Computernetzwerke

Sicherungsschicht

Sebastian Bauer

Wintersemester 2022/2023

Computer Engineering Curriculum

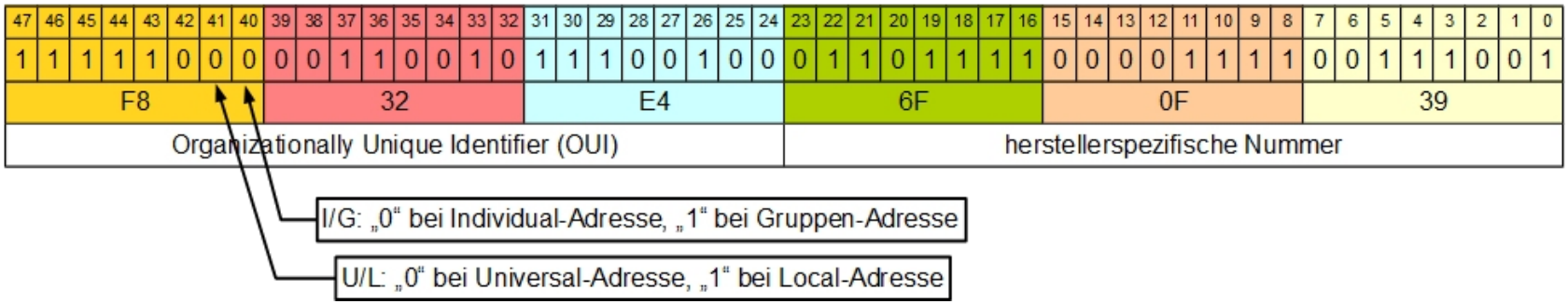
Hybrides Referenzmodell: Sicherungsschicht

Rahmen

- Protokollelemente, die hier ausgetauscht werden heißen Rahmen (engl. *frame*)
- Enthalten schichtspezifische Quell- und Zieladressen
- Informationen zur Datenflusssteuerung
- Nutzdaten der Vermittlungsschicht
- Prüfsumme zur Gewährleistung der Datenintegrität

Adressierung im Kontext der Sicherungsschicht

- Jeder Knoten hat eindeutige Kennung *Media Access Control Address* ≡ MAC-Adresse ≡ *physische Adresse*
- Dauerhaft assoziiert mit Gerät oder Netzwerkadapter, lässt sich aber auch per Software ändern
- Länge beträgt
  - bei Ethernet und Bluetooth 48 Bit = 6 Bytes (EUI-48)
  - bei ZigBee und FireWire 64 Bit = 8 Bytes (EUI-64)
- Aufbau einer 48 Bit MAC-Adresse:



- 48-Bit-Adresse wird oft mit sechs hexadezimalen Bytes notiert, z. B. 8c:16:45:67:35:b0

Spezielle MAC-Adressen

- Broadcast
  - Man möchte einen Rahmen an alle Teilnehmer schicken
  - Hierfür ist Adresse `ff:ff:ff:ff:ff:ff` vorgesehen
  - Solche Rahmen bleiben im Netzwerk (werden nicht geroutet)
- Spanning Tree Protocol (STP)
  - `01:80:c2:00:00:00` als Ziel für alle Bridges im Netz
- IP Multicast (siehe RFC 7042)
  - `01:00:5e:00:00:00` - `01:00:5E:7F:FF:FF` (IPv4)
  - `33:33:00:00:00:00` - `33:33:FF:FF:FF:FF` (IPv6)

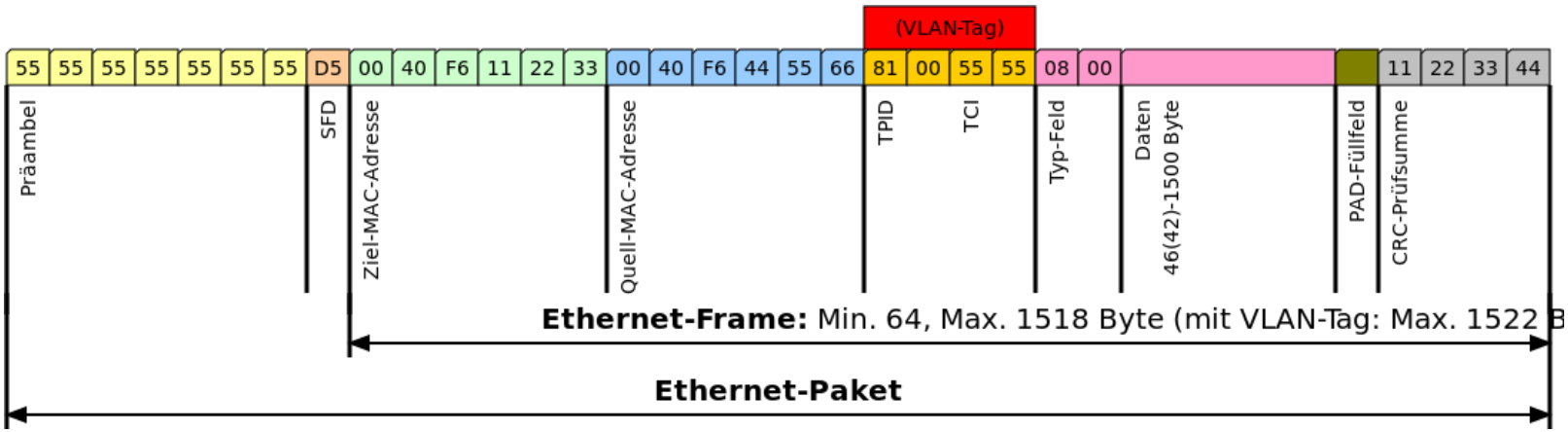
Outline

Ethernet

- Sowohl Bitübertragungsschicht (Kabel) als auch Sicherungsschicht
- Definiert in IEEE 802.3, es gibt Ethernet-I und -II
- Überträgt Daten byteweise, seriell beginnend mit niederwertigstem Bit eines Bytes
- `0xD5` wird somit als `10101011` gesendet
- Datenfelder mit mehr als 1 Byte: BigEndian

Wie wird die Zahl 1234 als 16-Bit-Wert bei Ethernet bitweise übertragen?

Ethernet-II-Rahmen (*Ethernet II Frame*)



- 7-Byte-Präambel zur Synchronisation: `0x55`
- Danach *Start Frame Delimiter* (SFD): `0xD5`

Das Frame (mind. 64 Bytes) enthält:

- MAC-Adressen (Ziel, Quelle)
- Optionales VLAN-Tag (durch spezielle Folge erkenntlich)
- Typ-Feld, dann bis zu 1500 Byte Nutzdaten (bei Jumbo-Frames auch mehr), ggf. aufgefüllt (PAD)
- Prüfsumme

Rahmenende auf Bitübertragungsebene (z. B. Steuercodes)

Effizienz / Overhead

$$\text{Protokol-Overhead} = \frac{\text{Gesamtgröße} - \text{Nutzdatengröße}}{\text{Gesamtgröße}}$$

$$\text{Protokol-Effizienz} = \frac{\text{Nutzdatengröße}}{\text{Gesamtgröße}}$$

Overhead und Effizienz des Ethernet-Rahmens

Wie groß sind Overhead und Effizienz beim Ethernet-Rahmen ohne VLAN-Tag bei

1. 64 Byte Nutzdaten sowie
2. 1500 Byte Nutzdaten?

Ethernet-II-Typfeld

- Gibt den Typ der Payload-Daten für die höhere Schicht an

Wert	Bedeutung
0x0800	IP Internet Protocol, Version 4 (IPv4)
0x0806	Address Resolution Protocol (ARP)
0x0842	Wake on LAN (WoL)
0x8035	Reverse Address Resolution Protocol (RARP)
0x8100	Es folgt das VLAN Tag (VLAN)
0x86DD	IP Internet Protocol, Version 6 (IPv6)

Prüfsumme (FCS, Frame Check Sequence)

- Auf Payload folgender 32-Bit-Wert erzeugt mit Polynom:

$$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$$

Berechnung arbeitet wie folgt:

- Betrachte Bitfolge als (binäres) Polynom p vom Grad n
- Teile mit x^k erweiterte Polynom durch Generatorpolynom vom Grad k
- Der Rest der Division = Prüfsumme, wird an p angehängt

Überprüfung kann so erfolgen:

- Erhaltene Bitfolge ist Polynom vom Grad $n + k$
- Teile Polynom durch Generatorpolynom vom Grad k
- Ist Rest $= 0 \rightarrow$ kein Fehler erkannt

Outline

Beispiel zum Ermitteln der Prüfsumme

Nutzdaten 1101101 werden um eine Prüfsumme aus Polynom $g = x^4 + x^2 + 1$ erweitert. Welche Bits werden übertragen?

1. Generatorpolynom g hat den Grad $k = 4$

$\rightarrow$ Korrespondierende Bitfolge ist

Beispiel zum Ermitteln der Prüfsumme

Nutzdaten 1101101 werden um eine Prüfsumme aus Polynom $g = x^4 + x^2 + 1$ erweitert. Welche Bits werden übertragen?

1. Generatorpolynom g hat den Grad $k = 4$

$\rightarrow$ Korrespondierende Bitfolge ist 10101

2. Nutzdaten entsprechen einem Polynom mit Grad $n =$

Beispiel zum Ermitteln der Prüfsumme

Nutzdaten 1101101 werden um eine Prüfsumme aus Polynom $g = x^4 + x^2 + 1$ erweitert. Welche Bits werden übertragen?

1. Generatorpolynom g hat den Grad $k = 4$

$\rightarrow$ Korrespondierende Bitfolge ist 10101

2. Nutzdaten entsprechen einem Polynom mit Grad $n = 6$

$\rightarrow$ Rahmen mit Erweiterung ist 11011010000 ($\equiv \cdot x^4$)

3. Rest aus Polynomdivision von erweitertem Rahmen und g

$\rightarrow$ 11011010000 mod 10101 =

Beispiel zum Ermitteln der Prüfsumme

Nutzdaten 1101101 werden um eine Prüfsumme aus Polynom $g = x^4 + x^2 + 1$ erweitert. Welche Bits werden übertragen?

1. Generatorpolynom g hat den Grad $k = 4$

$\rightarrow$ Korrespondierende Bitfolge ist 10101

2. Nutzdaten entsprechen einem Polynom mit Grad $n = 6$

$\rightarrow$ Rahmen mit Erweiterung ist 11011010000 ($\equiv \cdot x^4$)

3. Rest aus Polynomdivision von erweitertem Rahmen und g

$\rightarrow$ $11011010000 \mod 10101 = 1011$ bzw.
$x^{10} + x^9 + x^7 + x^6 + x^4 \mod x^4 + x^2 + 1 = x^3 + x^1 + 1$

Wir übertragen also 11011011011.

Überprüfen der Prüfsumme: Korrekter Fall

Wir erhalten die Daten von einer Quelle 11011011011, die mit durch ein Prüfsummenverfahren mit Generatorpolynoms $g = x^4 + x^2 + 1$ abgesichert wurden. Ist ein Fehler erkennbar?

1. Rest aus Polynomdivision von erhaltenden Daten mit g
2. Bitfolge von g: 10101

$\rightarrow$ $11011011011 \mod 10101 =$

Überprüfen der Prüfsumme: Korrekter Fall

Wir erhalten die Daten von einer Quelle 11011011011, die mit durch ein Prüfsummenverfahren mit Generatorpolynoms $g = x^4 + x^2 + 1$ abgesichert wurden. Ist ein Fehler erkennbar?

1. Rest aus Polynomdivision von erhaltenden Daten mit g
2. Bitfolge von g: 10101

$\rightarrow$ $11011011011 \mod 10101 = 0000$

Da sich kein Rest ergibt, trat (wahrscheinlich) kein Fehler bei der Übertragung auf.

Überprüfen der Prüfsumme: Fehlerhafter Fall

Wir erhalten die Daten von einer Quelle 11001011011, die mit durch ein Prüfsummenverfahren mit Generatorpolynoms $g = x^4 + x^2 + 1$ abgesichert wurden. Ist ein Fehler erkennbar?

1. Rest aus Polynomdivision von erhaltenden Daten mit g
2. Bitfolge von g: 10101

$\rightarrow$ 11001011011 mod 10101 =

Überprüfen der Prüfsumme: Fehlerhafter Fall

Wir erhalten die Daten von einer Quelle 11001011011, die mit durch ein Prüfsummenverfahren mit Generatorpolynoms $g = x^4 + x^2 + 1$ abgesichert wurden. Ist ein Fehler erkennbar?

1. Rest aus Polynomdivision von erhaltenden Daten mit g
2. Bitfolge von g: 10101

$\rightarrow$ $11001011011 \mod 10101 = 0010$

Da ein Rest bleibt, trat mit Sicherheit ein Fehler auf (bei den Nutzbits oder bei der Prüfsumme)

Fehlererkennungspolynom

Betrachten empfangene Daten als Originalbitfolge plus Fehlerpolynom:

		Kein Fehler	**Fehler**
Generator	g	10101	10101
Senden	s	11011011011	11011011011
Empfangen	r	11011011011	11001011011
Fehler	e	00000000000	00010000000
Rest	$r \mod g$	0000	0010

Beachte: Im $\mathbb{F}_2$ ist sowohl Addition als Subtraktion ein XOR

- s ist durch g teilbar

$\rightarrow$ $r = s + e$ ist durch g teilbar $\Leftrightarrow$ e durch g teilbar

Erkennung von Ein-Bit-Fehlern

Satz

Wenn das Generatorpolynom mindestens zwei Terme enthält, werden alle Ein-Bit-Fehler erkannt.

- Fehler hat an genau einer Stelle i eine 1, sonst 0
- Fehlerpolynom: $e =$

Erkennung von Ein-Bit-Fehlern

Satz

Wenn das Generatorpolynom mindestens zwei Terme enthält, werden alle Ein-Bit-Fehler erkannt.

- Fehler hat an genau einer Stelle i eine 1, sonst 0
- Fehlerpolynom: $e = x^i$

$\rightarrow$ Wenn g mind. zwei Terme hat, kann es e nie restlos teilen

Erkennung ungerader Anzahl von Fehlern

Satz

Ist $x + 1$ Teiler eines Generatorpolynoms g, dann wird jede ungerade Anzahl von Fehlern erkannt.

- Ungerade Anzahl von Fehlern entspricht Fehlerpolynom e mit ungerader Anzahl von Einsen
- Polynome mit ungerader Anzahl von Einsen sind nicht durch $x + 1$ ($=$ Bitfolge 11) teilbar

$\rightarrow$ e ist nicht durch $x + 1$ teilbar

$\rightarrow$ e ist auch nicht durch g teilbar

(Analogie zu $\mathbb{N}$: 1213 ist nicht durch 2 teilbar, erst recht nicht durch 8, da 2 Teiler von 8)

Bündelfehler

Bündelfehler (engl. *burst error*)

Ein *Bündelfehler* ist ein Kippen von zusammenhängenden Bits.

Beispiel

Bitnummer	01234567890123
Gesendete Bitfolge	00101010010101
Empfangene Bitfolge	00100101110101
Fehler e	

Bündelfehler

Bündelfehler (engl. *burst error*)

Ein *Bündelfehler* ist ein Kippen von zusammenhängenden Bits.

Beispiel

Bitnummer	01234567890123
Gesendete Bitfolge	00101010010101
Empfangene Bitfolge	00100101110101
Fehler e	00001111100000

Erkennung von Bündelfehler

Satz

Sei p ein Generatorpolynom mit Grad k. Durch das Verfahren werden alle Bündelfehler der Länge $r \leq k$ erkannt.

- Startet das Bündel bei i, dann lautet das Fehlerpolynom:

$$e = x^{i+k-1} + \ldots + x^i = x^i \left(x^{k-1} + \ldots + 1\right) = x^i b$$

- Lt. Voraussetzung enthält g immer $x^0 = 1$ als Summand

$\rightarrow$ g und x (und somit auch x^i) sind teilerfremd

$\rightarrow$ Falls also g Teiler von e wäre, dann nur von b

↯ Das geht auch nicht: Grad von b ist kleiner als Grad von g

$\rightarrow$ Bei einem Bündelfehler kommt immer ein Rest heraus

Fehlerbetrachtungen

Wird ein Fehler erkannt, trat

- ganz sicher ein Fehler im kompletten Datenstrom auf (Nutzdaten oder Prüfsumme),
- wahrscheinlich ein Fehler in den Nutzdaten (da diese in der Regel wesentlich länger als die Prüfsumme sind) oder

→ eher unwahrscheinlich ein Fehler in der Prüfsumme (aber nicht ausgeschlossen)

Wird kein Fehler erkannt:

- So wurden die Daten wahrscheinlich fehlerfrei übertragen
- Im unwahrscheinlicheren Fall traten mehrere Fehler in Kombination auf

Was passiert bei einem erkannten Fehler?

- Frame wird von der Schicht verworfen
- Das Protokoll der Schicht oder übergeordnete Schichten behandeln das Problem
  - Neuanforderung
  - Timeout auf Gegenseite

Umsetzung

- Heißt auch zyklische Redundanzprüfung (*cyclic redundancy check*, CRC)
- In HW umsetzbar mit (internen) Feedback-Schieberegister

Mögliche vereinfachte Umsetzung in C für CRC-32 (Ethernet)

```
uint32_t crc32(size_t len, const uint8_t *buf, uint32_t POLY = 0xEDB88320) {
  uint32_t crc = -1;
  while (len--) {
    crc = crc ^ *buf++;
    for (int bit = 0; bit < 8; bit++) {
      if (crc & 1) crc = (crc >> 1) ^ POLY;
      else  crc = (crc >> 1);
    }
  }
  return ~crc;
}
```

Adaptiert von https://github.com/Michaelangel007/crc32

Je nach Prozessorarchitektur ist man ggf. mit einer Tabelle schneller, siehe z. B. https://github.com/gcc-mirror/gcc/blob/master/libiberty/crc32.c

Aufgabe zu Fehlererkennung

Aufgabe zu CRC-5 Polynom

Sei $g = x^5 + x^2 + 1$ ein CRC-5 Polynom (dieses wird z. B. von USB verwendet):

1. Es soll der Text `CE` (ASCII-Kodierung) mit der Prüfsumme aus g erweitert werden. Wie lautet die gesamte übertragene Bitfolge?
2. Die Bitfolge von `01001000 01010100 01010111 11001` wird empfangen. Überprüfe, ob sie fehlerfrei übermittelt worden ist.
3. Nun wird eine Sequenz von `01001111 10001 10100` empfangen. Überprüfe, ob sie fehlerfrei übermittelt worden ist.

Outline

Flusskontrolle

- Fehler kann nicht korrigiert werden
  1. Rahmen wird verworfen und
  2. Rahmen wird erneut gesendet (außer bei verbindungslos)
- *Flusskontrolle* erlaubt dem Empfänger zu steuern, mit welcher Rate die Gegenseite Rahmen senden darf

→ Langsame Empfänger werden nicht überfordert

Beispiel

Ein Empfänger A kann 1 MiB pro Sekunde verarbeiten. Ein Sender B kann 5 MiB pro Sekunde verschicken. A und B müssen sich darauf verständigen, dass B an A nie mehr als 1 MiB pro Sekunde an Daten verschickt.

Flusskontrolle – Mechanismen

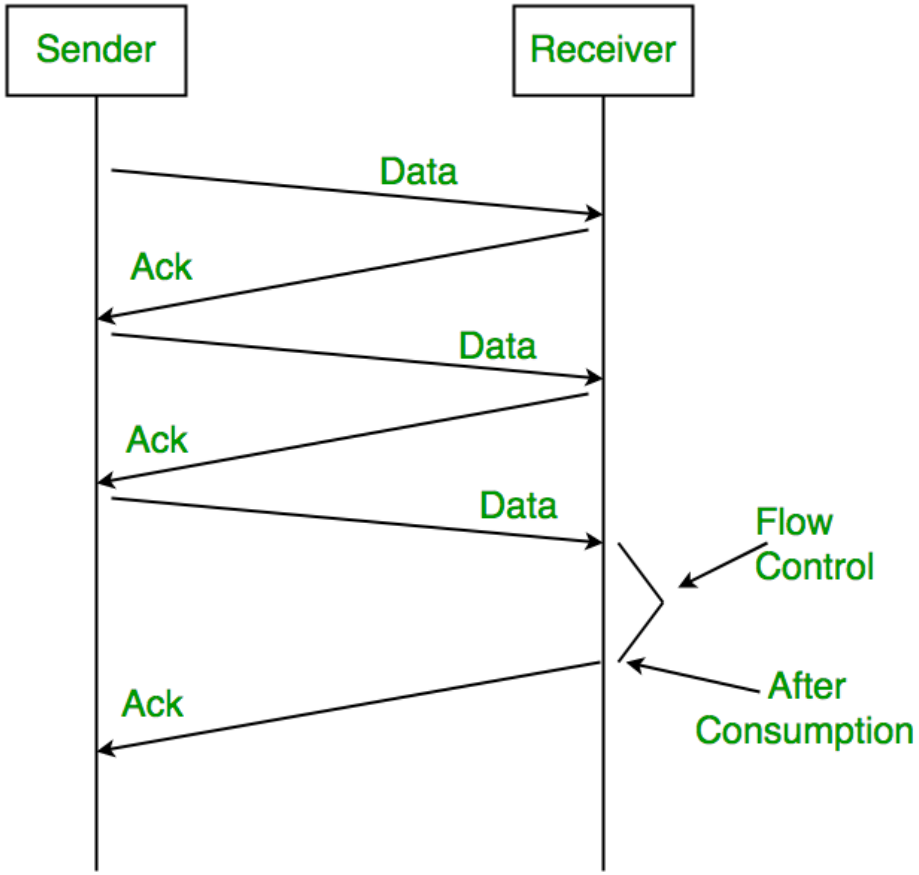
Grundlegende Mechanismen:

- Timeouts
- Bestätigungen (Acknowledgements, ACKs)

Bekannte Verfahren:

- Stop-and-Wait
- Schiebefenster (*sliding window*)

Stop-and-Wait (simplex)



https://www.geeksforgeeks.org/stop-and-wait-arq/

Senderschleife

1. Sendet genau ein Frame
2. Wartet auf Frame

Empfängerschleife

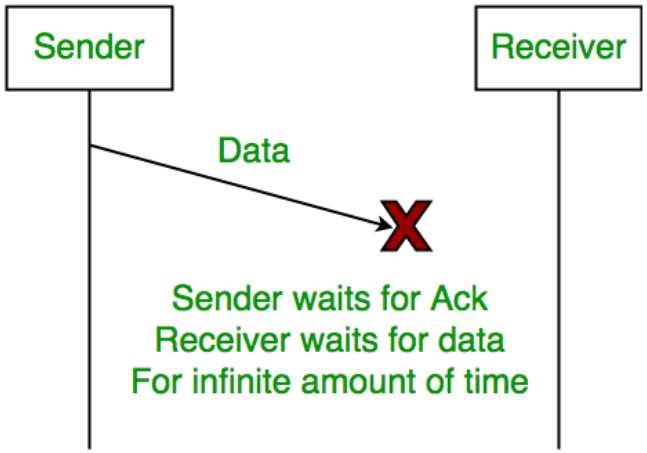
1. Wartet auf Frame
2. Sendet Ack

Welche Probleme können hierbei entstehen?

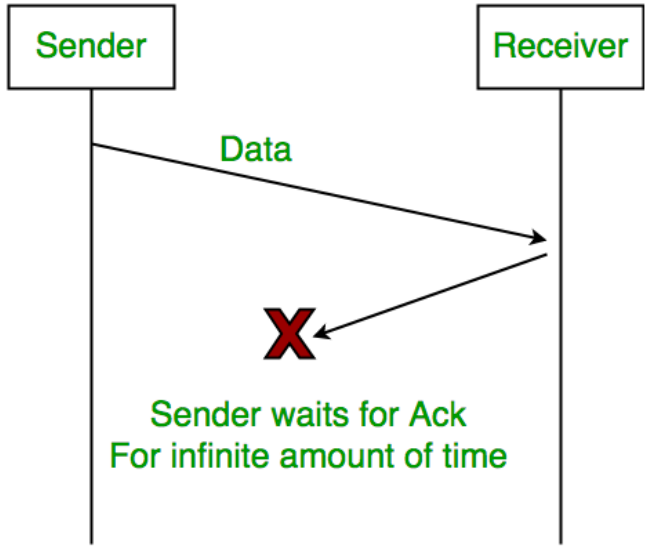
Stop-and-Wait (simplex): Probleme

Bei fehlerhaften Übertragungssystemen entstehen auch folgende Situationen:

1. Verlorenes Senderframe



2. Verlorenes ACK



3. Lang verzögertes ACK: Ein vorheriges, lang verzögertes ACK, wird für ein ACK eines kürzliches gesendeten Rahmens verstanden (wenn Sender überhaupt neuen Rahmen schickt).

Stop-and-Wait: ARQ

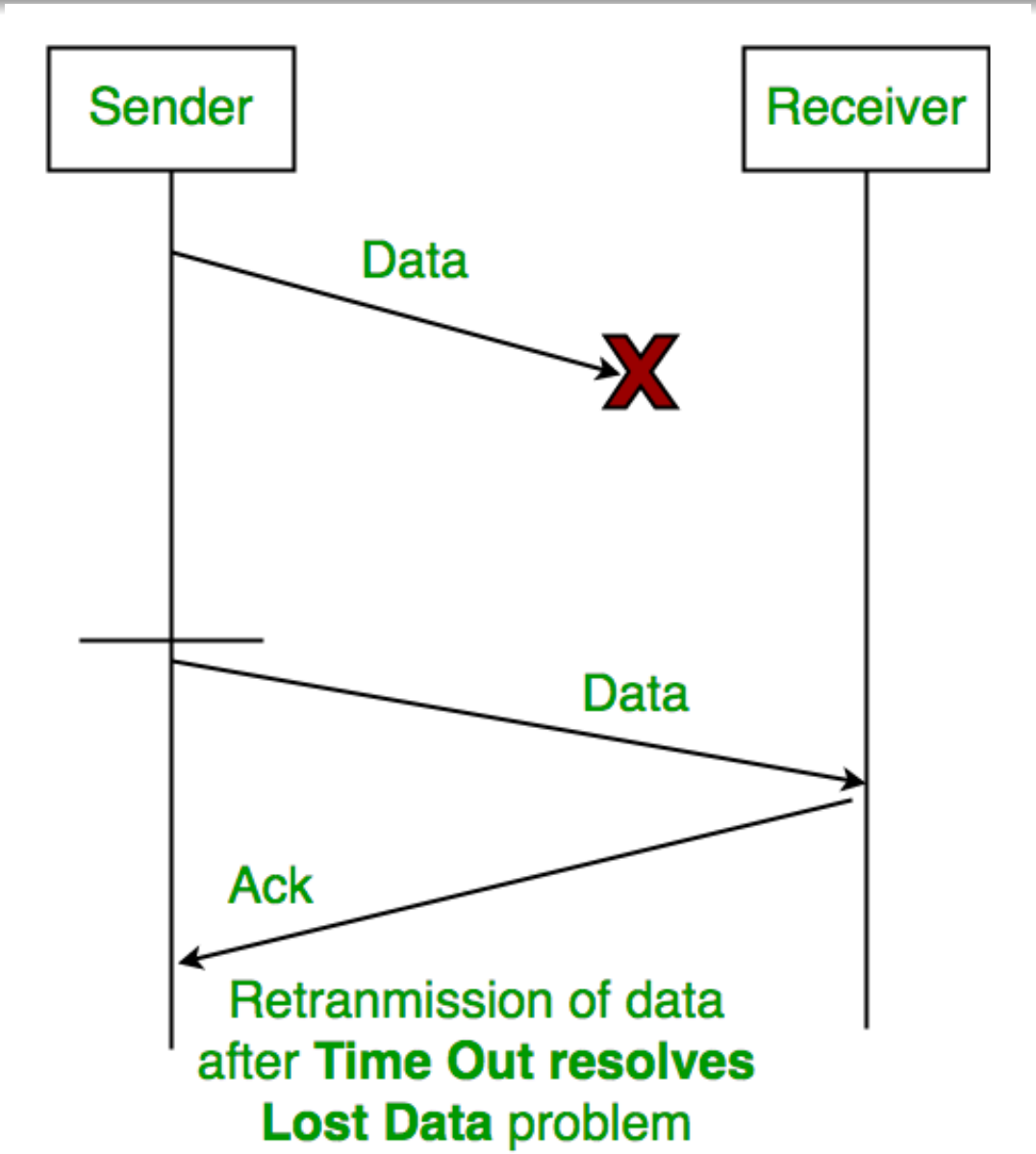
Um die Probleme zu lösen, führen wir ein:

1. Timeout
2. Sequenznummer für Daten
3. Sequenznummer für ACKs

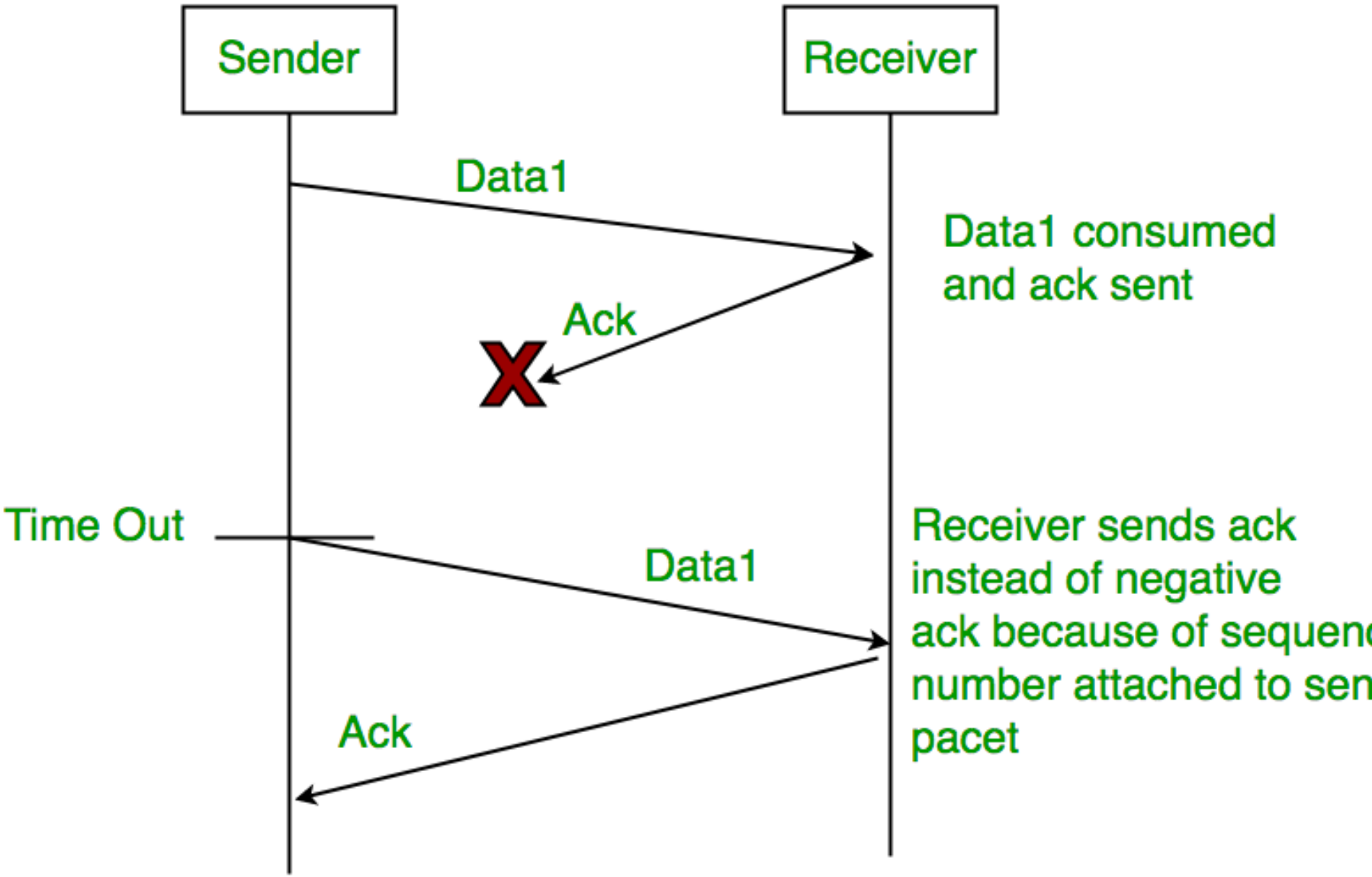
$\Rightarrow$ *Automatic Repeat Request* (ARQ)

Als Sequenznummer genügt zu Beginn ein Zyklus bestehend aus 0 und 1. Verfahren realisiert „Rückwärtsfehlerkorrektur“.

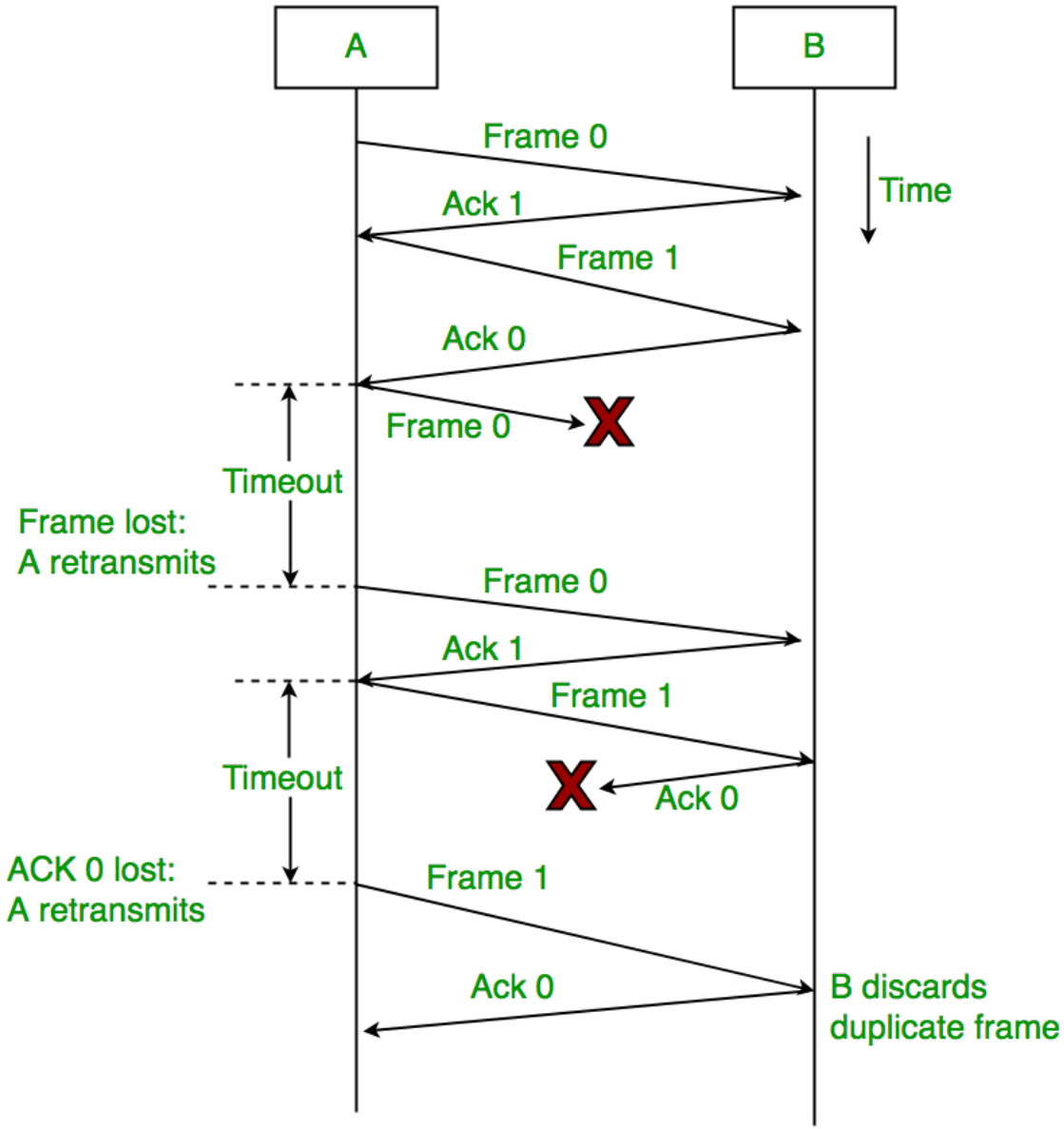
ARQ Stop-and-Wait: Timeout



ARQ Stop-and-Wait: Sequenznummer für Daten



ARQ Stop-and-Wait: Sequenznummer für ACKs



ARQ Stop-and-Wait: NAK

- Bisher: Fehler wird erkennt, Rahmen wird verworfen

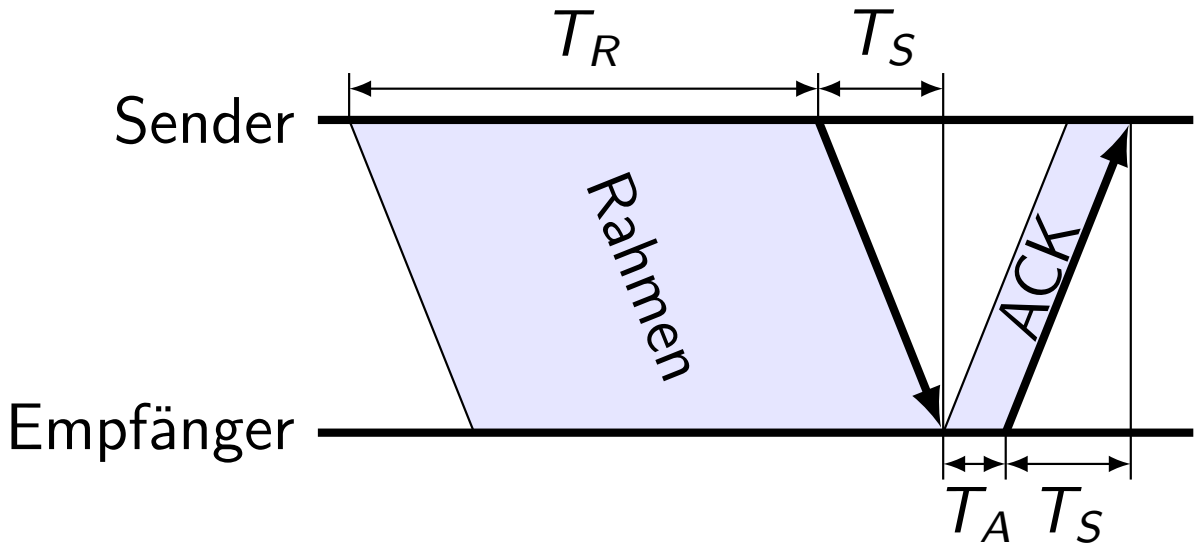
→ Timeout beim Sender wird getriggert

- Wenn Timeout > RTT, kann man auch NAKs senden
- Aber: Auch NAKs können verloren gehen

→ Timeout weiterhin notwendig

ARQ Stop-and-Wait: Effizienz

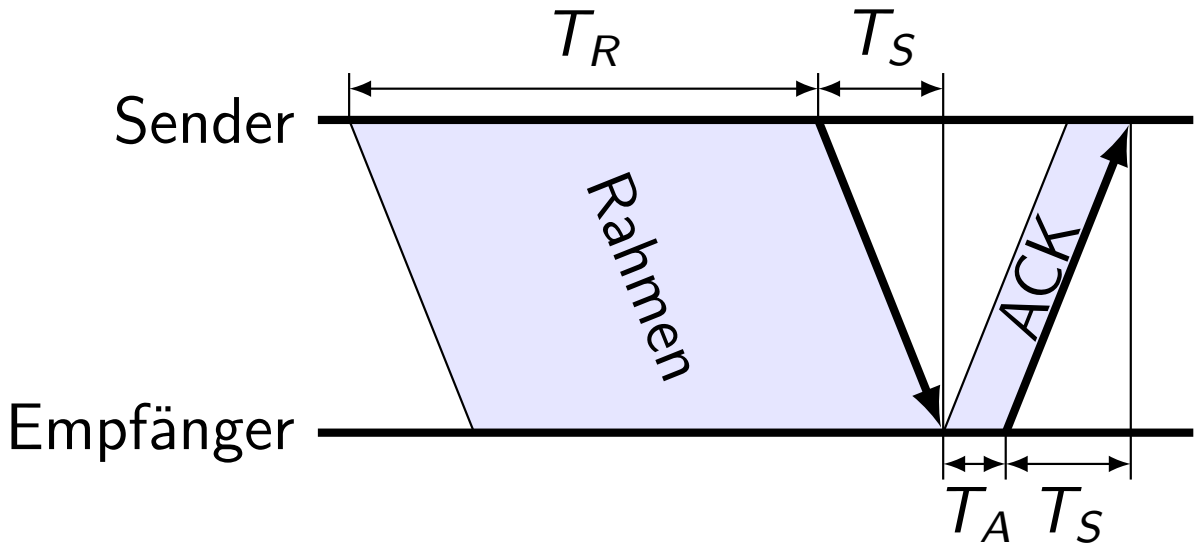
Sei die Datenrate R (Bit/s), die Rahmenlänge L (Bit), dann ist die Übertragungszeit: $T_R = \frac{L}{R}$. Weiterhin sei die Signallaufzeit T_S (s).



Rahmenumlaufzeit (*round trip time*, RTT) =

ARQ Stop-and-Wait: Effizienz

Sei die Datenrate R (Bit/s), die Rahmenlänge L (Bit), dann ist die Übertragungszeit: $T_R = \frac{L}{R}$. Weiterhin sei die Signallaufzeit T_S (s).



Rahmenumlaufzeit (*round trip time*, RTT) $= T_R + T_A + 2T_S$

Mit $T_A \approx 0$ ist die Effizienz: $U = \frac{T_R}{\text{RTT}} = \frac{T_R}{T_R + 2T_S}$

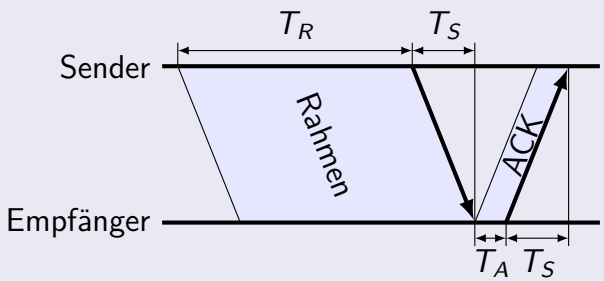
ARQ Stop-and-Wait: Aufgabe

Aufgabe

Zwei Rechner A und B sind mit einem 1 km langen Kabel verbunden, das Signale mit einer Geschwindigkeit von $2 \times 10^8\,\mathrm{m\,s^{-1}}$ überträgt. Rechner A sendet Rechner B Rahmen mit Länge 500 bit. Bestimme die Effizienz, wenn

1. die Datenrate $10\,\mathrm{Mbit\,s^{-1}}$ beträgt sowie
2. die Datenrate $1000\,\mathrm{Mbit\,s^{-1}}$ beträgt.

Cheat-Sheet



$$U = \frac{T_R}{T_R + 2T_S}$$

ARQ Stop-and-Wait: Analyse

- Ist Datenrate in Relation zur RTT groß, dann ist das Verfahren sehr ineffizient
- Bis zum nächsten Senden vergeht mind. die RTT
- Der Sender wartet mehr, als er sendet

→ Abhängigkeit des Senders zu einem ACK oder NAK führt zu stark verminderten effektiven Datenrate

- Wie können wir die effektive Datenrate verbessern?

Schiebefenster (engl. *sliding window*)

- Viele Kommunikationsformen benötigen keine sofortige Antwort, insbesondere bei einem Simplexverfahren (z. B. Streamen eines Videos)
- Idee: Anstatt nur eines Rahmens senden wir nach Bedarf mehrere Rahmen ab, ähnlich des Pipelining digitaler Schaltungen

→ Wir warten also nicht mehr auf jedes einzelne ACK

- Sequenzzähler haben jetzt größeren Wertebereich
- Empfänger sendet ein ACK oder mehrere davon gebündelt

Schiebefenster – Funktionsweise

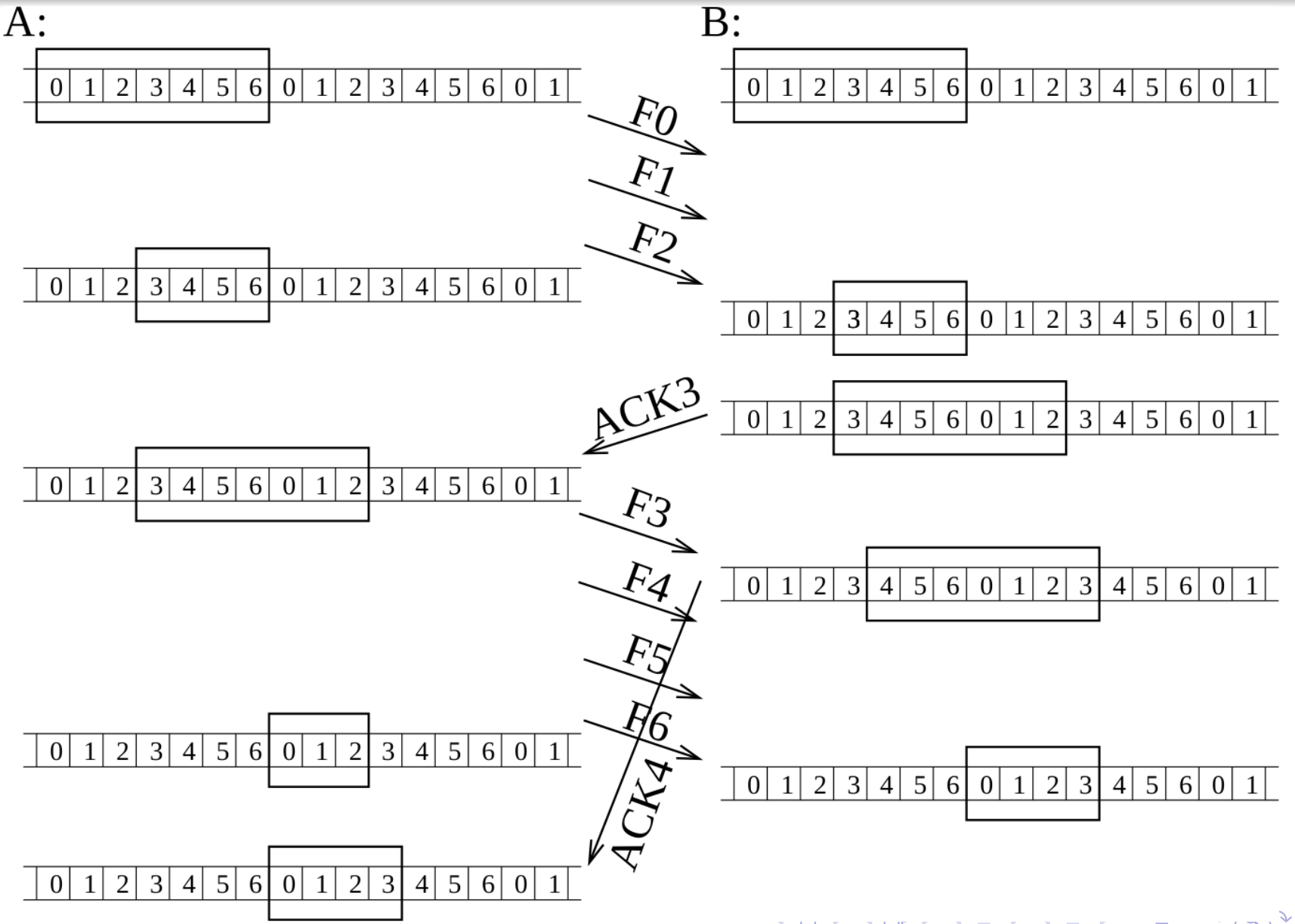
Sender managt Sequenznummern zum Senden (Sendefenster):

- Fenster hat Größe $n \Leftrightarrow$ Es können noch n Rahmen ohne ACK übertragen werden
- Sendefenster
  - verkleinert sich beim Senden von Rahmen
  - vergrößert sich bei Empfang von ACKs
  - Sender wartet, wenn das Fenster leer ist

Empfänger verwaltet empfangbare Sequenznummern

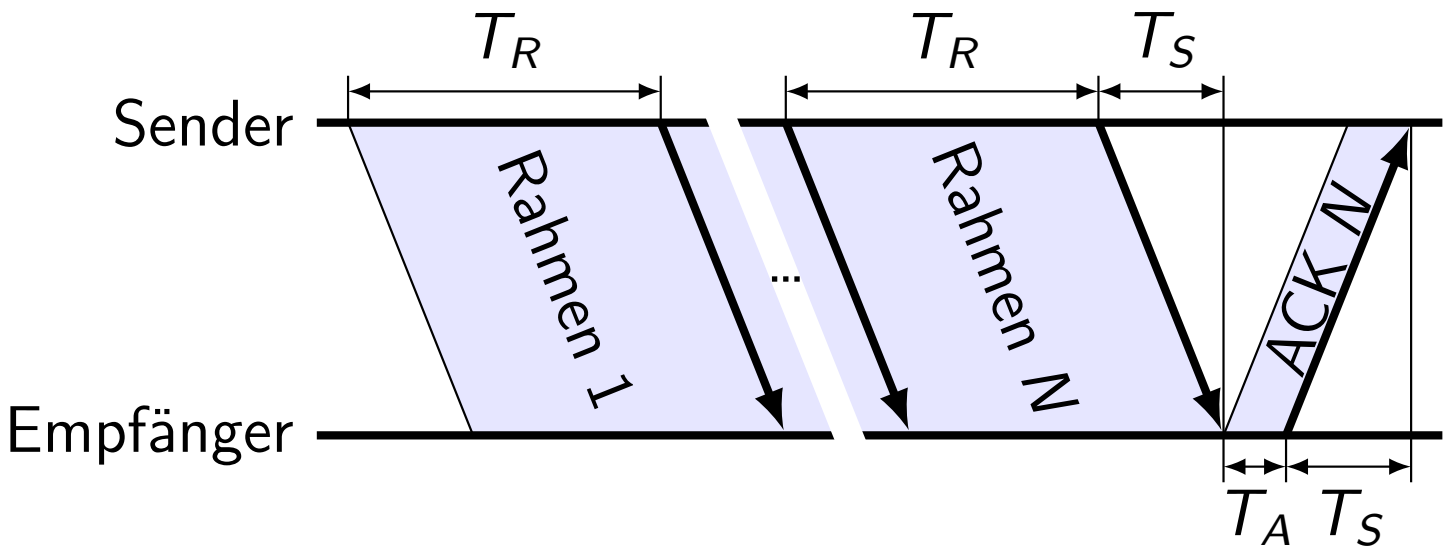
- Empfangsfenster wird nach Empfang von Rahmen kleiner
- Empfangsfenster vergrößert sich beim Senden von ACKs
- Ignoriert bereits erhaltende Rahmen, ACK wird wiederholt

Schiebefenster – Beispiel



Schiebefenster – Effizienz

Ohne Fehler und genau einem ACK für alle Frames:



Effizienz: $U = \frac{NT_R}{NT_R + 2T_S} = \frac{1}{1 + \frac{2T_S}{NT_R}}$

Schiebefenster – Aufgabe zu Effizienz

$$\text{Effizienz: } U = \frac{NT_R}{NT_R + 2T_S} = \frac{1}{1 + \frac{2T_S}{NT_R}}$$

Aufgabe

Zwei Rechner A und B sind mit einem Kabel verbunden, das eine Signallaufzeit von 5 µs hat. Rechner A sendet Rechner B Rahmen mit Länge 500 bit. Bestimme die Effizienz für das Schiebefensterverfahren mit $N = 16$, wenn

1. die Datenrate 10 $\text{Mbit}\,\text{s}^{-1}$ beträgt sowie
2. die Datenrate 1000 $\text{Mbit}\,\text{s}^{-1}$ beträgt.
3. Bestimme die minimale Fenstergröße für die zweite Datenrate, wenn eine Effizienz von 0.9 erreicht werden soll.

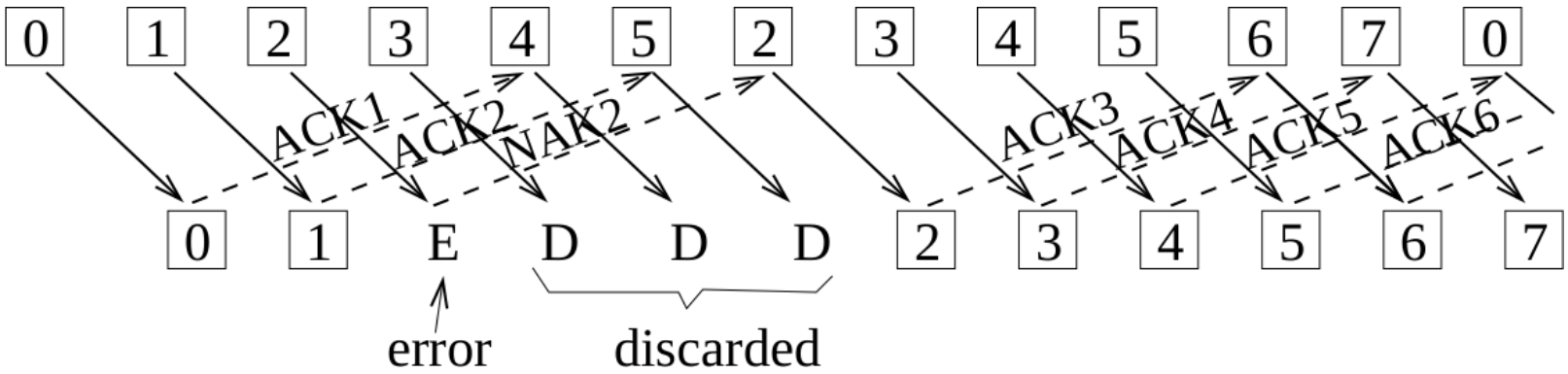
Schiebefenster – Go-back-N ARQ

Im Fehlerfall: Sender erhält NAK zurück oder ein Timeout

- Rahmen werden nur in Reihenfolge akzeptiert

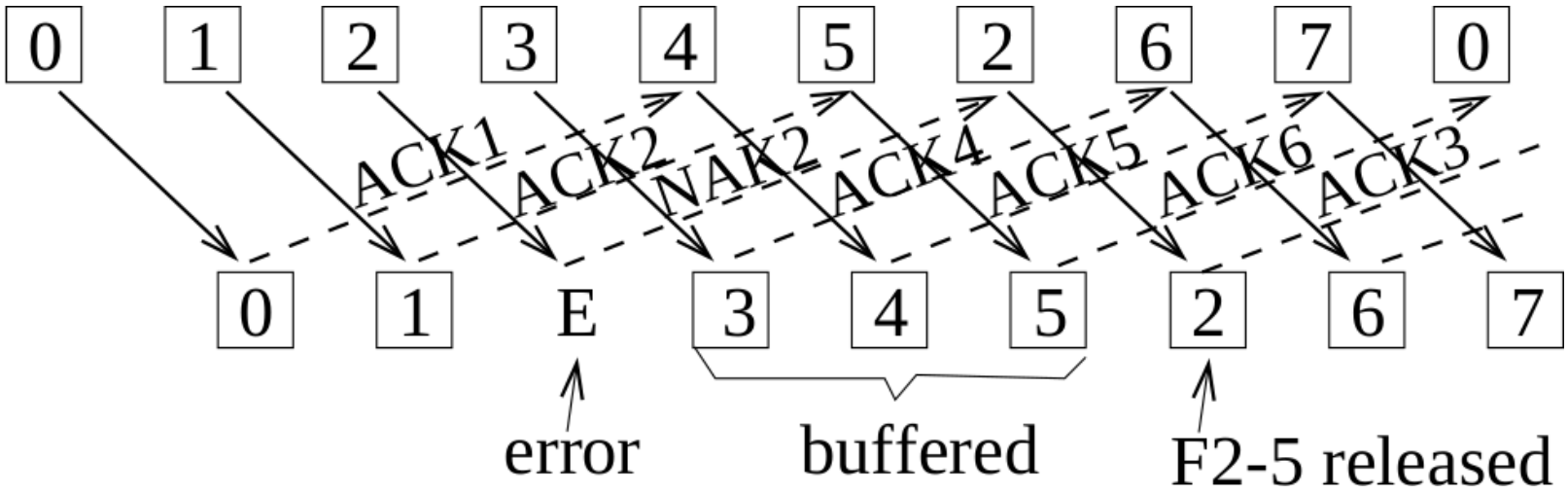
→ ACKs werden nur in Reihenfolge gesendet

- Ist ein Rahmen i korrupt, werden empfangsseitig alle nachfolgenden Rahmen gelöscht
- Sender sendet Rahmen i erneut und alle weiteren Rahmen



Schiebefenster – Selective-Repeat-ARQ

- Es werden nur die Rahmen erneut gesendet, für die ein NAK oder ein Timeout bekannt ist



- Üblicherweise ist das effizienter beim Übertragen
- Komplizierter in der Implementierung ($n + 1$-Bit-Sequenznummern sind für 2^n unterscheidbare Rahmen notwendig)
- Um Rahmen zu Puffern, braucht man mehr Speicher

Duplex-Verbindungen

- Jeder Teilnehmer kann senden oder empfangen

→ Jeder Teilnehmer muss auch ACKs vom Peer empfangen können

→ Art des Transfers (Anfrage oder Antwort) muss gekennzeichnet werden

- Beantwortet ein Empfänger einen Rahmen, so kann er selbst Daten in dieses Paket ablegen (*piggybacking*)

Flusskontrolle und Ethernet

Ethernet

Flusskontrolle ist auch bei Ethernet spezifiziert (Pause-Rahmen), spielt aber auf Sicherungsschicht in der Regel keine Rolle.

In der Internet-Protokoll-Suite ist es Teil der Transportschicht (TCP). Warum ist das keine schlechte Idee?

Flusskontrolle und Ethernet

Ethernet

Flusskontrolle ist auch bei Ethernet spezifiziert (Pause-Rahmen), spielt aber auf Sicherungsschicht in der Regel keine Rolle.

In der Internet-Protokoll-Suite ist es Teil der Transportschicht (TCP). Warum ist das keine schlechte Idee?

WLAN

WLAN hingegen bietet verbindungsloses Protokoll mit Bestätigung auf Sicherungsebene.

Flusskontrolle – Weitere Verfahren

Neben den hier vorgestellten Verfahren gibt es eine Reihe anderer:

- ASCII definiert XOFF und XON Steuercodes, wird bei seriellen Verbindung genutzt
- Dedizierte Leitungen

Outline

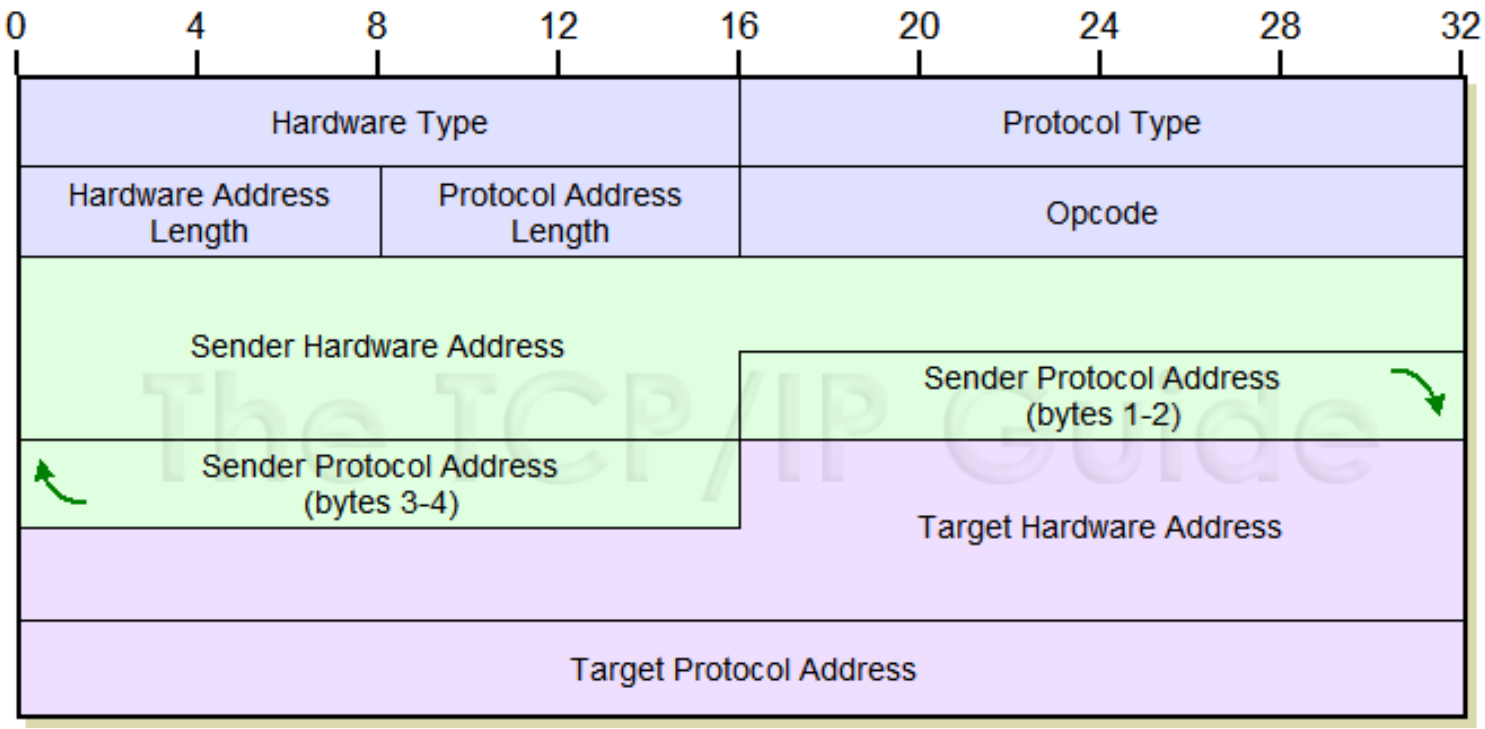
Adressen auflösen – Address Resolution Protocol

- Knoten ermitteln damit zu einer IPv4-Adresse (Vermittlungsschicht) physische Adresse der Sicherungsschicht
- Information wird lokal in einer Tabelle abgelegt
- Definiert in RFC 826
- Ist zwischen Sicherungsschicht und Vermittlungsschicht angesiedelt
- Anfrage ist eine Broadcast-Nachricht

Beispielanfrage

Wer hat IP Adresse x? Sende die Info zu IP-Adresse y mit MAC-Adresse z.

ARP-Format



Siehe http://www.tcpipguide.com/free/t_ARPMessageFormat.htm.

ARP-Anfrage

No.	Time	Source	Destination	Protocol	Length	Info
10	5.006735650	Avm_54:2a:ef	IntelCor_0e:b0:dc	ARP	56	Who has 192.168.178.60? Tell 192.168.178.1
11	5.006776458	IntelCor_0e:b0:dc	Avm_54:2a:ef	ARP	42	192.168.178.60 is at 0c:54:15:0e:b0:dc
29	32.216810051	IntelCor_0e:b0:dc	Avm_54:2a:ef	ARP	42	Who has 192.168.178.1? Tell 192.168.178.60
30	32.218093964	Avm_54:2a:ef	IntelCor_0e:b0:dc	ARP	56	192.168.178.1 is at 08:96:d7:54:2a:ef

```
▾ Frame 10: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface 0
  ▸ Interface id: 0 (wlan0)
    Encapsulation type: Ethernet (1)
    Arrival Time: May 17, 2019 09:42:37.891048268 CEST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1558078957.891048268 seconds
    [Time delta from previous captured frame: 1.356770925 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 5.006735650 seconds]
    Frame Number: 10
    Frame Length: 56 bytes (448 bits)
    Capture Length: 56 bytes (448 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:arp]
    [Coloring Rule Name: ARP]
    [Coloring Rule String: arp]
▾ Ethernet II, Src: Avm_54:2a:ef (08:96:d7:54:2a:ef), Dst: IntelCor_0e:b0:dc (0c:54:15:0e:b0:dc)
  ▸ Destination: IntelCor_0e:b0:dc (0c:54:15:0e:b0:dc)
  ▸ Source: Avm_54:2a:ef (08:96:d7:54:2a:ef)
    Type: ARP (0x0806)
    Trailer: 00000000000000000000000000000000
▾ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: Avm_54:2a:ef (08:96:d7:54:2a:ef)
    Sender IP address: 192.168.178.1
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 192.168.178.60
```

```
0000  0c 54 15 0e b0 dc 08 96  d7 54 2a ef 08 06 00 01   ·T······ ·T*·····
0010  08 00 06 04 00 01 08 96  d7 54 2a ef c0 a8 b2 01   ········ ·T*·····
0020  00 00 00 00 00 00 c0 a8  b2 3c 00 00 00 00 00 00   ········ ·<······
0030  00 00 00 00 00 00 00 00                           ········
```

ARP-Antwort

No.	Time	Source	Destination	Protocol	Length	Info
10	5.006735650	Avm_54:2a:ef	IntelCor_0e:b0:dc	ARP	56	Who has 192.168.178.60? Tell 192.168.178.1
11	5.006776458	IntelCor_0e:b0:dc	Avm_54:2a:ef	ARP	42	192.168.178.60 is at 0c:54:15:0e:b0:dc
29	32.216810051	IntelCor_0e:b0:dc	Avm_54:2a:ef	ARP	42	Who has 192.168.178.1? Tell 192.168.178.60
30	32.218093964	Avm_54:2a:ef	IntelCor_0e:b0:dc	ARP	56	192.168.178.1 is at 08:96:d7:54:2a:ef

```
▾ Frame 11: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
  ▸ Interface id: 0 (wlan0)
    Encapsulation type: Ethernet (1)
    Arrival Time: May 17, 2019 09:42:37.891089076 CEST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1558078957.891089076 seconds
    [Time delta from previous captured frame: 0.000040808 seconds]
    [Time delta from previous displayed frame: 0.000040808 seconds]
    [Time since reference or first frame: 5.006776458 seconds]
    Frame Number: 11
    Frame Length: 42 bytes (336 bits)
    Capture Length: 42 bytes (336 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:arp]
    [Coloring Rule Name: ARP]
    [Coloring Rule String: arp]
▾ Ethernet II, Src: IntelCor_0e:b0:dc (0c:54:15:0e:b0:dc), Dst: Avm_54:2a:ef (08:96:d7:54:2a:ef)
  ▸ Destination: Avm_54:2a:ef (08:96:d7:54:2a:ef)
  ▸ Source: IntelCor_0e:b0:dc (0c:54:15:0e:b0:dc)
    Type: ARP (0x0806)
▾ Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: IntelCor_0e:b0:dc (0c:54:15:0e:b0:dc)
    Sender IP address: 192.168.178.60
    Target MAC address: Avm_54:2a:ef (08:96:d7:54:2a:ef)
    Target IP address: 192.168.178.1
```

```
0000  08 96 d7 54 2a ef 0c 54  15 0e b0 dc 08 06 00 01   ···T*··T ········
0010  08 00 06 04 00 02 0c 54  15 0e b0 dc c0 a8 b2 3c   ·······T ·······<
0020  08 96 d7 54 2a ef c0 a8  b2 01                     ···T*··· ··
```

Reverse Address Resolution Protocol

- Ermittelt die IP-Adresse zu einer MAC-Adresse
- Definiert in RFC 903
- Wurde durch BOOTP und später DHCP ersetzt, die auf der Anwendungsschicht agieren