Chapter 7

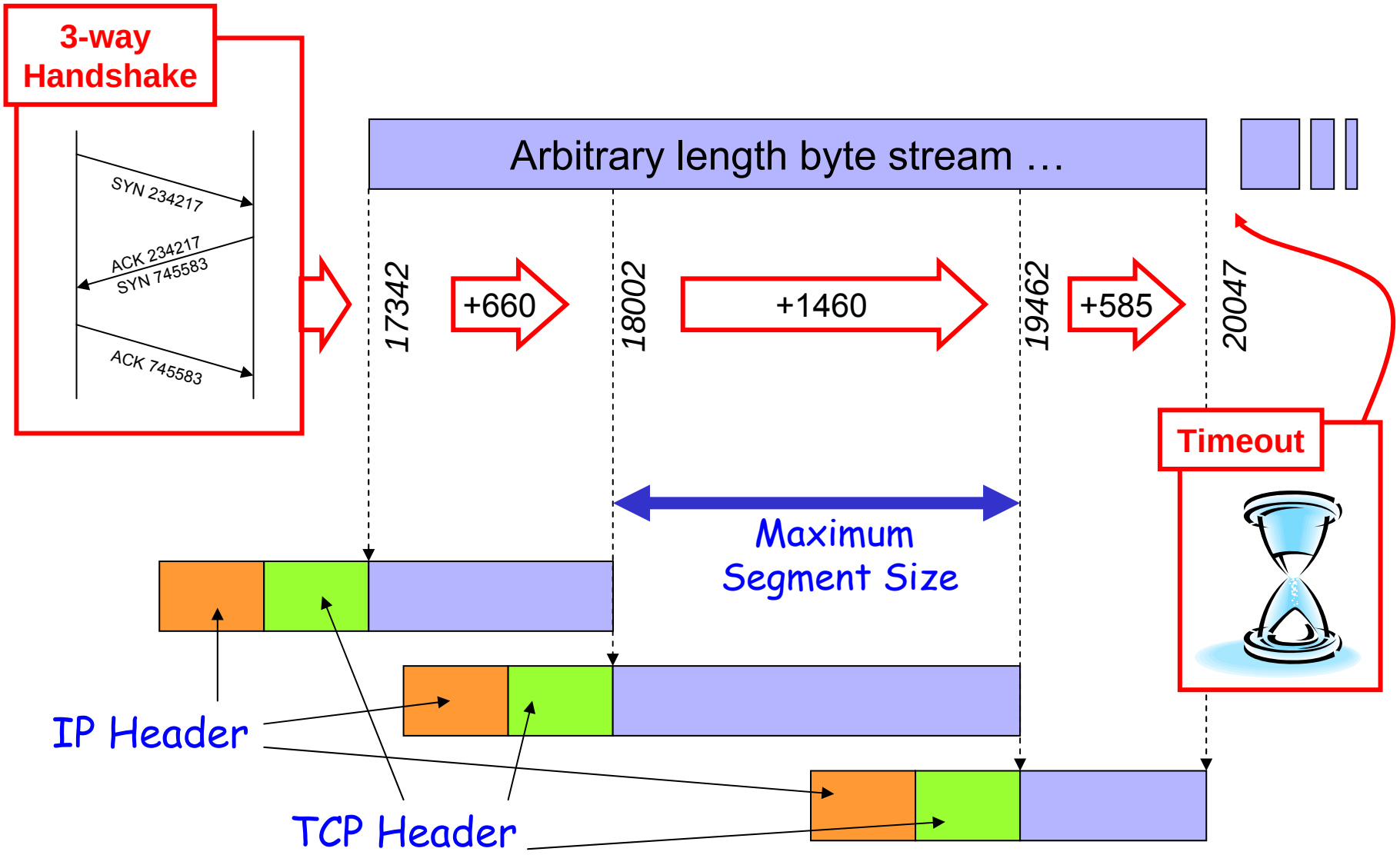
Advanced computer networking

Internet Protocols

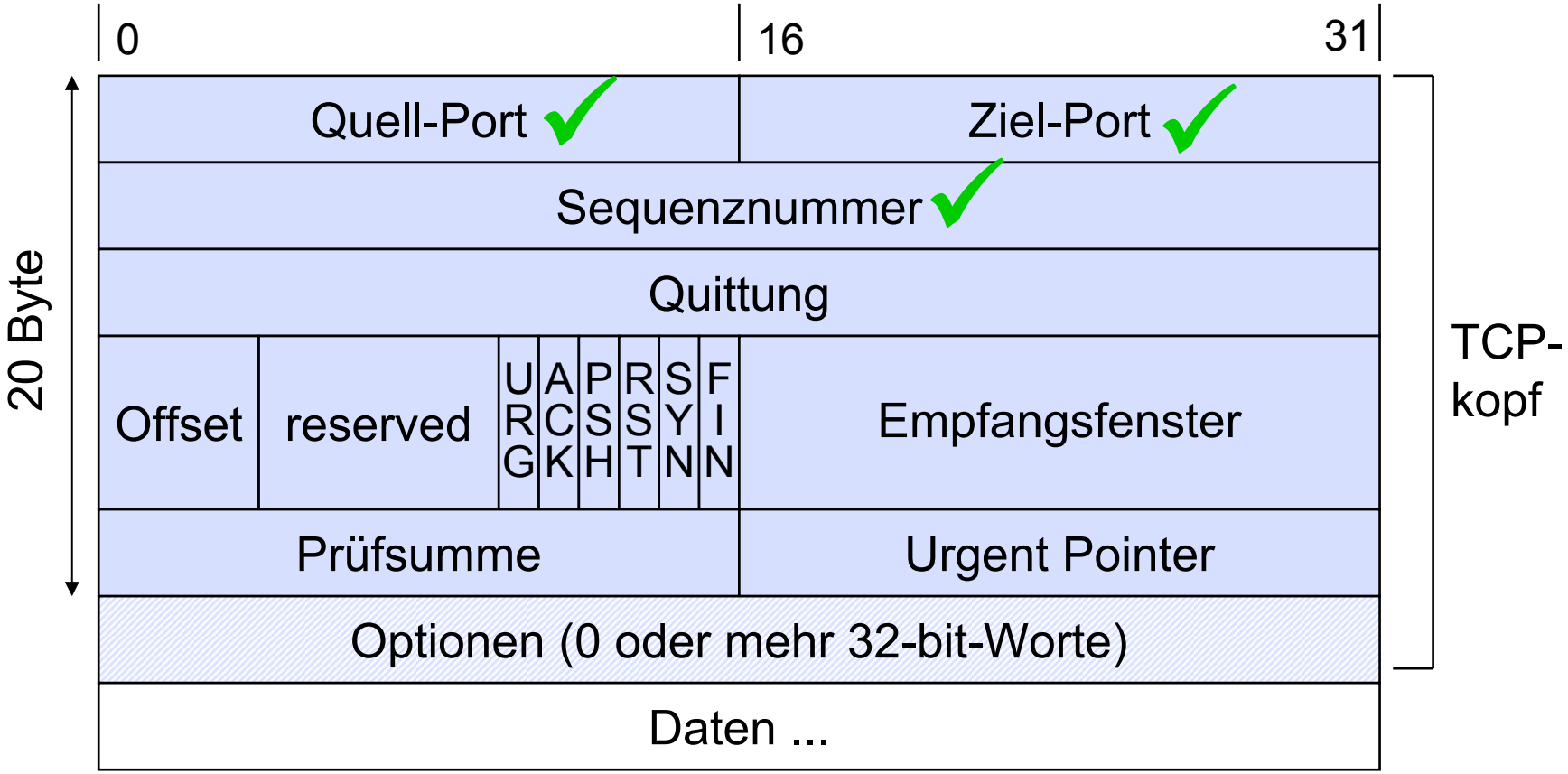
Thomas Fuhrmann

Network Architectures
Computer Science Department
Technical University Munich

Transport Protocol TCP – Segmentation



TCP Header


0
16
31
Quell-Port
Ziel-Port
Sequenznummer
Quittung
Offset
reserved
URG
ACK
PSH
RST
SYN
FIN
Empfangsfenster
Prüfsumme
Urgent Pointer
Optionen (0 oder mehr 32-bit-Worte)
Daten ...
20 Byte
TCP-kopf

Semantik der TCP Datenübergabe

TCP bietet gegenüber der Anwendung einen Byte-orientierten Dienst. Zur Netzwerkschicht hin muss es den Datenstrom segmentieren.

- Frage: Wann wird aus dem empfangenen Bytestrom eine TCP-Dateneinheit gebildet und an IP weitergegeben?
- RFC 793 sagt dazu: TCP should „send that data in segments at its own convenience“

Drei mögliche Kriterien sollten für die Lösung herangezogen werden:

- Maximum Segment Size (MSS):
  - Sobald Daten für ein Segment maximal zulässiger Größe vorhanden sind, kann gesendet werden.
  - Ist beim Bulk-Data-Transfer (FTP, HTTP, …) fast immer der Fall
- Push (PSH-Flag im TCP-Kopf bzw. Option an der Socket-Schnittstelle)
  - Sender verlangt hiermit das sofortige Versenden der übergebenen Daten
  - Wird im interaktiven Verkehr (SSH, Telnet, …) verwendet
- Zeitgeber
  - Nach einem gewissen Zeitintervall der Inaktivität werden die dem TCP-Protokollstapel übergebenen Daten gesendet

Reliability

… as provided by TCP

Zuverlässigkeit durch Quittungen

Problem

– Wie erfährt der Sender, dass eine Dateneinheit nicht bzw. nicht korrekt beim Empfänger angekommen ist?

Mechanismus

– Der Empfänger teilt dem Sender mit, ob er eine Dateneinheit empfangen hat oder nicht. Hierzu werden spezielle Daten-einheiten, so genannte Quittungen, versendet.
– Man unterscheidet zwei Quittungsarten:
  • Positive Quittung (engl. Acknowledgement, ACK), d.h. der Empfänger teilt dem Sender mit, dass er die entsprechenden Daten erhalten hat.
  • Negative Quittung (engl. Negative Acknowledgement, NACK), d.h. der Empfänger meldet dem Sender, dass er die entsprechenden Daten nicht erhalten hat.

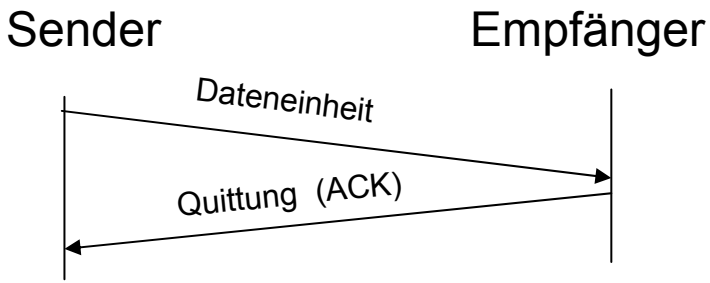
Die beiden Quittungsarten (ACK und NACK) kann man nochmal in zwei Varianten unterscheiden:

– Selektive Quittungen
  • Die Quittung bezieht sich auf eine einzelne Dateneinheit.
  • Engl. Selective Acknowledgement (SACK)

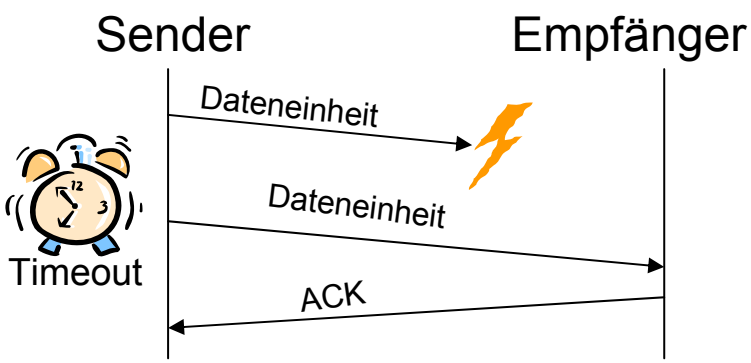
– Kumulative Quittungen
  • Die Quittung bezieht sich auf eine Menge von Dateneinheiten, die in der Regel durch eine obere Sequenznummer beschränkt ist.
  • Beispiel: Positive kumulative Quittung, die besagt, dass alle Dateneinheiten bis zur angegebenen Sequenznummer korrekt empfangen wurden.

Stop-and-Wait

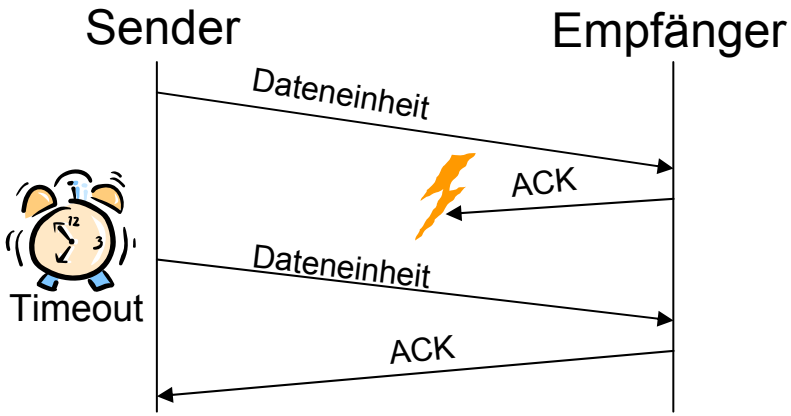
Normalfall



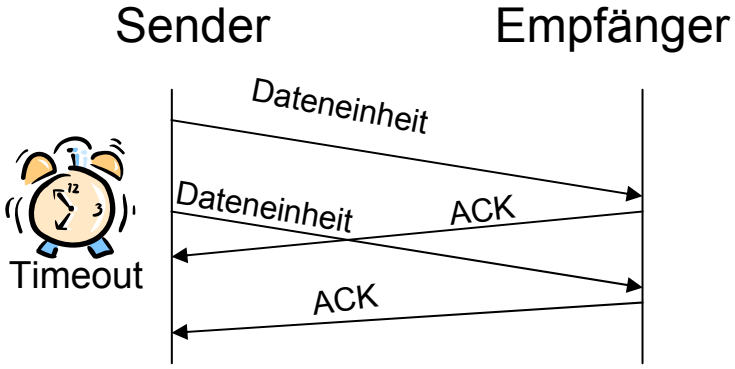
Daten-Verlust



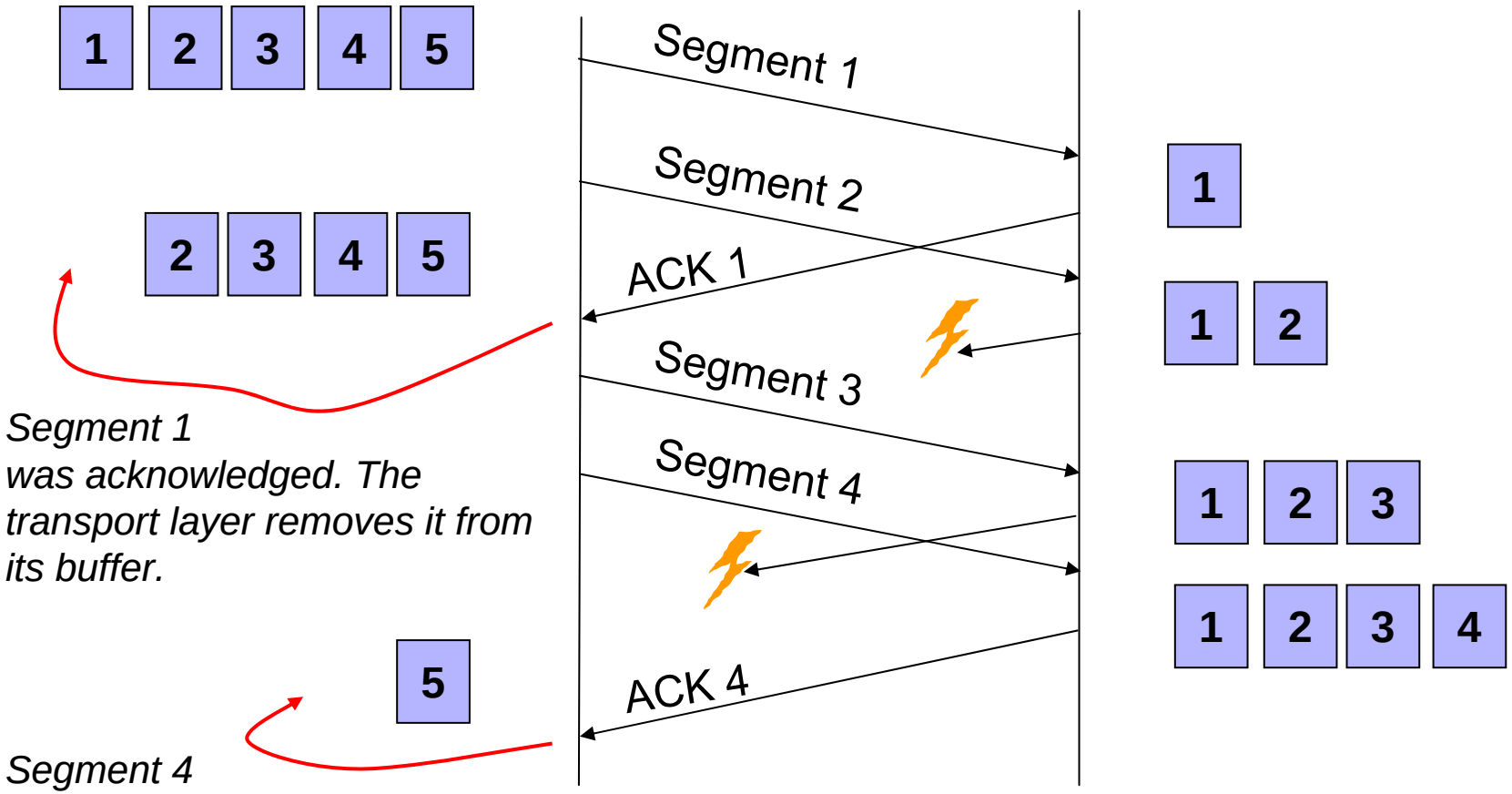
ACK-Verlust



Falscher Timer



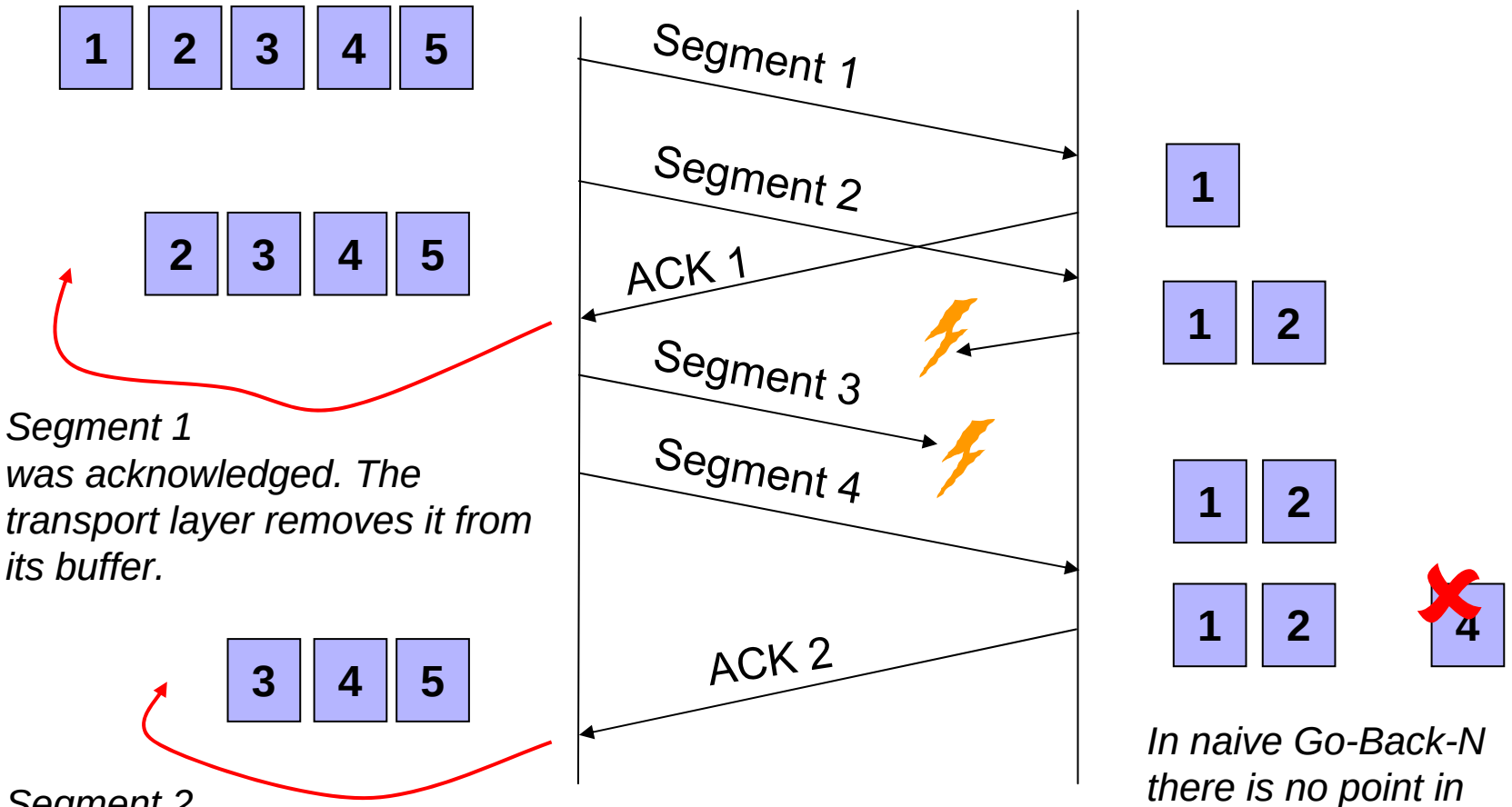
Go-Back-N



Segment 1 was acknowledged. The transport layer removes it from its buffer.

Segment 4 was acknowledged. The transport layer removes all segments up to and including segment 4.

Go-Back-N



Segment 1 was acknowledged. The transport layer removes it from its buffer.

Segment 2 was acknowledged. The transport layer removes it from its buffer.

In naive Go-Back-N there is no point in keeping segment 4. Why?

Go-Back-N

Go-Back-N erzielt einen höheren Durchsatz als Stop-and-Wait, indem es das Warten auf eine Quittung vor dem Senden der nächsten Dateneinheit vermeidet.

Mechanismus

- Sender kann mehrere Dateneinheiten senden bis er eine Quittung erhalten muss.
- Die maximale Anzahl der nicht quittierten Dateneinheiten ist begrenzt, typischerweise durch ein so genanntes Sliding Window.
- Variante 1:
  - Der Empfänger quittiert eine oder mehrere korrekt empfangene Dateneinheiten auf einmal.
  - Kumulative Sequenznummer gibt an, bis wohin die Daten korrekt empfangen wurden, d.h. es handelt sich um positive Quittungen.
  - Alle nicht quittierten aber gesendeten Daten werden vom Sender wiederholt
- Variante 2:
  - Nicht korrekt empfangene Dateneinheiten werden mit einer negativen Quittung (NACK) bestätigt.
  - Sender wiederholt daraufhin *ab* dieser Sequenznummer alle gesendeten Dateneinheiten.

Selective Reject ARQ

- Ziel: Erhöhung der Datenrate im Vergleich zu Stop-and-Wait. Reduzierung des Datenaufkommens im Vergleich zu Go-Back-N.

- Mechanismus
  - Der Sender kann mehrere Dateneinheiten senden, bis er eine Quittung erhalten muss. Die maximale Anzahl der nicht quittierten Dateneinheiten ist begrenzt. (Wie bei Go-Back-N)
  - Der Empfänger sendet eine negative Quittung, wenn er einen Fehler erkennt. Diese Quittung bezieht sich auf eine einzelne Dateneinheit.
  - Der Sender wiederholt genau die Dateneinheit mit der bei der negativen Quittung angegebenen Sequenznummer, d.h. nur die nicht korrekt empfangenen Dateneinheiten werden vom Sender wiederholt.

Quittungen bei TCP

Ereignis	Reaktion des TCP-Empfängers
Ankunft einer Dateneinheit in Reihenfolge. Alle Daten davor bereits quittiert. Keine Lücken.	Verzögerte Quittung. Bis zu 500 ms warten auf weitere reihenfolgetreue Dateneinheit. Wird keine weitere empfangen, dann Senden der Quittung.
Ankunft einer Dateneinheit in Reihenfolge. Alle Daten davor bereits quittiert. Bereits eine weitere reihenfolgetreue auf Quittung wartende Dateneineit. Keine Lücken.	Sofortiges Senden einer kumulativen Quittung. Beide Dateneinheiten werden quittiert.
Ankunft einer Dateneinheit mit einer nicht erwarteten höheren Sequenznummer. Erkennen einer Lücke.	Sofortiges Versenden einer duplizierten Quittung mit der Sequenznummer des nächsten erwarteten Bytes.
Ankunft einer Dateneinheit, die teilweise oder komplett eine Lücke bei den empfangenen Daten auffüllt.	Sofortiges Versenden einer Quittung, falls die Dateneinheit an der unteren Schranke der Lücke beginnt.

Umlaufzeit und Timeout

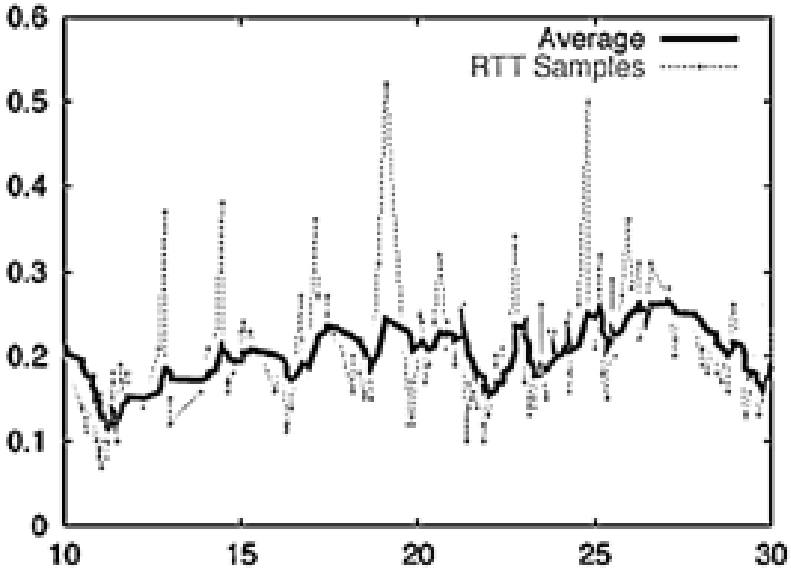
- Problem: Auf welchen Wert soll der TCP-Timeout gesetzt werden?
- Lösung: Wert sollte sich an der Umlaufzeit (Round Trip Time, RTT) orientieren und etwas größer sein
- Messung der Umlaufzeit
  - Timer (Granularität variiert, bis zu 500 ms) wird benutzt. Beim Ablauf wird ein Zähler jeweils inkrementiert.
  - SampleRTT
    - Gemessene Zeit vom Versenden der Dateneinheit bis zum Empfang der dazugehörigen Quittung.
    - Sendewiederholungen werden ignoriert.
    - Kumulativ quittierte TCP-Dateneinheiten werden nicht betrachtet.
- Glätten der gemessenen Werte, da diese stark schwanken
  - Es wird ein auf mehreren Messungen basierender Wert herangezogen: EstimatedRTT

Bestimmung der Umlaufzeit

- Nach jedem erfassten Wert für SampleRTT wird der geglättete Wert der Umlaufzeit folgendermaßen bestimmt

  EstimatedRTT := (1-α) * EstimatedRTT + α * SampleRTT

- Exponential Weighted Moving Average
  - Neue Werte werden höher gewichtet als alte Werte
  - Einfluss eines gemessenen Wertes sinkt exponentiell
- Typischer Wert für α: 0.125



Bestimmen des Timeout-Werts

- Timeout sollte auf einen etwas höheren Wert gesetzt werden als EstimatedRTT (= „Sicherheitszuschlag“)
- Dieser „Sicherheitszuschlag“ sollte bei hohen Schwankungen größer gewählt werden

  Deviation := (1- α) * Deviation + α * | SampleRTT – EstimatedRTT |

  Timeout := EstimatedRTT + 4 * Deviation

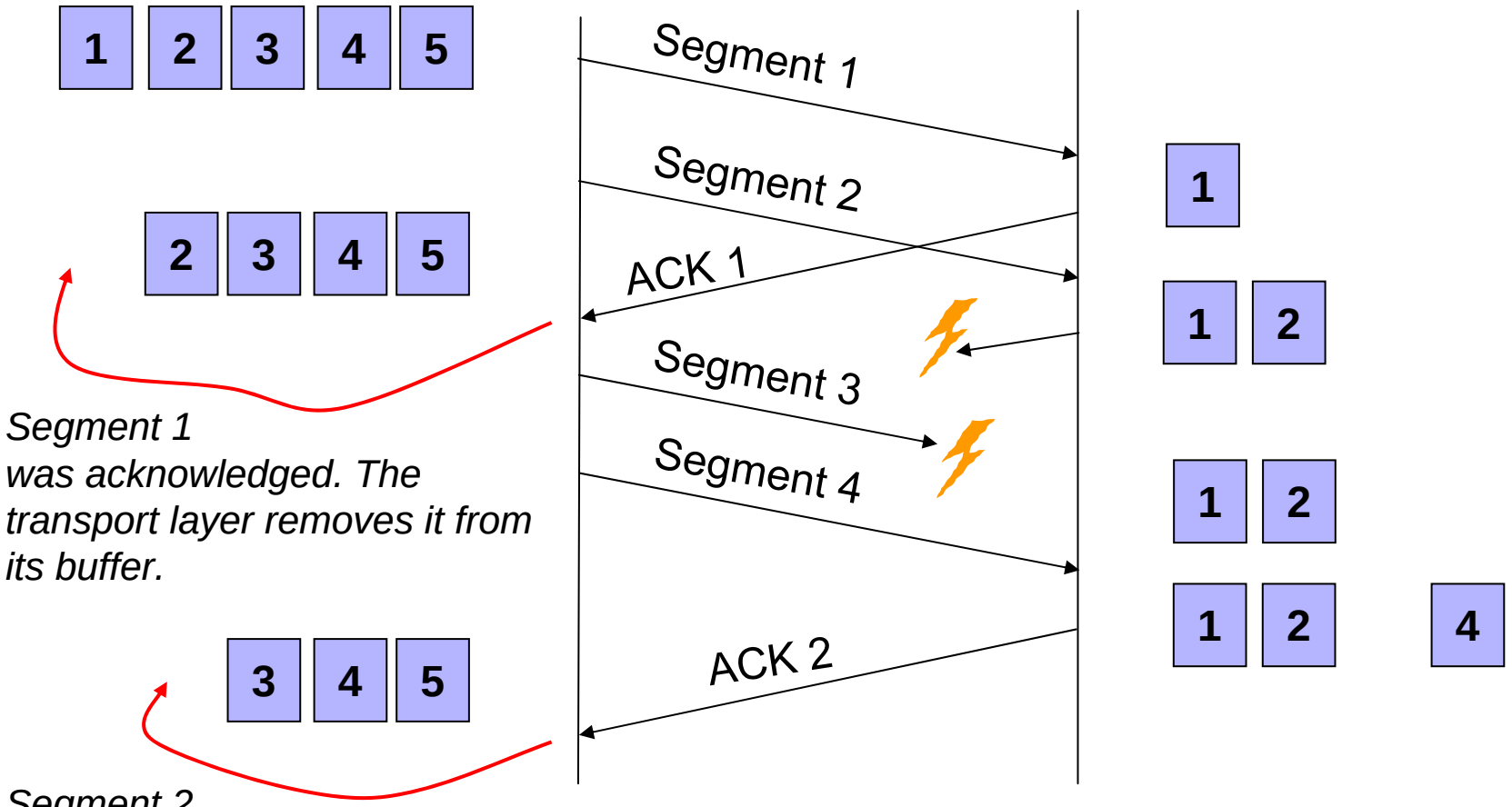
- Neben diesem so genannten Retransmission Timer verfügt TCP noch über eine Reihe weiterer Timer:
  - Persist Timer
  - Keepalive Timer
  - 2MSL-Timer

  Diese Timer werden statisch mit Werten belegt.

Fragen …

- Vergleichen Sie ACK und NACK Verfahren! – Lässt sich mit NACKs ein zuverlässiger Dienst bereitstellen?
- Diskutieren Sie die Grundprinzipien eines zuverlässigen Transportdiensts für Multicast, d.h. für eine Paketvermittlung, bei der Pakete im Netz dupliziert und an viele Empfänger versandt werden. Lässt sich ein solcher Dienst überhaupt effizient verwirklichen?
- Beschreiben Sie die Probleme die durch einen falsch gewählten Timer beim Stop-and-Wait entstehen!
- Beschreiben Sie das bei „normalem“ TCP verwendete ARQ-Verfahren (d.h. ohne SACK-Erweiterung). Wie werden Aspekte eines Selective Reject ARQ integriert?

Reliable Transmission with Go-Back-N



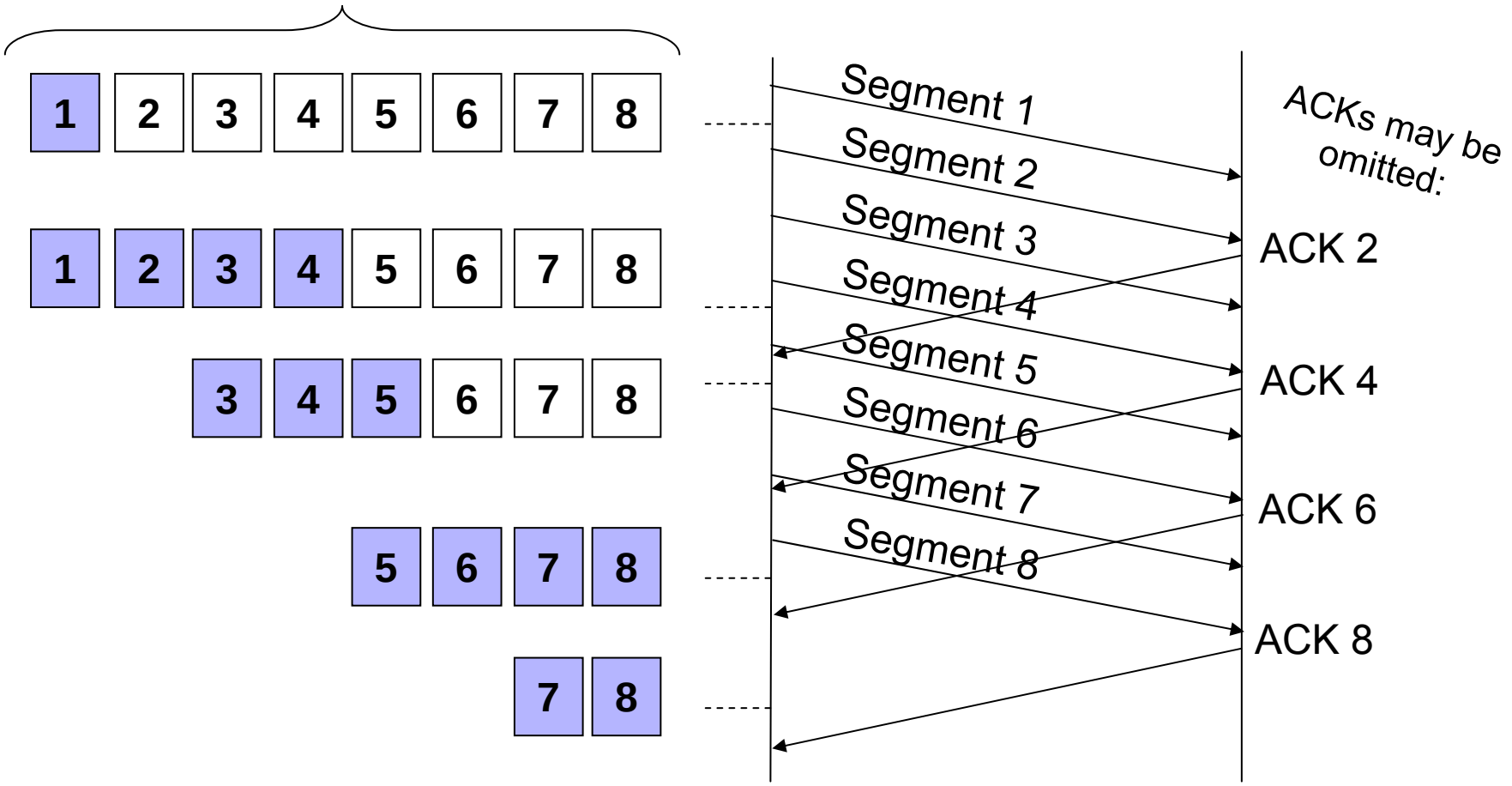
Segment 1 was acknowledged. The transport layer removes it from its buffer.

Segment 2 was acknowledged. The transport layer removes it from its buffer.

Note: TCP acknowledges by saying which byte it expects next in the stream.

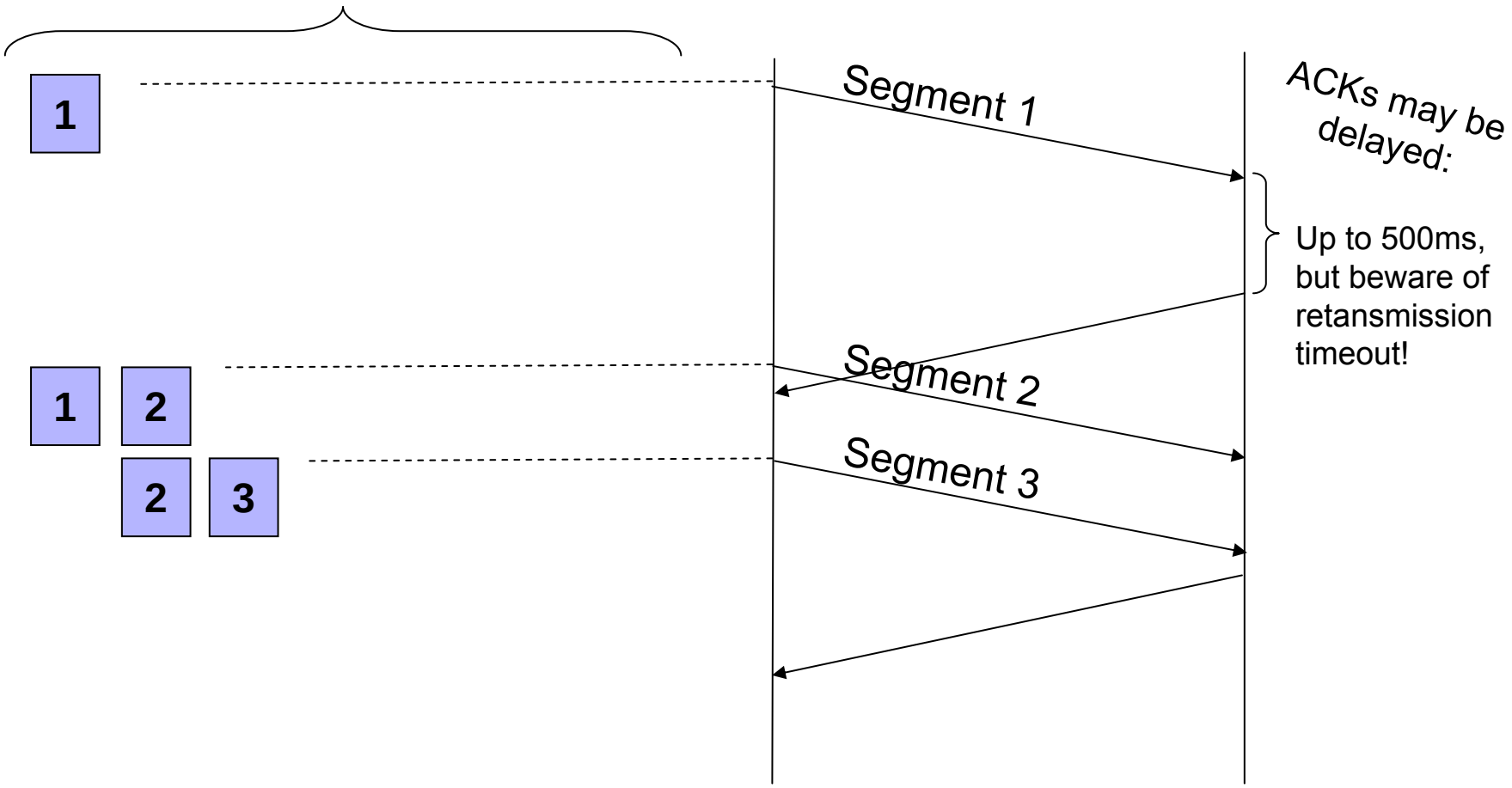
TCP Acknowledgement & Retransmission

Bulk data transfer: Application fills the send buffer with a large amount of data.



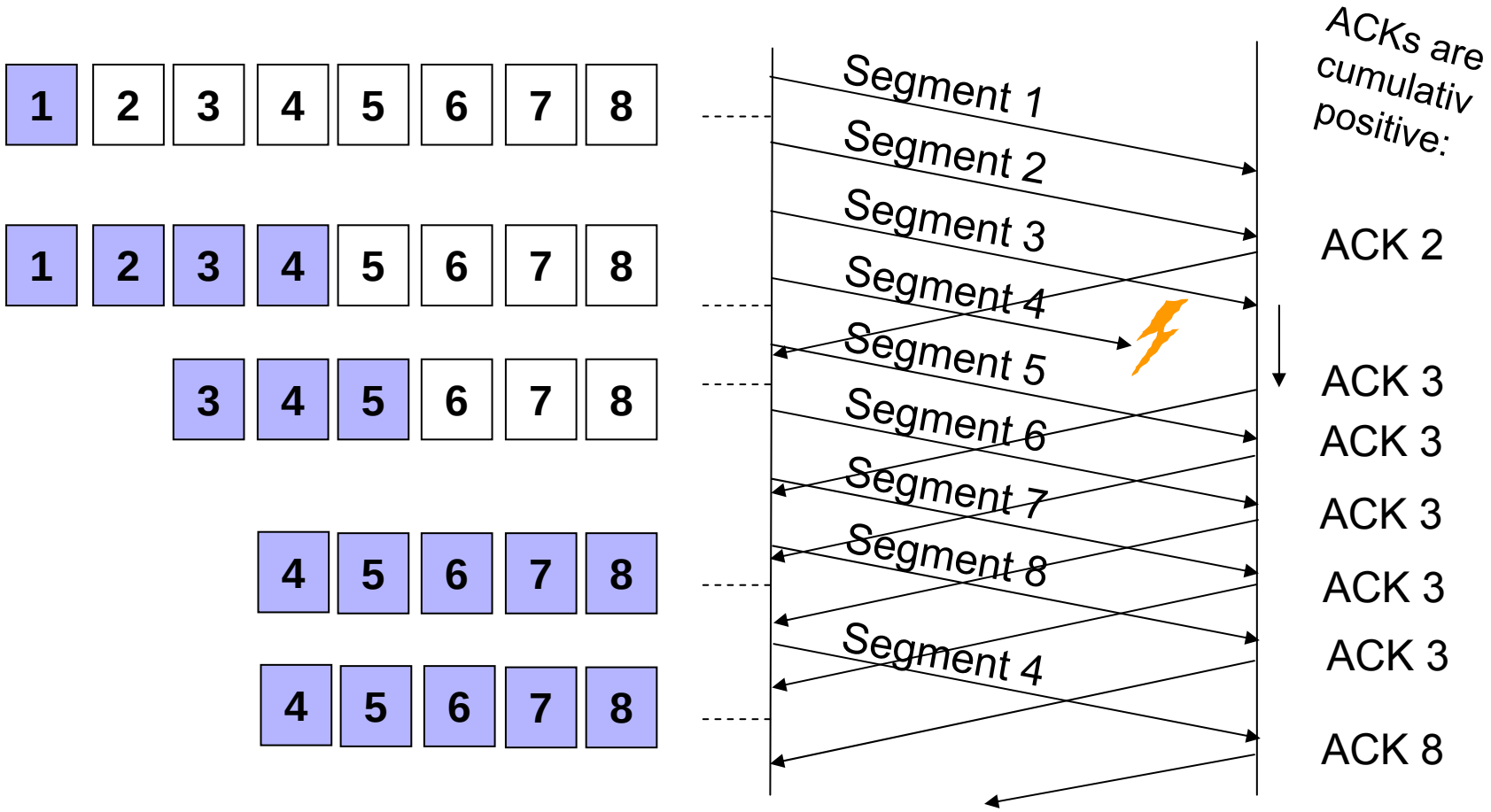
TCP Acknowledgement & Retransmission

Interactive traffic



TCP Acknowledgement & Retransmission

Single packet loss can be detected by reception of duplicate ACKs:



Flowcontrol

… as provided by TCP

Flusssteuerung – Übersicht

- Flusssteuerung adressiert das Problem, wie viele Dateneinheiten vom Sender hintereinander gesendet werden dürfen, ohne dass der Speicher beim Empfänger überläuft bzw. dessen Verarbeitungsgeschwindigkeit überschritten wird.
- Eine gute Flusssteuerung soll einfach, fair und stabil sein, sowie die Ressourcen des Netzes möglichst wenig belasten.
- Man unterscheidet zwei Varianten
  - ***Closed Loop***, d.h. eine Rückkopplung, bei der der Empfänger selbst verhindert, dass er "überschwemmt" wird. Hierzu adaptiert der Sender seinen Datenstrom entsprechend.
  - ***Open Loop***, d.h. eine Beschreibung des zulässigen Verkehrs mit entsprechender Ressourcenreservierung und Überwachung des eingehenden Verkehrs.
- Achtung: Die Flussteuerung verwendet oft ähnliche Mechanismen wie ARQ, dient aber einem völlig anderen Ziel.

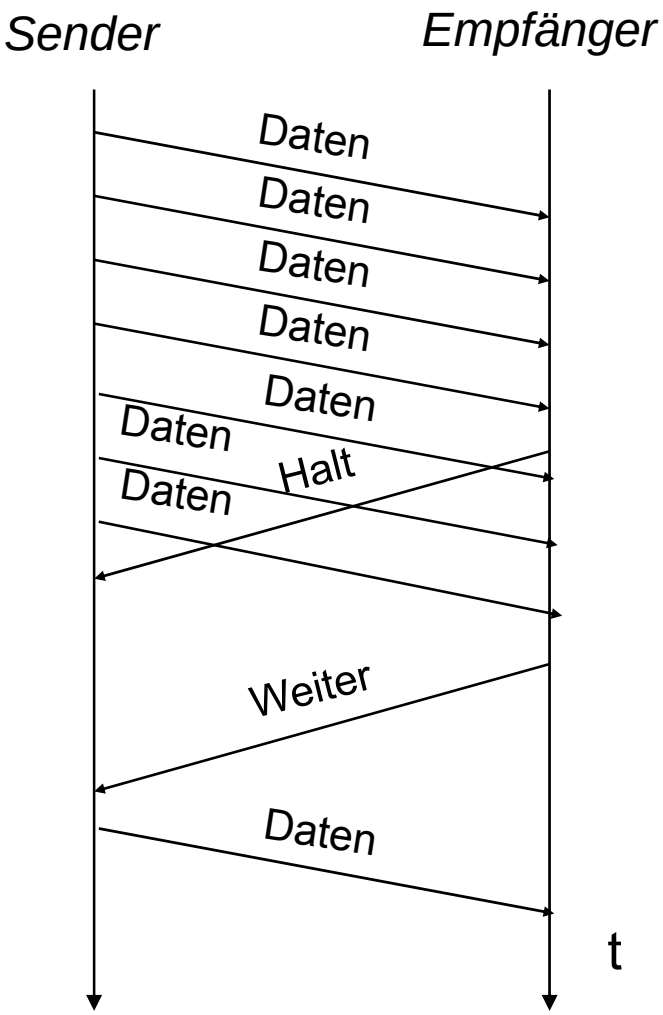
Halt-/Weiter-Meldungen

Flusssteuerung mit Halt-/Weiter-Meldungen ist die einfachste Methode:

- Sender-Empfänger-Flusssteuerung
- Kann der Empfänger nicht mehr Schritt halten, schickt er dem Sender eine **Halt**-Meldung.
- Ist ein Empfang wieder möglich, gibt der Empfänger die **Weiter**-Meldung.

Beispiel: Protokoll XON/XOFF

- Mit ISO 7-Bit-Alphabetzeichen.
- XON
- XOFF
- Nur auf Vollduplex-Leitungen verwendbar.



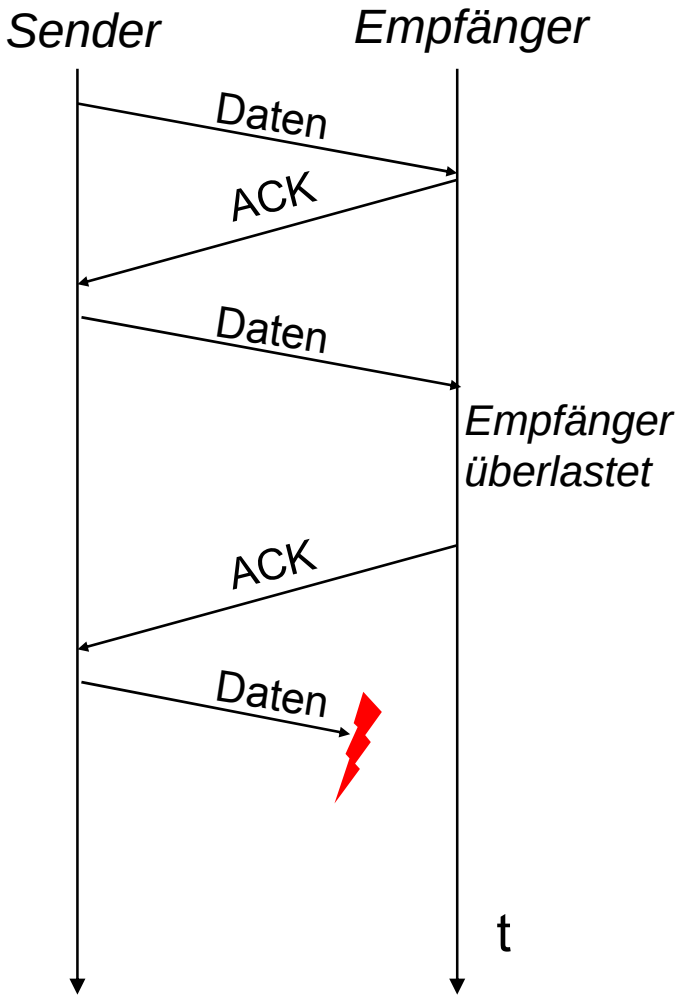
Implizite Flusssteuerung

Funktionsweise

- Durch Zurückhalten der Quittung (z.B. ACK/NAK) kann der Sender gebremst werden.
- Das bedeutet, dass ein Verfahren zur Fehlererkennung für die Flusskontrolle mitbenutzt wird.

Problem

- Der Sender kann nicht mehr unterscheiden,
  - ob sein Paket völlig verloren ging, oder
  - ob der Empfänger die Quittung wegen Überlast zurückgehalten hat.



Kreditbasierte Flusssteuerung

Funktionsweise

- Der Empfänger räumt dem Sender einen mehrere Transfereinheiten umfassenden Sendekredit ein.
- Ist der Kredit (ohne neue Kreditgewährung) erschöpft, stellt der Sender den Transfer ein.
- Dazu ist aber eine verstärkte Fehlerkontrolle, z.B. für den Verlust der neuen Kreditgewährung, erforderlich.

Realisierungsmöglichkeiten

- Explizite Kreditgewährung:
  - Empfänger teilt dem Sender explizit den aktuellen Kredit mit.
- Kreditfenster („***Sliding Window***“):
  - Mit jedem quittierten Paket wird das Kreditfenster verschoben.

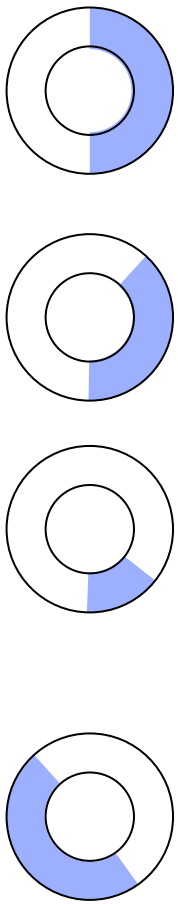
Beispiel zum Sliding Window

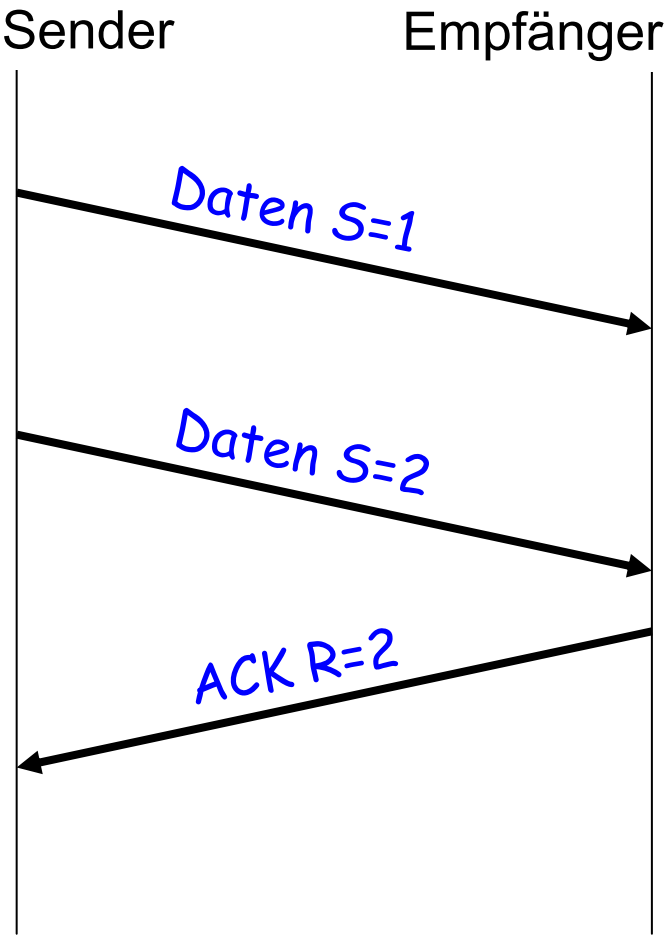
Empfänger teilt Sender die Größe seines Empfangs-puffers mit.

Sender sendet mit Sequenz-nummern am Anfang des offenen Sendefensters.

Dabei schließt sich das Fenster Schritt für Schritt.

Der Empfang einer Quittung öffnet das Fenster wieder.





Achtung: Fehler- und Flusskontrolle werden vermischt!

Bemerkungen zum Sliding Window

- Das Sendefenster bzw. Sliding Window ist zunächst ein abstraktes Konstrukt zur Flusskontrolle.
- Wäre nicht die Vermischung mit der Fehlerkontrolle, müssten keine Daten gepuffert werden, sondern nur die aktuelle Sequenznummer und die Fenstergröße gespeichert werden.
- Aufgrund der Vermischung mit der Fehlerkontrolle müssen aber tatsächlich Daten in der Transportschicht des Senders gepuffert werden, damit diese für eventuelle Übertragungswiederholgungen zur Verfügung stehen.
- Umgekehrt puffert typischerweise der Empfänger die Daten ebenfalls innerhalb der Transportschicht, um den Übertragungsmechanismus von der Anwendung zu entkoppeln.
- Insbesondere bei Multimedia-Anwendungen ziehen sich Flusssteuerungsmechanismen bis tief in die Anwendung hinein. Dort werden jedoch häufig ratenbasierte Verfahren verwendet.

Zero Window Probing bei TCP

Problem einer Fenstergröße von Null:

- Endsystem B hat an Endsystem A gemeldet, dass sein Empfangspuffer voll ist, damit ist das Empfangsfenster leer, d.h. RcvWindow=0
- Endsystem A kann keine Daten mehr senden.
- Da Größe des Empfangsfensters aber in Dateneinheiten bzw. Quittungen übermittelt wird, kann Endsystem B Endsystem A erst dann informieren, dass wieder freier Empfangspuffer existiert, wenn Daten fließen. Widerspruch!

Lösung:

- Endsystem A muss auch bei Fenstergröße Null noch Dateneinheiten (mit 1 Byte) senden, die von Endsystem B quittiert werden.
- Zero Window Probing wird durch den so genannten Persistent-Timer gesteuert.

Weitere Timer in TCP

- Retransmission Timer:
- Persist Timer
  - initiiert regelmäßige Nachfragen nach der Fenstergröße, auch wenn der Empfänger sein Empfangsfenster schließt
  - TCP verwendet ein Exponential Backoff Verfahren zur Berechnung der Timer-Werte
- Keepalive Timer
  - Erkennt, wenn der Partner Probleme hat
  - über eine TCP-Verbindung im Idle-Zustand fließen keine Daten
  - ist nicht Bestandteil der TCP-Spezifikation, stellt eine Option dar
  - kann dazu führen, dass bestehende Verbindungen terminieren (z.B. bei einem temporären Problem auf der Vermittlungsschicht)
  - kann Servern helfen, Ressourcen nicht unnötig zu belegen, falls Client abgestürzt ist
- 2MSL-Timer
  - Misst die Zeit, die eine Verbindung im TIME_WAIT-Zustand verbringt

Congestion Control

… as provided by TCP

Staukontrolle in TCP (1)

Die Staukontrolle soll Überlastsituationen im Netz vermeiden

- 1986 kam es zu ernsthaften Problemen im Internet, die durch Überlast hervorgerufen wurden. Der Durchsatz einzelner Verbindungen sank um ca. drei Größenordnungen auf nur noch 40bit/s.
- Seit 1989 ist die Staukontrolle fester Bestandteil des TCP-Protokolls

TCP verbindet Fluss-, Fehler- und Staukontrolle in einem einheitlichen Mechanismus

- Flusskontrolle vermeidet Überlast beim Empfänger
- Fehlerkontrolle garantiert die vollständige, korrekte, reihenfolgetreue Übertragung
- Staukontrolle versucht vorhandene Ressourcen im Netz möglichst optimal aufzuteilen

TCP definiert „optimal“ als:

- Bandbreite am Engpass wird zu gleichen Teilen auf alle Verbindungen verteilt (= „TCP-Fairness“)

Staukontrolle in TCP (2)

Herausforderung an die Staukontrolle im Internet

- Es gibt keine direkte Unterstützung durch das Netz, d.h. Überlastsituationen können nur durch Beobachtung vermutet werden.

Mechanismus

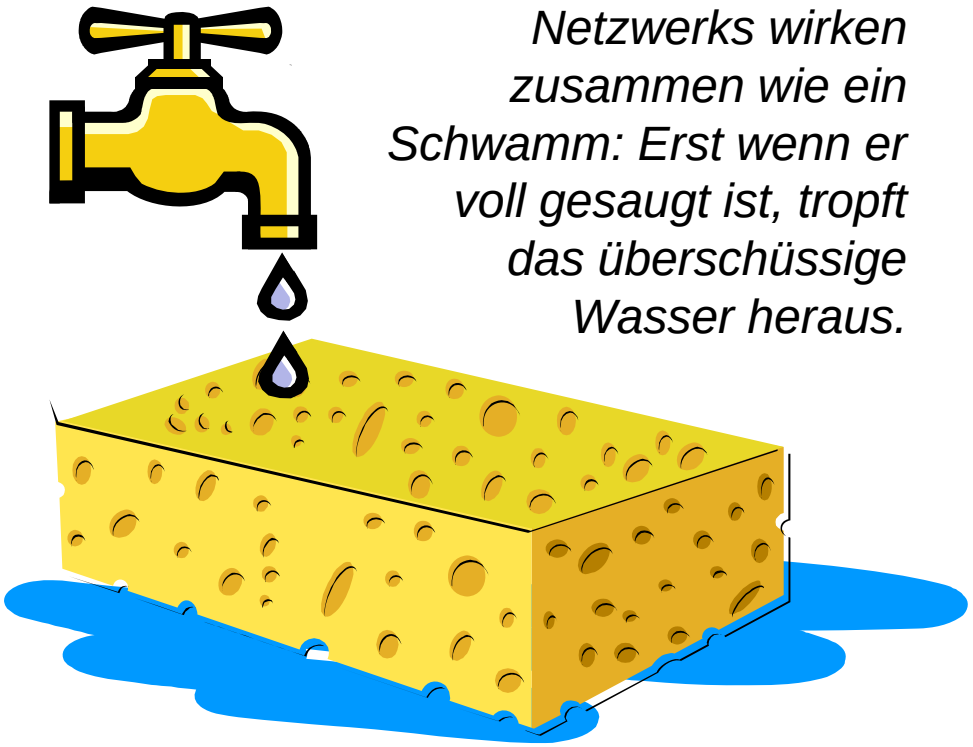
- Bei ausbleibender Quittung wird Stausituation vermutet
- Daraufhin erfolgt eine Begrenzung der Datenrate, die ein TCP-Sender senden darf
- Sender testet, mit welcher maximalen Datenrate er senden kann, indem er die Datenrate langsam erhöht
- Hierzu müssen zwei neue Variablen in den Sliding-Window-Mechanismus von TCP eingeführt werden
  - *Staukontrollfenster* (CWnd)
  - *Schwellenwert* (SSTresh)

Ursachen für Stau im Internet (1)

- Zwei Gründe für Stau im Internet:
  - Bei steigender Last steigt die RTT und die Schwankung der RTT (vgl. Warteschlangentheorie), somit wächst die Zahl unnötiger Übertragungswiederholungen. (=Staukollaps Typ 1)
  - Bei noch größerer Last steigt auch die Zahl verworfener Pakete, somit wächst die Zahl der schlussendlich erfolglos über ein Teilstück transportierten Pakete. (=Staukollaps Typ 2)
- Im ersten Fall wird das Grundgesetz einer konstanten Zahl von im Netz befindlichen Paketen verletzt:
  - Noch bevor ein Paket vom Empfänger entnommen wurde, oder bevor es von einem Router entfernt wurde, schickt der Sender ein neues Paket ins Netz.
- Im zweiten Fall wird die Leistungsfähigkeit des Netzes noch weiter durch unsinnige Arbeit reduziert.

Ursachen für Stau im Internet (2)

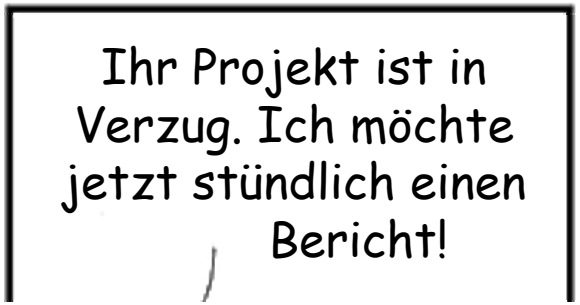
Staukollaps – Typ 1



Warteschlangen des Netzwerks wirken zusammen wie ein Schwamm: Erst wenn er voll gesaugt ist, tropft das überschüssige Wasser heraus.

Staukollaps – Typ 2

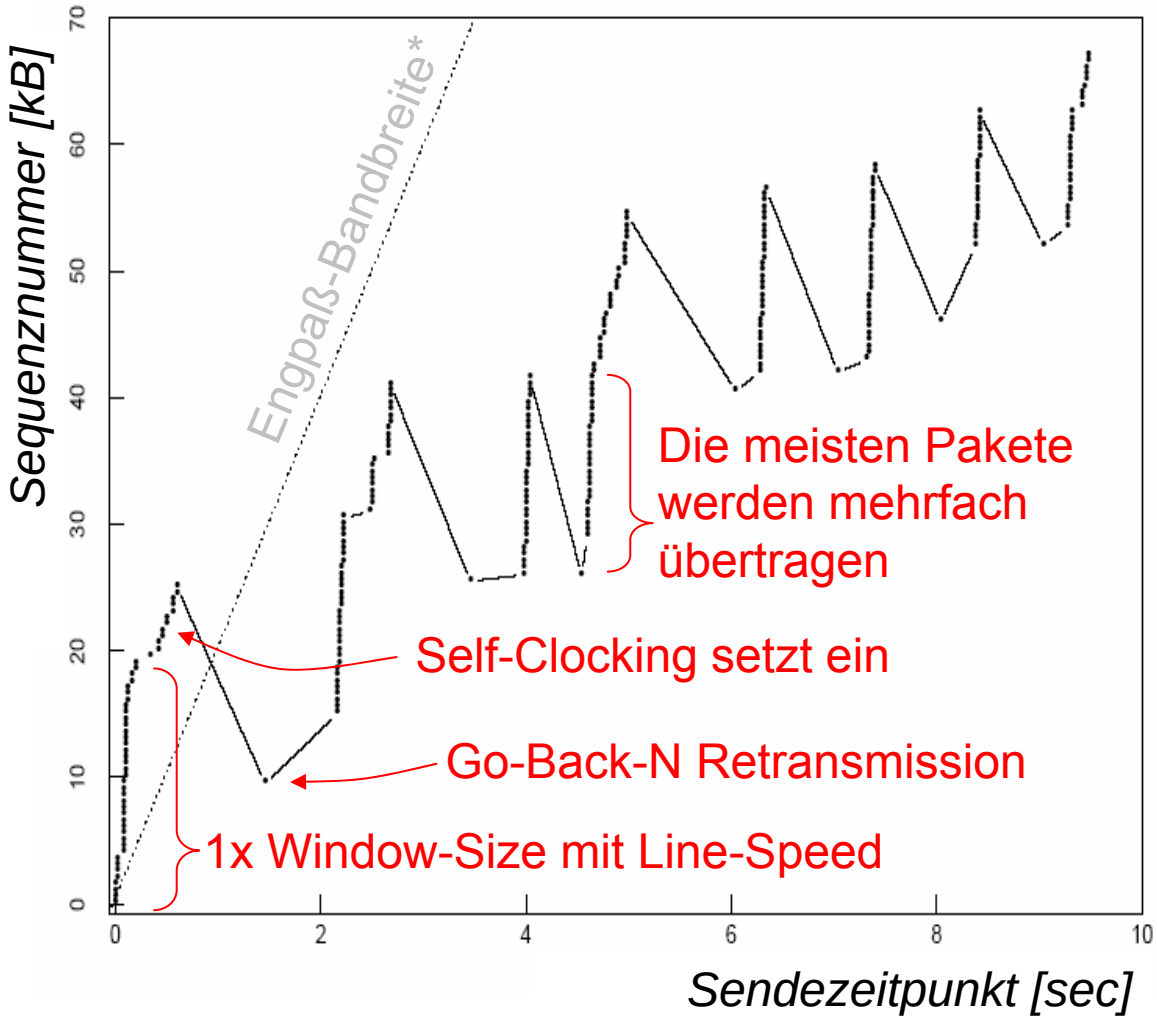
Unter Last wird die Leistungsfähigkeit des Netzes zusätzlich geschwächt.



Go-back-N TCP (1)

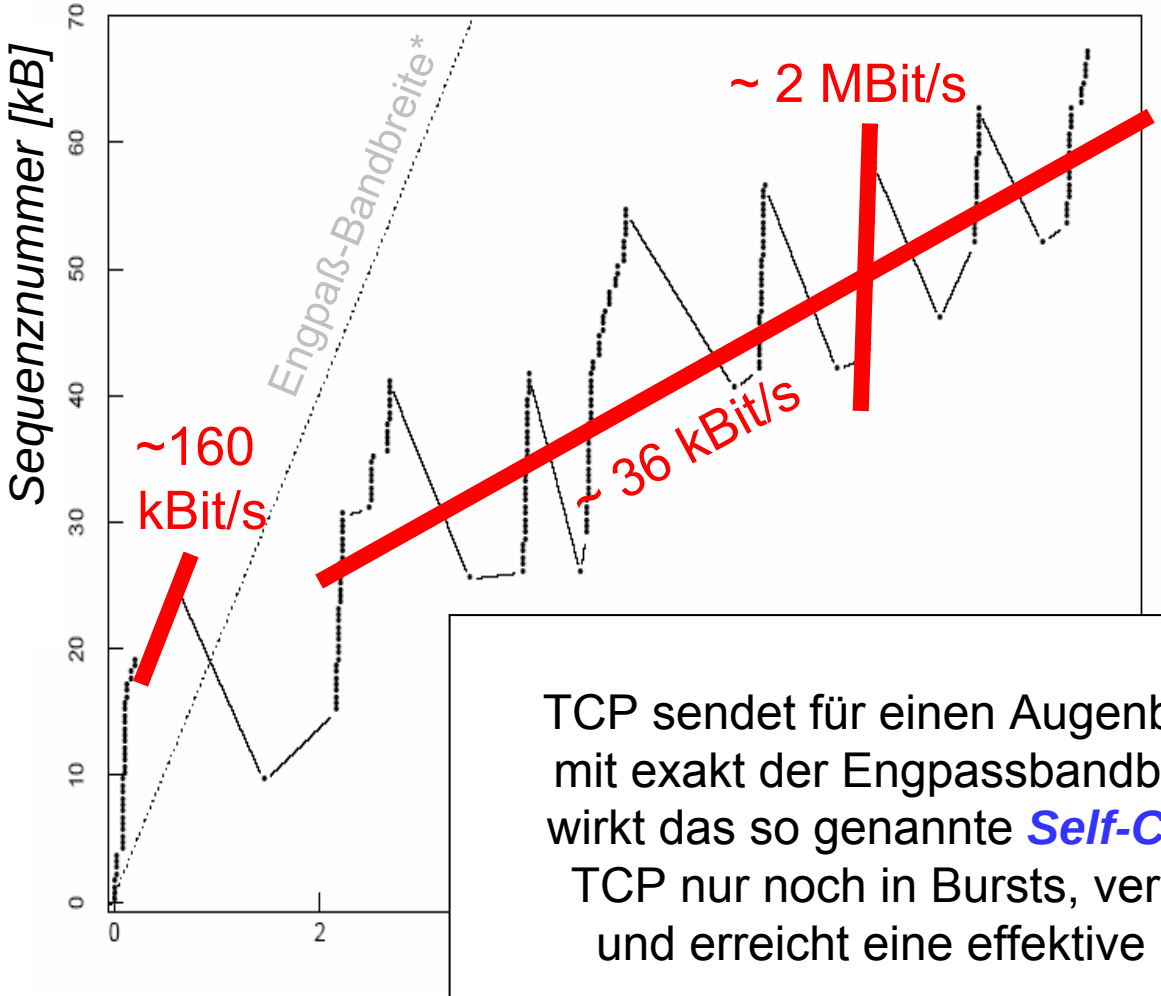
TCP vor 1988:

- Fehlerkontrolle mit Go-back-N, d.h. nach Timeout beim ersten unbestätigten Paket starten
- Flusskontrolle, d.h. Empfänger muss regelmäßige Window-Updates schicken, um den Datenfluss aufrecht zu erhalten.
- Aber keine Staukontrolle!



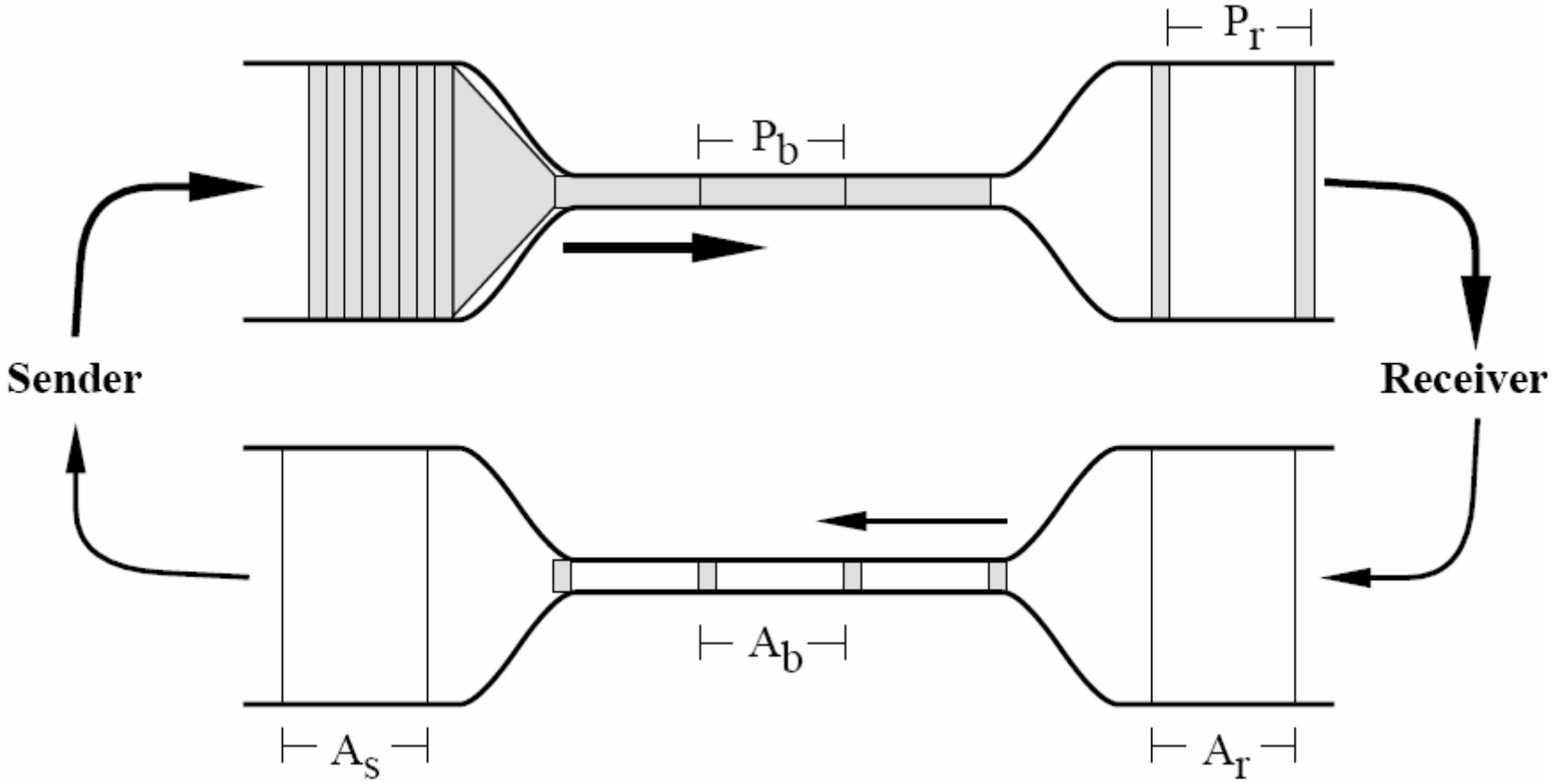
*Topologie: Zwei 10MBit/s Ethernet-Subnetze mit 0.16MBit/s Mikrowellen-Link verbunden

Go-back-N TCP (2)



TCP sendet für einen Augenblick kurz nach dem Start mit exakt der Engpassbandbreite. In diesem Moment wirkt das so genannte ***Self-Clocking***. Danach sendet TCP nur noch in Bursts, verliert die meisten Pakete und erreicht eine effektive Nutzung von nur 23%.

TCP Self-Clocking



Quelle: Van Jacobson, Michael J. Karels. *Congestion Avoidance and Control*. Proceedings of SIGCOMM'88, Stanford, CA, August 1988.

Staukollaps (1)

- Das klassische Go-back-N TCP (vor 1988) ist anfällig für den Staukollaps:
  - Lastschwankungen und verfrühte Übertragungswiederholungen treiben das Netz in einen gesättigten Zustand mit vollen Warteschlangen und hohen Paketumlaufzeiten.
  - Die effiziente Self-Clocking Phase ist also instabil.
- Dabei wirkt die an sich wünschenswerte Eigenschaft des Self-Clockings Stau-verstärkend:
  - Vorteil des Self-Clocking: Ein neues Paket darf erst gesendet werden, wenn ein alter Paket bestätigt wurde.
  - Nachteil des Self-Clocking: Sobald das Netz ein Paket aus den vollen Warteschlangen ausgeliefert hat, kommt schon wieder das nächste nach.
- Der durch Fehler- und Flusskontrolle erzeugte Self-Clocking-Mechanismus ist anfällig für den Staukollaps.

Staukollaps (2)

- Ziel muss es sein …

  … das System rasch in den Bereich des Self-Clocking zu bringen und dauerhaft dort zu halten.

- Damit wird erreicht …

  … dass TCP die Senderate beibehält, die der Engpass momentan verkraftet (unter Berücksichtigung anderer Verkehrsströme).

- Self-Clocking kann jedoch nicht diese optimale Rate finden:
  - Wird sie unterschritten bleibt Kapazität ungenutzt.
  - Wird sie überschritten bauen sich lange Warteschlangen auf, die durch Self-Clocking nicht mehr abgebaut werden können.
- Nur solange die Flusskontrolle den Sender stärker begrenzt als ein eventueller Engpass im Netz, arbeitet Self-Clocking stabil.

TCP Congestion Control (1)

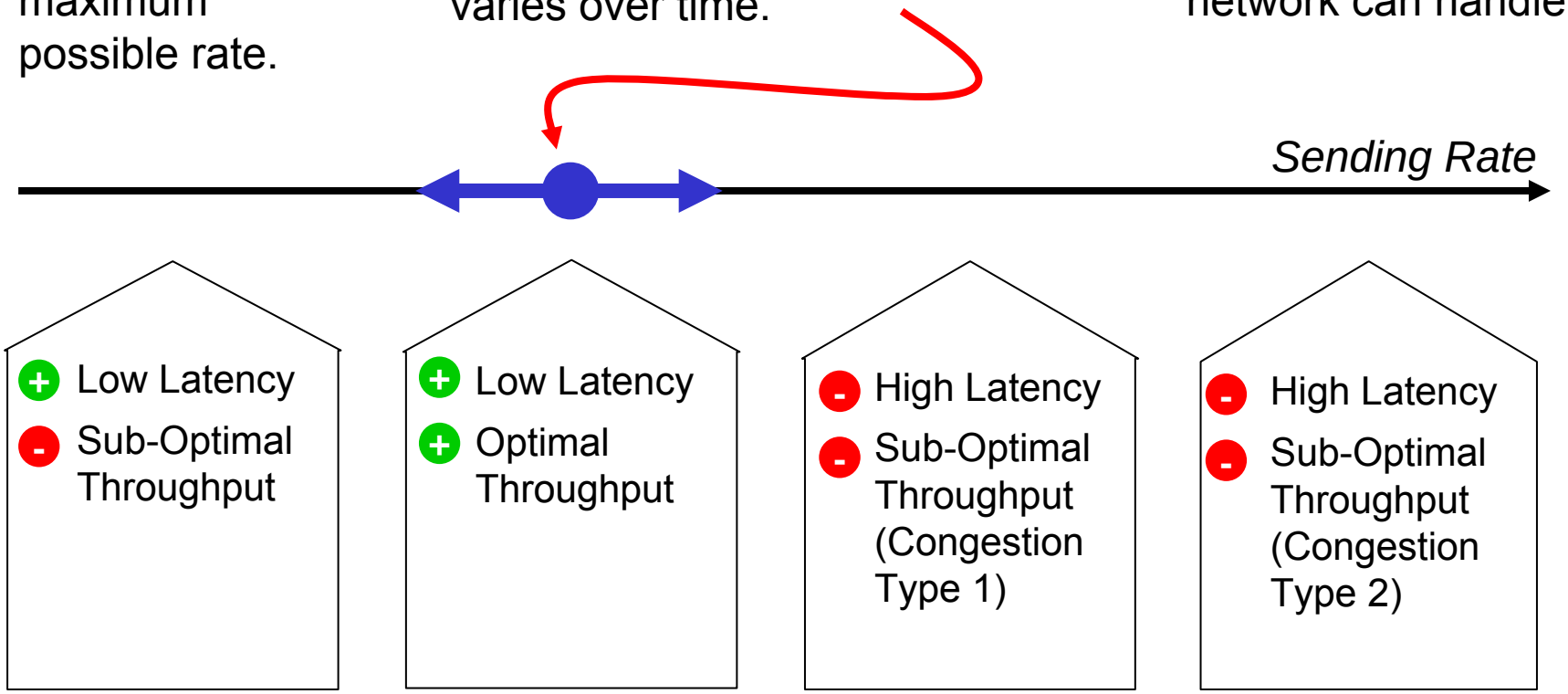
Underutilization:

TCP does not send at the maximum possible rate.

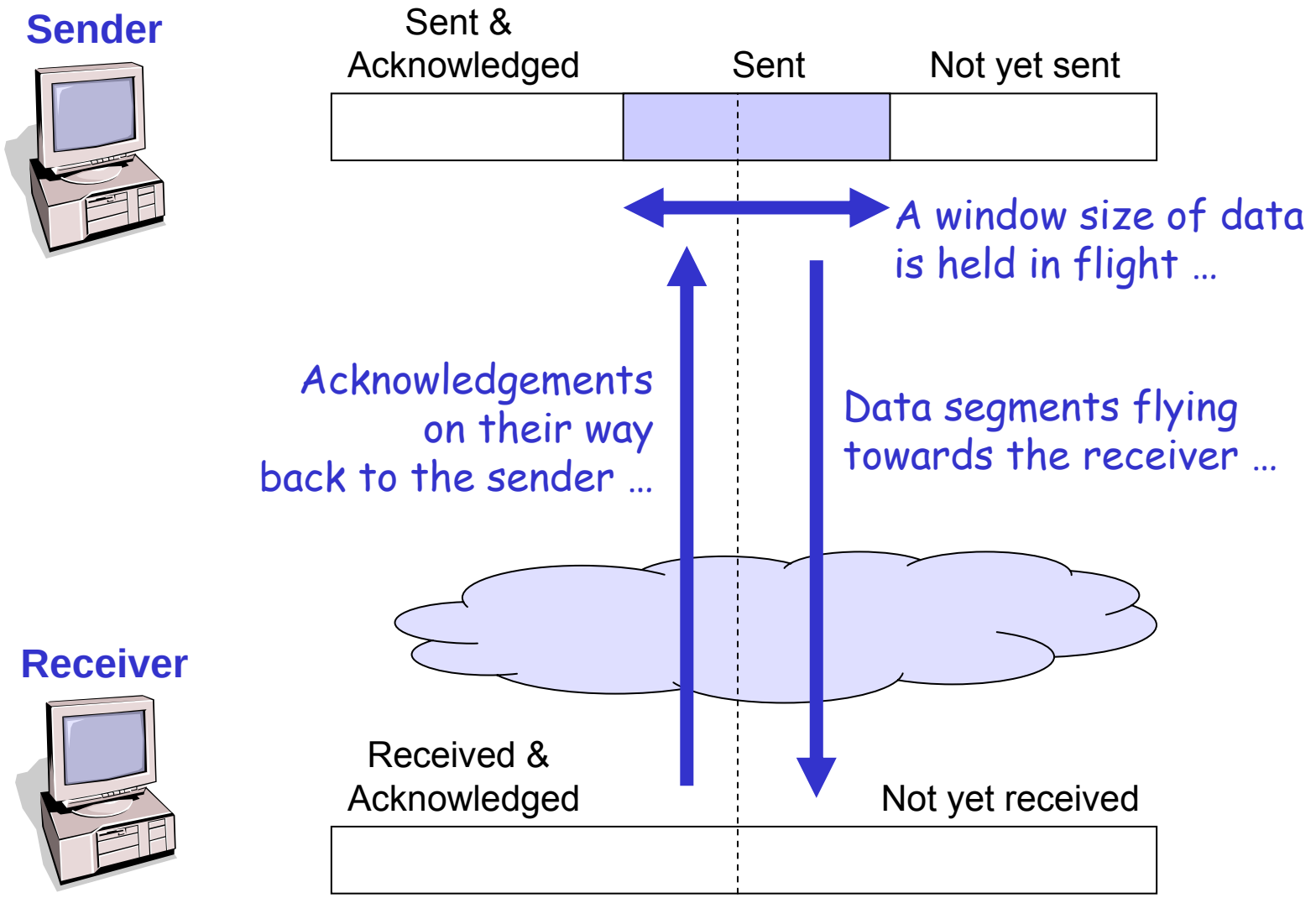
The ideal sending rate is determined by the bandwidth demand at the bottleneck. It depends on all connections sharing the bottleneck and quickly varies over time.

Congestion:

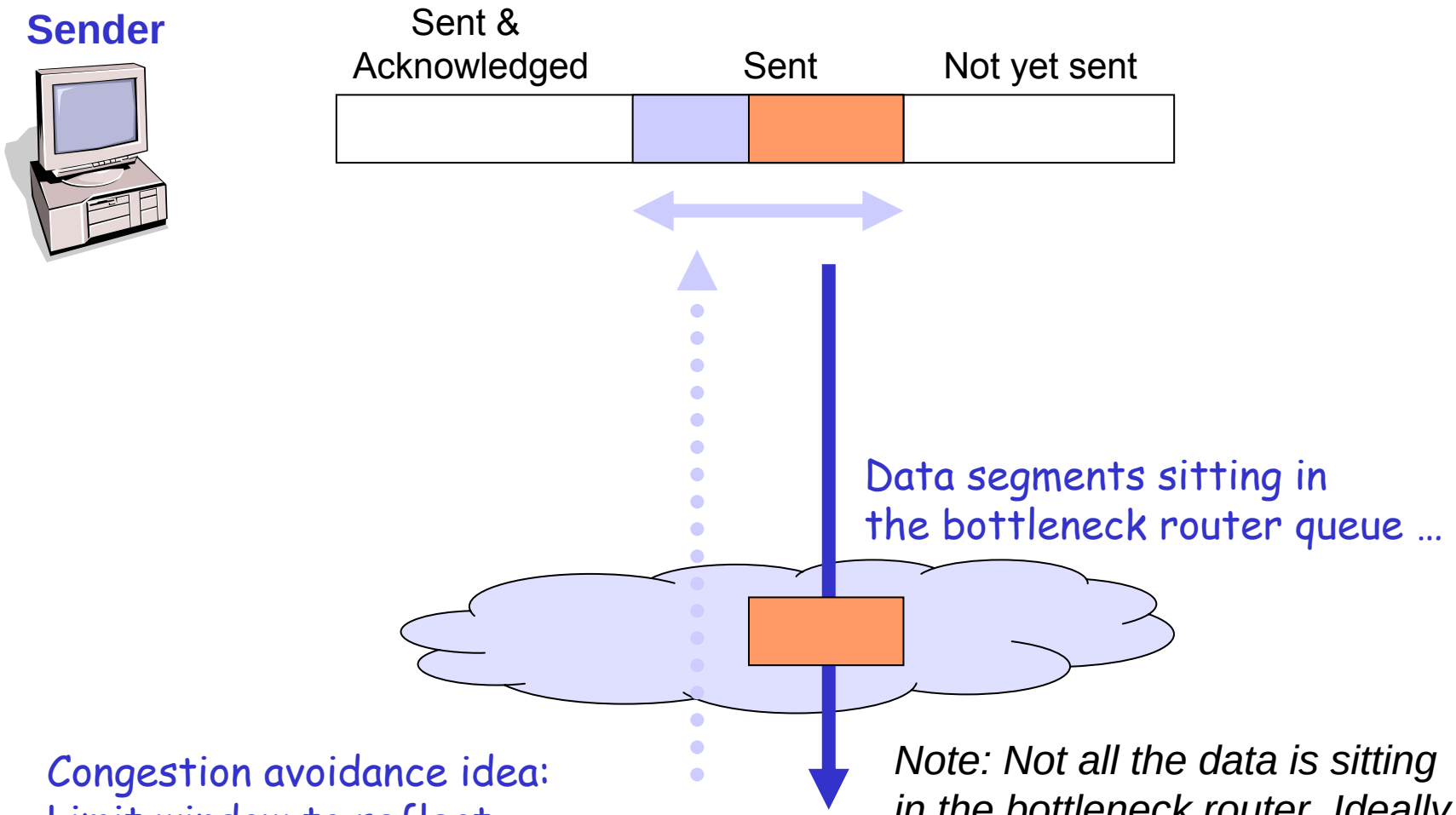
TCP sends more data than the network can handle.



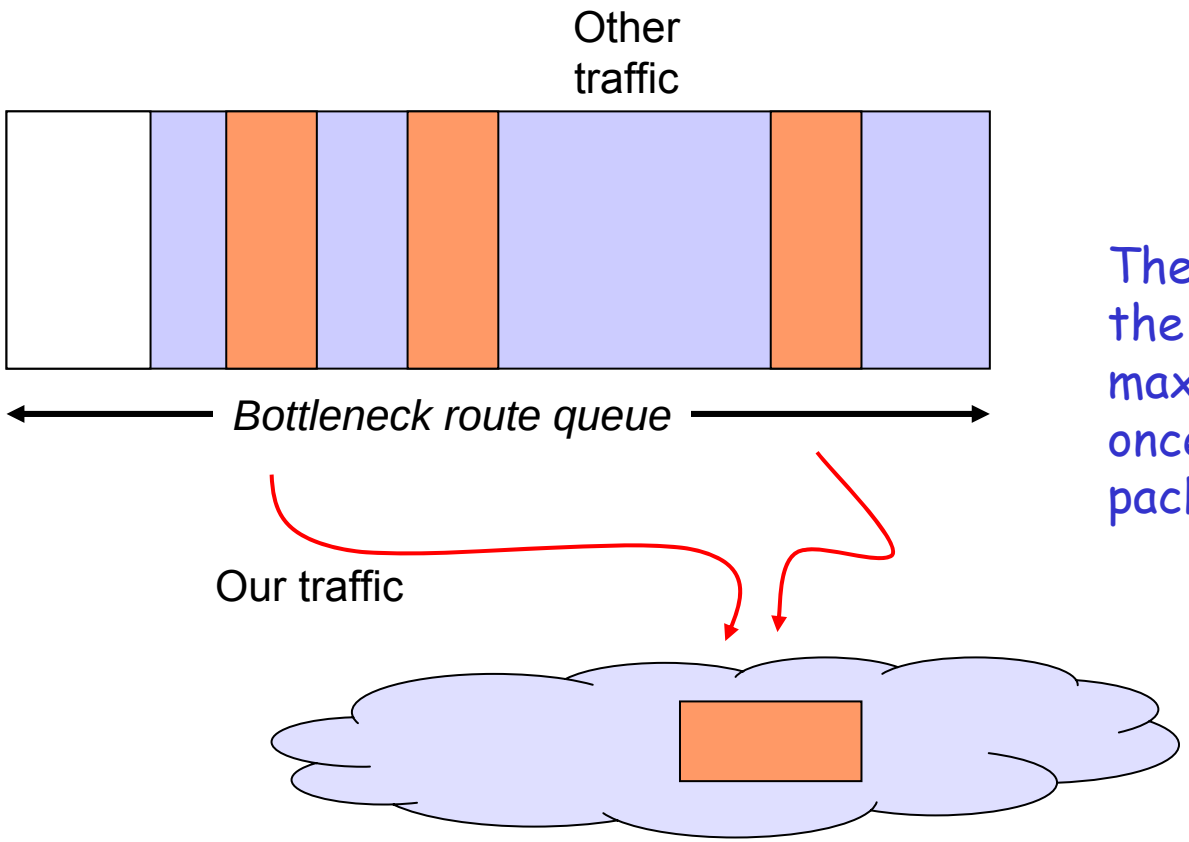
TCP Congestion Control (2)



TCP Congestion Control (3)

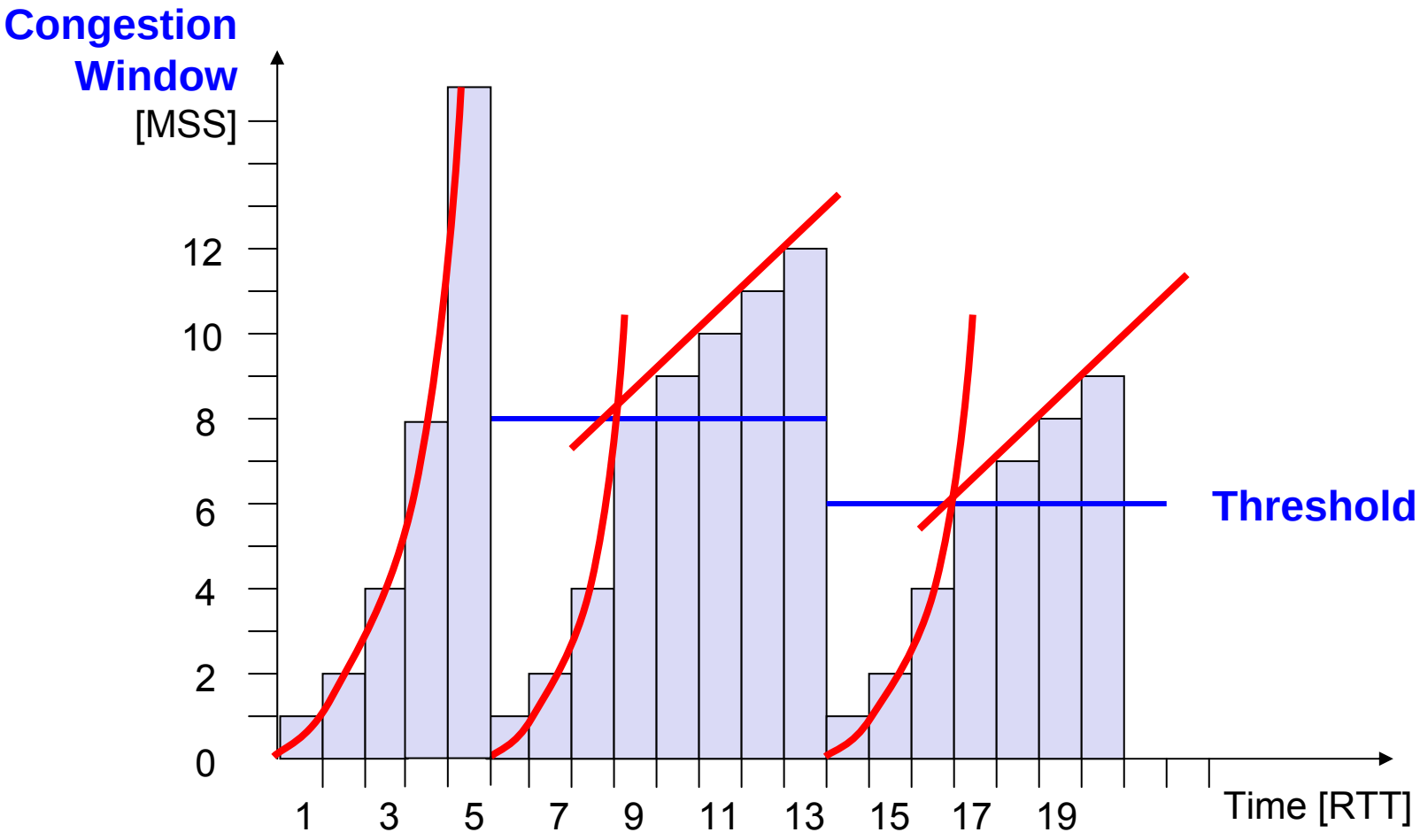


TCP Congestion Control (4)



The queue is full, i.e. the window is maximally open when once in a while a packet is dropped.

TCP Tahoe Overview



TCP Tahoe

Zwei Grundmechanismen vermeiden den Staukollaps:

- Berücksichtigung der ***RTT Schwankung*** bei der Berechnung der Retransmission Timeouts. Damit werden verfrühte Übertragungswiederholungen vermieden.
- Erweitere TCP um ein ***Staukontrollfenster***. Dieses begrenzt die im Netz befindliche Datenmenge (vgl. „Wasser im Schwamm“) unabhängig vom Flusskontrollfenster, das der Empfänger festsetzt.

Das Staukontrollfenster hält die Senderate im Self-Clocking-Bereich. Es wird zu Beginn einer TCP-Verbindung anders kontrolliert, als danach:

- ***Slow-start*** – Fange immer langsam an, statt gleich ein ganzes Flusskontrollfenster ins Netz zu feuern, d.h. setze Staukontrollfenster anfangs auf ein Segment. Dadurch setzt sofort das Self-Clocking ein.
  Um rasch die maximal von Engpass verkraftete Rate zu erreichen, verdopple das Staukontrollfenster pro Paketumlaufzeit (=exponentieller Zuwachs)
- ***Congenstion Avoidance*** – Interpretiere jedes verlorene Paket als Staunachricht* und halbiere das Staukontrollfenster. Läuft die Übertragung wieder fehlerfrei erhöhe das Staukontrollfenster um nur noch ein Segment pro Paketumlaufzeit (=linearer Zuwachs).

* Messungen ergaben, dass weniger als 1% der Pakete durch Übertragungsfehler verloren gehen.

TCP Tahoe Details

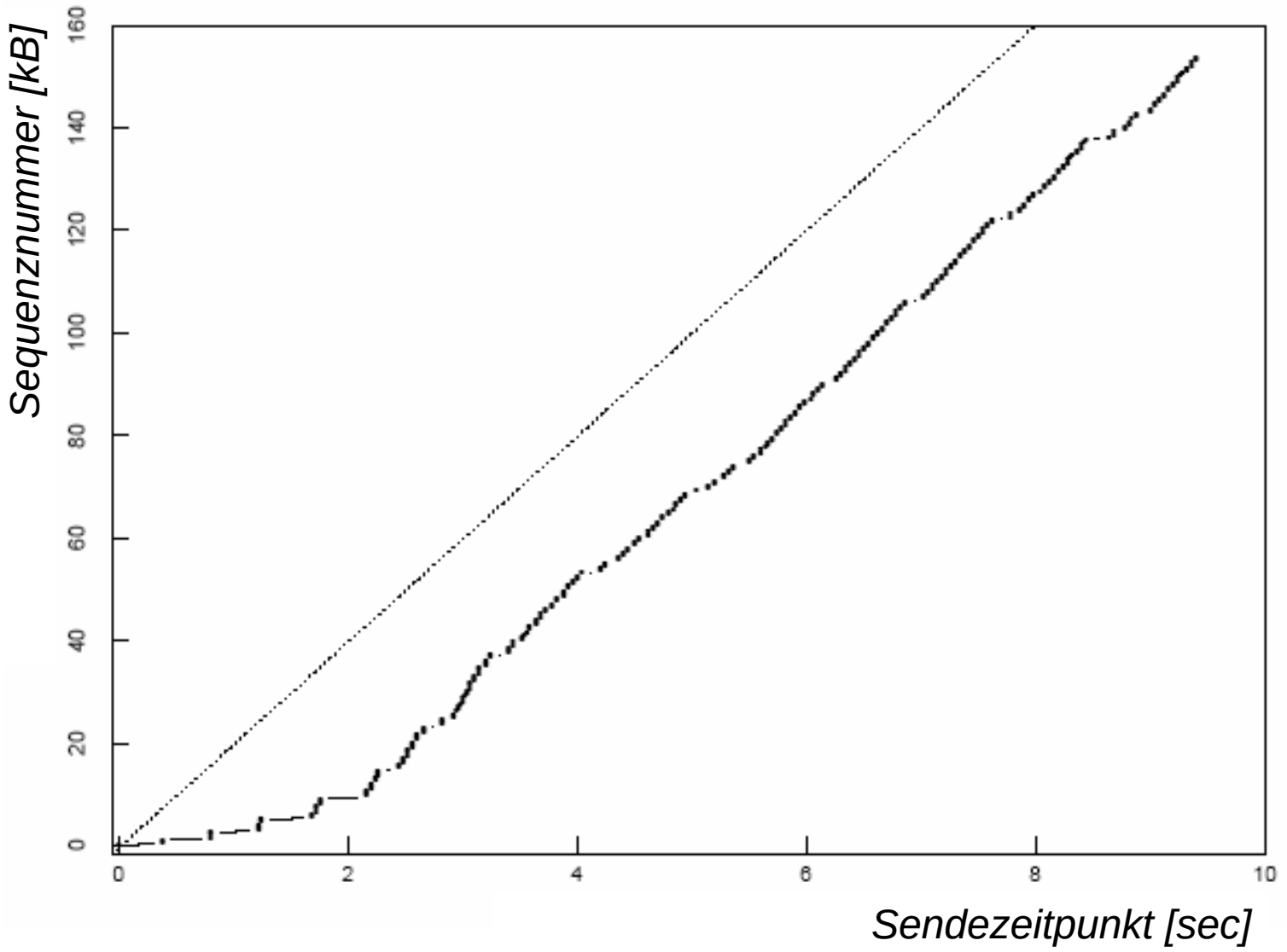
Größe des Sendefensters ist Minimum aus Fluss- und Staukontrollfenster (CWnd)

Schwellenwert (SSTresh) beeinflusst, wie Staukontrollfenster vergrößert wird

Ablauf

- Start: CWnd = 1 MSS und SSTresh = ∞
- Solange CWnd <= SSTresh wird so genanntes ***Slow-Start*** angewandt
  - Bei jeder empfangenen Quittung: CWnd + = 1
  - Entspricht Verdopplung des Staukontrollfensters nachdem alle gesendeten Segmente mit ACKs bestätigt wurden (und diese ACKs empfangen wurden)
- Sobald CWnd > SSTresh wird so genanntes ***Congestion Avoidance*** angewandt
  - Bei jeder empfangenen Quittung: CWnd + = 1 / CWnd
  - Lineares Erhöhen des Staukontrollfensters, d.h. erhöhen des Staukontrollfensters um eine Einheit pro vollständig gesendetem und bestätigtem Sendefenster
- Falls ein Timeout auftritt
  - SSTresh := CWnd/2
  - CWnd = 1 MSS

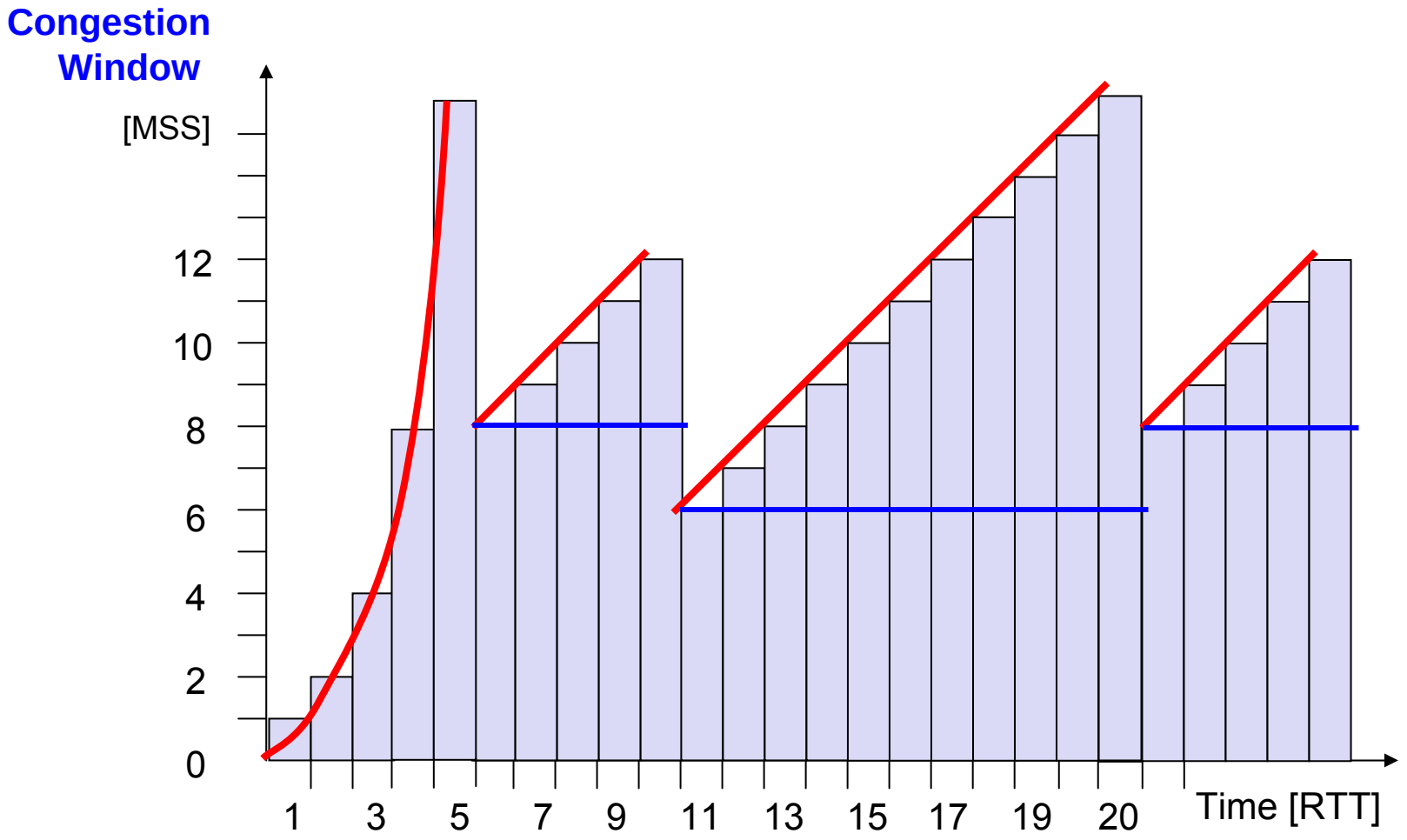
TCP Tahoe Throughput



TCP Tahoe & TCP Reno

- Zusätzlich kann man die Leistungsfähigkeit (nicht Staukontrolle) von TCP durch selektive Quittungen erweitern:
  - Fast Retransmit und Fast Recovery (RFC 2001) – Interpretiere das dritte Duplicate ACK als negative Quittung für das auf das ACK folgende Segment. In diesem Fall wiederhole sofort das entsprechende Segment und halbiere das Staukontrollfenster.
  - Selective Acknowledgement (RFC 2018) – Nutze eine neue TCP Option um explizit einzelne Segmente positiv zu bestätigen.
- SACK verbessert die Leistungsfähigkeit, falls mehre Segmente verloren gegangen sind.

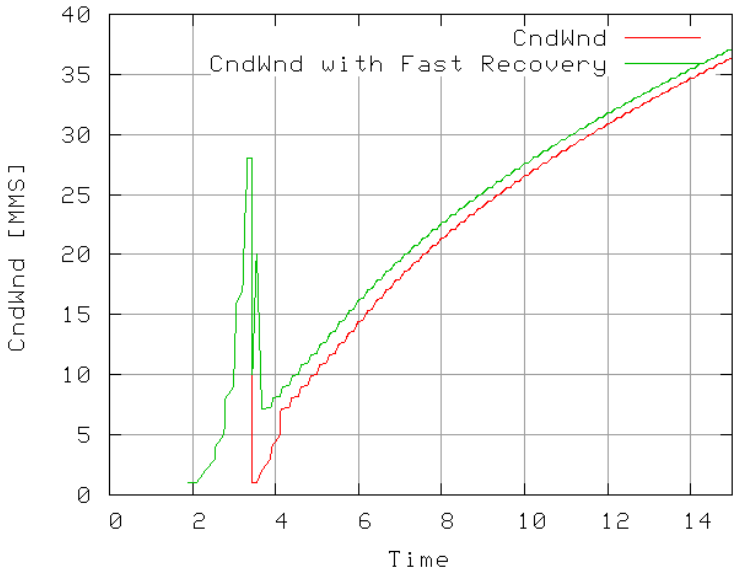
TCP Reno Overview



TCP-Varianten

Tahoe

- der vorher beschriebene Algorithmus
- Problem
  - Sender muss bei Datenverlust evtl. lang auf den Timeout warten
  - Nicht jeder Datenverlust geht auf eine Stausituation zurück



Reno

- Fast-Retransmit: Sendewiederholung wird angestoßen, falls drei gleiche Quittungen empfangen wurden
- Fast-Recovery: Nach Fast-Retransmit wird kein Slow-Start verwendet, d.h. keine Reduktion auf nur ein Segment, aber lineares Wachstum

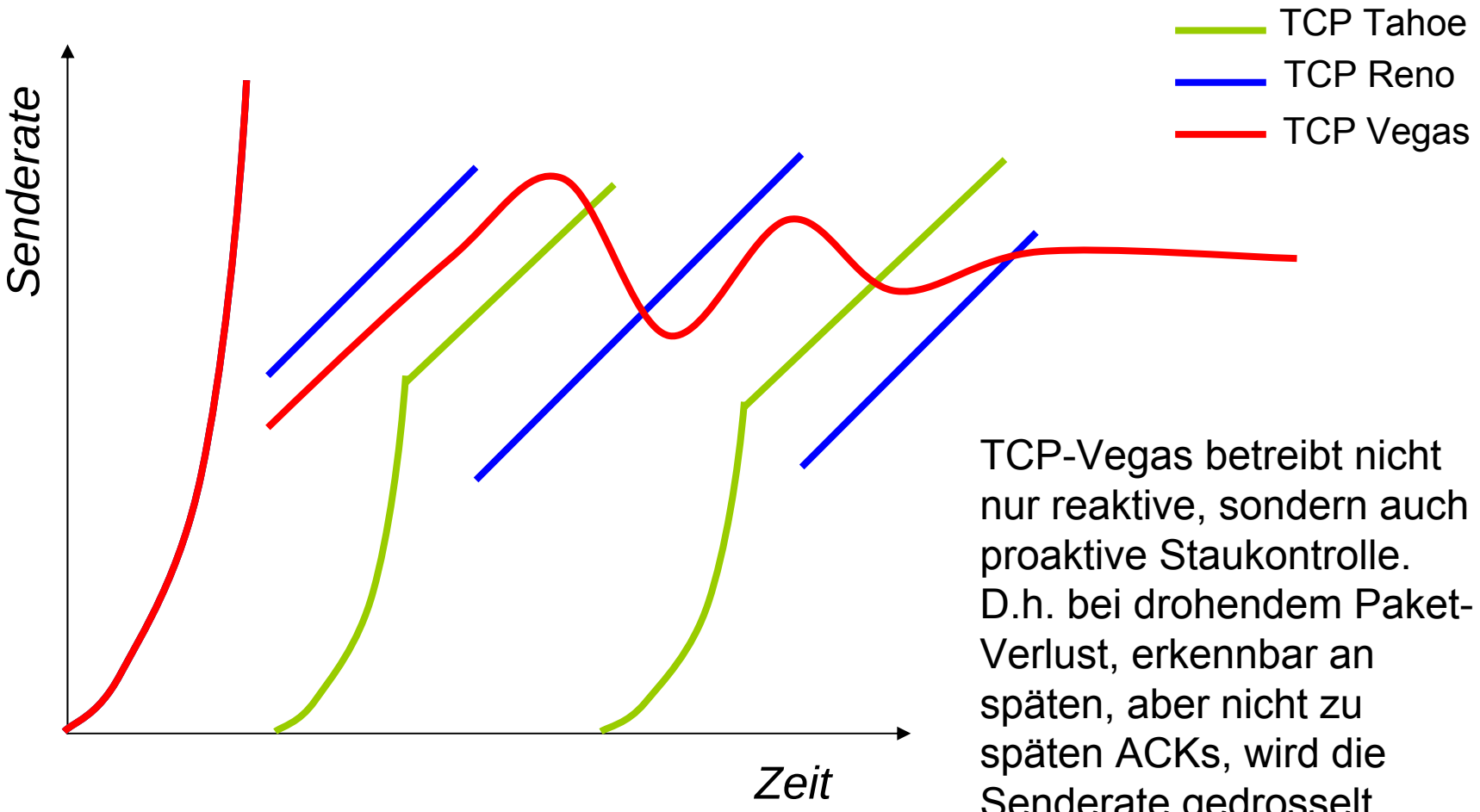
NewReno

- Nochmals verbessertes Fast-Recovery

Vegas

- Idee: Stausituationen erkennen bevor Datenverluste auftreten, dann lineare Reduzierung (anstelle der multiplikativen Erniedrigung bei Tahoe und Reno)
- Vermuten einer Stausituation basierend auf steigenden Umlaufzeiten

Ausblick auf TCP Vegas

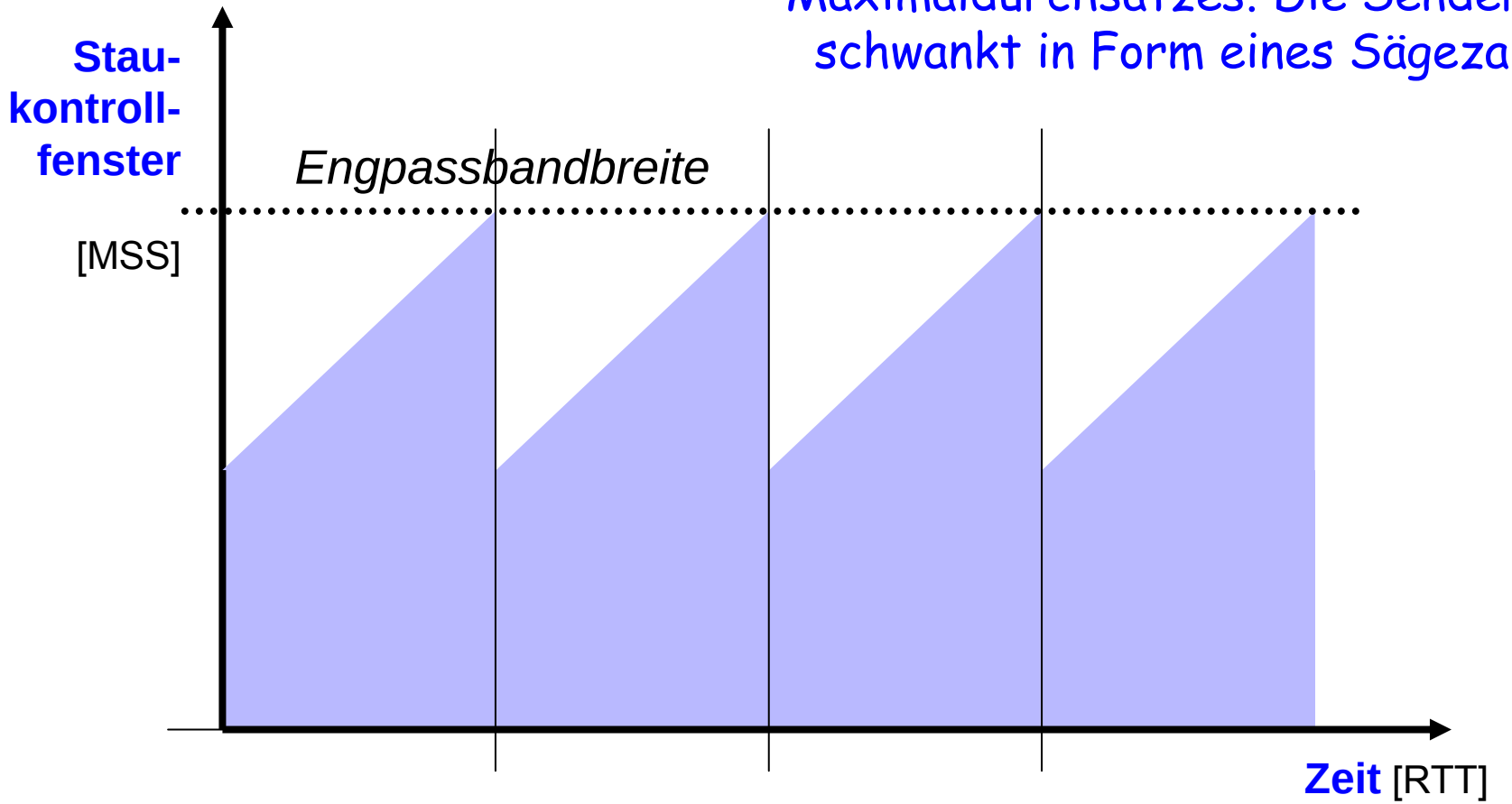


TCP-Vegas betreibt nicht nur reaktive, sondern auch proaktive Staukontrolle. D.h. bei drohendem Paket-Verlust, erkennbar an späten, aber nicht zu späten ACKs, wird die Senderate gedrosselt.

TCP Throughput & TCP Fairness

TCP Durchsatz

TCP* erzielt im Mittel nur 75% des Maximaldurchsatzes. Die Senderate schwankt in Form eines Sägezahns.

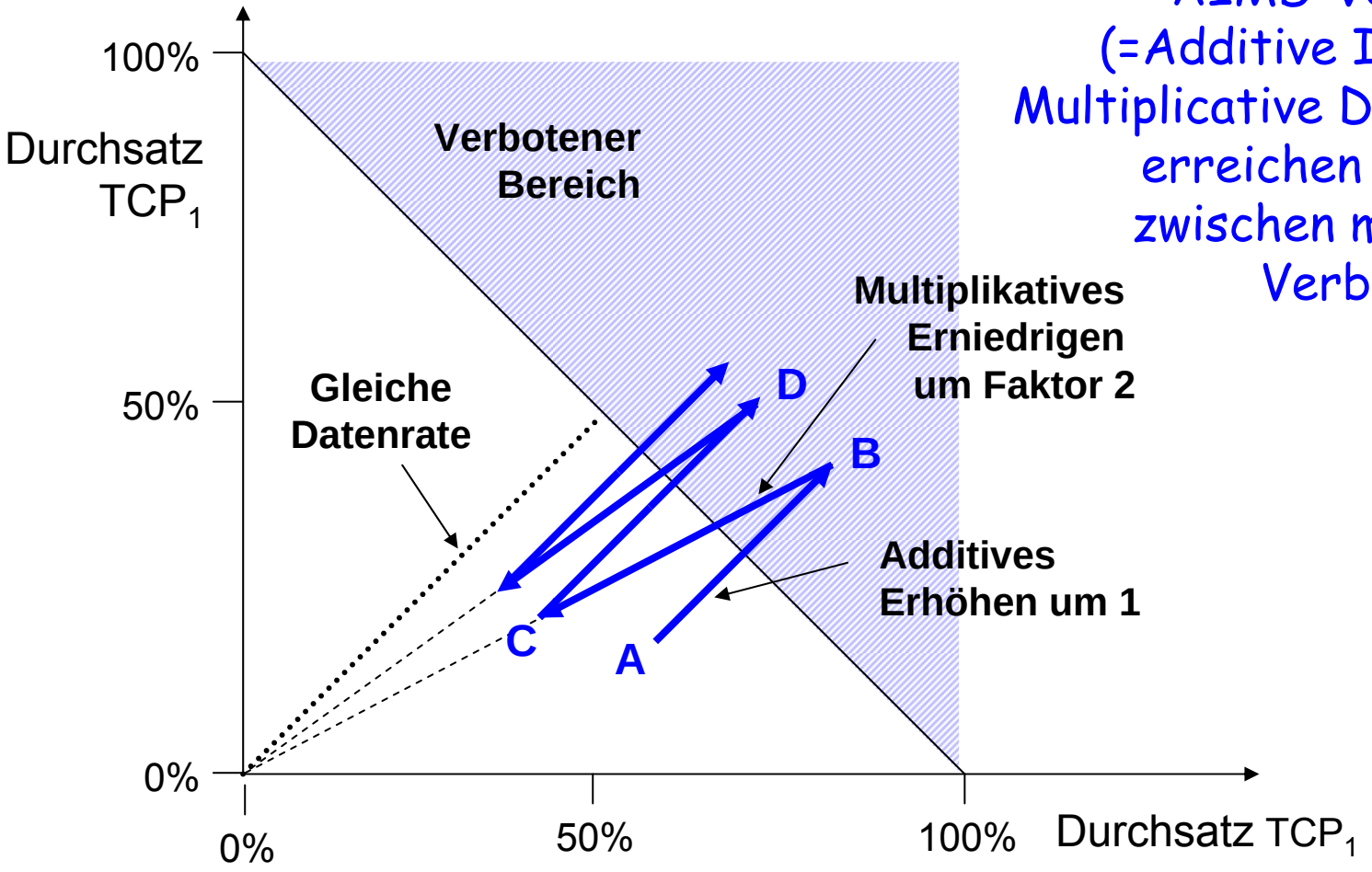


* TCP-Reno, d.h. mit Fast Recovery

TCP Fairness (1)

- Ein Protokoll sollte Fairness gewährleisten. – Aber, was ist Fairness?
  - Keine allgemeine Definition möglich
  - Kontextabhängig
  - Fairness kann (nur) durch gemeinsame Übereinkunft aller an einer Kommunikationssystem beteiligten Partner definiert werden
- Im Internet hat sich der Begriff ***TCP-Fairness*** etabliert:
  - Definiert Senderate in Abhängigkeit der aktuellen Netzwerkbedingungen (Round Trip Time und Paketverlustrate) auf dem jeweiligen Kommunikationspfad
  - Fair ist die Rate, die TCP in Abhängigkeit dieser Bedingungen erreichen würde
- TCP-Fairness beurteilt also Transportprotokolle mit Hilfe der faktisch von TCP erreichten Senderate

TCP Fairness (2)

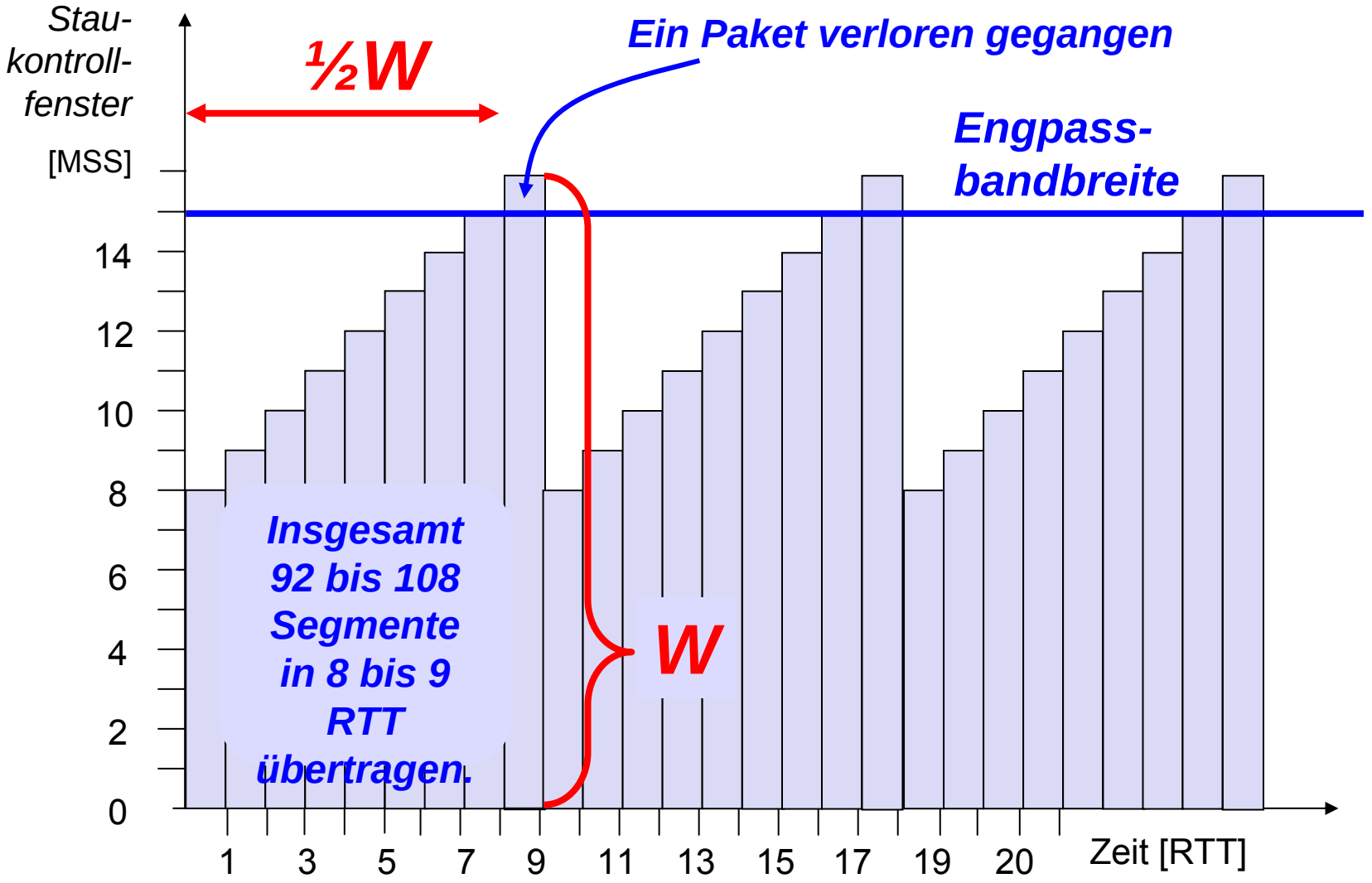


AIMD-Verfahren (=Additive Increase, Multiplicative Decrease) erreichen Fairness zwischen mehreren Verbindungen

TCP Fairness (3)

- TCP Implementierungen sind (im allgemeinen) gleich, d.h. kein Partner wird bevorzugt.
  - TCP teilt die zur Verfügung stehende Bandbreite gleichmäßig auf alle TCP *Verbindungen* auf (vgl. Max-Min-Fair-Share)
  - Achtung: Definitionsgemäße Grundlage von TCP-Fairness ist eine Verbindung, d.h. eine Anwendung mit doppelt so vielen Verbindungen erhält doppelt so viel Bandbreite!
- Probleme entstehen aus der gleichzeitigen Anwesenheit von TCP und anderen Protokollen im Netz
  - Typisches Entwurfsziel: Neue Protokolle sollen TCP-fair sein
  - Dazu erforderlich: Von TCP abstrahierte Beschreibung der von TCP erreichten Senderate (→***TCP-Formel***)
  - Die TCP-Formel basiert auf der Analyse des ***TCP Staukontrollmechanismus***

Herleitung der TCP-Formel (1)



Herleitung der TCP-Formel (2)

$$\text{TCP-Senderate} = \frac{\frac{3}{8}W^2 \cdot \text{MSS}}{\frac{1}{2}W \cdot \text{RTT}} = \frac{3}{4}W \cdot \frac{\text{MSS}}{\text{RTT}}$$

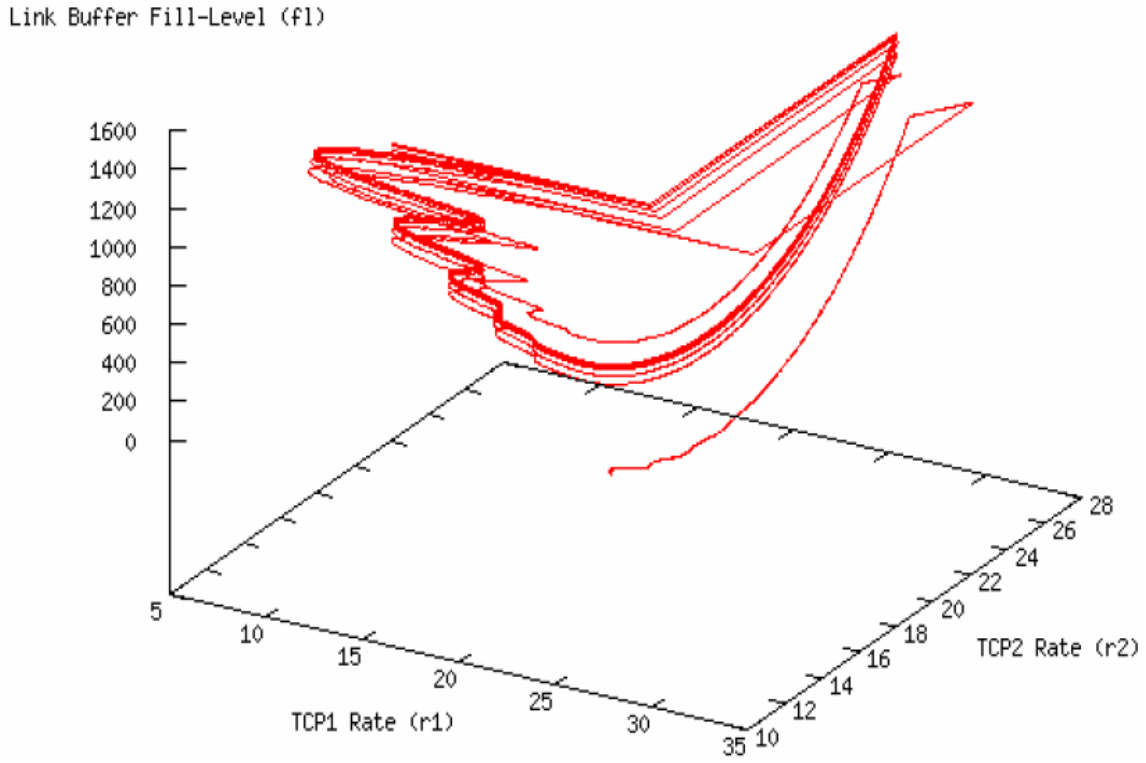
Verlustrate:
(ein Paket von allen gesendeten Paketen)

$$p = \frac{1}{\frac{3}{8}W^2}$$

$$\text{TCP-Senderate} = \sqrt{\frac{3}{2p}} \cdot \frac{\text{MSS}}{\text{RTT}}$$

Bemerkungen zum TCP-Durchsatz

- Die TCP-Senderate oszilliert wie eine Sägezahnkurve. TCP schöpft dabei im Mittel nur 75% der eigentlich zur Verfügung stehenden Engpassbandbreite aus.
- Bei Routern, die nur wenige TCP-Verbindungen zusammenfassen, schwingen sich TCP-Verbindungen ein und oszillieren dann in Phase.



Ratenbasierte Staukontrolle

- Mit der TCP-Formel ist ein alternativer Ansatz möglich, die TCP-Oszillationen zu vermeiden:
  - Messe die Paketverlustrate (p) und Paketumlaufzeit (RTT) und messe bzw. schätze die Segmentgröße (MSS)
  - Die TCP-Formel gibt dann die TCP-faire Senderate.
  - Diese Rate kann mit einem Leaky-Bucket umgesetzt werden.
  - Während der Übertragung werden Verlustrate und RTT gemessen und die Rate allmählich an etwaige Veränderungen angepasst.
- Dieser Ansatz ist besonders für Multimedia-Datenströme geeignet:
  - Daten laufen gleichmäßig statt in Bursts.
- Mit der TCP-Formel kann TCP-faires Mulitcast realisiert werden.

Token Bucket & Leaky Bucket

Token Bucket

- Zähler („Eimer“) mit max. Größe B
- Zähler wird mit Rate R erhöht
- Wenn ein Paket der Größe L gesendet wird, wird der Zähler um L verringert
- Ist der „Eimer leer“, d.h. der Zähler kleiner L, wird das Paket gepuffert, bis der Zähler wieder groß genug ist.
- Garantiert die Einhaltung der mittleren Senderate R

Leaky Bucket

- Puffer („Eimer“) mit max. Größe B
- Pakete werden in den Puffer gepackt und „tröpfeln“ mit der Rate R heraus, d.h. ein Paket der Größe L wird erst gesendet, wenn L Byte aus dem „Eimer getropft“ sind
- Garantiert die Einhaltung der Senderate R und verhindert Bursts

Token Bucket und Leaky Bucket werden oft in Zwischensystemen eingesetzt, um Datenströme an einen „Verkehrsvertrag“ anzupassen.

Fragen

- Erläutern Sie wie TCP einen Staukollaps im Internet auslösen kann!
- Was leistet das Self-Clocking? Warum kann es dennoch keinen Stau verhindern?
- Welche Änderungen bringt TCP Tahoe zur Stauvermeidung?
- Welchen Vorteil bringt Reno gegenüber Tahoe?
- Erläutern Sie, warum sich zwei TCP-Verbindungen die Bandbreite fair teilen können!
- Leiten Sie die TCP Formel her!
- Erläutern Sie die Funktionsweise von ratenbasierter Staukontrolle!

TCP Support in the Network Layer

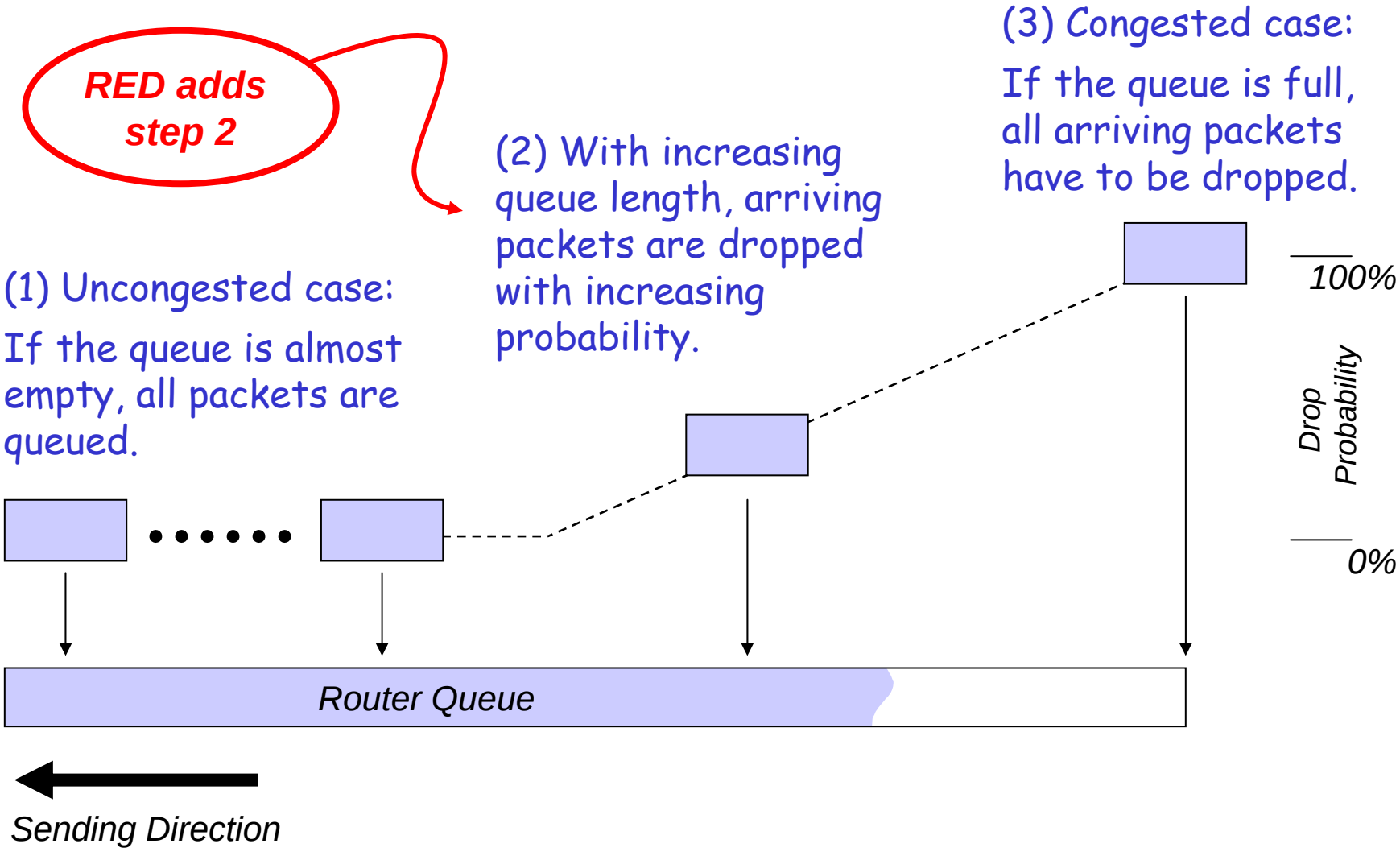
TCP Unterstützung im Netz (1)

Genaue Untersuchungen zeigen, dass es vorteilhaft ist, die Puffer der Router nicht immer ganz auszuschöpfen, bevor die TCP-Senderate reduziert wird.

- Random Early Detection (RFC 2309) ist ein Verfahren, das aktives Warteschlangenmanagement in den Routern anwendet
- Mit steigender Länge der Warteschlange werden Pakete mit steigender Wahrscheinlichkeit verworfen
- Dadurch reduzieren zufällig ausgewählte einzelne TCP-Verbindungen ihre Senderate, anstatt dass alle Verbindungen gleichzeitig ihre Rate reduzieren

Ursprünglich wurde das Verfahren als „Random Early Drop“ (RED) bezeichnet. In Verbindung mit „Early Congenstion Notification“ (ECN) müssen Paket aber nicht verworfen werden (siehe unten). Man hat den Namen daher so geändert, dass die Abkürzung RED erhalten blieb, aber das Wort „drop“ ersetzt wurde.

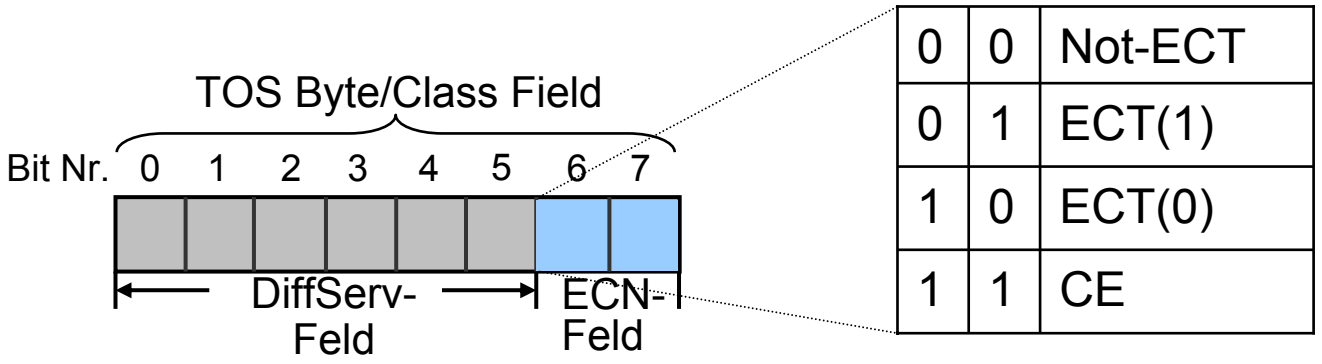
Random Early Drop



TCP Unterstützung im Netz (2)

- Ohne ECN ist das Netzwerk für TCP eine „Black Box“. Die Endsysteme können nur indirekt über Paketverlust auf Stausituationen schließen.
- Explicit Congestion Notification (RFC 3168, Sep. 2001)
  - Vermeidet Paketverluste durch explizite Stauanzeige des Netzes: Statt ein Paket frühzeitig zu verwerfen wird das ECN-Bit im Header gesetzt.

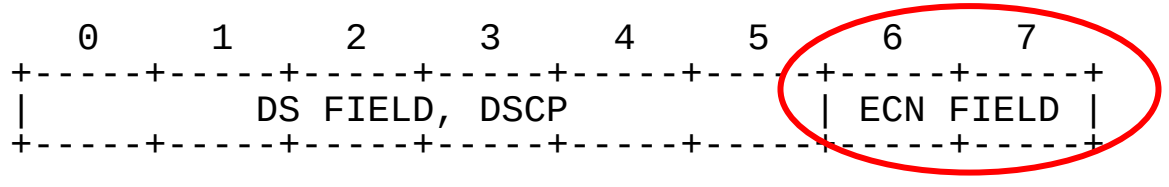
  TCP reagiert darauf mit Reduktion seiner Senderate.
  - ECN setzt Active Queue Management (z.B. RED) im Router voraus.
  - Markierung des IP-Pakets mittels Congestion Experienced (CE) Bit.
  - Diese Anzeige muss erfolgen, bevor Warteschlange wirklich voll ist.
  - ECN-Fähigkeit muss signalisiert werden, um TCP Fairness sicherzustellen: ECN-Capable Transport (ECT) Bits



0	0	Not-ECT
0	1	ECT(1)
1	0	ECT(0)
1	1	CE

Explicit Congestion Notification (1)

IP Header

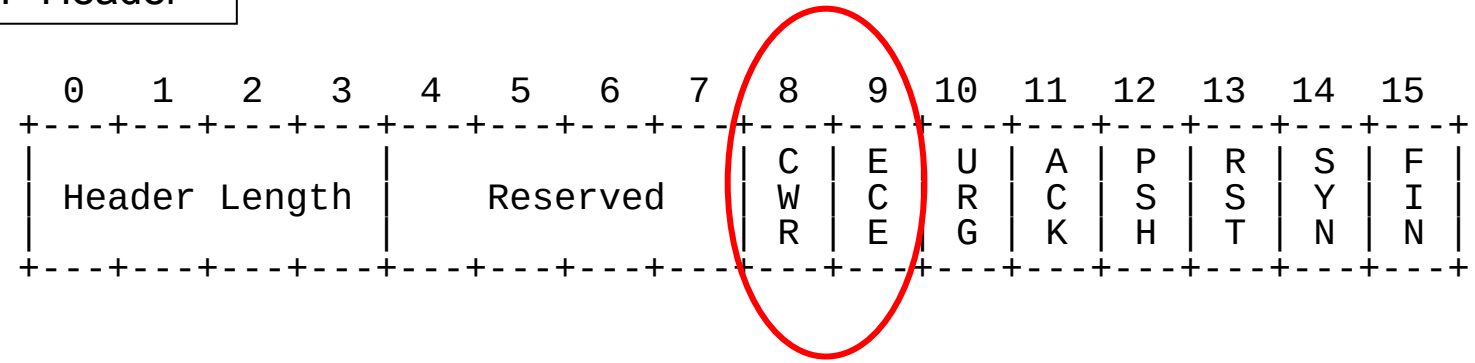


DSCP: differentiated services codepoint

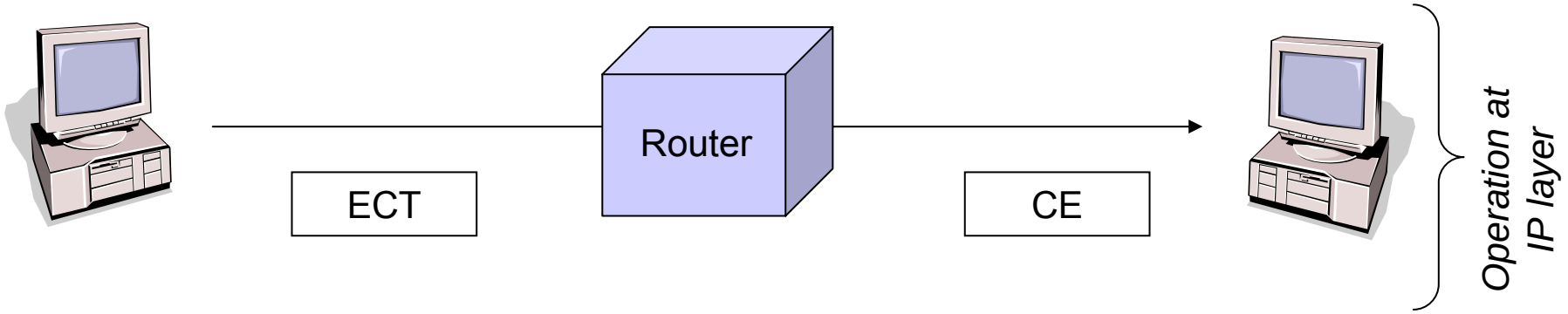
ECN: Explicit Congestion Notification

0	0	Not-ECT
0	1	ECT(1)
1	0	ECT(0)
1	1	CE

TCP Header

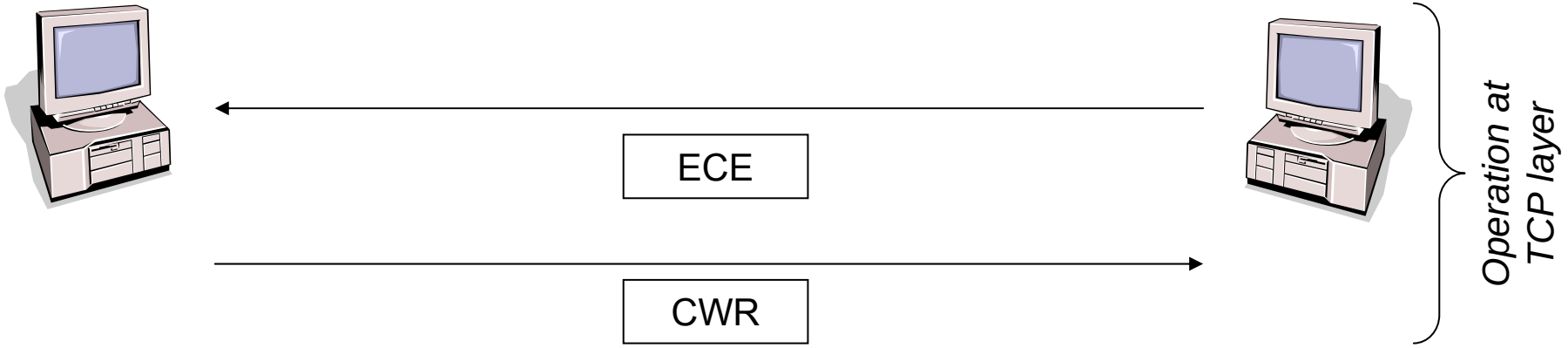


Explicit Congestion Notification (2)



1. ECN capable hosts set ECT in the IP header. ECN capable routers set ECT to CE to indicate congestion.

2. ECN capable receivers echo the CE in the TCP header as ECE (=echo congestion experienced). ECN capable senders acknowledge the reception ECE as CWR (=congestion window reduced).



More TCP Design Issues …

(How to cheat with TCP)

Misbehaving TCP Empfänger (1)

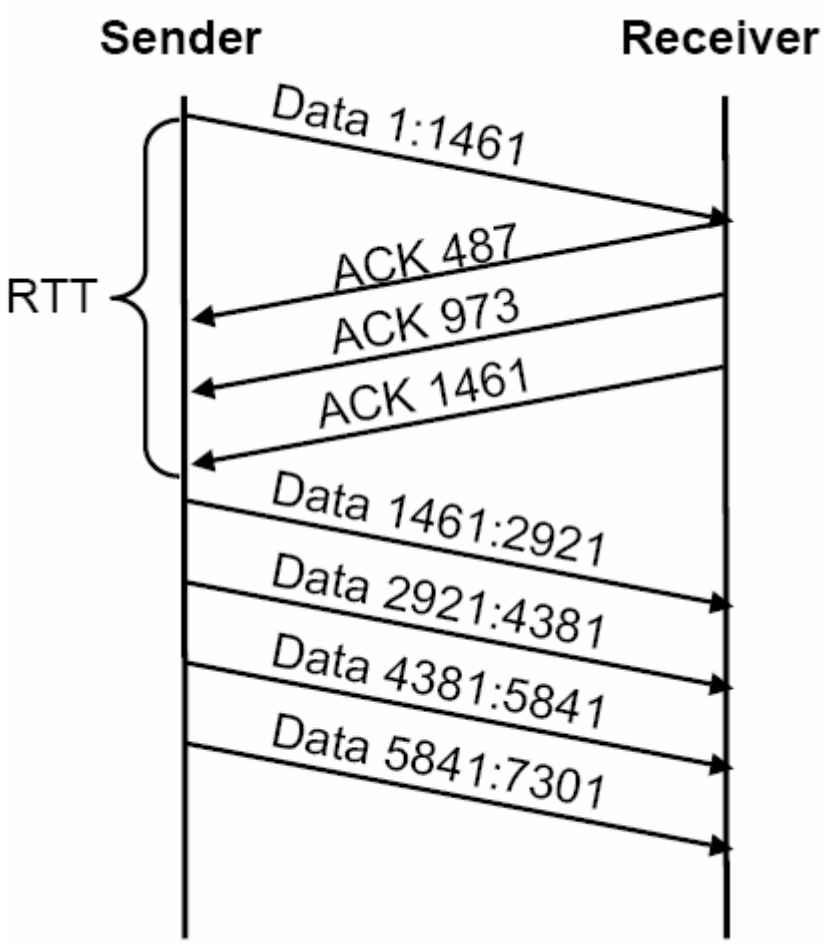
- TCP ist anfällig gegen unfairen Bandbreitenkonsum:
  - Die Möglichkeit zu unfairem Verhalten ist Best Effort Netzen inhärent, aber TCP ermöglicht es auch einer einzelnen Partei (und zwar dem Empfänger), sich einen Vorteil zu verschaffen.
  - Ein entsprechender Web-Browser kann also z.B. schneller Web-Seiten saugen, ohne dass er dafür einen Web-Server ändern müsste.
  - Aus der Spieltheorie ist bekannt, dass sich dieses Verhalten durchsetzt, da es ja nicht von der Gruppe kontrolliert werden kann.
- Der Ansatzpunkt für diesen „Betrug“ sind Inkonsistenzen in der TCP-Spezifikation.

Misbehaving TCP Empfänger (2)

- TCP Sequenznummern zählen Bytes, das Staukontrollfenster zählt in Segmenten.
- Angriff:

  Bestätige jedes Segment in einzelnen Häppchen!
- Dadurch öffnet sich das Staukontrollfenster schneller.
- Bemerkung:

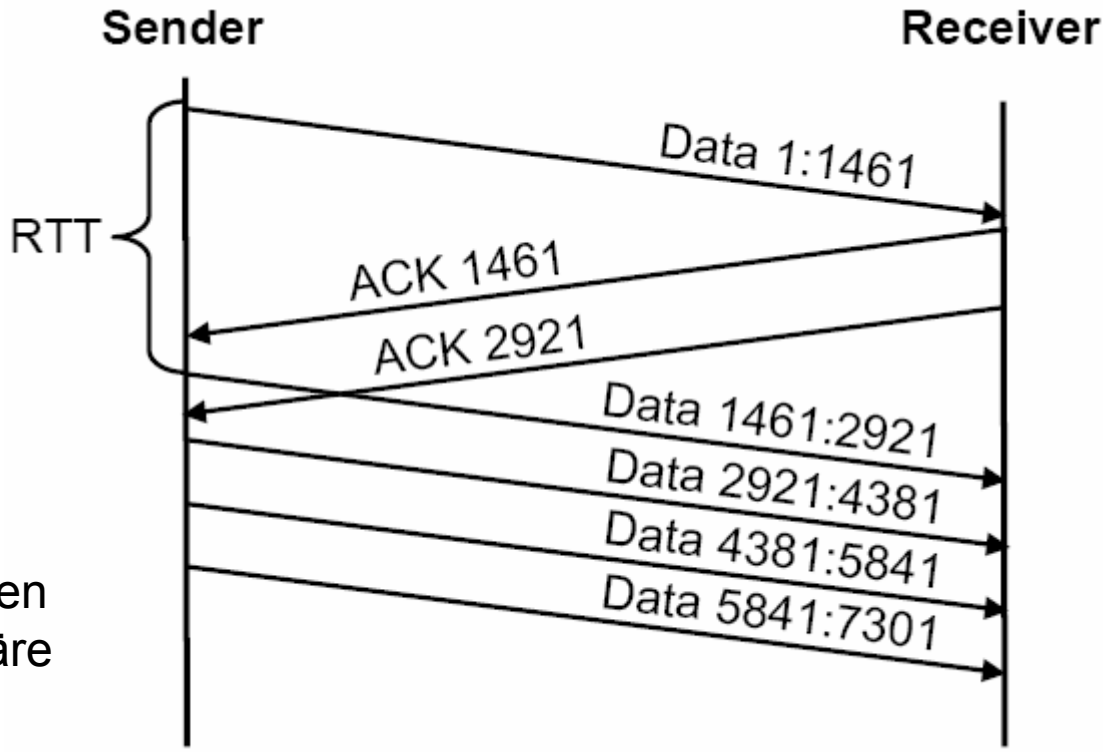
  Würde das Staukontrollfenster ebenfalls in Bytes zählen, wäre der Angriff unmöglich.



Misbehaving TCP Empfänger (3)

- Paketverluste sind selten und Quittungen beweisen nichts.
- Angriff:

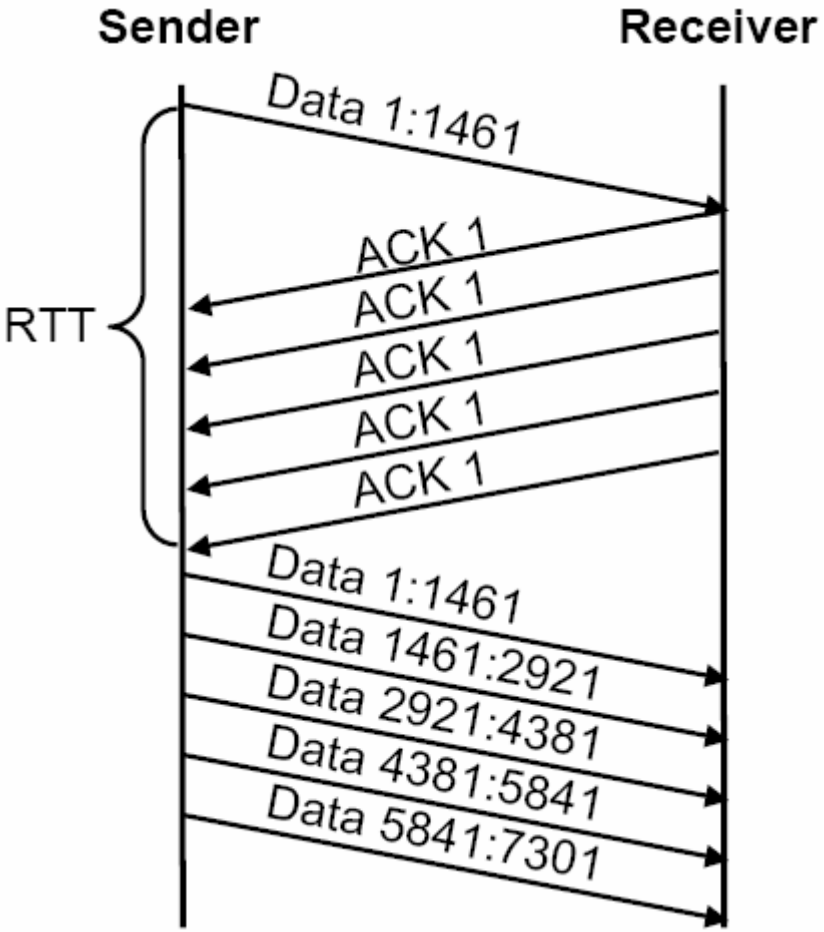
  Bestätige Segmente schneller als sie eintreffen.
- Bemerkung:
  - Würden Quittungen tatsächlich den korrekten Empfang beweisen, wäre der Angriff unmöglich.



Anders als im ersten Fall wäre zur Lösung hierfür eine Protokolländerung erforderlich. Alternativ würde aber bereits eine Erhöhung der Paketverlustwahrscheinlichkeit, d.h. das gelegentliche auslassen eines Segments beim Sender, einen solchen Angriff unrentabel machen.

Misbehaving TCP Empfänger (4)

- Duplicate ACKs vermischen negative und positive Quittungen:
  - Segment x soll wiederholt werden
  - Segment x+1, x+2, x+3, … wurde empfangen
- Gemäß TCP Reno läuft das Self-Clocking weiter und auch das Staukontrollfenster wird weiter geöffnet, d.h. es werden immer weiter neue Segmente übertragen bis das Flusskontrollfenster erschöpft ist.



Questions?

Thomas Fuhrmann

Department of Informatics
Self-Organizing Systems Group
c/o I8 Network Architectures and Services
Technical University Munich, Germany

fuhrmann@net.in.tum.de