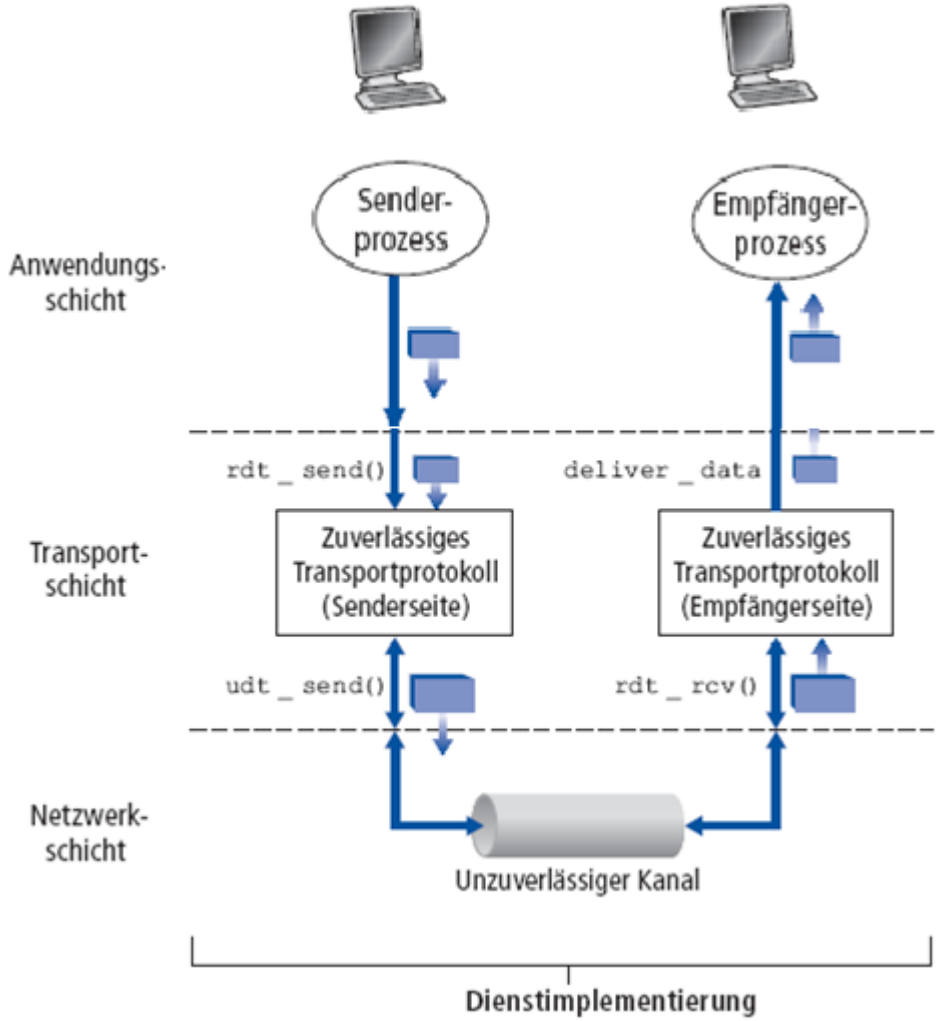
Block 6

Entwicklung eines zuverlässigen Datentransportprotokolls

Grundlagen der zuverlässigen Datenübermittlung

- Wichtig für Anwendungs-, Transport- und Sicherungsschicht
- Gehört zu den Top 10 der wichtigsten Netzwerkprobleme!
- Eigenschaften des unzuverlässigen Kanals bestimmen die Komplexität des Protokolls zur zuverlässigen Datenübertragung (rdt)

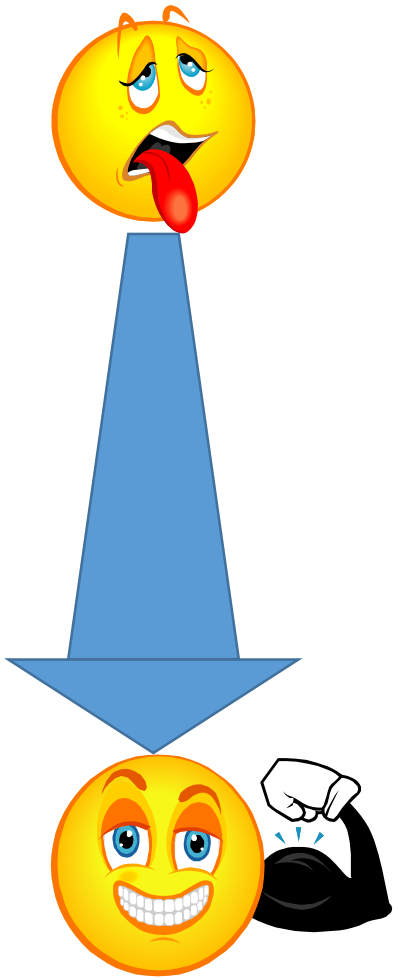


Bildquelle: frei nach Kurose/Ross, S.230

Entwicklung einer zuverlässigen Datenübermittlung

Wir wollen uns schrittweise Gedanken machen, wie ein zuverlässiger Datentransport realisiert werden könnte.

Dazu zeichnen wir ein Flussdiagramm, das wir sukzessive ergänzen, in dem wir Fragen stellen und Probleme aufwerfen, für die Lösungen zu erarbeiten sind.



Idee: Kurose/Ross, S.229 ff., Umsetzung und Grafiken: Kowalk

Version 1: Datenübermittlung minimal

Sender

Empfänger

Fragen an Version 1

- Wie können wir eine Informationsverfälschung bei der Übertragung bemerken (Bitfehler)?

- Was können wir tun, wenn beim Empfänger ein Bitfehler festgestellt wird?

Antworten zu den Fragen an Version 1

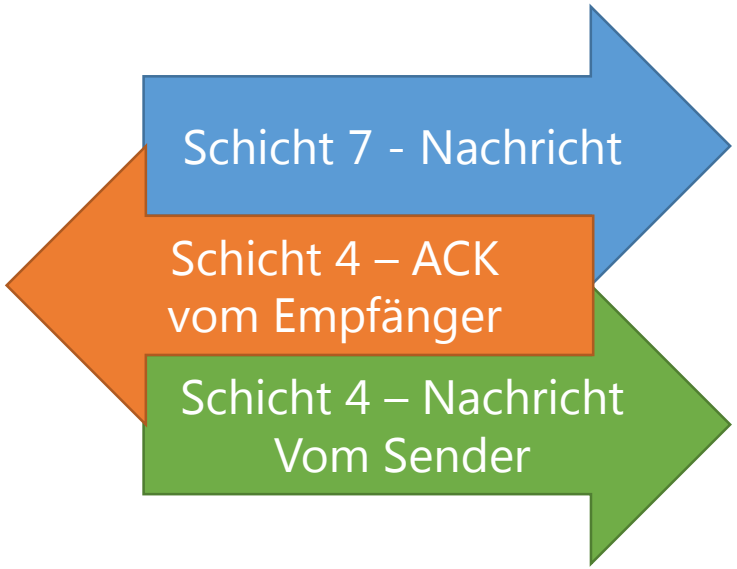
- Wie können wir eine Informationsverfälschung bei der Übertragung bemerken (Bitfehler)?
  - Wir gehen davon aus, dass dies durch eine **Prüfsumme** (Checksum) erkannt wird.
- Was können wir tun, wenn beim Empfänger ein Bitfehler festgestellt wird?
  - **Acknowledgement (ACK):** Empfänger sagt dem Sender explizit, dass das Paket erfolgreich empfangen wurde.
  - **Negative Acknowledgement (NAK):** Empfänger sagt dem Sender explizit, dass das Paket fehlerbehaftet war.
  - Sender **wiederholt Übertragung** für diese Pakete.

Neu in Version 2: Der Empfänger sendet!

- Obwohl „nur“ die Schicht 7-Information vom Sender zum Empfänger übertragen wird (unidirektional), kommunizieren auf Schicht 4 beide Seiten (bidirektional).

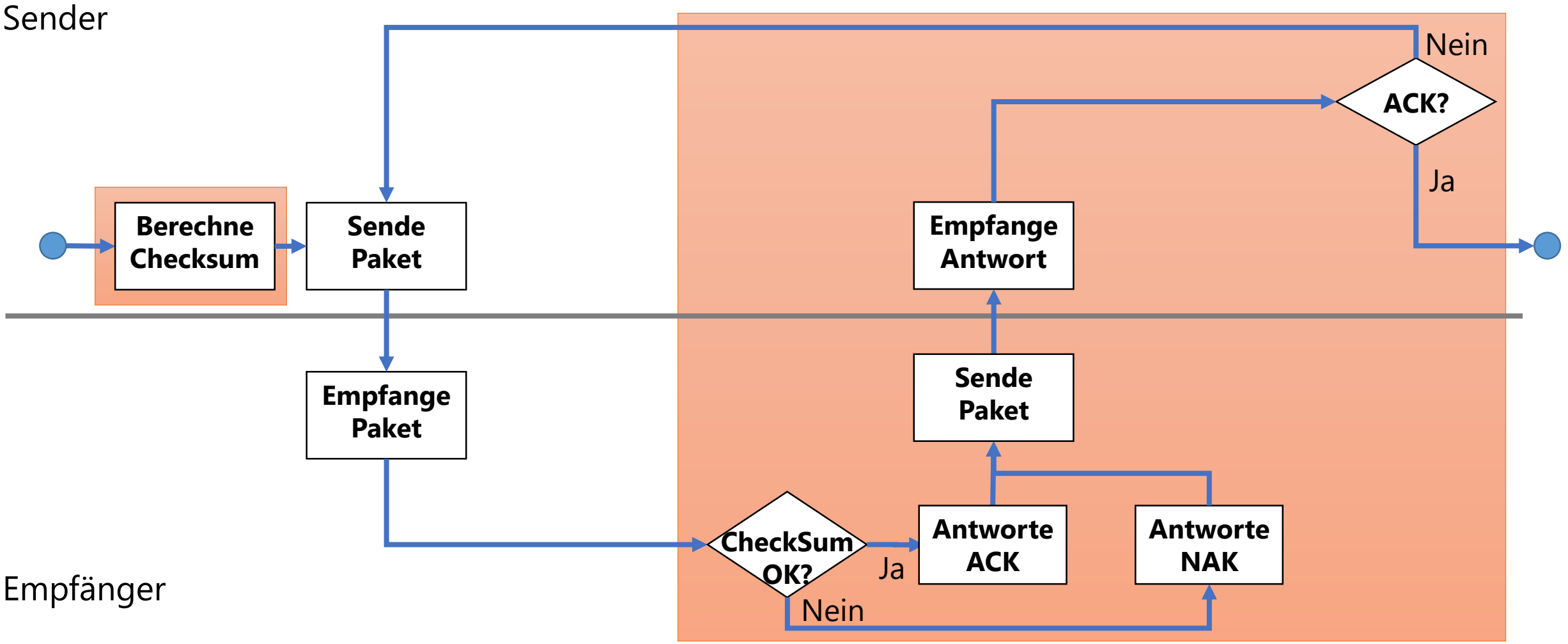
Wir haben eine **bidirektionale Abstimmung** für eine **unidirektionale Nachricht**!

Quelle: http://www.hanne.voorgang.de/rufen.htm

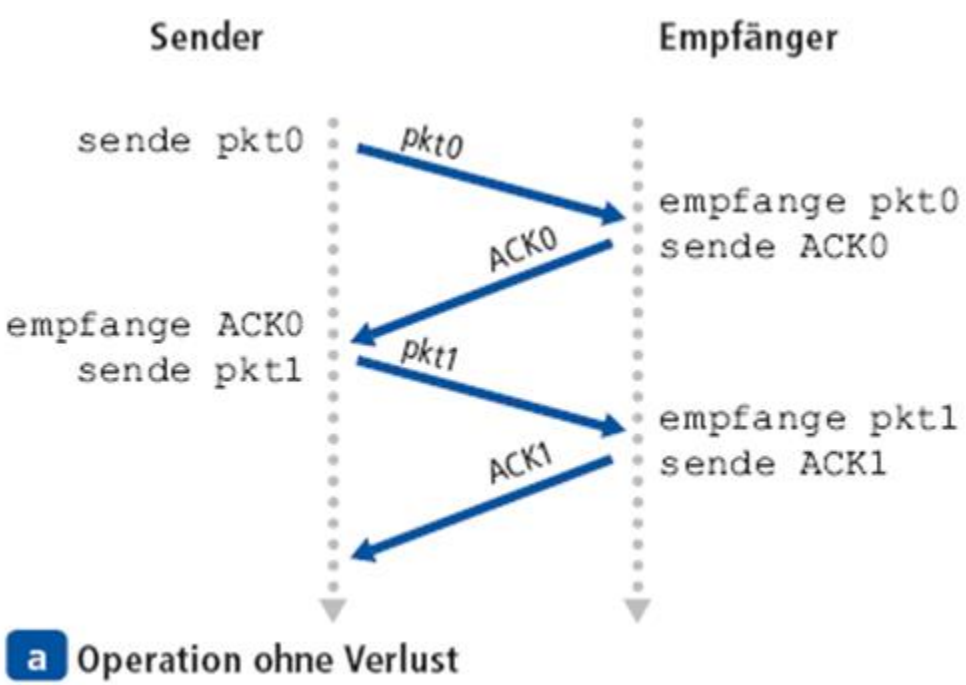


Quelle: http://data.heimat.de

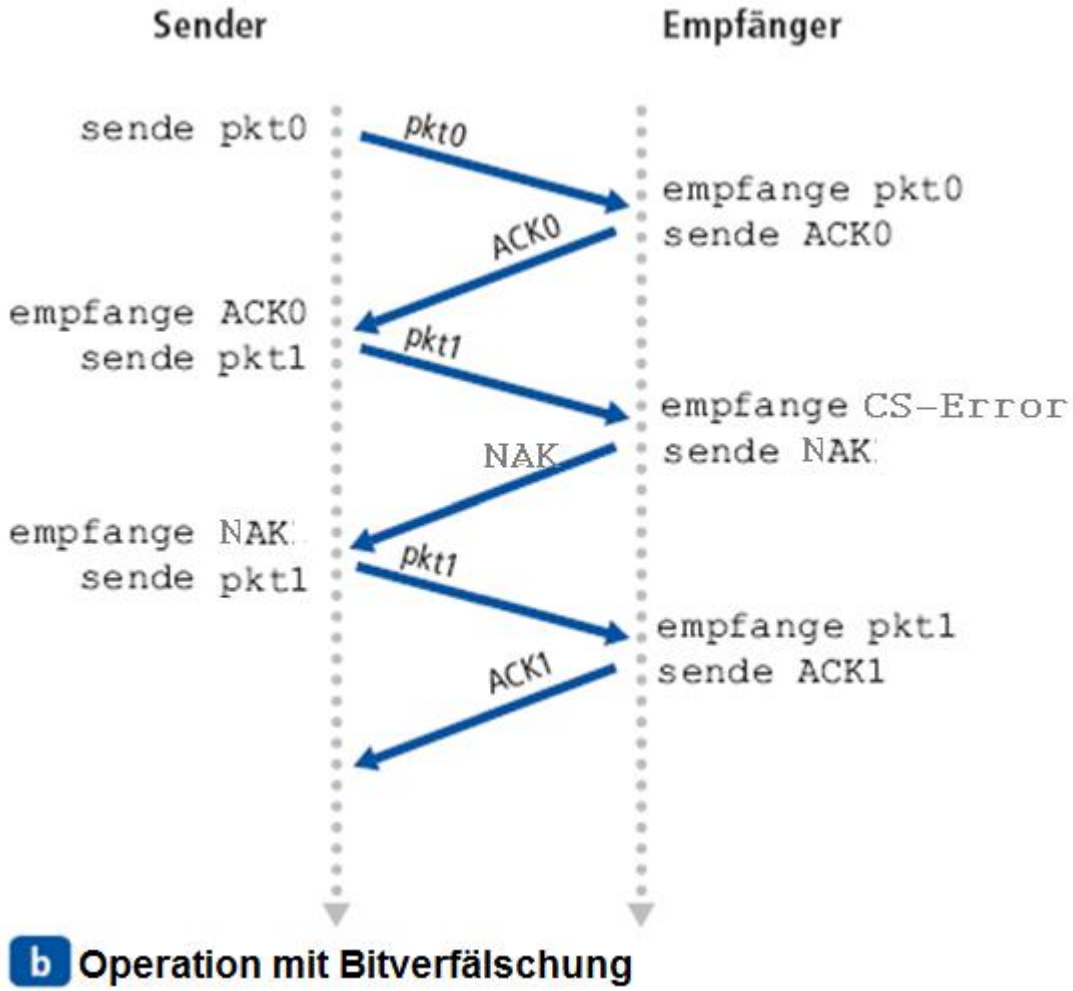
Version 2: Datenübermittlung mit Checksumme und Bestätigung zum Schutz vor Bitfehlern in der Nachricht



Version 2: Ablaufbeispiele



a Operation ohne Verlust



b Operation mit Bitverfälschung

Bildquelle: Kurose/Ross, 241, modifiziert von Kowalk

Fragen an Version 2

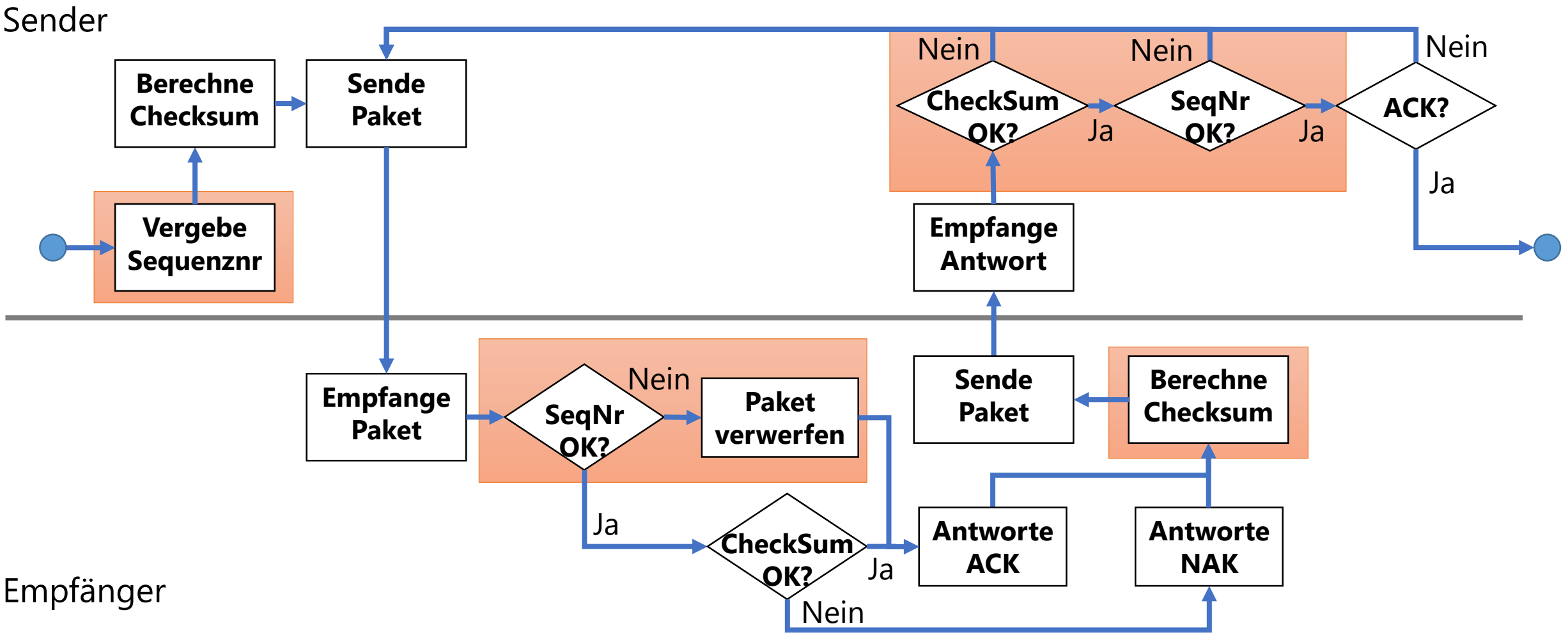
- Status:
  - Der Empfänger weiß nicht, ob sein ACK/NAK beim Sender ankommt.
  - Der Sender weiß nicht, ob die ACK/NAK-Nachricht korrekt ist.
- Was passiert, wenn ein ACK/NAK verfälscht wird?
- Wie kann ein ACK/NAK gesichert werden?
- Wenn ein ACK versehentlich zum NAK wird, sendet der Sender erneut. Der Empfänger erhält 2x die gleiche Nachricht. Und nun?
- Wenn ein NAK versehentlich zum ACK wird, ist die Nachricht verloren.

Antworten auf die Fragen an Version 2

- Auch ACK/NAK werden per Prüfsumme gesichert.
- Sender wiederholt die Übertragung, wenn ACK/NAK verfälscht wurde.
- Behandlung von Duplikaten:
    - Sender überträgt eine eindeutige Sequenznummer im Paket.
    - Empfänger verwirft Pakete mit der gleichen Sequenznummer.

Wichtig: Der Empfänger weiß NICHT, ob der Sender das letzte ACK/NAK unverfälscht empfangen hat! Das lässt sich erst am nächsten empfangenen Datenpaket erkennen.

Version 3: Datenübermittlung mit Sequenznummer und Prüfsumme in ACK/NAK



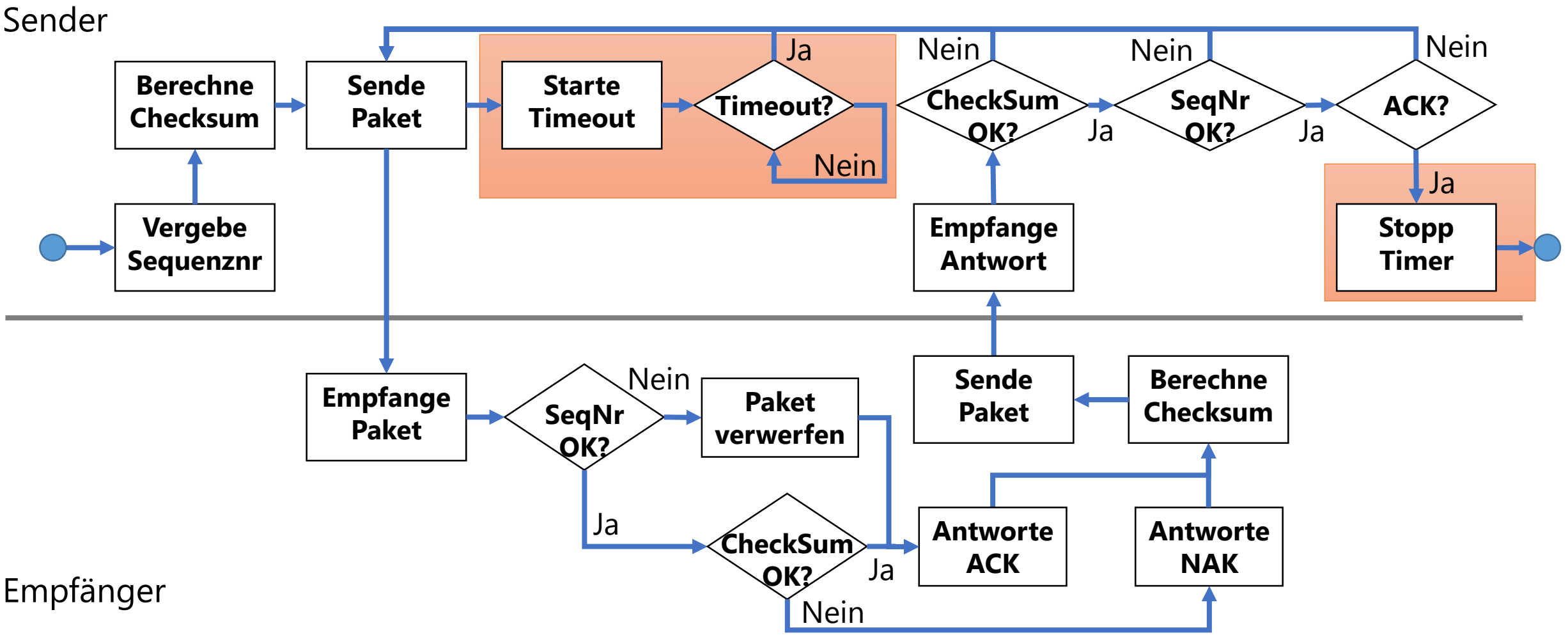
Fragen an Version 3

- Wie könnte wer feststellen, dass ein Paket (seien es Daten, ACK oder NAK) komplett verloren gegangen ist?

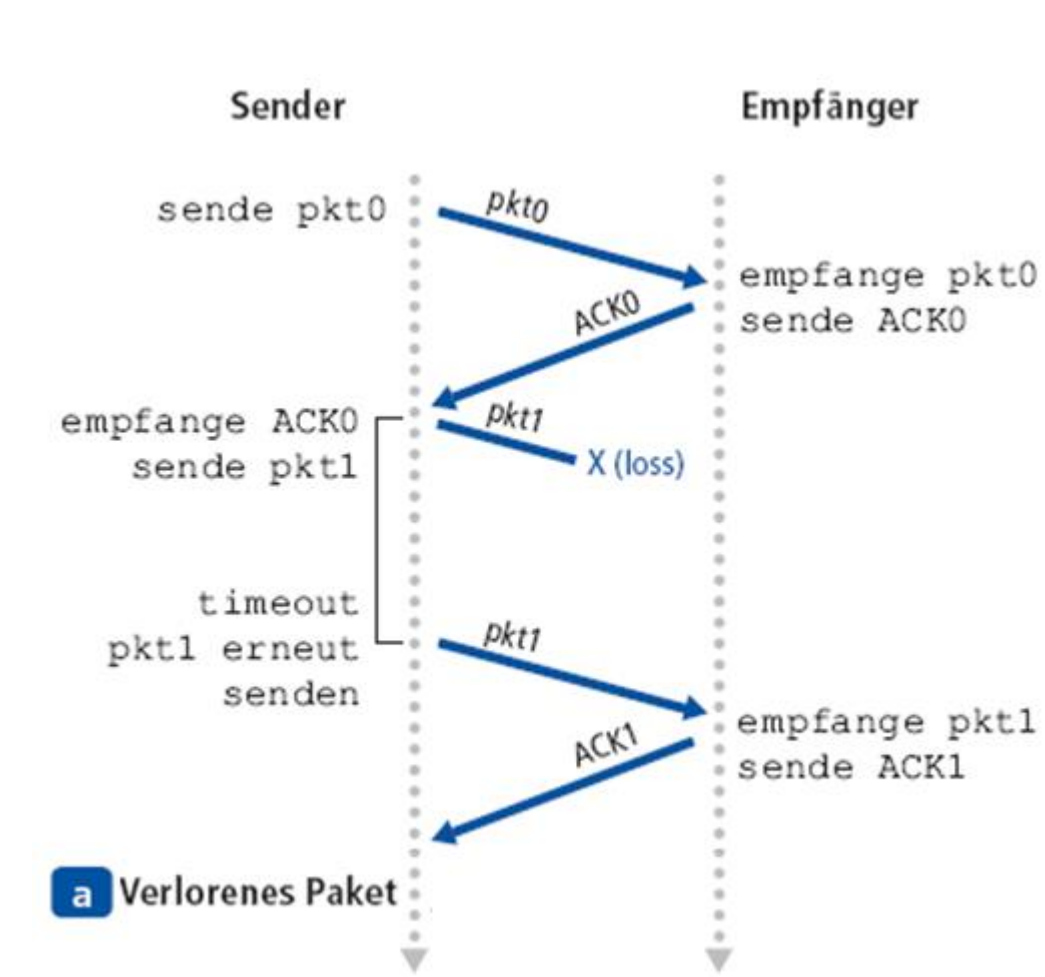
Antworten auf die Fragen an Version 3

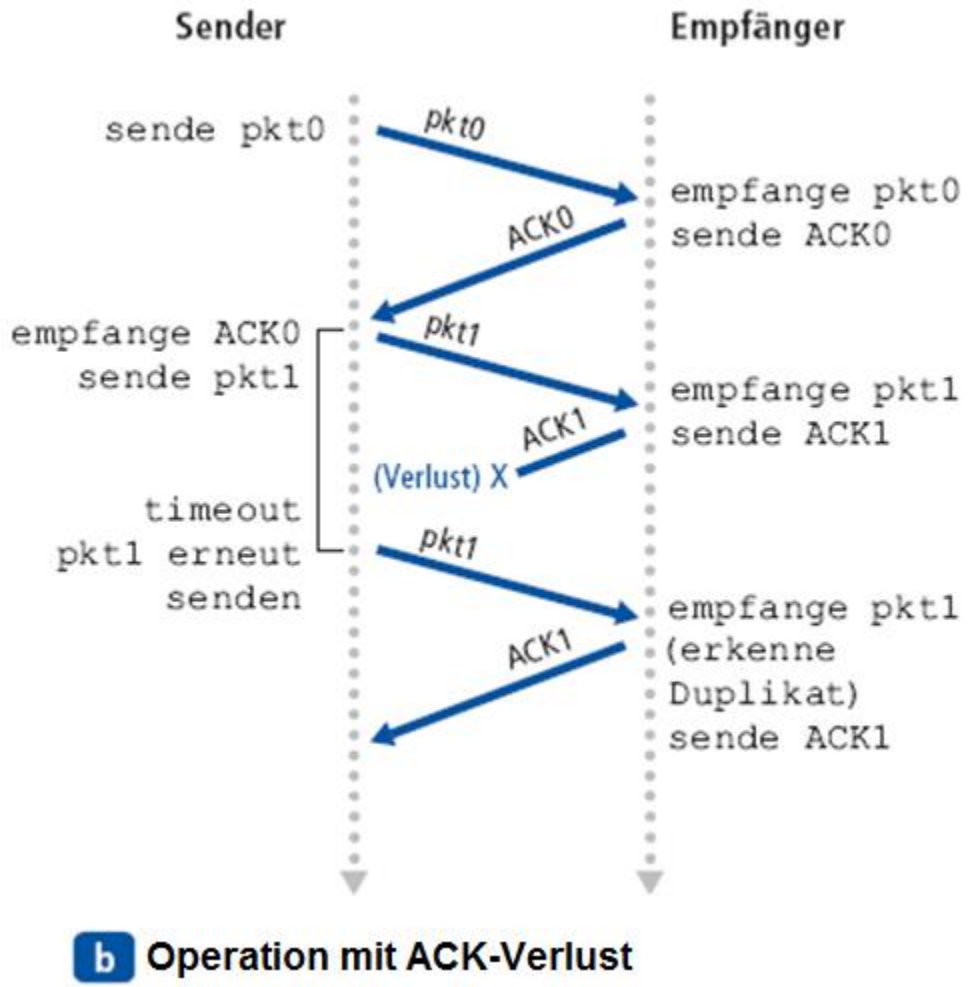
- Wie könnte wer feststellen, dass ein Paket (seien es Daten, ACK oder NAK) komplett verloren gegangen ist?
    - Ein Timer beim Sender wartet eine „sinnvolle“ Zeit. Kommt kein ACK, wird neu gesendet.

Version 4: Datenübermittlung mit Timer



Version 4: Ablaufbeispiele





Bildquelle: Kurose/Ross, 241, modifiziert durch Kowalk

Fragen an Version 4

- Um die Komplexität zu reduzieren und Zustände zu sparen wollen wir die Zahl der möglichen Antworten von 2 auf 1 halbieren.

- Wie können wir ohne NAK auskommen?

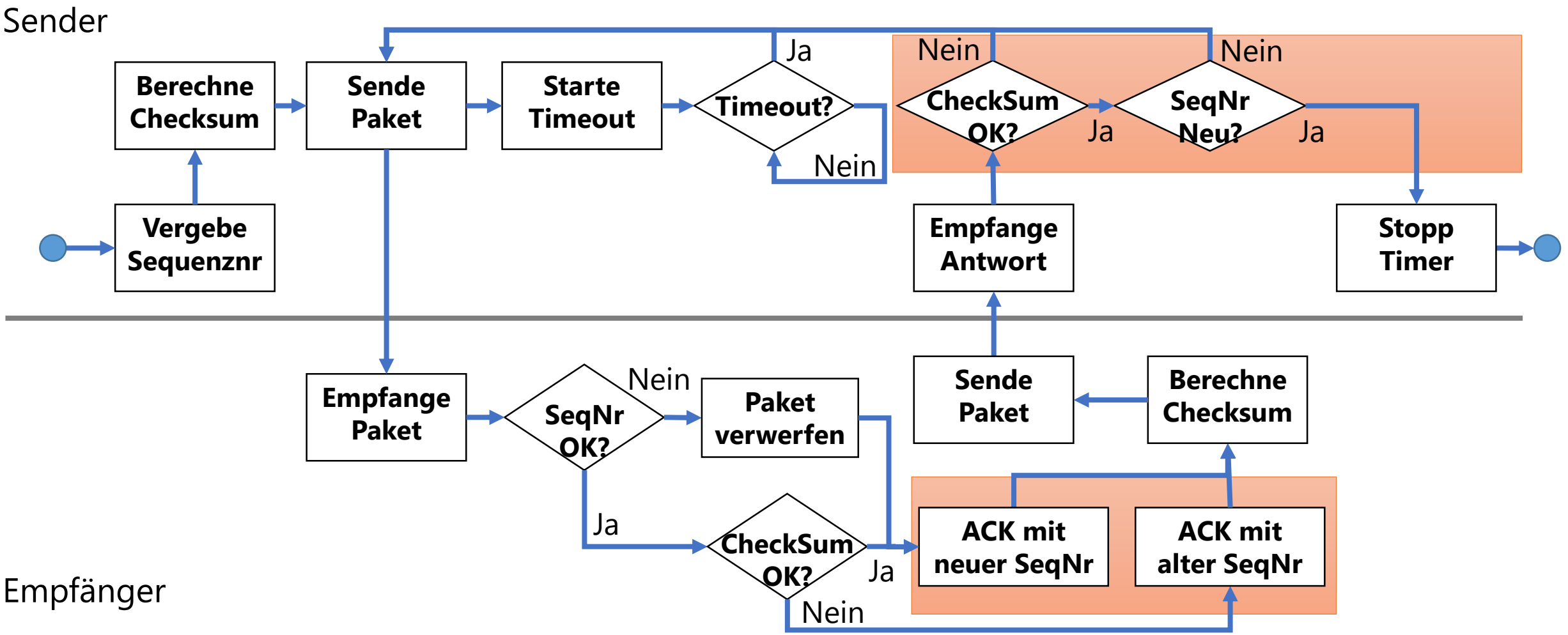
Antworten auf die Fragen an Version 4

- Um die Komplexität zu reduzieren und Zustände zu sparen wollen wir die Zahl der möglichen Antworten von 2 auf 1 halbieren.

- Wie können wir ohne NAK auskommen?

    - Der Empfänger liefert im ACK die letzte erfolgreich empfangene Sequenznummer zurück. Wird eine alte Sequenznummer empfangen oder die Checksumme ist ungültig, wird dies als NAK gedeutet und neu gesendet.

Version 5: Datenübermittlung ohne NAK



Fragen an Version 5

- Zu Version 2 schrieben wir:
  - „**Wichtig:** Der Empfänger weiß NICHT, ob der Sender das letzte ACK/NAK unverfälscht empfangen hat! Das lässt sich erst am nächsten empfangenen Datenpaket erkennen.“

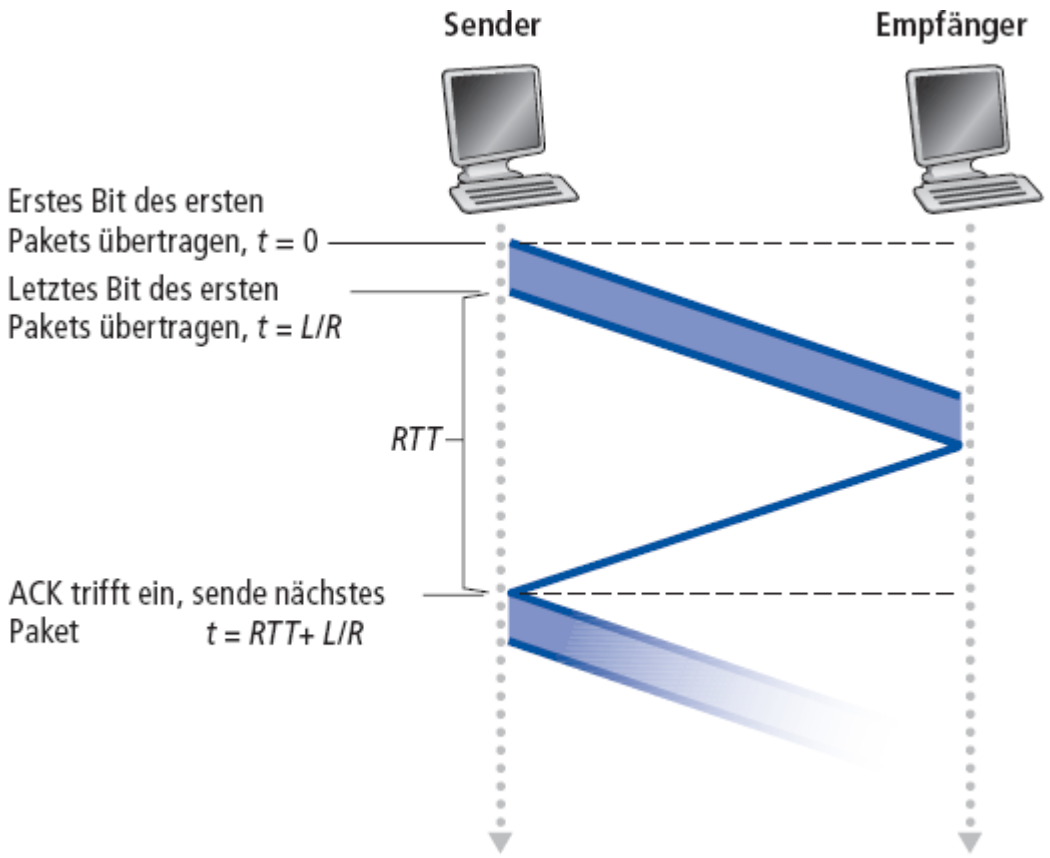
- Wie weiß der Empfänger, dass er alles empfangen hat und seine letzte Bestätigung angekommen ist?

- Wie weiß der Empfänger, dass die erste empfangene Nachricht auch die erste gesendete ist?

Antworten auf die Fragen an Version 5

- Damit der Empfänger die Vollständigkeit der Kommunikation nachvollziehen kann, brauchen wir eine Erweiterung des Protokolls:
  - Eröffnung er Kommunikation (Handshake)
  - Abschluss der Kommunikation

Performance unserer Datenübermittlung



Ablauf bei Stop-and-Wait

Die Datenübermittlung funktioniert, aber die Performance ist schlecht!

Beispiel: R=1 GBit/s Link, 15 ms Ausbreitungs-verzögerung (RTT=30 ms), L=8000 Bit Paketgröße:

$$T_{transmit} = \frac{L}{R} = \frac{8000\ bit}{10^9\ Bit/s} = 8 \cdot 10^{-3}\ ms$$

$$U_{sender} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

U_{sender}: Utilization (engl. für Auslastung) – Anteil der Zeit, in der der Sender tatsächlich sendet.

Einmal 8000 Bit alle ~30 ms -> 33KBit/s Durchsatz über einen Link mit 1 GBit/s. Das Protokoll beschränkt die Ausnutzung physikalischer Ressourcen!

Bildquelle: Kurose/Ross, 243

Frage zur Performance

- Welche Möglichkeiten haben wir, die keinen Einfluss auf das Netzwerk in der Mitte haben, die Performance deutlich zu steigern?

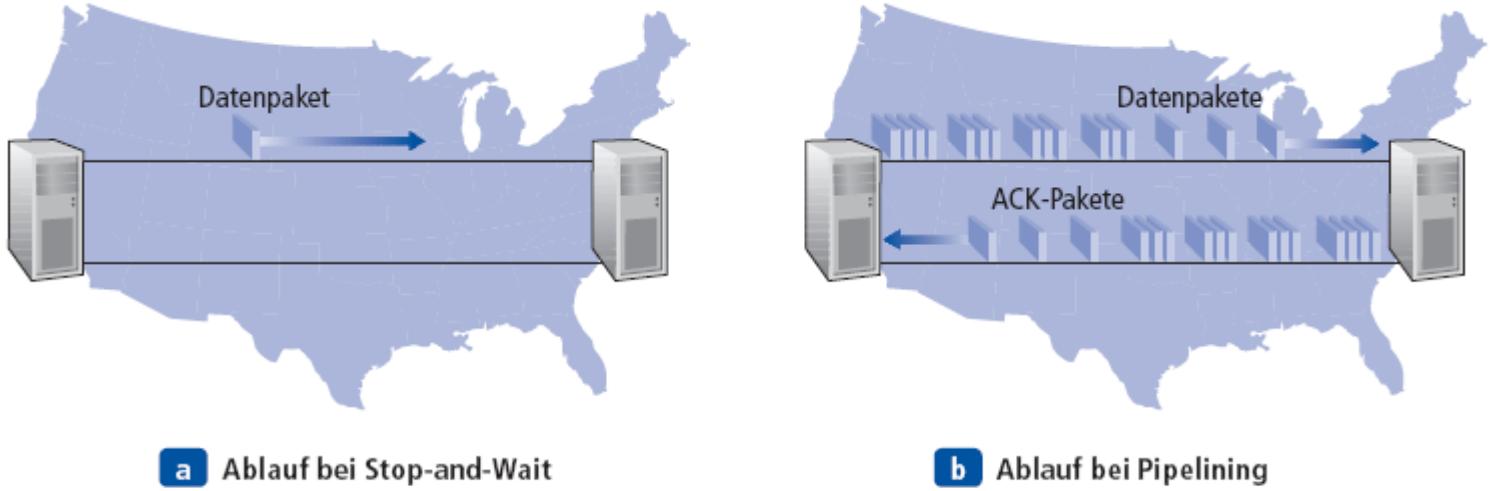
Antwort zu der Frage zur Performance

- Welche Möglichkeiten haben wir, die keinen Einfluss auf das Netzwerk in der Mitte haben, die Performance deutlich zu steigern?
  - Wir können mehrere Pakete abschicken, ohne auf eine Bestätigung zu warten (Pipelining).
  - Aufwand beim Sender:
    - Wir müssen Pakete puffern, um sie nochmal senden zu können.
    - Wir müssen mehrere Timer verwalten.
  - Aufwand beim Empfänger:
    - Wir müssen Pakete Puffern, da sie in falscher Reihenfolge ankommen können.
  - Aufwand gesamt:
    - Im Handshake muss geklärt werden, wer den kleineren Puffer hat. Das begrenzt den Umfang der maximal unbestätigten Pakete.

Protokolle mit Pipelining

Pipelining: Sender lässt nicht nur eines, sondern mehrere unbestätigte Pakete zu

- Die Anzahl der Sequenz-nummern muss erhöht werden.
- Pakete müssen beim Sender und/oder Empfänger gepuffert werden.

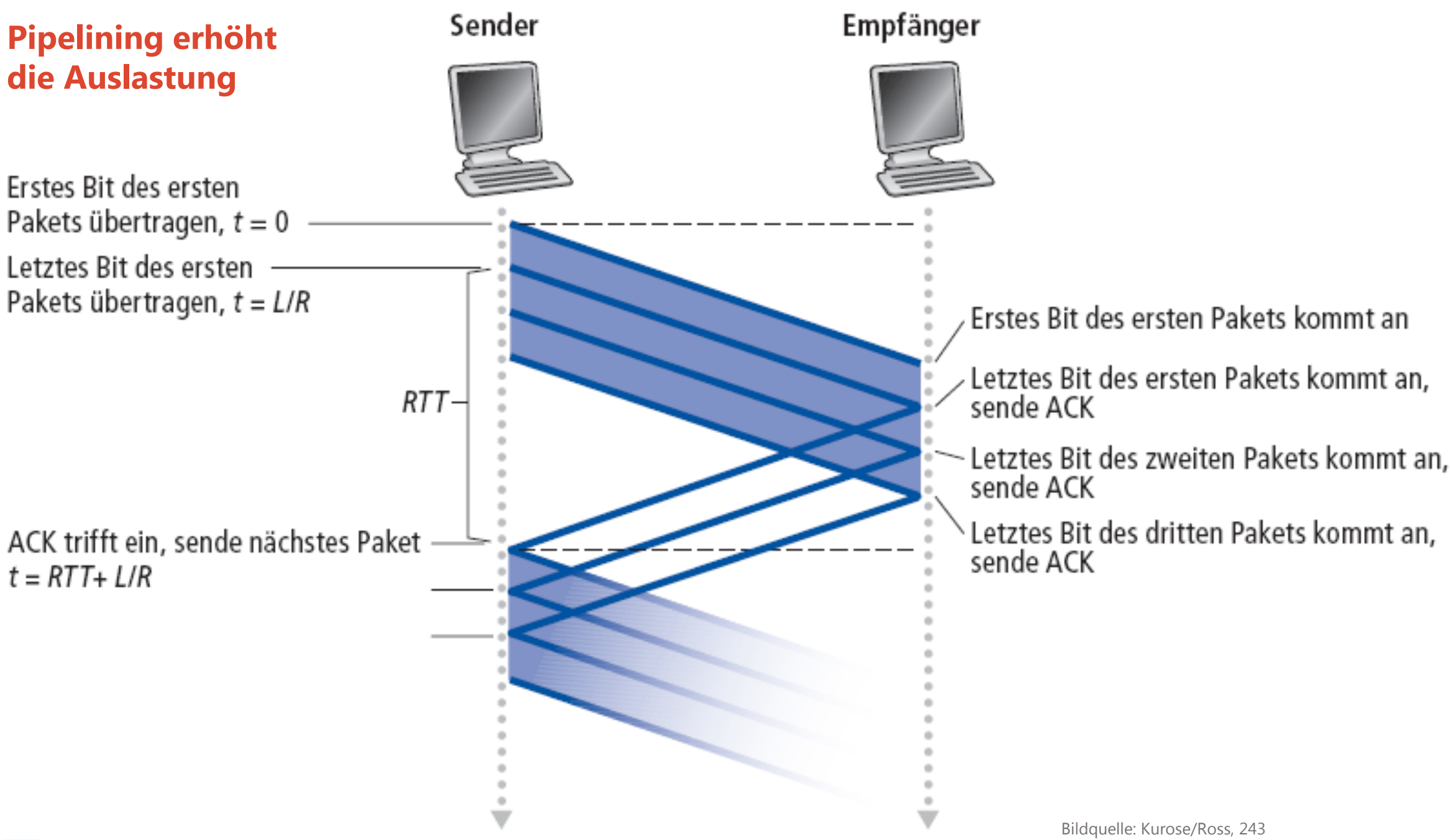


a Ablauf bei Stop-and-Wait

b Ablauf bei Pipelining

Bildquelle: Kurose/Ross, 242

Pipelining erhöht die Auslastung



b Ablauf bei Pipelining

Bildquelle: Kurose/Ross, 243

Frage zum Pipelining

- Wie kann das Pipelining umgesetzt werden?

Antwort auf die Frage zum Pipelining

- Wie kann das Pipelining umgesetzt werden?

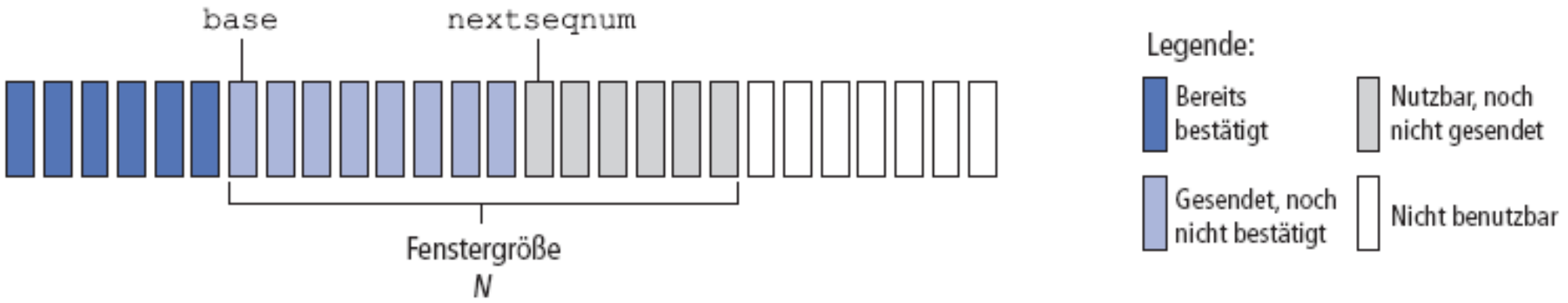
- Zwei prinzipielle Arten von Protokollen mit Pipelining:
  - **Go-Back-N**
    - ACK(n): Bestätigt **alle Pakete** bis zu (und einschließlich) dem Paket mit Sequenznummer n
    - Timeout(n): alle Pakete mit der Sequenznummer n und höher neu übertragen

  - **Selective Repeat**
    - Empfänger bestätigt **jedes empfangene Paket** einzeln.
    - Nur das einzelne Paket wird beim Timeout wiederholt

Go-Back-N (GBN)

Sender:

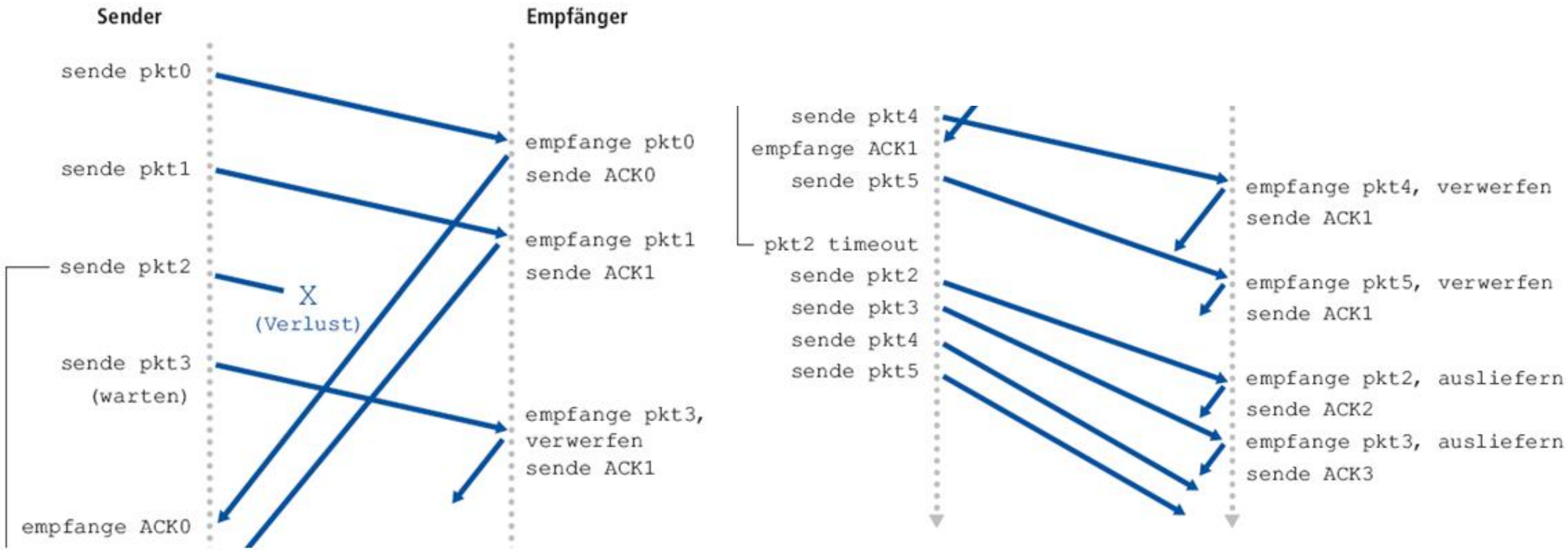
- k-Bit-Sequenznummer im Paketkopf
- Ein „Fenster" von bis zu N aufeinanderfolgenden unbestätigten Paketen wird zugelassen



- ACK(n): Bestätigt alle Pakete bis zu (und einschließlich) dem Paket mit Sequenznummer n
- Doppelte ACKs sind möglich
- Ein Timer für jedes unbestätigte Paket
- Timeout(n): alle Pakete mit der Sequenznummer n und höher neu übertragen

Bildquelle: Kurose/Ross, 245

GBN: Ablauf

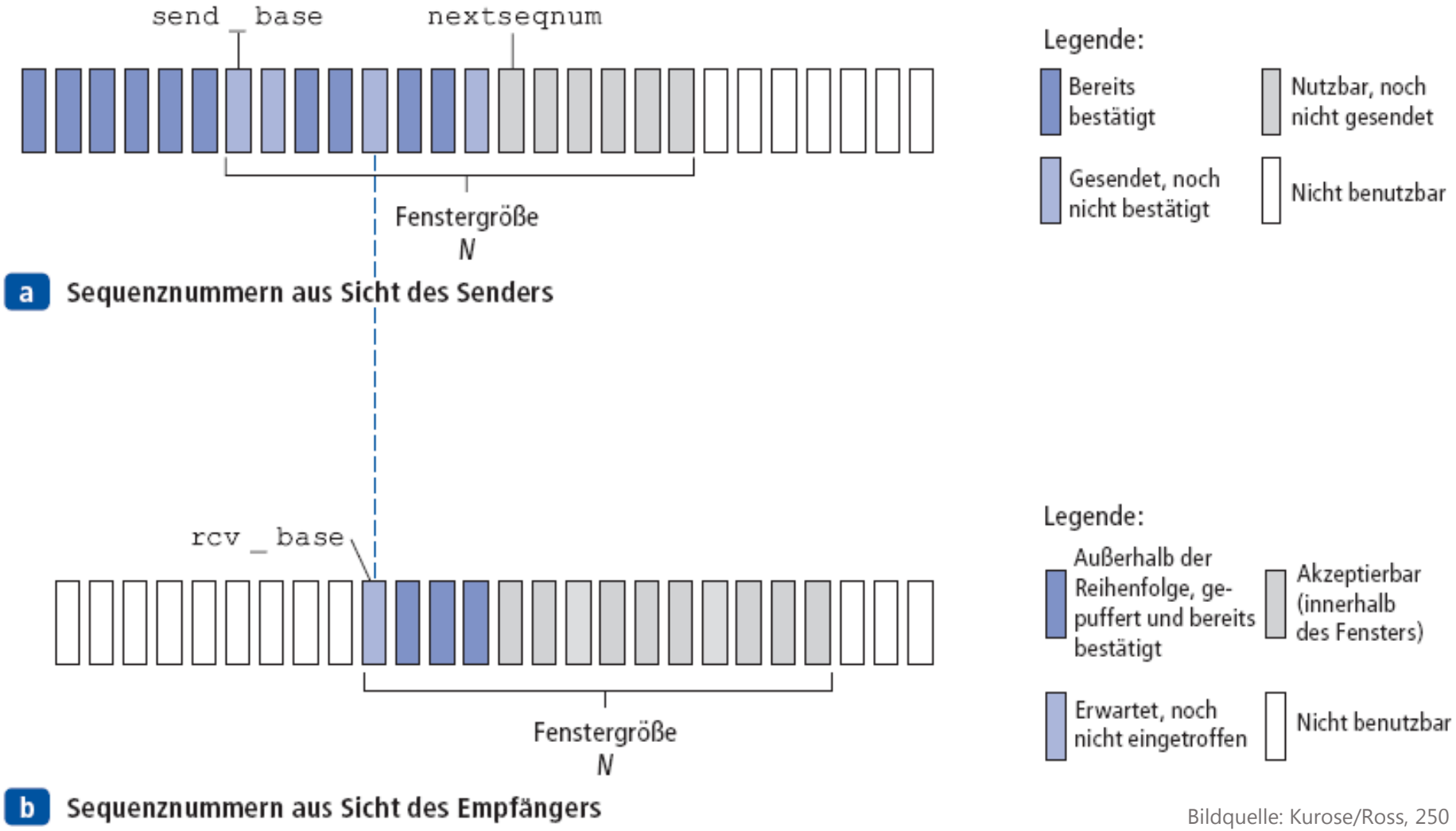


Bildquelle: Kurose/Ross, 248

Selective Repeat (SR)

- Empfänger bestätigt jedes empfangene Paket einzeln.
- Empfänger puffert korrekt empfangene Pakete, die außer der Reihe empfangen wurden, in einem Empfängerfenster:
  - Ausliefern an die nächste Schicht, wenn dies in der richtigen Reihenfolge möglich ist
- Sender wiederholt eine Übertragung nur für individuelle Pakete:
  - Ein Timer für jedes unbestätigte Paket
- Fenster des Senders:
  - N aufeinanderfolgende Sequenznummern
  - Beschränkt wieder die unbestätigten, ausstehenden Pakete

Selective Repeat: Sender- und Empfängerfenster



Bildquelle: Kurose/Ross, 250

Selective Repeat

Sender

Daten von oben:

- Wenn nächste Sequenznummer im Fenster liegt: Paket senden, Timer(n) starten!

Timeout(n):

- Paket n erneut übertragen, Timer(n) starten

ACK(n) aus Fensterpuffer

- Paket n als empfangen markieren
- Wenn n die kleinste unbestätigte Sequenznummer ist, Fenster zur neuen kleinsten unbestätigten Sequenznummer verschieben

Empfänger

Paket im erwarteten Bereich

- Sende ACK(n)
- Außer der Reihe: Puffern
- In der Reihe: Ausliefern (auch alle gepufferten Pakete ausliefern, die jetzt in der Reihe sind), Fenster zum nächsten erwarteten Paket verschieben

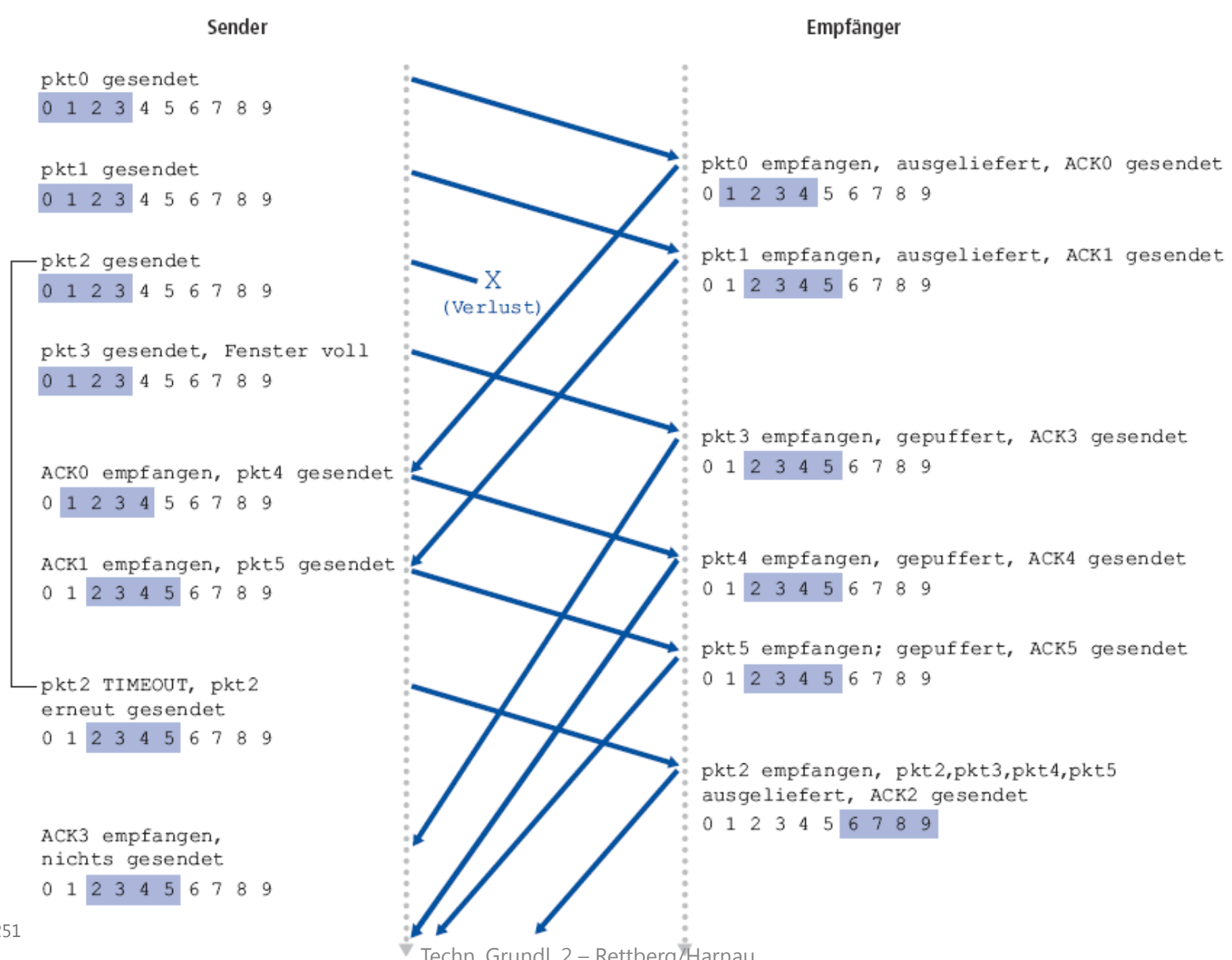
Paket schon empfangen

- ACK(n)

Sonst (völlig aus der Reihe):

- Ignoriere das Paket

Selective Repeat: Ablauf



Bildquelle: Kurose/Ross, 251

Problem: Sequenznummern

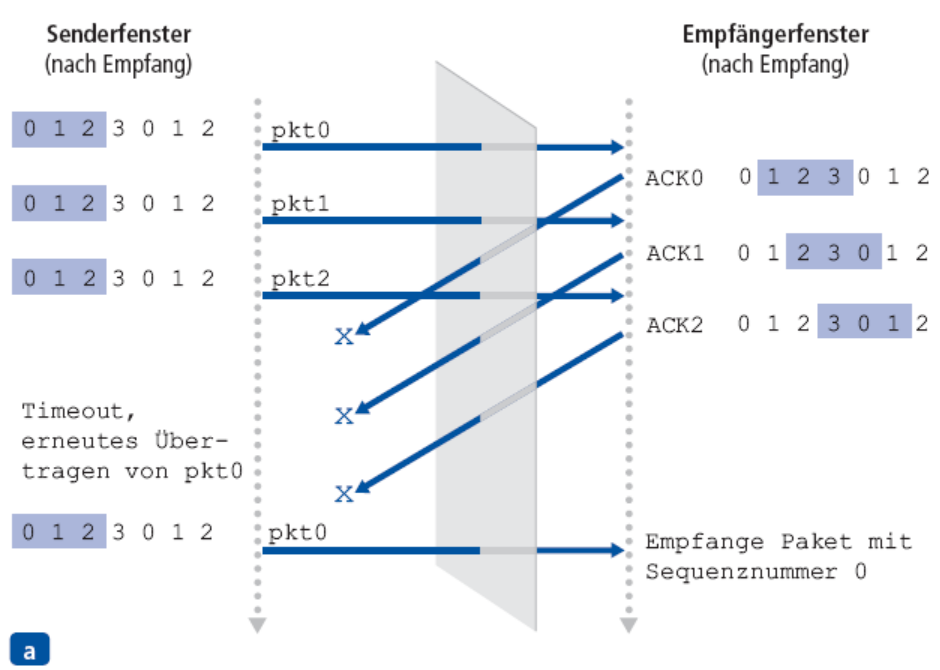
- Beispiel:
  - Sequenznummern: 0-3
  - Fenstergröße=3

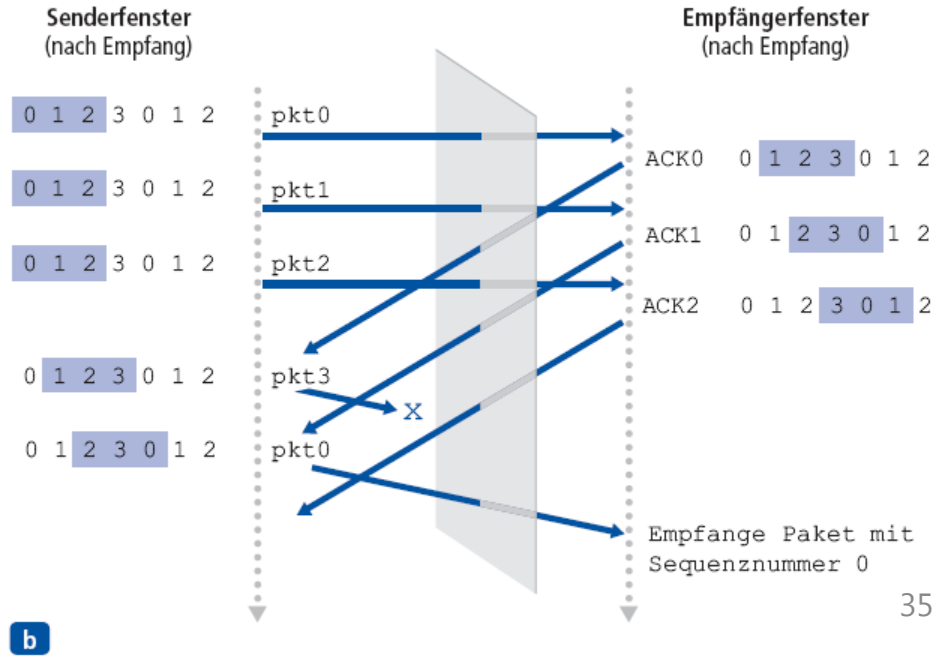
- **Fall a:** Erste Paket mit Sequenznummer 0 kommt erneut.

- **Fall b:** Zweites Paket mit Sequenznummer 0 kommt erstmalig.

- Empfänger kann beide Fälle nicht unterscheiden und gibt verdoppeltes altes Paket als neue Daten nach oben!

- Frage: Welche Relation muss zwischen Fenstergröße und der Anzahl an Sequenznummern bestehen?





Bildquelle: Kurose/Ross, 253

Rekapitulieren Sie: Mechanismen für die zuverlässige Datenübertragung

Mechanismus	Einsatzzweck, Kommentare