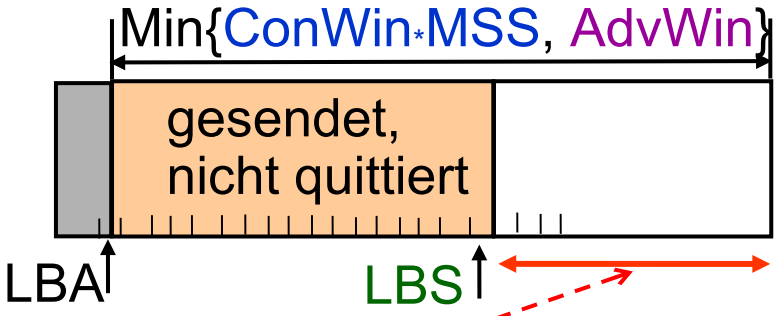
Slow Start (Wiederholung)

Congestion Window

- Der Sender verwendet ein **Congestion Window (ConWin)**
- Der Sender darf nie mehr als **ConWin * MSS Bytes** ohne Quittierung senden
- ConWin wird in der Einheit Maximum Segment Size (MSS) angegeben
  - Default nach RFC 1122: MSS = 536 Byte
- Congestion Control und Flow Control sind im Sender gekoppelt
  - Maximal sendbar ist das **Effective Window = Minimum aus ConWin * MSS und AdvWindow**



Congestion Control, „Feedback des Netzwerks“

Flow Control, durch Empfänger gesteuert

Effective Window = Min{ConWin*MSS, AdvWindow} – (LBS – LBA)

noch nicht quittierte Anzahl von Bytes

Slow Start

Prinzip

- Mit einem kleinen Wert des Congestion Windows beginnend, wird dieses exponentiell erhöht, solange keine Überlast auftritt
- Slow Start wird bei Überschreitung eines Grenzwertes beendet
  - **Threshold (Thres)**: Grenzwert in Byte (default: 64 kByte)

Verfahren

- **Start:** Beim Verbindungsstart oder nach einem Segmentverlust

  ConWin = 1 MSS

- **Eingehendes ACK:**

  ConWin = ConWin + 1 MSS

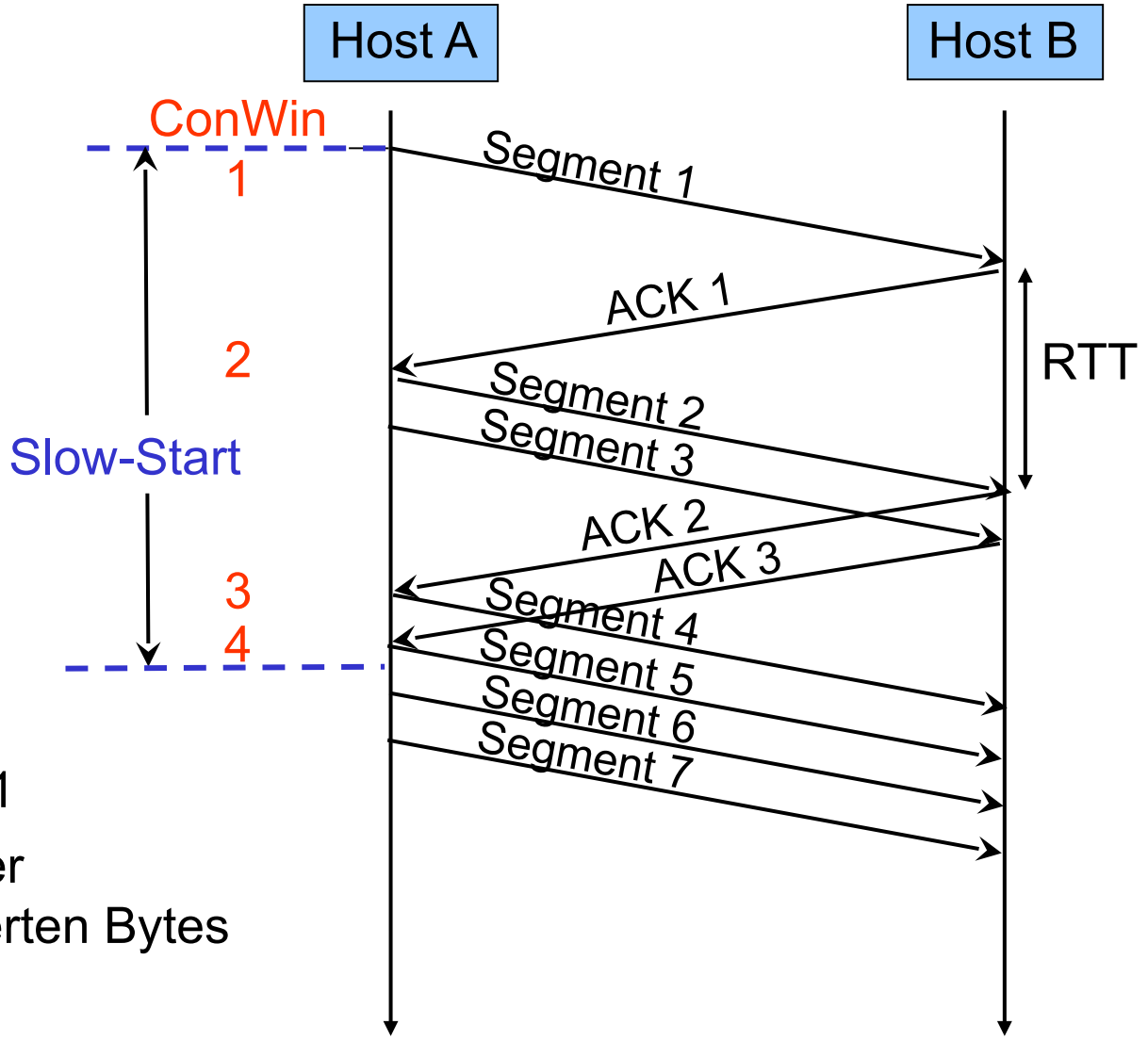
- **Stop:** Slow Start endet, falls ConWindow den Grenzwert überschreitet

  ConWin > Thres → stop Slow Start

Besipiel: Slow Start

Initialisierung:

- AdvWin > 50 MSS
- Tres = 3 MSS
- kein delayed Ack



- Jedes ACK erhöht das ConWin um 1
- unabhängig von der Anzahl der quittierten Bytes

Slow Start – Analytischer Ansatz

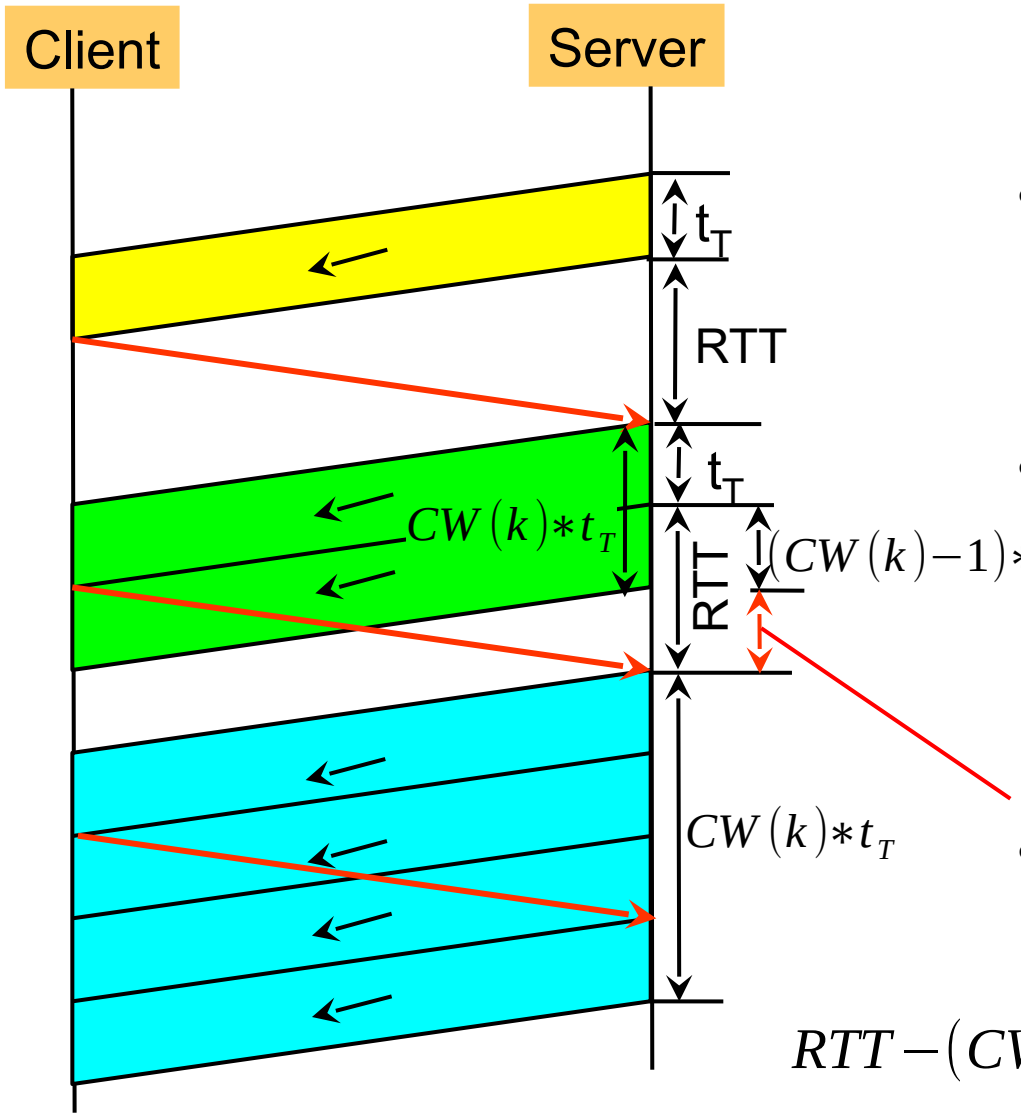
Variablen

- $CW(t)$ = Congestion Window zum Zeitpunkt t
- $AW(t)$ = Advertised Window zum Zeitpunkt t
- D = Gesamte zu übertragende Datenmenge in bit
- EFF(t) = Größe des Effective Window
- $k \in \{1,2,3,...\}$ = Anzahl von Fensterzyklen für Übertragung
- S = Anzahl der zur Übertragung benötigten Segmente D/MSS
- t_T = Zeiteinheit für Slots auf Graphen $(= MSS / v_B)$
- Q = Anzahl an Fensterzyklen während derer auf ACK gewartet wird
- thres = Grenze für Übergang von Slow Start zu Congestion Avoidance
- RTT = Round Trip Time, v_B = Leitungsgeschwindigkeit
- K = Gesamtzahl der Zyklen, MSS = Maximum Segment Size

Initialwerte

- $CW(0) = 1$

Slow Start - Übertragungsdauer



- Im k-ten Fenster können
  $CW(k) = 2^{k-1}$
  Segmente gesendet werden
- Das ACK für das erste Segment des Fensters erreicht den Server nach der Dauer $RTT + t_T$
- Bevor das nächste Fenster gesendet wird, muss der Sender auf dieses ACK warten, falls

$$CW(k)*t_T < RTT + t_T$$

- Die Wartedauer bis zum Empfang des ACKS beträgt:

$$RTT - (CW(k)-1)*t_T = RTT - (2^{k-1}-1)*t_T$$

Slow Start - Übertragungsdauer (2)

- Sind genügend Daten zu senden, so muss der Server nur während der ersten Q Fensterzyklen auf ein ACK warten, wobei

$$Q = max\{k : CW(k) * t_T \leq RTT + t_T\} = max\{k : 2^{k-1} * t_T \leq RTT + t_T\}$$

$$= max\{k : 2^{k-1} \leq 1 + \frac{RTT}{t_T}\} = max\{k : k \leq 1 + \log_2(1 + \frac{RTT}{t_T})\}$$

Lies: maximales k das die Bedingung k<=1+[…] erfüllt

- Es werden insgesamt K Fenster-Zyklen an Daten übertragen.
  - Der Server muss nur während der ersten

$$F = min\{Q, K-1\}$$

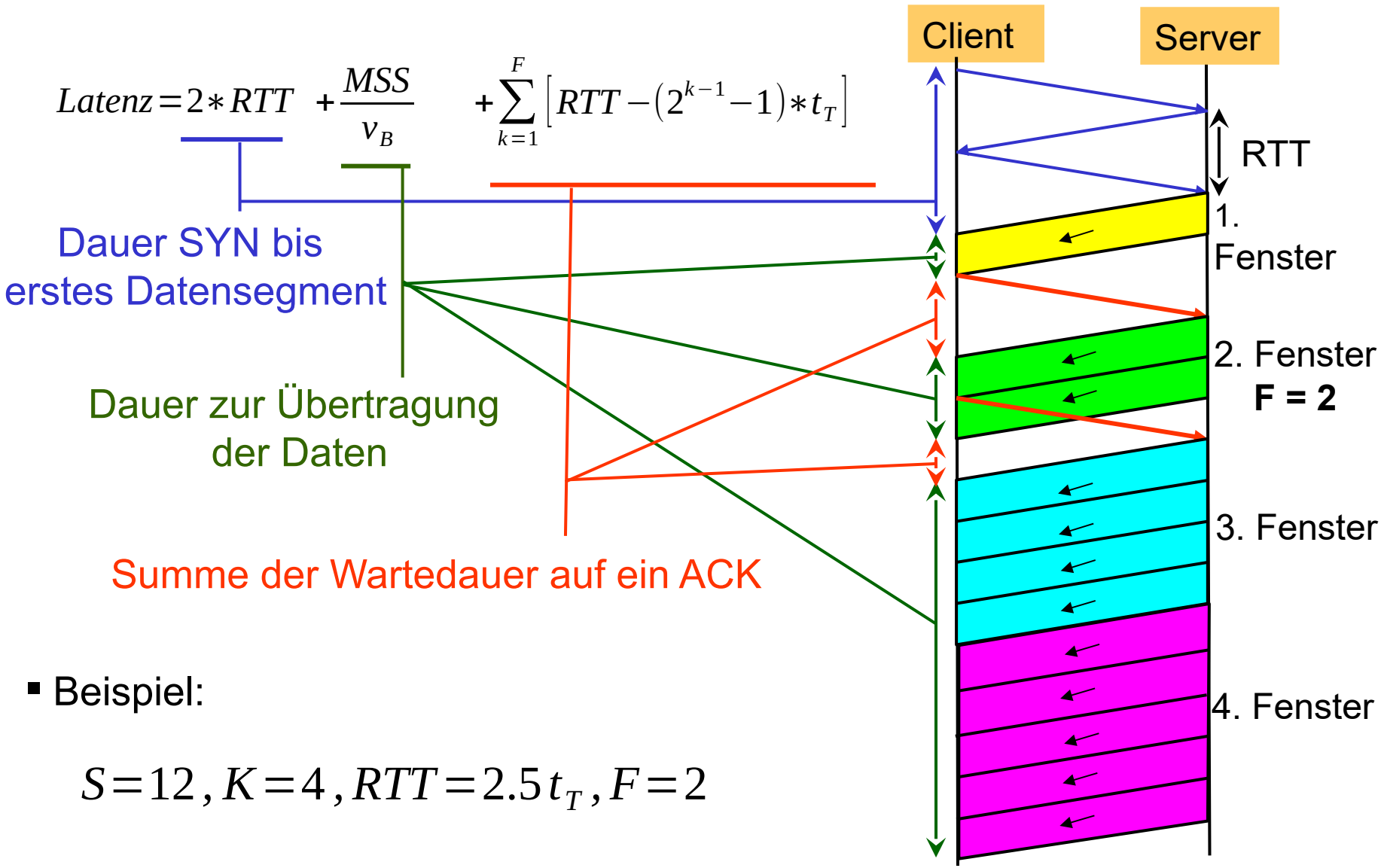
Fenster-Zyklen warten

- Beispiel

$$S = 12, K = 4, RTT = 2.5\,t_T \rightarrow Q = max\{k : k \leq 1 + \overbrace{\log_2(3.5)}^{1.8}\}$$

$$F = 2$$

Slow Start - Übertragungsdauer (3)



- Beispiel:

$$S = 12, K = 4, RTT = 2.5\, t_T, F = 2$$

Slow Start - Übertragungsdauer (4)

- Es gilt die geschlossene Form die wie folgt vereinfacht werden kann:

$$Latenz = 2 * RTT + \frac{D}{v_B} + \sum_{k=1}^{F} \left[RTT - \left(2^{k-1} - 1\right) * t_T \right]$$

$$= 2 * RTT + \frac{D}{v_B} + F * \left(t_T + RTT\right) - t_T \sum_{k-1}^{F} 2^{k-1}$$

$$= 2 * RTT + \frac{D}{v_B} + F * \left(t_T + RTT\right) - t_T \left(2^F - 1\right)$$

- Beispiel:

$$S = 12, D = S * MSS = 12\, MSS, K = 4, F = 2$$

$$t_T = \frac{MSS}{v_B}, RTT = 2.5\, t_T = 2.5 \frac{MSS}{v_B}$$

$$Latenz = \frac{MSS}{v_B}\left(5 + 12 + 2\left(1 + 2.5\right) - 3\right) = 21 \frac{MSS}{v_B} = 1.75 \frac{D}{v_B}$$

Slow Start - Übertragungsdauer (5)

- Minimale Latenz: Congestion Window schränkt Senderate nicht ein

$$minimale\ Latenz = 2 * RTT + \frac{D}{v_B}$$

- Beispiele: mit MSS = 536 Bytes, SIS=Slow Start

	RTT= 100 ms, D = 100 Kbyte, K = 8			RTT= 100 ms, D = 5 Kbyte, K = 4			RTT= 1 s, D = 100 Kbyte, K = 8		
v_B [Mbit/s]	F	minimale Latenz	Latenz mit SIS	F	Minimale Latenz	Latenz mit SIS	F	Minimale Latenz	Latenz mit SIS
0,1	2	8,4 s	8,5 s	2	0,6 s	0,76 s	5	10,2 s	14,1 s
1	5	1 s	1,5 s	3	0,24 s	0,52 s	7	2,8 s	9,3 s
10	7	0,28 s	**0,93 s**	3	0,2 s	**0,5 s**	7	2,1 s	9,0 s

- Negativer Einfluss von Slow Start bei sehr großer RTT

Congestion Avoidance

Fast-Retransmit und Fast-Recovery

Beobachtung

- Nicht jeder Verlust eines Segmentes beruht auf einer Überlastsituation
- Der Empfang von „duplicate ACKs“ deutet auf einzelne Segmentverluste hin

Forderung

- Vermeidung langer Timeout-Intervalle bei einzelnen Segmentfehlern
  - Lösung: Fast-Retransmit; Neuübertragung einzelner Segmente
- Liegt keine Überlast vor kann die Slow Start Phase entfallen
  - Lösung: Fast-Recovery; zurücksetzen des ConWin auf Thres

Congestion Avoidance (TCP RENO)

Verfahren: **A**dditive **I**ncrease - **M**ultiplicative **D**ecrease (AIMD)

ConWin > Thres (=Zustand CA):

- Neues ACK vor Ablauf des Retransmission Timers

  ConWin = Min{ ConWin + 1 / ConWin, TCP_MaxWin}

- Timeout: Retransmission Timer abgelaufen → Überlast angenommen

  Thres = max{ ConWin/2, 2 MSS} und **Retransmission**

  ConWin = 1 (→ zurück zu Phase Slow Start)

- 3 dACKs empfangen: → Einzelne Segmentfehler angenommen

  Fast Retransmit

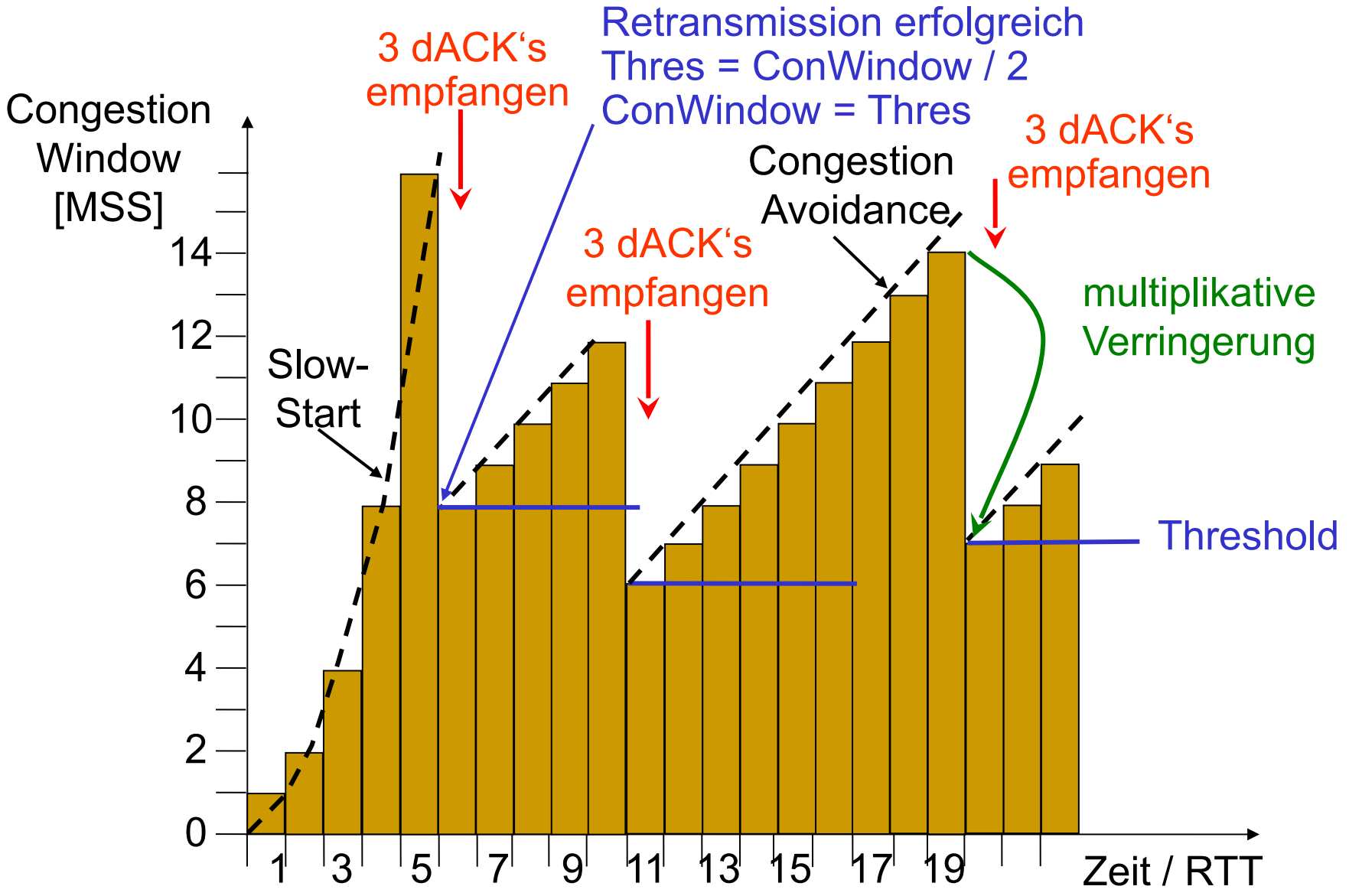
  Thres = max{ ConWin/2, 2 MSS}

  - Ack für Retransmit (oder neues ACK): → Fast Recovery

    ConWin = Thres

    Zustand: Congestion Avoidance

Beispiel: RENO mit Fast – Recovery (vereinfacht)



Zustandstabelle (TCP RENO)

Zustand	Ereignis	TCP Sender Aktion	Kommentar
Slow Start (SS)	neues ACK für nicht quittierte Daten	ConW = ConW + MSS, If (ConW > Threshold) → Zustand CA	Das ConW wird je RTT verdoppelt
Con. Avoid. (CA)	neues ACK	ConW = ConW + 1 / ConW	Additive Erhöhung, ConW ≈ 1 MSS je RTT erhöhen
SS oder CA	Duplicate ACK (dACK)	erhöhe dACK Zähler für jedes dupliziert quittiertes Segment	ConW und Threshold bleiben erhalten
	3 dACKs	Überlast → Zustand FR	Einzelsegmentverlust
SS / CA / FR	Timeout	Thr* = max{ConW/2, 2 MSS} ConW = 1 → Zustand SS	Überlast des Netzes
Fast Recovery (FR)	drittes dACK	Thresh=Thr*=max{ConW/2, 2} Fast-Retransmission ConW = Thresh+3 MSS	Fast Retransmit und verringertem neuem Threshold.
FR	dACK	ConW = ConW + MSS nächstes Segment senden	DACK → ein Segment hat den Empfänger erreicht
FR	ACK für Re-transmission	ConW = Thr* → Zustand CA	wiederholtes Segment quittiert, FR beendet

Congestion Avoidance – Analytischer Ansatz

Mathematische Betrachtung

$$CW_0 = Thres$$

- If ACK received:

$$CW(t) = CW(t-1) + \frac{1}{CW(t-1)}$$

- CW wächst um 1/CW mit jedem eintreffenden ACK

- Hinweis: Sie dürfen diese Werte in der Klausur abrunden und das CW inkrementieren sobald „alle“ ACKs angekommen sind (=Nachkommastellen ignorieren)

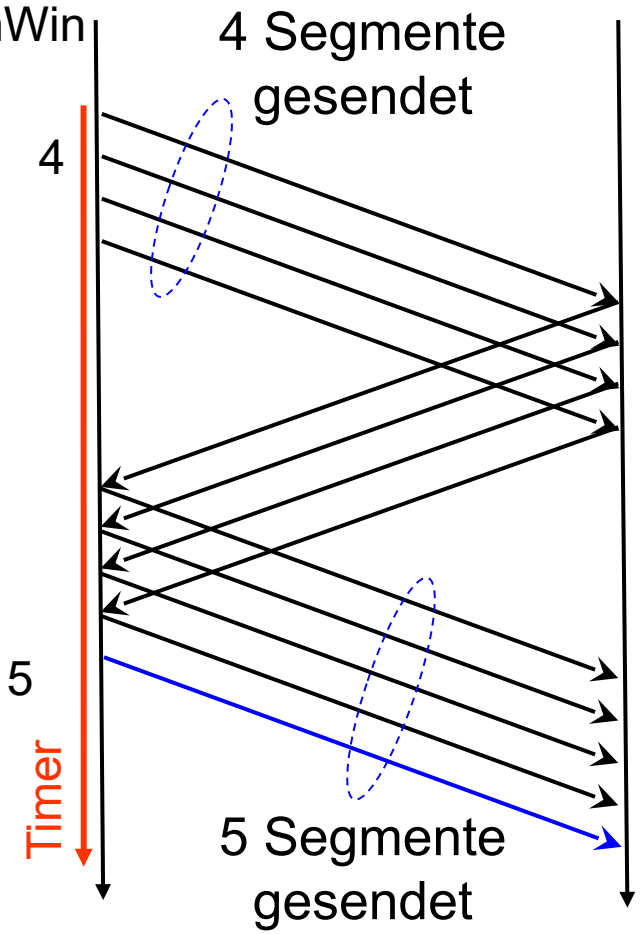
- Beispiel: CW = 4 → 4 Segmente gesendet, nach Empfang von Acknowledgement 4 → CW = 5

Congestion Avoidance - Beispiel

Beispiel:

- Annahme: TCP sendet nur Segmente von MSS Bytes

ConWin [MSS]	gesendete Segmente	empfangene ACKs	ConWin neu
$CW_0 = 4$	4	-	-
CW_0	-	1	$CW_1=CW_0+1/CW_0 = 4{,}25$
CW_1	-	2	$CW_2=CW_1+1/CW_1 \approx 4{,}5$
CW_2	-	3	$CW_3=CW_2+1/CW_2 \approx 4{,}7$
CW_3	-	4	$CW_4=CW_3+1/CW_3 \approx 4{,}9 \approx 5$
CW_4	5	-	-



Netzwerkunterstützte Congestion Control

Problem:

- Ende-zu-Ende-Überlastkontrolle schließt durch Paketverluste auf Überlast
- Paketwiederholungen sind für verzögerungszeitsensitive Anwendungen ungünstig

Lösung:

Aktives Warteschlangenmanagement:

- Verhindern, dass eine Warteschlange in den Überlastbereich gelangt
- **Random Early Detection**
  - Netzwerkgestützte Überlastcontroller:
  - Netzwerk teilt Endsystemen eine bestehende Überlast mit
  - Explicit Congestion Notification

Random Early Detection (RED)

Prinzip

- Implizite Benachrichtigung der Übertragungspartner
  - Verwerfen von Paketen durch den Router (führt zu Timeouts)
    → Halbierung des Congestion Windows, Reduktion der Senderate
- Explizite Benachrichtigung durch Markieren von Paketen möglich

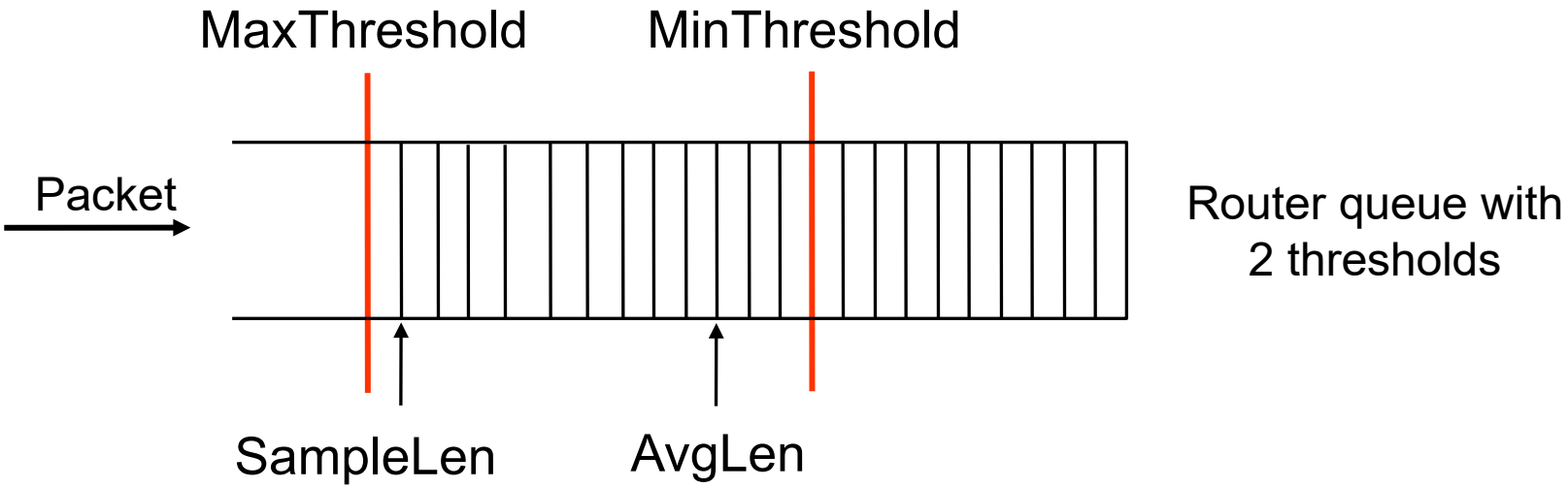
Frühzeitiges Verwerfen

- Anstatt darauf zu warten, dass die Queue vollständig gefüllt ist wird jedes Paket mit einer **drop propability** verworfen sobald die Queue eine definierte Länge überschreitet

Eigenschaften

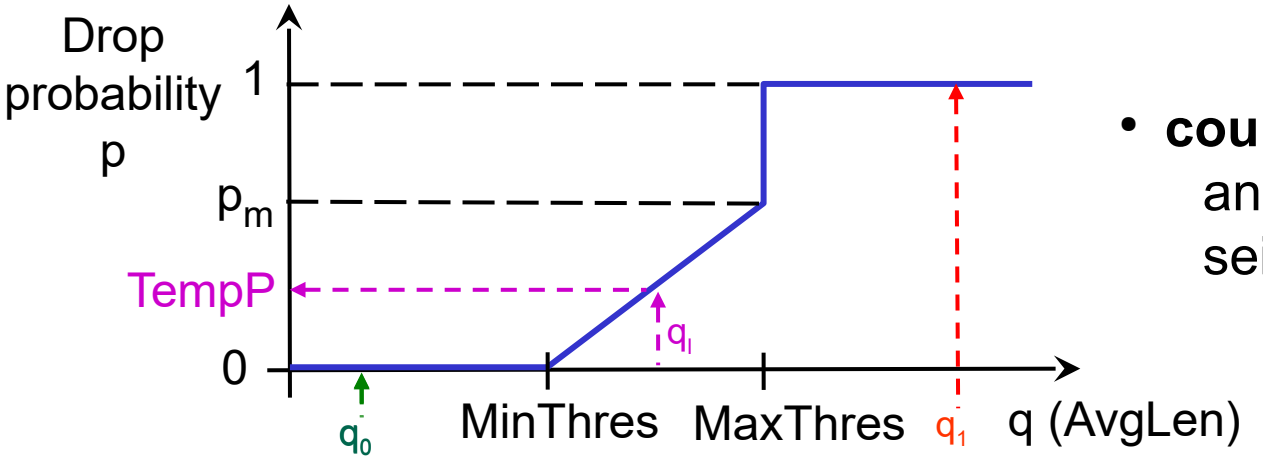
- Verhindert Synchronisation der Sendepartner
- Reduziert die mittlere Queuelänge im Router

Random Early Detection (RED)



- SampleLen ist die Länge der Queue zum Ankunftszeitpunkt eines Pakets
- Average queue length (AvgLen)
  - AvgLen = (1 - Weight) * AvgLen + Weight * SampleLen
  - 0 < Weight < 1 (usually 0.002)
- AvgLen wird meist für jedes ankommende Paket errechnet

Random Early Detection (RED)



- **count** = Anzahl der angekommenen Pakete seit letztem Drop

Average queue length q	Drop probability p
0 ≤ q ≤ MinThres	p = 0
MinThres < q < MaxThres	p steigt linear bis p_m an
q ≥ MaxThres	p = 1 (all packets are dropped)

- **Berechnung der Wahrscheinlichkeit P**

  TempP = p_m * (q - MinThres)/ (MaxThres - MinThres)

  P = TempP/(1 - count * TempP)

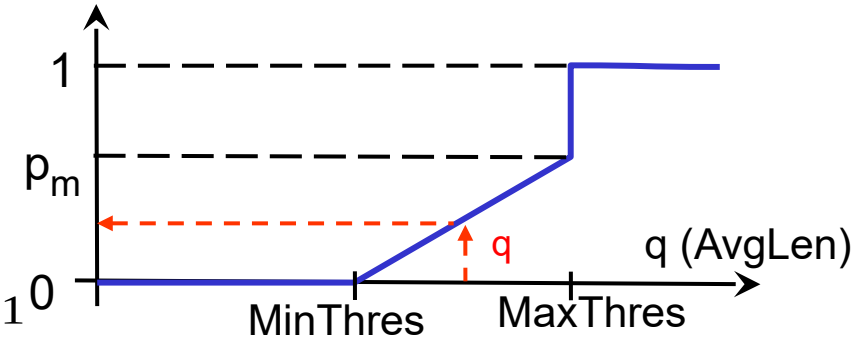
Random Early Detection (RED) - Beispiel

$p_m = 0{,}02$

$$q = \frac{MaxThres + MinThres}{2} \approx const\,, da\ Mittelwert$$

$$TempP = p_m \frac{q - MinThres}{MaxThres - MinThres} = 0{,}5\ p_m = 0{,}01$$

$$P = \frac{TempP}{1 - count \cdot TempP} = \frac{0{,}01}{1 - 0{,}01 \cdot count}$$



wäre P = TemP = 0,01, würde im Mittel jedes 100‘tes Paket gelöscht

count = 0

Anzahl Pakete	count	P	Aktion
1	0	0,01	nicht löschen
50	49	1/49	nicht löschen
51	50	0,02	nicht löschen
99	98	0,5	nicht löschen
100	99	1,0	löschen

- count: Number of arriving packets since the last packet has been dropped
- Aktion: drop packet or accept packet

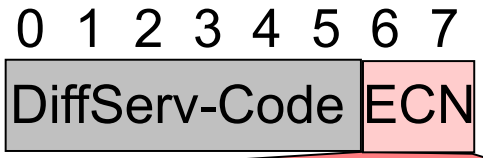
Random Early Detection (RED) - Parameter

- Drop-Wahrscheinlichkeit für einen Flow ist grob proportional zum Anteil der Bandbreite die der Flow bekommt
- **MaxP** (p_m) ist typischerweise auf 0.02 gesetzt, wenn also die durschschnittliche Queuelänge zwischen beiden Thresholds liegt, wird ca. jedes 50. Paket verworfen
- Wenn der Verkehr „bursty“ ist, sollte **MinThreshold** ausreichend groß sein um ausreichend hohe Verbindungsauslastung zu gewährleisten
- Die Differenz zwischen zwei Thresholds sollte größer sein als der typische Zuwachs der durchschnittlichen Queuelänge während einer RTT (oft: MaxThreshold = 2*MinThreshold)
- Weitere Anmerkungen:
  - RED erfordert komplexes Queue-Management
  - Ist heutzutage in quasi allen Routern implementiert

Explicit Congestion Notification (ECN, RFC 3168)

- Router im Netz zeigt Überlast im TOS-Feld des IP-Headers an
- Setzt ein Warteschlangen-Management im Router voraus (z.B. RED)
  - Signalisierung der Überlast muss vor einem Paketverlust erfolgen
  - Im Überlastfall werden die IP-Datagramme markiert
- Die ECN-Verwendung muss im IP-TOS-Feld angezeigt werden
  - ECN-Capable Transport (ECT) Bits

TOS-Feld im IP-Header:

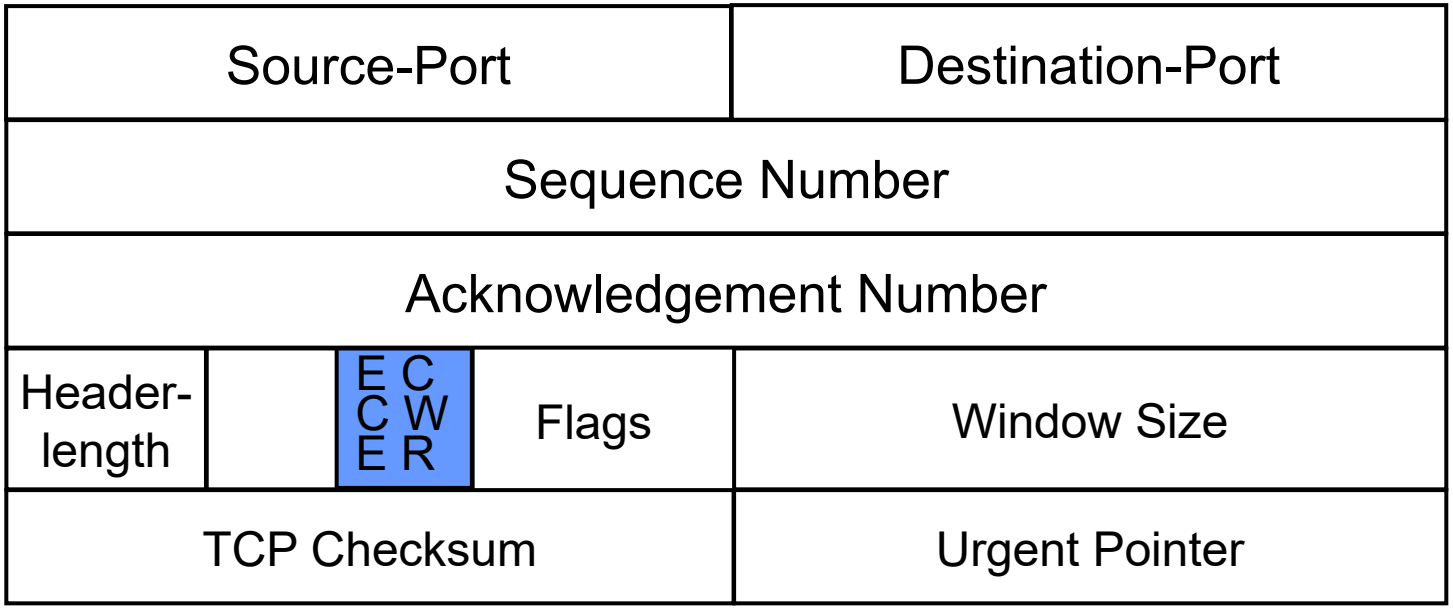


0 0	kein ECT	Sender verwendet kein ECN
0 1	ECT(1)	Sender zeigt ECN - Verwendung an
1 0	ECT(0)	alternative ECN - Anzeige
1 1	CE	Überlastanzeige (Congestion Experiences)

ECN-Erweiterung von TCP

Erweiterung des TCP-Headers (2 Bit des reservierten Bereiches)

- ECN-Echo-Flag (ECE)
- Congestion-Window-Reduced Flag (CWR)



Verbindungsaufbau

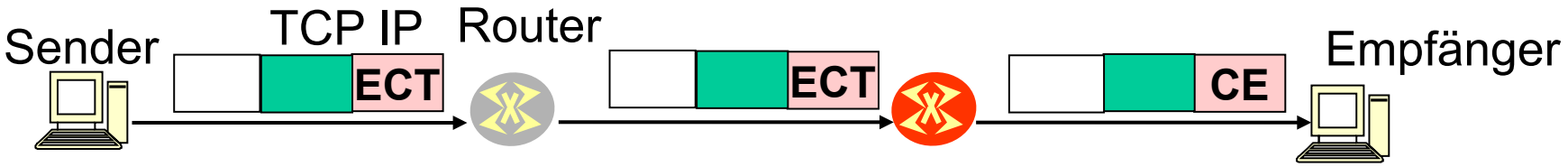
- ECN-Verwendung wird durch SYN+ECE+CWR signalisiert

ECN-Erweiterung von TCP (2)

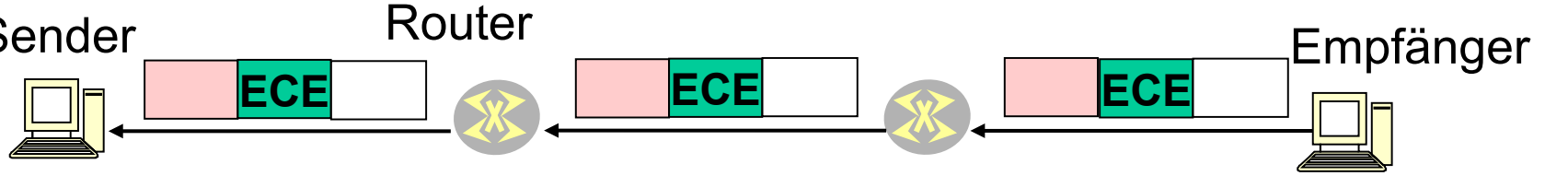
Congestion Control:

- TCP-Sender:
  - setzt ECT(0) oder ECT(1) im ECN-Feld des IP-Headers
- Router
  - setzt im Überlastfall ECN-Feld des IP-Headers auf CE
- TCP Empfänger
  - Erkennt Überlast und sendet die folgenden ACKs mit gesetztem ECE-Flag, bis er ein Sender-Segment mit gesetztem CWR-Flag erhält
- TCP-Sender empfängt ACK mit ECE-Flag
  - halbiert das Überlastfenster und den Grenzwert (Thres)
  - setzt das CWR-Flag im nächsten Segment

ECN-Erweiterung von TCP (3)

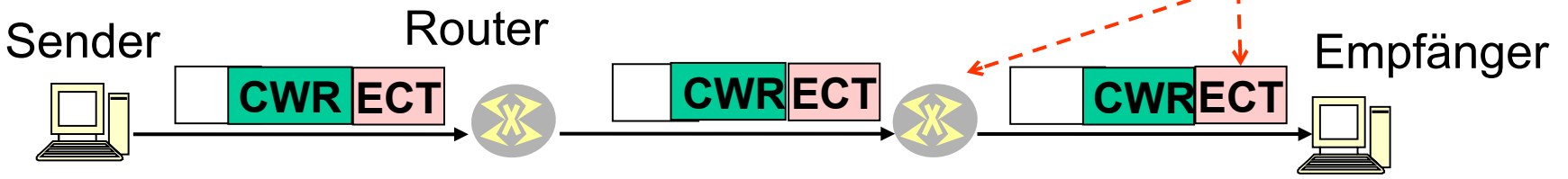


Empfänger erkennt Überlast
sendet ACK mit ECE-Flag



Sender erkennt Überlast
setzt ConWin/2 und Thres/2
setzt das CWR-Flag im nächsten Segment

Ergebnis:
Überlast reduziert



ECT: ECN-Capable Transport
ECE: ECN-Echo-Flag
CWR: Congestion-Window-Reduced Flag

Empfänger erkennt, dass Sender
das Überlastfenster reduziert hat