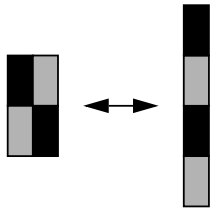
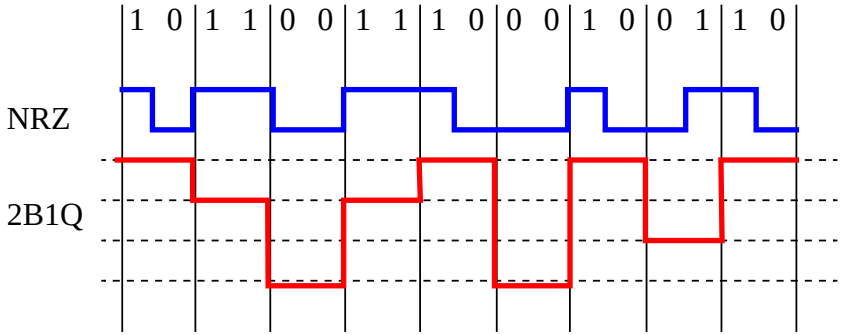
Gruppencodierung

- Eine Gruppe von g *Zeichen* wird auf eine Gruppe mit k *Codesymbolen* abgebildet
  - zweistufig: Gruppencodierung wird vor Leitungscodierung angewendet
  - ggf. überzählige Symbole, die keine Zeichen kodieren, können z.B. zur Leitungsüberwachung oder -Steuerung eingesetzt werden (z.B. als Begrenzungssymbole für Datenpakete)
  - auch magnetische Speichermedien und CDs verwenden Gruppencodes

- Beispiel: *2B1Q-Codierung*
  - 2 Binärstellen codieren
  - 1 quaternäres Zeichen übertragen





- Weitere gebräuchliche Gruppencodes, z.B. 4B5B
  - 4 Bitblöcke --> 5 Bitblöcke mit 80% Effizienz

    (32 verschiedene Symbole für 16 Zeichen --> einige der restlichen Symbole verwendet für die Übertragungssteuerung; nicht mehr als 3 konsekutive Nullen in den Symbolen --> Taktrückgewinnung; Verwendung z.B. bei FDDI sowie 100 Base TX und FX-Ethernet)

Quellencodierung

- Auf höherer Ebene (Darstellungsebene) angesiedelt
  - von Leitungscodierung (und Gruppencodierung) zu unterscheiden!

- Ziele:
  - kompakte Datendarstellung (Komprimierung)
  - Hinzufügen von Redundanz

  } z.T. antagonistisch

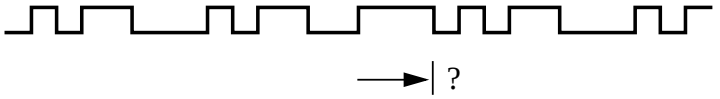
- Beispiele:
  - Huffman-Codierung
  - Lauflängencodierung
  - Lempel-Ziv-Kompression
  - diskrete Kosinustransformation (DCT)
  - Bildkompression mit JPEG
  - wavelet-Bildkompression
  - fraktale Bildkompression

Synchronisation

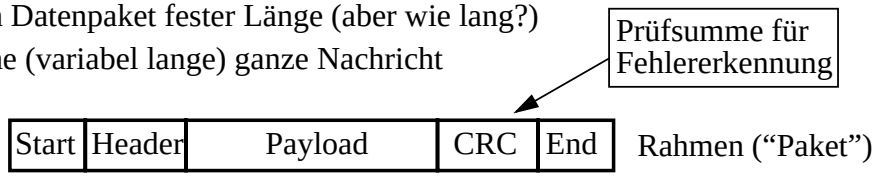
- Zeitliche Abstimmung von Sende- / Empfangsstationen zur geregelten Signalübertragung
  - "Gleichlauf" von Sender und Empfänger
  - nach einer Synchronisation sollen Sender und Empfänger die gleiche Vorstellung vom Zustand der Kommunikation haben
- *Taktsynchronisation*
  - notwendig zur korrekten Interpretation übertragener Signale (Zeitpunkt zum Abtasten des Signals)
  - hierzu verschiedene Techniken: Taktsignal über getrennte Leitung oder Regenerierung des Taktes aus dem Übertragungssignal (dann genügend viele Wertänderungen im Signalstrom notwendig!) und punktuelle Resynchronisation eines unabhängigen Taktgenerators
  - mehrmaliges Abtasten pro Bit, um Abweichungen zu kompensieren
  - Sender und Empfänger müssen sich über Taktrate verständigen

Rahmenbildung ("framing")

- Wo beginnt / endet eine Informationseinheit?



- Informationseinheit kann sein
  - ein einzelnes Zeichen (z.B. ASCII-Byte)
  - ein Datenpaket fester Länge (aber wie lang?)
  - eine (variabel lange) ganze Nachricht

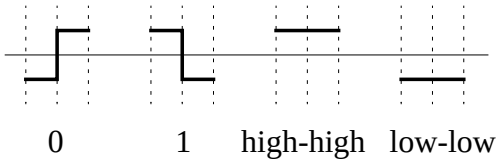


- Header enthält typischerweise die Länge des Pakets, Adressinformation und Steuerinformation (Flags etc.)
- Wozu Informationseinheit bzw. Rahmenbildung?
  - Bündelung für Bitfehler-Prüfung mit CRC
  - Einheit für Quittierung und Flusskontrolle ("bis Rahmen n alle Daten korrekt erhalten!" oder "bitte maximal 7 Rahmen auf einmal!)
  - selektives Wiederholen bei Übertragungsfehlern ("alle Daten ab Rahmen k wiederholen!")
  - Rahmengrenze als Aufsetzpunkt nach Störung

Erkennung von Rahmengrenzen

- Längenangabe im Header
  - Problem: Bei Übertragungsfehlern kann das Längenfeld verfälscht werden; eine Bitfehlererkennung ist ohne Blockgrenzen aber kaum möglich (ggf. in Kombination mit anderen Methoden sinnvoll)

- Verwendung illegaler Codezeichen



z.B. Manchestercodierung: die beiden rechten Zeichen kommen in den Daten nicht vor und können so Rahmengrenzen darstellen (da "selten": kein Synchronisationsverlust)

- Verwendung von speziellen Steuerzeichen
  - z.B. STX ("start of text") bzw. ETX ("end of text")
  - in den Nutzdaten ("payload") darf kein Steuerzeichen vorkommen
  - "character stuffing" als Lösung für "Datentransparenz":
    - Voranstellen eines Fluchtzeichens ("escape"); i.a. DLE
    - DLE ("data link escape") selbst im Datenteil wird dann verdoppelt
    - dadurch "transparenter Übertragungsmodus"
    - Variante: DLE STX schaltet Transparenzmodus ein; DLE ETX aus

- Verwendung eines speziellen Bitmusters
  - z.B. 01111110 und "bit stuffing" als Lösung für Datentransparenz (Erläuterung von "bit stuffing" später)

- Bei character stuffing oder bit stuffing verlängert sich allerdings die "Nutzlast"
  - insbesondere keine festen Blockgrössen möglich

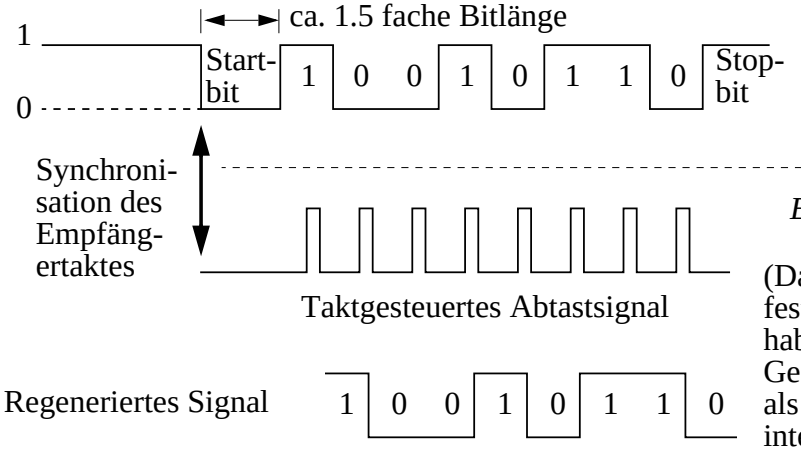
Asynchrone Datenübertragung

- Einzelne Zeichen werden unabhängig voneinander übertragen (jedes Byte "einzeln" in einem Rahmen)
  - beliebig lange Pausen zwischen den Byte-Rahmen ("asynchron")
  - i.a. bitserielle Übertragung; ggf. Paritätsbit am Zeichenende angefügt

- Rahmen ("frame") hat am Anfang bzw. Ende ein Start- bzw. Stopbit zur Synchronisation ("Start-Stop-Verfahren")
  - Synchronisationsoverhead ca. 30% bei einzelnen Bytes!

- Da Rahmen klein, sind die Gleichlaufbedingungen nicht strikt --> einfache, preiswerte Realisierungen
  - alte Technik (Telex-Netze: Rahmen mit ca. 10 Bits)

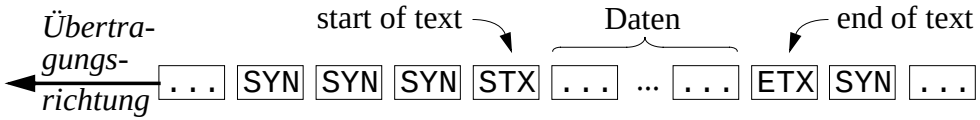
- Klassischerweise eingesetzt u.a. bei Datenübertragung von der Tastatur bzw. ASCII-Terminal zum Rechner
  - lange idle-Zeiten --> Empfänger muss sich bei Beginn des nächsten Zeichens synchronisieren (dazu dient das "Startbit")
  - während einer Zeichenübertragung laufen Sender / Empfänger synchron
  - typische Übertragungsraten: 9600 b/s oder 19200 b/s; z.B. über V.24



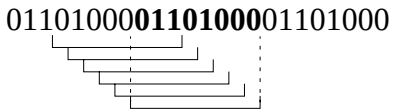
(Da Rahmen feste Länge haben --> keine Gefahr, Stopbit als Daten zu interpretieren)

Synchrone Datenübertragung

- Gleichlauf Sender / Empfänger über lange Zeit garantiert
  - Rückgewinnung des Taktes aus dem Signalwert (oder eigene Taktleitung)
- Damit auch grosse Datenblöcke “am Stück” übertragbar
  - Maximalgrösse von Blöcken notwendig, da Blöcke beim Empfänger zwischengespeichert werden und Fehlererkennung blockweise erfolgt
  - ein Block (auch oft als *Rahmen* oder “*frame*” bezeichnet) kann unabhängig vom letzten Block gesendet werden (d.h. “blockweise asynchron”!)
  - damit Synchronisation aufrecht erhalten werden kann, sollten auch zwischen Blöcken Signale gesendet werden
  - daher Anfang / Ende von Blöcken markieren (“framing”), z.B. so:



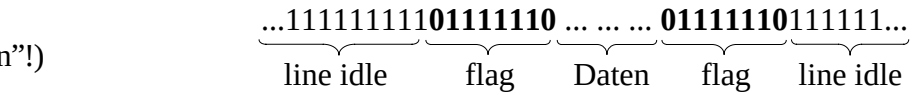
- Vor einem Datenblock z.B. zwei oder mehr SYN-Zeichen
  - ggf. auch zwischen zwei Blöcken ständig SYN-Zeichen senden
  - SYN = “synchronous idle”; Bitmuster **01101000**
  - erhält *Bitsynchronisation* über Blockgrenzen (ggf. wie bei asynchroner Übertragung anhand dieser Bitsignale neu synchronisieren)
  - ermöglicht (nach Bitsynchronisation) die *Zeichensynchronisation*
    - “hunt mode”: Jedes Fenster von 8 Bits auf SYN überprüfen:



    - damit die Byte-Grenzen gefunden (=Zeichensynchronisation)!
  - *Blocksynchronisation* mit STX- und ETX-Zeichen: begrenzen Blöcke aus Folgen ganzer Zeichen

Bitorientierte Übertragung

- Obiges Verfahren war *zeichenorientiert*
  - spezielle Kontrollzeichen für Blockbegrenzung
- Daneben existiert die *bitorientierte* Übertragung:

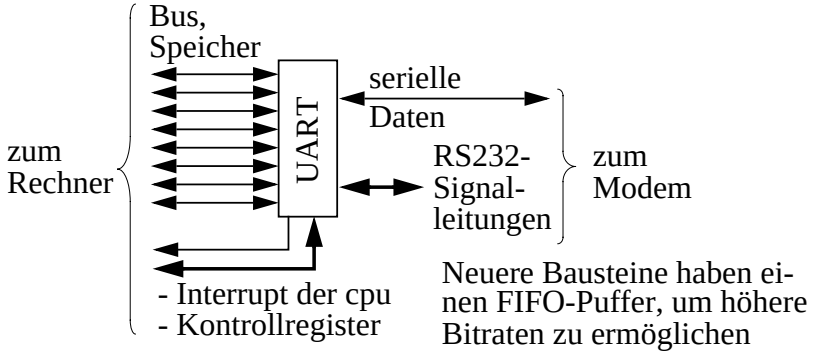


  - Daten werden durch ein flag 01111110 eingerahmt
  - Empfänger sucht (“bitweise”) das flag im Bitstrom
  - Sender fügt in die Nutzdaten nach 5 aufeinanderfolgenden Einsen eine Null ein (“bit stuffing”)
  - Empfänger macht Bitfolge 111110 zu 11111 (löscht die 0)
  - kann (effizient) durch Hardware erledigt werden
  - garantiert Transparenz
- Bitorientierte Übertragung i.a. effizienter als zeichenorientierte
  - beliebige (Binär-)Daten, nicht an Bytegrenzen gebunden, codeunabhängig
  - wegen bit stuffing ist allerdings keine feste Blockgrösse möglich!

UART-Baustein

- “Universal Asynchronous Receiver/Transmitter”
- Schnittstellenbaustein (z.B. für RS232) auf einem IC, der die meisten Funktionen serieller Kommunikation realisiert
- Wesentliche Funktion: Parallel/Seriell-Umwandlung einzelner Zeichen (mit Bit- und Zeichensynchronisation)

- Bsp.: 8250, 8251, 8450, 16450, 16550A, 16650, 16750, 16950...
(Intel, National Semiconductor...)



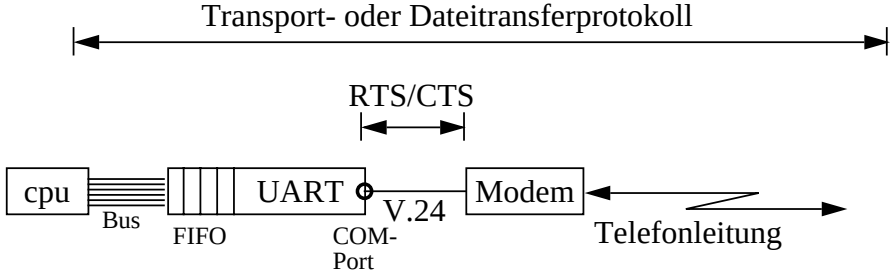
- verwendet zur Ansteuerung von Modems bzw. des “COM-Ports” bei PCs (Signalleitungen TxD, RxD, DSR, DTR, CTS, RTS...) etc.

- “universal” heisst programmierbar für verschiedene Funktionen, i.a. realisiert durch Setzen von Bits in Steuerregistern, z.B.:
  - synchron / asynchron
  - Übertragungsrate
  - Anzahl der Bits pro Zeichen (z.B. 5 oder 8)
  - parity Bit (ja / nein bzw. even / odd)
  - Stop-Bit (ja / nein bzw. Länge von 1, 1.5 oder 2 Bits)

- gemeldet werden (i.a. über ein Statusregister) auch Fehlerbedingungen wie z.B. parity error oder framing error (z.B. ungültiges Stop-Bit, was auf eine falsche Datenrate hindeutet)

“UART Overrun Errors”

- Mögliche Symptome bei der Modembenutzung:
  - Rechner meldet “CRC-Fehler”, oder
  - Übertragung ist extrem langsam, da durch übergeordnetes Protokoll CRC-Fehler durch Wiederholungen automatisch maskiert werden

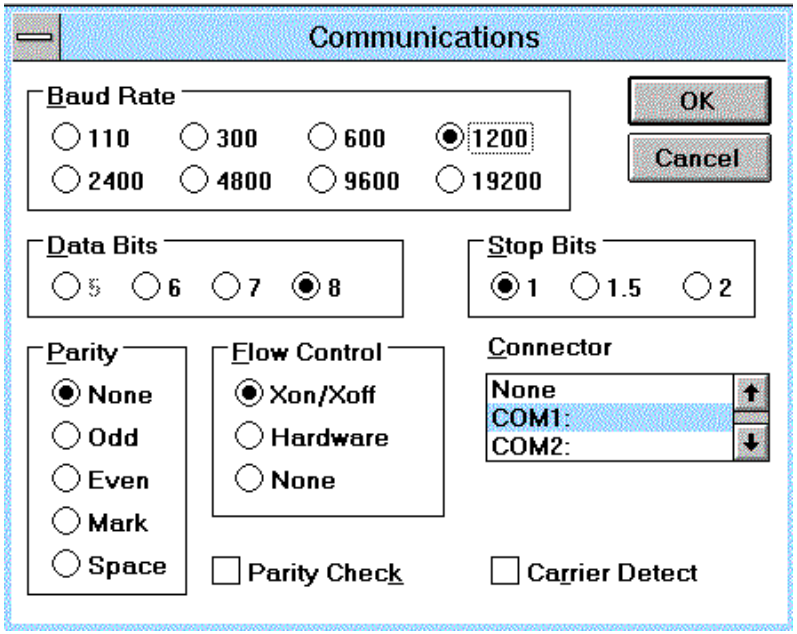


- Mögliche Ursache:
  - Prozessor reagiert zu spät auf UART-Interrupt (z.B. bei Überlastung oder Deaktivierung des Interrupts durch ein Anwendungsprogramm)
  - nächstes ankommendes Zeichen überschreibt alte, noch nicht aus dem Puffer des UART abgeholte Zeichen

- Beispiel: UART-Chip 16550 mit 16-Byte-FIFO-Puffer
  - kann über Software (Treiber des Betriebssystems) konfiguriert werden
  - z.B. Interrupt erzeugen, wenn FIFO mit 14 Byte gefüllt ist
  - bei V.90-Modem und Kompressionsfaktor 4 --> ca. 200 kb/s --> restliche 2 Bytes sind schon nach 80 µs verbraucht!
  - UART 16650: 32-Byte-Puffer, 16750: 64 Byte, 16950: 128 Byte...

- Lösung: Interrupt früher generieren, FIFO vergrössern, Bus verbreitern, cpu beschleunigen, Flusssteuerung verbessern...

COM - Der “Communication Port”

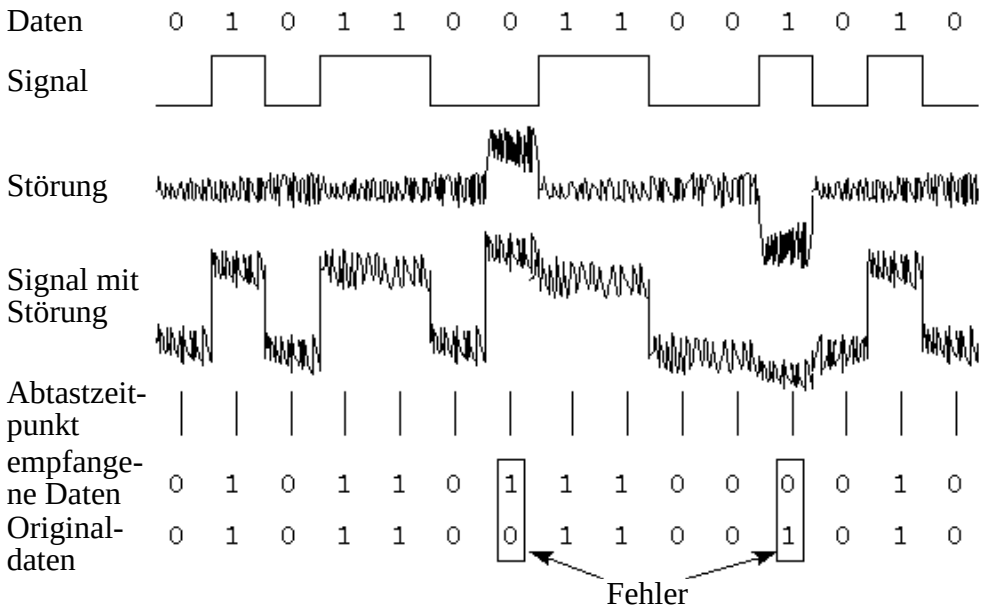


This is the communications configuration panel from Windows Terminal. It raises a number of interesting questions. Why would someone use 5 bits per character? Because a 5-bit code was used by very early Teletype equipment that was already obsolete in the 1950’s. What is the right number of Stop Bits? Well, if you have a Teletype Model 33, the right answer is 2. If you have a Teletype Model 35, the right answer is 1.5. However, no device built in the last 20 years has needed more than 1 stop bit. What is Xon/Xoff Flow Control? XON and XOFF are byte values. The Teletype had a device to read punched paper tape. The XON character turned the tape reader on, and the XOFF character turned it off. Long after the last paper tape was burned, computers have maintained the tradition that XOFF can optionally mean “stop sending data,” in which case XON means “begin sending again.” What is parity? Before modems did error correction, parity provided a simple mechanism to detect characters corrupted by phone line noise. Today it is unnecessary and is typically disabled.

So in current use, the correct setting for the COM port is always 8-bit characters, no parity, 1 stop bit, hardware pacing and some speed faster that the native transmission speed of the modem. The panel to configure the COM port is left around because everyone is scared to get rid of it.

Übertragungsfehler

- Bitfehler durch verrauschten Kanal



- Bitfehler durch fehlerhafte Synchronisation

Übertragungsfehler (2)

- Ursachen für Fehler bei der Datenübertragung
  - thermische Elektronenbewegungen in Halbleitern oder Leitungen
  - elektromagnetische Einstrahlungen (Motoren, Zündanlagen, Blitze)
  - radioaktive Einstrahlungen (z.B. bei Zwischenstationen)
  - schadhafte Geräte (z.B. Versorgungsspannung), Leitungen (Kontakte)
  - zu hohe Leitungsdämpfung; sporadisch fehlerhafte Bitsynchronisation
  - Zugriffskollisionen in Ethernet-LANs
  - Pufferüberlauf beim Empfänger oder einer Zwischenstation
  - unvollständige Nachrichten bei absichtlichem Sendeabbruch
- Oft sind dann mehrere Bits nacheinander falsch
  - Fehlerbündel ("bursts") --> Bitfehler also oft nicht statistisch unabhängig voneinander!
  - einfache Paritätsprüfung versagt dann!
- Fehlerkorrektur bzw. -erkennung ist notwendig
  - selbst bei einer Fehlerrate von "nur" 10^{-7} würde bei 10 Mb/s (z.B. Ethernet) jede Sekunde ein Bitfehler auftreten (nicht akzeptabel!)

- Fehlererkennung
  - einfach: Paritätsbit (z.B. "Byteparität"; geht mit XOR in Hardware effizient)
  - es gibt leistungsfähigere Verfahren (z.B. CRC-Codes)
  - typischerweise Datenblock erneut senden bei erkanntem Fehler (wenn Datenblock "beschädigt" wurde oder vermutlich verloren ging)
- Fehlerkorrigierende Codes (z.B. bei Satelliten)
  - "Forward Error Control" (FEC): Empfänger korrigiert verfälschte Bits
  - viel Redundanz notwendig (Hammingdistanz); Probleme bei Fehlerbursts

Datensicherung und Quittungen

- Wesentliche Aufgaben der Ebene 2 (Internet: Ebene 4) sind *sichere Datenübertragung* und die *Flusssteuerung*
  - *sichere Datenübertragung*: Datenverfälschung und Datenverlust der Ebene 1 ausgleichen
  - *Flusssteuerung*: Sendegeschwindigkeit an Geschwindigkeit des Empfängers anpassen
- Daten*verfälschung* i.a. auf Daten*verlust* zurückgeführt
  - Empfänger "vernichtet" Datenblöcke mit erkannten Bitfehlern
- Hilfsmittel für beide Aufgaben sind i.w. *Bestätigungsnachrichten* und *Zeitüberwachungen* (timeouts)
  - positive Bestätigungen ("ACK")
  - negative Bestätigungen ("NAK")

  } also zumindest Halbduplex nötig
  - timeout tritt typischerweise bei Ausbleiben eines ACK ein
- Daher beide Aufgaben oft in *einem* Protokoll "verwoben"

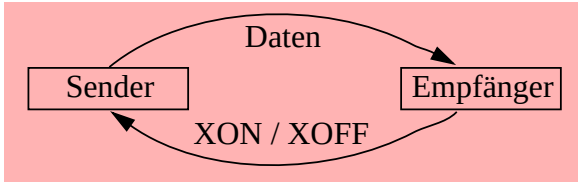
- Bei Duplexbetrieb kann ein ACK ggf. zusammen mit einer Nachricht in Gegenrichtung verschickt werden
  - "Huckepack"-Prinzip ("piggybacking")
  - ACKs oft nur einige Bits lang (z.B. Sequenznummer des letzten erhalten Blocks); eigene Nachricht dafür wäre relativ aufwendig
  - ggf. vor dem Versenden eines ACK kurze Zeit warten, ob dieses nicht huckepack mit der nächsten Nachricht übermittelt werden kann
- Erneutes Versenden eines Blocks bei Ausbleiben eines ACK (oder Erhalt eines NAK) wird als "Automatic Repeat Request" (ARQ) bezeichnet

Flusssteuerung

- Aufgabe: Empfänger vor einem zu grossen Zufluss von Paketen eines Senders schützen

- Angesiedelt typischerweise auf der Sicherungsschicht
  - ggf. aber auch (zusätzlich) auf höheren Schichten

- Einfachste Methode: “Halt”- und “Weiter”-Meldung vom Empfänger zum Sender



  - Beispiel: XON/XOFF-Protokoll:
    XON ist ASCII-Zeichen 017 (oktal): DC1 (“Device Control 1”);
    XOFF ist ASCII-Zeichen 019 (oktal): DC3 (“Device Control 3”)
    (auf ASCII-Tastaturen: XON/XOFF mit “control Q” bzw. “control S”)
  - muss rechtzeitig eintreffen (beachte Bandbreite-Delay-Produkt!)

- Implizite Flusssteuerung, z.B.:
  - Zurückhalten der Quittung (ACK)
  - Empfänger räumt dem Sender einen mehrere Transfereinheiten umfassenden Sendekredit ein; bei Erschöpfung (ohne neue Kreditgewährung) stoppt der Sender
  - Beachte: Kreditmethode erfordert aufwendigere Fehlerkontrolle (z.B. bzgl. Verlust einer Kreditgewährung)

- Ratenbasierte Flusssteuerung
  - Empfänger erlaubt dem Sender, eine bestimmte Menge an Daten innerhalb eines Zeitintervalls zu senden
  - bei vorzeitiger Ausschöpfung der Datenmenge muss der Sender bis zum nächsten Zeitintervall warten

Netzüberlastung und Laststeuerung

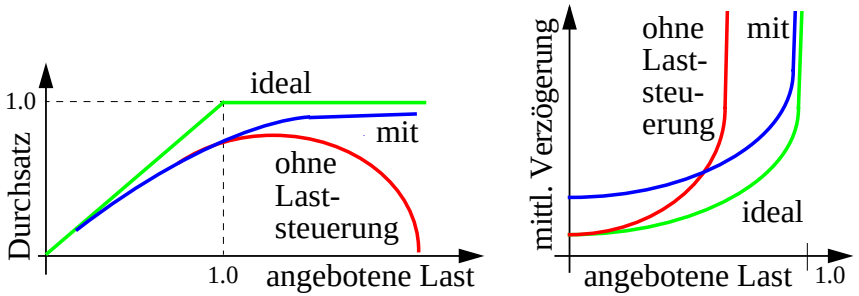
network congestion — congestion control

- Bei Netzüberlastung: grosse Verzögerungen, sinkender Durchsatz; Datenverlust durch Pufferüberläufe

- Gründe für (lokale oder globale) Netzüberlastung:
  - zu starker Zufluss von Datenpaketen
  - Umkonfiguration aufgrund von Teilausfällen des Netzes
  - Zunahme transienter Störungen in Netzteilen

- Zweck von Laststeuerung (bzw. “Staukontrolle”):
  - “vernünftiges” Netzverhalten bei Hochlast-/ Überlastbetrieb
  - Erhaltung bzw. Optimierung der Durchsatzleistung (Last so begrenzen, dass keine Reduktion des Durchsatzes erfolgt)
  - Verhinderung von Pufferüberläufen aufgrund von Überlastung

- Ohne Laststeuerung kann es zu einem Kollaps kommen
  - weggeworfene Pakete (Pufferüberlauf!) werden wiederholt und vergrössern noch die Netzlast



  - Laststeuerung hat einen Overhead (spürbar im Niedriglastbereich); darf aber nicht zur Verschlimmerung im Hochlastbereich beitragen!