R = b/s Bitrate
L = Paketgröße
l = Länge (Meter)
v = (Ausbreitungs-)Geschwindigkeit
D = l/v Ausbreitungsverzögerung
a = (RD)/L Kanalpuffergröße in Paketen
W = Fenstergröße

Kilo = 10^3
Mega = 10^6
Giga = 10^9

b = Bit
B = Byte = 8 Bits

RD > 1: (a > 1:)
t = D: erstes Bit erreicht Empfänger, RD Bits (a Pakete) gesendet

RD < 1: (a < 1:)
t = D: Anfang des ersten Bits erreicht Empfänger,
und Anteil RD des Bits (Anteil a des Pakets) gesendet
t = 1/R: (L/R:) Bit erstes (Paket) komplett gesendet
t = 1/R + D: (L/R + D:) erstes Bit (Paket) komplett empfangen

TCP-Leistungsanalyse
Verbindungsaufbau (Drei Wege Handshake): 2RTT
a) C -> S : SYN (seqnum: client_isn)
b) S -> C : SYN + ACK (seqnum: server_isn, acknum: client_isn+1)
c) C -> S : ACK + Request (seqnum: client_isn+1, acknum: server_isn+1)

Verbindungsabbau:
a) C -> S : FIN (seqnum: client_sqn)
b) S -> C : ACK (seqnum: server_sqn, acknum: client_sqn+1)
c) S -> C : FIN (seqnum: server_sqn)
d) C -> S : ACK (seqnum: client_sqn+1, acknum: server_sqn+1)
e) C wartet noch 2 Segmentlebensdauern auf mögliche alte Segmente („time wait“), S ist nach dem Erhalt des letzten ACKs fertig

Festes Fenster:
bei WL/R ≥ L/R + RTT -> keine Wartezeiten: d = 2RTT + O/R

bei WL/R < L/R + RTT -> mit Wartezeiten:

$$d = 2RTT + \frac{O}{R} + (K-1)\left(\frac{L}{R} + RTT - \frac{WL}{R}\right)$$

K = Anzahl benötigter Fenster = $\lceil \frac{O}{WL} \rceil$; dabei ist O die Objektgröße

Dynamisches Fenster:
d = 2RTT + O/R + d_{SSW}

warten, wenn $\frac{WL}{R} \leqslant \frac{L}{R} + RTT \Rightarrow$ Wartezeit nach K. Fenster: $\max\{\frac{L}{R} + RTT - 2^{K-1} \cdot \frac{L}{R}; 0\}$
Wartezeiten bis Fenster Q: $\max\{k : T \cdot \frac{L}{R} + RTT - 2^{k-1} \cdot \frac{L}{R}; 0\}$
Anzahl Fenster K um O zu übertragen: $K = \min\{k : \sum_{i=1}^{k} x^{i-1} \cdot L >= 0\}$
Übung 5, 9)

$$Q = \left\lfloor \log_2\left(1 + \frac{RTT}{\frac{L}{R}}\right) \right\rfloor + 1$$ bis zu welchem Fenster Wartezeiten

$$K = \left\lceil \log_2\left(\frac{O}{L} + 1\right) \right\rceil$$ Anzahl Fenster um O zu übertragen

$$P = \min\{ \underbrace{Q}_{\text{Nicht mehr warten, O groß genug}} ; \underbrace{K-1}_{\text{Warten, wenn O zu klein}} \}$$ Wartezeit bei endlich großem Objekt O
(P: Anzahl SSW)

$$d = 2RTT + \frac{O}{R} + P \cdot \left(RTT + \frac{L}{R}\right) - \left(2^P - 1\right) \cdot \frac{L}{R}$$

mehrere Links T:

$$d = 2 \cdot RTT_T + \frac{O}{R} + (T-1) \cdot \frac{L}{R} + P_T \cdot \left(RTT_T + \frac{TL}{R}\right) - \left(2^{P_T} - 1\right) \cdot \frac{L}{R}$$

$$Q_T = \left\lfloor \log_2\left(T + \frac{RTT}{\frac{L}{R}}\right) \right\rfloor + 1$$

K = wie normal

um Faktor 2 erhöhen, also Fenstergröße W^3:

$$d = 2 \cdot RTT + \frac{O}{R} + P \cdot \left(\frac{L}{R} + RTT\right) - \frac{P-1}{2} \cdot \frac{L}{R}$$

$$Q = \left\lfloor \log_3\left(1 + \frac{RTT}{\frac{L}{R}}\right) \right\rfloor + 1 \quad ; \quad K = \left\lceil \log_3\left(\frac{2O}{L} + 1\right) \right\rceil$$

d_{trans} = L/R Übertragungsverzögerung
d_{prop} = D Ausbreitungsverzögerung
d_{proc} = Verarbeitungsverzögerung
d_{queue} = Warteschlangenverzögerung
Paketverzögerung = $d_{trans} + d_{prop} + d_{proc} + d_{queue}$
Paketverzögerung mit mehreren Links:

$$\left(\sum_{i=1}^{E} d_{trans_i} + d_{prop_i} + d_{proc_i} + d_{queue_i}\right) + d_{proc}$$

Übertragung in Paketen über mehrere Links:

$$d_{trans} = n_{links} \cdot \frac{L}{R} + (n_{pakete} - 1) \cdot \frac{L}{R}$$

(wenn leitungsvermittelt: $n_{links} = 1$)

mittlere Warteschlangenverzögerung:

$$d = \frac{d_{ges}}{N}$$

$$d_{ges} = \sum_{i-1}^{N} (i-1) \cdot \frac{L}{R} = \frac{N \cdot (N-1)}{2} \cdot \frac{L}{R}$$

Verkehrsintensität:
λ: durchschnittliche Paketankunftsrate (Pakete pro Sekunde)

$$\rho = \frac{L \cdot \lambda}{R} \begin{cases} d_{queue} \text{ fällt} & \text{falls } \rho < 1 \\ d_{queue} \text{ steigt} & \text{falls } \rho = 1 \\ d_{queue} \to \infty, \text{ da mehr Pakete ankommen als abgearbeitet werden} & \text{falls } \rho > 1 \end{cases}$$

Wahrscheinlichkeit, dass n Nutzer gleichzeitig senden:
$P_n = \binom{max}{n} \cdot (P_{user})^n \cdot (1 - P_{user})^{max-n}$ $\quad \binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$

Wahrscheinlichkeit, dass mehr als n Nutzer gleichzeitig senden:
$\sum_{n}^{max} P_i = 1 - \sum_{i=0}^{n} P_i$

Bitfehlerwahrscheinlichkeiten:
1. mindestens 1 Bitfehler im Segment: $1-(1-p)^L$
2. 2 Bitfehler im Segment: $(L-1)L / 2\ P_{bestimmtes\ Paar\ fehlerhaft}$
3. mittlere Anzahl Pakete bis 2 Bitfehler im gleichen Segment: $1/P_{2\_Fehler}$

Verzögerung HTTP mit M Objekten und 1 Basisseite (alles O Bits groß):
nicht-persistent: (M+1)(2 RTT + O/R + SSW)
persistent mit Pipelining (Q und K konstant): 2 RTT + RTT + (M+1) O/R + gem. SSW
gem. SSW = SSW + RTT - max{L/R + RTT - (2^{K-1} L/R); 0}
nicht-persistent mit X parallelen Verbindungen (M/X ganze Zahl):
(M/X + 1) 2 RTT + (M+1) O/R + SSW_{normal} +$SSW_{parallel}$ (in $SSW_{parallel}$ R durch R/X ersetzen)

Flusskontrolle (Steuerung der Senderate, ohne Empfänger zu überlasten):
a) effective window > 0 -> senden
b) LastByteWritten - LastByteAcked ≤ MaxSendBuffer -> Anwendung schreibt
c) Freier Speicher bei Empfänger:
advertised window = maxRcvBuffer - ((NextByteExpected-1)-LastByteRead)
d) Falls mehr als 16 Bit für Fenstergröße benötigt werden: Scaling Factor F ≤ 14,
dann gilt window = advertised window * 2^F

Überlastkontrolle (Steuerung der Senderate, um das Netz vor Überlast zu schützen):
a) max window = min{congestion window;advertised window}
b) effective window = max window - (LastByteSent - LastByteAcked)
c) Bitrate R ≈ congestion window/RTT
d) Slow-Start:
i. anfangs congestion window = MSS, nach jedem ACK congestion window += MSS
(also exponentiell), bis 3 doppelte ACKs eintreffen oder Threshold (anfangs bei ∞)
ii. Multiplicative Decrease (3 doppelte ACKs):
Threshold = congestion window/2; congestion window /= 2
iii. Additive Increase: Mit jedem ACK: congestion window += MSS^2 / congestion window
iv. nach Timeout: Threshold = congestion window / 2 ; congestion window = MSS
(dann wird Slow-Start bis zum Threshold und danach AIMD durchgeführt)

Stop-And-Wait:
(Leitungs-)Durchsatz: L / N*(L/R +2)
Normierter Durchsatz: S = 1 / N*(1+2a)
Mittlere Anzahl der Sendeversuche: N = 1 / (1-p)
keine Fehler -> N = 1

Schiebefensterprotokolle ohne Fehler:
W ≥ 1+2a -> Fenster groß genug: S = 1
W < 1+2a -> muss auf ACK warten: S = W / (1+2a)

Selective-Repeat (mit Fehlern):
W ≥ 1+2a -> S = 1-p
W < 1+2a -> S = W*(1-p) / (1+2a)

Go-Back-N (mit Fehlern):
Mittlere Anzahl der Sendeversuche: (1-p+K*p)/(1-p)
W ≥ 1+2a -> S = (1-p) / (1+2ap) ; K = 1+2a
W < 1+2a -> S = W(1-p) / (1-p+W*p)*(1+2a) ; K=W

Sequenznummernraum mit m= 2n Werten:
1. Empfangsfenstergröße = 1: W <m hinreichend
2. Sendefenstergröße = Empfangsfenstergröße > 1:
W < (m+1)/2

(Slotted-)ALOHA (N = $\#_{Stationen}$):
Wahrscheinlichkeit für Senden ohne Kollision: p $(1-p)^{2N-2}$
Normalisierter Durchsatz:
S = N p $(1-p)^{2(N-1)}$; S_{Amax}=37%; S_{SAmax}= 18%
Sendeversuche pro Slot:
G = N*p -> p = G/N
i. ALOHA: G = 0,5
ii. Slotted-ALOHA: G = 1

CSMA:
- verwendet ACKs
- notwendige Voraussetzung: d_{prop} < L/R
- bei fehlendem ACK nach Timeout -> Backoff -> Resend
1-persistent: Bei belegtem Medium warten bis frei, dann sofort senden
nicht-persistent: Bei belegtem Medium Backoff
p-persistent: Wenn Medium wieder frei, Senden mit Wahrscheinlichkeit p oder noch einen Slot abwarten (1-p)

CSMA/CD (N = $\#_{Knoten}$):
- mit allen CSMA-Varianten kombinierbar
- keine ACKs
- verwendet „listen while talking“

Normierter Durchsatz: S = 1 / (1 + 4,4a)
daher bei a < 1,04 besser als SlottedA.
Minimale Rahmengröße: L > 2RD
Erfolg: $P_{Erfolg} = N*p*(1-p)^{N-1}$
$(P_{Erfolg})^{max} = 1/e$ -> alle e Slots erfolgreich

Ethernet:
- 1-persistentes CSMA/CD
- Hub: eine Kollisionsdomäne, Signal wird auf alle Ports weitergeleitet
- Bridge: Aufteilung in versch. Kollisions-domänen, enthält „Switching Fabric“
- VLAN entweder port- oder tag-basiert

Verteilte Hash-Tabellen
- jeder Peer hat Bezeichner p∈[0;2m) bei m Bits
- jedes Datenelement hat Schlüssel k∈[0;2m)
- Speicherung und Auslesen von k auf Peer p
  succ(k) = (k+a) mod 2^m mit minimalem a

Effizienzsteigerung durch Fingertabellen auf jedem Peer p:
m Einträge: $FT_p[i] = (p+2^{i-1}) \bmod 2^m$

Lookup von k beginnend auf beliebigem Peer p:

$$\text{lookup}(k,p) = \begin{cases} p & ,p = k \\ FT_p[1] & ,p < k \leqslant FT_p[1] \\ \text{lookup}(k, FT_p[i]) & ,FT_p[i] \leqslant k < FT_p[i+1] \text{ mit } 1 < i < m \\ \text{lookup}(k, FT_p[m]) & ,FT_p[m] \leqslant k \end{cases}$$

Shannons Theorem
1. Maximale Datenrate R bei Bandbreite B: $R = B \cdot \log_2(1 + \frac{S}{N})$
2. $x\ dB = 10 \log_{10} \frac{S}{N} \Rightarrow \frac{S}{N} = 10^{\frac{x}{10}}$

signal to noise ratio

Cyclic Redundancy Check
- Nutzdaten D mit d Bits
- Prüfdaten R mit r Bits
- Generatorpolynom G mit r+1 Bits

z.B. x^4+x^2+x+1 = 10111
1. R ist Rest bei $(D*2^r)/G$ (Länge r)
   (Div: schriftliches bitweises xor)
2. (D,R) ist folglich $(D*2^r) \oplus R$
3. Korrekt falls (D,R)/G = 0

Routing
Dijkstra: Distanz, last hop
Link-State: (Ziel, Distanz, next hop) BestL beginnt leer
Fingertabelle: Bezeichner (nur die Zahl)
Distanz-Vektor: | Ziel | Distanz | next hop |
- Initial: direkte Nachbarn; davor nur self bekannt
- Poisoned reverse: Sich selbst dem Nexthop auf diesem Pfad nicht anbieten. -> Kosten = ∞

Sendezeiten
h Bits Header und d sec. Verbindungsaufbau:
t= d + (N+E-1)*(L+h)/R
Verbindungslos und 2h Header:
t= d + (N+E-1)*(L+2h)/R (d ist vermutl. falsch)
Verbindungslos, ohne Segmentierung, 2h Header:
t = (L*N+2h) * E/R
leitungsvermittelt, ohne Segmentierung, h Header:
t = d + (N*L+h)/R

TCP-Server
```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
public class TCPServer {
    private static int port = 2021;
    public static void main(String args[]) throws IOException {
        ServerSocket welcomeSocket = new ServerSocket(port);
        // Diese Schlaufe ermöglicht neue Verbindungen nach Ende der ersten.
        while (true) {
            Socket socket = welcomeSocket.accept();
            // Schlaufe ermöglicht mehrere Zeilen (z.B. mit flush()) zu schicken
            BufferedReader br = new BufferedReader(new InputStreamReader(new
DataInputStream(socket.getInputStream())));
            PrintWriter pw = new PrintWriter(new OutputStreamWriter(new
DataOutputStream(socket.getOutputStream())));
            while ( condition_for_looping ) {
                // Arbeit mit den Daten
            }
            pw.close();
            br.close();
            socket.close();
}   }   }
```

TCP-Client
```
public class TCPClient {
    private static String servername = "Hier steht der Servername";
    private static int port = 2021;
    public static void main(String args[]) throws IOException {
        Socket socket = new Socket(servername, port);
        PrintWriter pw = new PrintWriter(new OutputStreamWriter(new
DataOutputStream(socket.getOutputStream())));
        BufferedReader br = new BufferedReader(new InputStreamReader(new
DataInputStream(socket.getInputStream())));
        // Arbeit mit den Daten
        pw.close();
        br.close();
        socket.close();
}   }
```

UDP-Server
```
import java.net.DatagramPacket; import java.net.DatagramSocket;
public class UDPServer {
    static int port = 2021;
    public static void main(String args[]) {
        DatagramSocket serverSocket = new DatagramSocket(port);
        // Schleife falls mehrere Zeilen geschickt werden
        while ( condition_for_looping ) {
            byte[] receiveData = new byte[???];
            DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
            serverSocket.receive(receivePacket);
            // Arbeit mit den Daten
        }
        serverSocket.close();
    }

    public static int byteArrayToInt(byte[] b) {
    // Hier stehen die Konvertierungsprozeduren fuer alle möglichen Typen
}   }
```

UDP-Client
```
import java.net.*;
public class UDPClient {
    private static String servername = "Hier steht der Servername";
    private static final int port = 2021;
    public static void main(String[] args) {
        DatagramSocket clientSocket = new DatagramSocket();
        InetAddress ipA = InetAddress.getByName(servername);
        byte sendData[] = new byte[???];
        // Arbeit mit den Daten
        DatagramPacket packet = new DatagramPacket(sendData,
sendData.length, ipA, port);
        clientSocket.send(packet);
        clientSocket.close();
}   }
bw.write('hi srvr!\r\n');
bw.flush();
---
String user = br.readLine();
pw.print("331 User okay, need password.\r\n");
pw.flush();
String un = user.substring(user.indexOf(" ") + 1); // "USR: Sebi"
if (un.startsWith("B...")) do sth; // ...itches
```

Schichtenarchitektur im Internet:
- Anwendungsschicht (HTTP, FTP, SMTP) Kommunikation der Anwendungsprozesse mit anwendungss-pezifischen Informationen; hat Sitzungs- und Darstellungsschicht aus OSI-Referenzmodell absorbiert
- Transportschicht (TCP, UDP) zuverlässiger Ende-zu-Ende Transport von Segmenten zw. Anwendungen
- Netzwerkschicht Datagramme zwischen Hosts über Router (IP), Verbindungsaufbau, Weiterleitung, We gewahl (Routing), Betriebsmittelverwaltung
- Sicherungsschicht (IPv4, IPv6) Medienzugriff und gesicherte Übertragung von Rahmen (Frames) zw. benachbarten Geräten: Rahmensynchronisation, Fehler- und Flusskontrolle
- Bitübertragungsschicht (Ethernet) mechanische, elektrische und prozedurale Eigenschaften zur Über tragung von Bits: Zeitsynchronisation, Kodierung, Modulation

Cross-Layer Optimierung: Trennung in Schichten nicht eingehalten, z.B. zur Steigerung der Effizienz Mechanismen der Bitübertragungs-und der Sicherungsschicht gekoppelt
Kommunikationsarten: Unicast, Multicast, Broadcast, Anycast **Übertragungsarten:** simplex, halbduplex, (voll-)duplex **Multiplexverfahren:** Frequenzmultiplex, Zeitmultiplex
Vermittlungsarten: Leitungsvermittlung: zwischen Sender und Empfänger wird mittels Signalisierung ein Kanal zur Übertragung aufgebaut (z.B. durch Zeit- oder Frequenzmultiplex) die zur Verfügung stehende Bitrate muss fest auf die Kanäle aufgeteilt werden. bei schwankenden Datenaufkommen mit vielen Pausen ineffizient Paketvermittlung: Sender schickt Daten in Paketen, die einzeln zum Empfänger gelangen die Bitrate wird effizienter aufgeteilt kurzfristiges höheres Datenaufkommen kann über Puffer abgefangen werden dies kann zu Verzögerungen und Pufferüberläufen führen
Netzwerktopologien: Bus (linear), Ring, Stern, Baum, 2D-Torus, vollständig vermascht
Dienstgüte (QoS): quantitative Aspekte v. RK: Latenz, Jitter, Durchsatz, Verlust- & Fehlerrate, Verfgbarkeit
Transportprotokolle: TCP: verbindungsorientiert (abstrakte Sicht des Versendens eines Bytestroms), zuverlässig, feste Bitrate, z.B. Dateitransfer, Online-Banking
UDP: verbindungslos (Versenden einzelner Datagramme), unzuverlässig, geringe verzögerung, z.B. Videostreaming, online gaming, Echtzeit-Multimedia (mit lag?)
Uniform Resource Identifier (**URI**); Domain Name System (**DNS**); Simple Network Mngmt Protocol (**SNMP**); Internet Control Message Protocoll (**ICMP**); Maximum Transmission Unit (**MTU**); Interior Gateway Protokolle (**IGP**) bzw. Exterior (**EGP**) z.B. Border Gateway Protokoll (**BGP**); Carrier Sense Multiple Access (**CSMA**)
HTTP: Dynamische Inhalte via Rumpf von Anfragenachricht mit POST und CGI-Skripten (Common Gateway Interface) *oder* durch Scripting: PHP (serverside), Javascript (clientside), weitere Verbesserung durch Ajax
Web-Caching: Verringerung der Wartezeit des Users und des Netzwerkverkehrs durch Zwischenspeicher & bei Server erfragen, ob sein Objekt noch aktuell ist
HTTP/2: Multiplexing: Vermeidung von Head-of-Line (HOL) Blockierung durch große Objekte durch Aufteilung in kleinere Frames und Interleaving
Out-of-Band-Control: FTP-Steuerbefehle gehen nicht über die normale Datenverbindung, sondern über extra aufgebaute Verbindungen. Vorteil: Steuerung während einer Datenübertragung ist möglich.
Netzwerksicherheit: Funktionssicherheit: nicht durch technisches Fehlverhalten eine Bedrohung für materielle Güter und die körperliche Unversehrtheit darstellt Informationssicherheit: Schutz eines IT-Systems gegen unautorisierte Nutzung Datenschutz: Fähigkeit einer natürlichen Person, die Weitergabe von Informationen, die sie persönlich betreffen, zu kontrollieren
Schutzziele Informationssicherheit: Authentizität, Datenintegrität, Informationsvertraulichkeit, Verfügbarkeit, Verbindlichkeit, Anonymisierung und Pseudomisierung
P2P: Upload-Bitrate der Peers wird mitgenutzt, flexibler Beitritt, meist dezentral, Fluten skaliert schlecht
Content Distribution Network (CDN)**:** Video-Streaming für Milliarde Nutzer weltweit. Statt singulärem, massivem Server -> sehr viele (bis zu mehrere 100k) Spiegel-Server geografisch verteilt & näher am User -> Engpässe auf Strecke zum User minimieren: erste/ letzte Meile, Peering-Punkte (Übergänge zwischen ISPs); Keine mehrfache Übertragung des selben Inhalts über eine Strecke; Vermeiden eines Single Point of Failure
UDP-Pseudo-Header: Quell- & Ziel-IP, Protokollnr. (17 = UDP) und Segmentlänge -> Kontrolle der Prüfsumme erkennt auch Fehler in den IP-Adressen *Aber:* Verletzung Schichtenprinzip (aber nur auf Endsyst)
Interdomain-Routing: Typen von autonome Systeme (AS): Stub AS: nur eine Verbindung zu anderen AS. Multihomed AS: mehrere Verbindungen zu anderen AS, aber *kein* Durchgangsverkehr. Transit AS: mehrere Verbindungen zu anderen AS und auch Durchgangsverkehr.
Hub: elektrische Signale werden an alle angeschlossenen Geräte verteilt; evtl. Kollisionen (Layer 1)
Switch: Pakete werden genau zu Zieladresse weitergeleitet; *keine* Kollisionen (Layer 2)
Network Address Translation (NAT): Adressknappheit -> Netzwerke verwenden intern global ungültige Adressen über *eine* global gültige Adresse; NAT-Router überschreibt Quelladresse/ -port bei Übertragung von innen nach außen & vice versa; bessere Abschirmung; Nachteil: Schichtenprinzip wird verletzt
Address Resolution Protocol (ARP): Netzwerkadr. -> Hardware-Adr.; A's ARP-Tabelle hat keine HW-A für B:
A: ARP-Anfrage als Broadcast (FF-FF-FF-FF-FF) mit seiner physikalischen Adr. und der IP-Adr. von B
B: sendet in ARP-Antwort seine physikalische Adr. an die physikalische Adr. von A
A: speichert die Zuordnung der Adressen von B in seiner ARP-Tabelle -> Autonomität, „Soft State“
TCP: Sender: 1. wenn Daten zum Senden und Platz im Fenster: erstelle Segment mit nextSQN und sende es mit IP, erhöhe nextSQN um Länge der Daten; wenn es das erste Paket im Fenster ist, starte Timer 2. wenn ein ACK mit ACK-Nr. im Fenster zurückkommt, schiebe das Fenster bis zu dieser ACK-Nr.; wenn das Fenster leer ist, stoppe den Timer, sonst starte den Timer neu 3. wenn der Timeout abläuft, sende das erste unbestätigte Paket des Fensters erneut, starte den Timer erneut **Empfänger:** wenn ein Segment ankommt und SQN = Fensteranfang ist und alle vorherigen Segmente bereits bestätigt sind: schiebe Fensteranfang bis zum nächsten erwarteten Byte und warte ein Timeout, wenn bis dahin kein neues Segment ankommt, schicke ein ACK mit dem Fensteranfang (delayed ACK); SQN = Fensteranfang ist und ein vorheriges Segment noch nicht bestätigt wurde, schiebe Fensteranfang bis zum nächsten erwarteten Byte und schicke sofort ein kumulatives ACK mit dem Fensteranfang SQN > Fensteranfang ist, puffere die Daten und schicke sofort ein kumulatives ACK mit dem Fensteranfang es eine Lücke teilweise oder ganz füllt, puffere die Daten, schiebe Fensteranfang bis zum nächsten erwarteten Byte und schicke sofort ein kumulatives ACK mit dem Fensteranfang