

- HTTP version _ status code _ phrase _ cr lf (= "/r/n") header field name: _ value (beliebig oft) ... cr lf RUMF (wird meist bei POST benutzt)	Stop-And-Wait: 1. Sende Paket mit aktueller SQN + Timer 2. ACK (ohne Bitfehler und aktueller SQN) vor Timer → SQN+1 und 1 3. Timeout: sende Paket erneut + Restart Timer Empfänger: Paket ohne Fehler mit aktueller SQN, sende ACK mit aktueller SQN sonst letzte ACK neu.	TCP-Server //WelcomeSocket ServerSocket welcomeSocket = new ServerSocket(port); while(true) { //Schleife für den Socket //erzeuge Socket blockiert bis Client verbindet Socket connectSocket = welcomeSocket.accept(); //Alle anderen Socket identisch zum Client connectSocket.close(); //Schliesst Verbindung } UDP-Server //erzeuge Datagrammsocket DatagramSocket serverSocket = new DatagramSocket(9876); //Senden String byte[] data = myString.getBytes("UTF-8"); //Senden primitiv byte[] data = ByteBuffer.allocate(8).putDouble(1729.1729).array(); DatagramPacket sendPkt = new DatagramPacket (data, data.length, ipAddress, port); sock.send(sendPkt); //Empfangen byte[] receiveData = new byte[1024]; DatagramPacket rcvPkt = new DatagramPacket(rcvData, rcvData.length); sock.receive(rcvPkt); String s = new String(rcvPkt.getData(), 0, rcvPkt.getLength(), "UTF-8"); double d = ByteBuffer.wrap(rcvPkt.getData(), 0, rcvPkt.getLength()).getDouble(); //IP und PORT fürs sneden InetAddress ipAddress = rcvPkt.getAddress(); int port = rcvPkt.getPort();
DHCP C: DHCP discovery src. 0.0.0.0, dest: 255.255.255.255; S: DHCP offer src. own ip, dest: 255.255.255.255, yiaddr:clientIP C: DHCP Request: src: 0.0.0.0, dest: serverIp S: DHCP ACK: src: serverIP, dest: 255.255.255.255 TCP-Fehlerkontrolle: 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 (500 ms), 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 Fensteranfg. - 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 Überlastkontrolle - Slow Start 1 setze CongestionWindow= MSS 2 nach Erhalt eines ACKs: CongestionWindow+= MSS 3 bis Thresholderreicht ist, dann Additive Increase - nach 3 doppelten ACKs: 1 MultiplicativeDecrease:Threshold= CongestionWindow/2; CongestionWindow/= 2 2 Additive Increase: bei Erhalt eines ACKs: CongestionWindow+= MSS x (MSS/CongestionWindow) - nach Timeout 1 Threshold= CongestionWindow/2; CongestionWindow= MSS	Go-Back-N: - S: Paket bis zu einem Wert x - E: kumulative ACKs (ACK mit SQN → alle Pakete bis SQN ack'd) • Sendepuffer: base = SQN ältestes Paket; nextSQN = SQN next Paket; W: Fenstergröße; Fenster[base,base+W-1], [base, nextSQN-1]: unbestätigte Pakete, [nextSQN, base+W-1]: bisher ungesendete Pakete 1. Wenn Daten + Platz in W → sende Packet mit nextSQN & nextSQN++; wenn nur 1 in Fenster → starte Timer 2. ACK ohne Bitfehler und SQN in W → schiebe Fenster bis SQN; Wenn W leer → stoppe Timer, sonst restart 3. Timeout: Sende alle unbestätigten Pakete des Fensters erneut Empfänger: Paket ohne Fehler + aktuelle SQN → sende ACK mit aktueller SQN & SQN++, sonst ACK neu Selective Repeat 1. Sende Paket & Timer (pro Paket) & nextSQN++ 2. Wenn ACK (SQN in W, keine Fehler) →markiere Paket als bestätigt + move W 3. Timeout: Sende Paket erneut (wofür Timeout) Empfänger: Paket ohne Bitfehler + SQN in W → sende ACK mit SQN, Puffer Paket, Schiebe W SQN von vorigem Fenster: sende ACK neu	TCP-Client //Eingabestrom für Standardeingabe BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in, "UTF-8")); //Clientsocket Socket clientSocket = new Socket("hostname", port); //Ausgabe-Strom für Socket OutputStream out = sock.getOutputStream(); DataOutputStream dataOut = new DataOutputStream(out); dataOut.writeDouble(3.14159); //Einfach BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(out, "UTF-8")); writer.write("Beispieltext" ‘\n’); writer.flush() //String //Eingabe InputStream in = sock.getInputStream(); BufferedReader reader = new BufferedReader(new InputStreamReader(in, "UTF-8")); String s = reader.readLine(); //Einfach Datentype DataInputStream dataIn = new DataInputStream(in); double d = dataIn.readDouble() //readInt,readBoolean; connectSocket.close(); //Schliesst Verbindung UDP-Client DatagramSocket clientSocket = new DatagramSocket(); //Übersetzte hostname mit DNS InetAddress ipAddress = InetAddress.getByName("hostname"); //Identisch zu Server Java Integer.parseInt(str); str.equals(str2); str.startsWith(x) str.substring(y,x); str.indexOf("x") public static void main (String argv[]) throws Exception){}
	ALOHA: sofort senden, auf ACK warten, wenn kein ACK, wiederhole CSMA: 1-persistent: 1. wenn das Medium frei ist, sendet der Knoten sofort 2. wenn das Medium belegt ist, wartet der Knoten bis es frei ist und geht dann zu 1. 3. wenn nach Senden kein ACK kommt, geht der Knoten der Knoten in Backoff, danach zu 1. nicht-persistent 1. wenn das Medium frei ist, sendet der Knoten sofort 2. wenn das Medium belegt ist oder nach Senden kein ACK kommt, geht der Knoten in Backoff, danach zu 1. p-persistent: 1. wenn das Medium frei ist, sendet der Knoten jeweils mit Wahrscheinlichkeit p oder wartet noch einen Slot mit Wahrscheinlichkeit 1-p 2. wenn das Medium belegt ist, wartet der Knoten bis es frei ist 3. wenn nach Senden kein ACK kommt, geht der Knoten der Knoten in Backoff, danach zu 1. CSMA/CD: (listen while talking) - Senden, wenn Killision erkannt wird, abbrechen, anderen sagen später senden	

Übertragungsverzög.	$d_{Transport} = \frac{L}{R}$
Ausbreitungsverz.	$d_{prop} = \frac{l}{v}$
Kanalpuffergröße	$a = \frac{R \cdot D}{L} = \frac{\frac{l}{v}}{\frac{L}{R}}$
Leitungsvermittlung	$R_{eff} = \frac{R}{x}$

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

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

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

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:

$$\rho = \frac{L \cdot \lambda}{R} \begin{cases} d_{queue} \text{ fällt} \\ d_{queue} \text{ steigt} \\ d_{queue} \rightarrow \infty, \end{cases}$$

λ : (Pakete pro Sekunde)

Verzögerung HTTP mit M Objekten und 1 Basisseite:
M = Basisseite = O Bits

nicht-persistent:

$$(M + 1) \left(2 \cdot RTT + \frac{O}{R} + \text{SSW} \right)$$

persistent

$$2 \cdot RTT + RTT + (M + 1) \cdot \frac{O}{R} + \text{SSW}$$

persistent mit Pipelining

$$2 \cdot RTT + RTT + (M + 1) \cdot \frac{O}{R} + \text{gem. SSW}$$

$$\text{gem. SSW} = \text{SSW} + RTT - \max \left\{ \frac{L}{R} + RTT - \left(2^{K-1} \cdot \frac{L}{R} \right); 0 \right\}$$

nicht-persistent mit X parallelen Verbindungen ($\frac{M}{X}$ ganze Zahl):
in $SSW_{parallel}$ R durch $\frac{R}{X}$ ersetzen

$$\left(\frac{M}{X} + 1 \right) \cdot 2 \cdot RTT + (M + 1) \cdot \frac{O}{R} + SSW_{normal} + SSW_{parallel}$$

TCP-Leistungsanalyse

Festes Fenster

$$\text{1. Fall} \quad \frac{WL}{R} \geq \frac{L}{R} + RTT$$

$$d = 2 \cdot RTT + \frac{O}{R}$$

$$\text{2. Fall} \quad \frac{WL}{R} < \frac{L}{R} + RTT$$

$$d = 2RTT + \frac{O}{R} + \frac{(K-1) \cdot \left(\frac{L}{R} + RTT - \frac{WL}{R} \right)}{\text{\#Fenster}} \quad \text{Wartezeit pro Fenster}$$

$$K = \left\lceil \frac{O}{WS} \right\rceil \quad L = \text{MaxSegmentSize (MSS)} = S$$

Slow-Start

$$d = 2RTT + \frac{O}{R} + \underbrace{P \cdot \left(RTT + \frac{L}{R} \right) - (2^P - 1) \cdot \frac{L}{R}}_{\text{Slow Start Wartezeit}}$$

P = min(Q, K-1)

$$K = \left\lceil \log_2 \left(\frac{O}{L} + 1 \right) \right\rceil$$

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

mehrere Links T :

$$d = 2 \cdot RTT_T + \frac{O}{R} + (T - 1) \cdot \frac{L}{R} +$$

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

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

um Faktor 2 erhöhen, also Fenstergröße W^3 :

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

$$Q = \left\lfloor \log_3 \left(1 + \frac{RTT}{\frac{L}{R}} \right) \right\rfloor + 1$$

$$K = \left\lfloor \log_3 \left(\frac{2O}{L} + 1 \right) \right\rfloor$$

Wahrscheinlichkeit, dass n Nutzer gleichzeitig senden:

$$P_n = \binom{\max}{n} \cdot (P_{user})^n \cdot (1 - P_{user})^{\max-n}$$

Fehlerkontrolle

1. Stop-And-Wait:

$$\text{Durchsatz:} \quad \frac{L}{N \cdot \left(\frac{L}{R} + 2D \right)}$$

$$\text{Normierter Durchsatz:} \quad S = \frac{1}{N(1 + 2a)}$$

Falls keine Fehler: $N = 1$

$$N = \frac{1}{1 - p}$$

p = Fehlerwahrscheinlichkeit pro Packet

2. Selective-Repeat (mit Pakete auf Kanal $a = \frac{R \cdot D}{L}$)

$$W > 1 + 2a: S = \frac{W \cdot L}{W \cdot \frac{L}{R}} \cdot \frac{1}{R} = 1$$

Fehlerwahrscheinlichkeit p : $S = 1 - p$

$$W < 1 + 2a: S = \frac{W \cdot L}{\frac{L}{R} + 2D} \cdot \frac{1}{R} = \frac{W}{1 + 2a}$$

Go-Back-N (ohne Fehler wie Selective-Repeat)

Mittlere Anzahl Sendeveruche pro Paket:

$$N = \frac{1 - p + K \cdot p}{1 - p}$$

Fall 1: $W \geq 1 + 2a$: ($K = 1 + 2a$)

$$S = \frac{1 - p}{1 + 2ap}$$

Fall 2: $W < 1 + 2a$: ($K = W$)

$$S = \frac{W \cdot (1 - p)}{(1 - p + W \cdot p) \cdot (1 + 2a)}$$

Timeout Schätzung anhand RTT, RTT Durchschnitt

$$\text{EstimatedRTT} = (1-\alpha) \times \text{EstimatedRTT} + \alpha \times \text{SampleRTT}$$

typischer Wert: $\alpha = 0,125$

$$\text{DevRTT} = (1-\beta) \times \text{DevRTT} + \beta \times |\text{SampleRTT} - \text{EstimatedRTT}|$$

typisch: $\beta = 0,25$

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 \times \text{DevRTT}$$

Sequenznummernraum mit $m = 2^n$ Werten

1. Falls Empfangsfenstergröße = 1: $W < m$ hinreichend

2. Falls Sendefenstergröße = Empfangsfenstergröße > 1: $W < \frac{m+1}{2}$

Leistungsanalyse Medienzugriff

ALOHA:

Wahrscheinlichkeit für Senden ohne Kollision bei Sendewahrscheinlichkeit p :

Normalisierter Durchsatz:

$$S = N \cdot p(1 - p)^{2(N-1)}$$

Sendeversuche pro Slot: $G = N \cdot p \Rightarrow p = \frac{G}{N}$

ALOHA: $G = 0,5$

Slotted-ALOHA: $G = 1$

N = #Knoten

CSMA

Voraussetzung: ^{*} $d_{prop} < \text{Rahmensendezeit} \cdot \frac{L}{R}$

CSMA/CD

$$S_{max} = \frac{\text{Sendezeit}}{\text{Sendezeit} + \text{Ausbreitungszeit} + \text{Wettbewerbszeit}}$$

$$= \frac{\frac{L}{R}}{\frac{L}{R} + D + (e - 1) \cdot 2D} = \frac{1}{1 + 4,4a}$$

Minimale Rahmengröße L :

$$\frac{L}{R} > 2 \cdot D \Rightarrow L > 2 \cdot R \cdot D \quad \rightarrow \text{Aus}^*$$

$$\text{Erfolg} \quad p = \frac{1}{N} \quad (P_{\text{erfolg}})^{\max} = \frac{1}{e}$$

$$P_{\text{erfolg}} = N \cdot p(1 - p)^{N-1}$$

Cyclic Redundancy Check

- Nutzdaten D mit d Bits, Prüfdaten R mit r Bits, Generatorpolynom G mit $r + 1$ Bits
- R ist Rest bei $(D \cdot 2^r)/G$
- (D, R) ist folglich $(D \cdot 2^r) \oplus R$
- Korrekt falls $(D, R)/G = 0$

DHCP

- C: DHCP discovery src. 0.0.0.0, dest: 255.255.255.255;
- S: DHCP offer src. own ip, dest: 255.255.255.255, yiaddr: cIp
- C: DHCP Request: src: 0.0.0.0, dest: serverIp
- S: DHCP ACK: src: serverIP, dest: 255.255.255.255