., **FAST TCP: Motivation, Architecture, Algorithms, Performance**, IEEE Transactions on Networking, 2007
- E. Kohler, et al., **Designing DCCP: Congestion Control Without Reliability**, ACM SIGCOMM 2006
- D. Wishik, et al., **Design, Implementation and Evaluation of Congestion Control for Multipath TCP**, USENIX NSDI 2011