Inhaltsübersicht Autor: Julian Kotzur

1. Einführung:
   (a) Klassifikation von Kommunikationssystemen
   (b) Netzwerksicherheit
2. Anwendungsschicht:
   Stellt eine effiziente Kommunikation mit anderen Anwendungsprogrammen innerhalb eines Netzwerks sicher
   (a) Client-Server-Paradigma
   (b) HTTP / FTP
   (c) SMTP und POP3 mit Programmierung
   (d) Netzwerkmanagement: FCAPS,SNMP, MIB, BER
   (e) DNS und CDNs
   (f) Peer-to-Peer
3. Transportschicht:
   übernimmt die Daten von der Sitzungsschicht, verkleinert sie wenn nötig und übergibt sie der Vermittlungsschicht
   (a) UDP
   (b) Stop-and-Wait / Go-Back-N / Selective Repeat
   (c) TCP: Vollduplex, zuverlässiges Transportprotokoll im Internet
      i. Segmentformat / Multiplexen und Demultiplexen
      ii. Fehlerkontrolle / Fast Retransmit
      iii. Multipath: Erweiterung von TCP, zusätzlich API zur Nutzung der Funktionen von MPTCP
      iv. SACK
      v. Verbindungsaufbau und Verbindungsabbau
      vi. RTT-Schätzung und Latenzzeit
      vii. Fluss und Überlastkontrolle
   (d) TLS: hybrides Verschlüsselungsprotokoll zur sicheren Datenübertragung im Internet
4. Netzwerkschicht:
   steuert den Austausch von Datenpaketen, da Zwischenziele notwendig sind.
   (a) IP
      i. IPv4 / IPv6 / Weiterleitungstabelle
      ii. ICMP: Kontrollnachricht von Router an andere Teilnehmer
      iii. Subnetze und klassenlose Adressierung
      iv. Fragmentierung
   (b) DHCP: Client-Server-Protokoll
   (c) NAT / VPN / MPLS / SDN
   (d) IPsec: Sicherheit auf Netzwerkschicht, ermöglicht VPN
   (e) Aufbau eines Routers / Routing Allgemein / Interdomain-Routing / Routing-Vergleich
   (f) Link-State / Distanzvektor / Peer-to-Peer-Routing: $succ(p + 2^{(i-1)} \% m)$
   (g) Poisened Reverse: Lösung des Count to infinity Problems bei zyklen der Länge 2.
5. Sicherungsschicht
   fehlerfreie Übertragung von Frames über Übertragungsabschnitte
   (a) Adressierung: MAC-Adresse
   (b) Datensicherung
   (c) Medienzugriff
   (d) Ethernet: dominierende Technologie für kabelgebundene LANs
   (e) Drahtlose LANs: Hidden-Terminal-Problem, Medienzugriff gemäß Basic Access
6. Physikalische Schicht
   elektrische oder mechanische Schnittstelle zum physischen Medium
   (a) Signale
   (b) Leitungskodierung binärer Signale: Zuordnung von Signalwerten zu Nullen und Einsen einer Bitsequenz
   (c) Maximale Datenrate: Nyquist-Theorem, Shannon-Theorem
   (d) Modulation: Änderung von Parametern eines Trägersignals durch ein aufgeprägtes Signal
   (e) Übertragung über elektrische Leiter
   (f) Signalübertragung über Lichtwellenleiter: Signal mit LED in optisches Signal
   (g) Funkübertragung: Kabellos durch elektromagnetische Wellen

Programmierhinweise

Socket

- Socket(String host, int port): Erzeugt einen Socket und verbindet ihn mit der Port-Nummer am angegebenen Host.
- InputStream getInputStream(): Liefert den Eingabestrom für den Socket.
- OutputStream getOutputStream(): Liefert den Ausgabestrom für den Socket.
- void close(): Schließt den Socket.
- void connect (SocketAddress endpoint, int timeout): Verbindet den Socket mit einem Server
- boolean isConnected(): Gibt True zurück, wenn Socket verbunden ist.

DatagramSocket

- DatagramSocket(int port): Erzeugt einen neuen UDP-Socket, der am lokalen Port port angeschlossen ist.
- void bind(SocketAddress addr): Bindet den Socket an die übergebene lokale Adresse
- void close(): Schließt den Datagram-Socket.
- void send(DatagramPacket p): Sendet das Datagrammpaket p an die in p eingetragene Adresse.
- int getReceiveBufferSize(): Liefert die Größe des Empfangspuffers zurück.
- void receive(DatagramPacket p): Empfängt ein Paket über den Socket und trägt es in p ein, wobei die Daten gegebenenfalls auf die Pufferlänge von p gekürzt werden (der Rest wird verworfen).

DataOutputStream

- DataOutputStream(OutputStream out): Erzeugt einen neuen DataOutputStream, der mit out initialisiert wird.
- void flush(): Bewirkt, dass noch gepufferte Daten in den Stream geschrieben werden.
- int size(): Liefert die Anzahl der bisher in den Stream geschriebenen Bytes.
- void write(int b): Schreibt das niederwertigste Byte von b in den Stream.

BufferedWriter (schreibt Tipp \n = new line)

- BufferedWriter(Writer out):Erzeugt neuen BufferedWriter mit Puffergröße von 8.192 Zeichen
- void close(): Schließt den Stream. Zuvor wird der Pufferinhalt in den darunterliegenden Stream geschrieben.
- void flush(): Schreibt alle im Puffer befindlichen Bytes in den Stream und ruft dessen flush()-Methode auf.
- void write(int c): Schreibt die niederwertigen 2 Bytes von c als char-Wert in dem Stream.

BufferedReader (liest)

- BufferedReader(Reader in): Erzeugt neuen BufferedReader mit Puffergröße von 8.192 Zeichen
- void close(): Schließt den Stream.
- int read(): Liest ein Zeichen und liefert es in der Unicode-Codierung zurück. Der Rückgabewert ist -1, falls das Ende des Streams erreicht ist.
- int read(char[] b, int off, int len): Versucht, len Bytes aus dem Stream zu lesen und speichert sie ab dem Index off in b. Wenn beim Versuch, das erste Byte zu lesen, das Stream-Ende bereits erreicht war, ist der Rückgabewert -1, ansonsten wird die Anzahl der tatsächlich gelesenen Bytes zurückgeliefert.

Try-and-Catch und Exeptions

```
try { ...
} catch (IOException e) { e.printStackTrace();
} finally { ... }
```

Rechnerkommunikation

Klassifikation von Kommunikationssystemen

Kommunikationsarten:
→ Unicast: Ein Sender und ein Empfänger
→ Multicast: Ein Sender und Empfängergruppe
→ Broadcast: An alle Teilnehmer des Netzes
→ Anycast: Ein Empfännger aus Gruppe möglicher Ziele

Übertragungsarten:
→ Simplex: Nur in eine Richtung möglich
→ Halbduplex: Beide Richtungen möglich, nicht gleichzeitig
→ Vollduplex: gleichzeitig in beide Richtungen

Multiplexverfahren:
→ Methoden zur Signalübertragung, wobei mehrere Signale zusammengefasst und simultan über ein Medium übertragen werden. z.b. Zeitmultiplex und Frequenzmultiplex

Leitungsvermittlung:
→ Signalisierung baut einen Kanal zur Übertragung auf
Statistische Kanalaufteilung:
⇒Vorteil: keine Peaks möglich
⇒Vorteil: Zusicherung an Verzögerung möglich
⇒Nachteil: Ineffiziente Nutzung des Mediums
→ Vorhandene Bitrate wird fest auf Kanäle aufgeteilt
→ Ineffizient bei schwankenden Datenaufkommen mit Pausen
→ FDMA = Parallel zur Zeit (Frequenzintervall)
→ TDMA = Senkrecht zur Zeit (Zeitintervall)

Paketvermittlung:
→ Gesendete Datenpakete gelangen einzeln zum Empfänger
→ die Bitrate wird effizienter aufgeteilt
→ Zu viele Daten können über Puffer gesichert werden
→ dies kann zu Verzögerungen und Pufferüberläufen führen

Übertragunsmedium:
→ Leitungsgebunden (z.b. Glasfaser / Kuperdrähte):
⇒ Bitraten von Kbps bis viele Gbps
⇒ kleine Bitfehlerraten
→ Drahtlos (z.b. Funk / Infrarot):
⇒ Bitfehlerraten hoch
⇒ außerdem treten Bitfehler oft in Schüben (Bursts) aufc

Protokolle

Grundlagen:
→ Legt Nachrichtenformat und Teilnehmerverhalten fest
→ Protokolle sind in Schichten aufgeteilt
⇒Das sind die Schichten aus der VL
→ Instanz: Stellt Kommunikation sicher
→ Dienst: Beschreibt, was eine Instanz anbietet
→ Protokoll: Plan um Dienste mit Instanzen zu realisieren
→ Schicht:Zusammenfassung von Instanzen
→ Schichtenarchitektur: Festgelegtes System von Schichten

Cross-Layer Optimierung:
Die Trennung in Schichten wird oft nicht eingehalten, z.B. werden zur Steigerung der Effizienz Mechanismen der Bitübertragungs- und der Sicherungsschicht gekoppelt

Dienstgüte (QoS = Quality of Service):
→ Aspekte von Rechnernetzen und Protokollen
⇒ Latenz (Zeit, bis Paket Empfänger erreicht)
⇒ Jitter (Maß für die Variabilität der Latenz)
⇒ Durchsatz (übertragene Daten pro Zeiteinheit)
⇒ Verlust- und Fehlerrate
⇒ Verfügbarkeit (Zeitanteil eines einsatzfähigen Systems)

Netzwerksicherheit

Funktionssicherheit:
Eigenschaft eines IT-Systems, dass es durch Fehler nicht zu Sachschäden oder Körperschäden kommt.
Informationssicherheit:
Schutz eines IT-Systems gegen unautorisierte Nutzung
Datenschutz:
Fähigkeit einer natürlichen Person, die Weitergabe von Informationen, die sie persönlich betreffen, zu kontrollieren
Schutzziele:
→ Authenzität
→ Datenintegrität
→ Informationsvertraulichkeit
→ Verfügbarkeit
→ Verbindlichkeit
→ Anonymisierung und Pseudomisierung

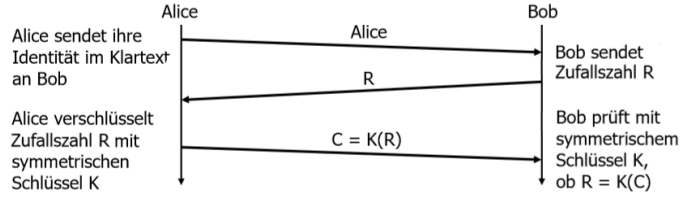
Kryptosysteme:
→ Symmetrisches Kryptosystem:
⇒ Jeder hat den gleichen Schlüssel
⇒ Beispiel: Cäsar
→ Asymmetrisches Kryptosystem:
⇒ Jeder hat seinen eigenen Schlüssel
⇒ Beispiel: RSA

RSA-Verfahren:
→ Man hat Zahlen, die Bedingungen erfüllen
→ Damit erhält man einen öffentlichern und privaten Key
→ Andere nutzen den öffentlichen Key für Nachricht N
→ Man selbst entschlüsselt N mit dem privaten Key

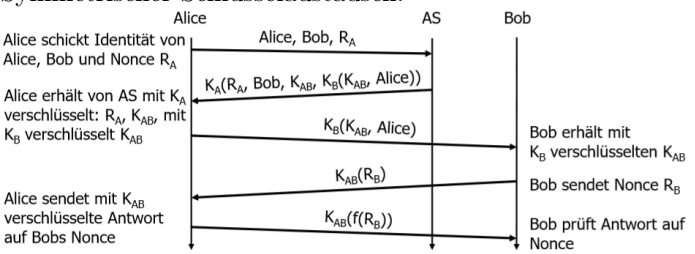
Kryptografische Hashfunktionen:
→ Fingerabdruck zur Unveränderlichkeitüberprüfung
→ Beispiel: Message Authentication Code
⇒ Von symmetrischem Schlüssel abhänge Hashfunktion

Digitale Signatur:
→ Signaturaufbau: privater Schlüssel und Hashfunktion
→ Sichert Urheberschaft und Integrität

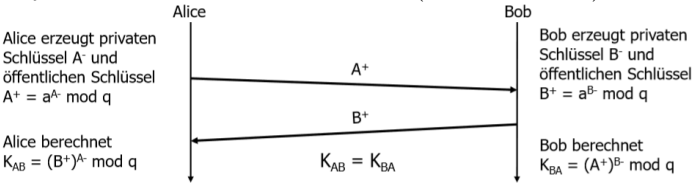
Challenge-Response-Verfahren:
→ Zur Authentifikation des Kommunikationspartners



Schlüsselaustausch:
Symmetrischer Schlüsselaustausch:



Asymmetrischer Schüsselaustausch (Diffie Hellman):



Anwendungsschicht

Client-Server-Paradigma
→ Server stellt Dienste bereit, Client fordert Dienste an
→ Server ist somit leistungsfähig und immer verfügbar
→ Client manchmal verfügbar und kommuniziert mit Server
⇒ Es gibt weitere ähnliche Paradigmen

Anforderungen von Anwendungen

Anwendung	Verlust	Bitrate	Verzögerungszeit
Dateitransfer	kein Verlust	elastisch	keine harte Grenze
E-Mail	kein Verlust	elastisch	keine harte Grenze
Web-Dokumente	kein Verlust	elastisch	keine harte Grenze
Multimedia	Verlust tolerierbar	10 Kbps – 100 Mbps	einige s
Interaktive Spiele	Verlust tolerierbar	Kbps – 10 Kbps	einige 100 ms
Automatisierung	kein Verlust	Kbps	einige ms
Smartphone Messaging	kein Verlust	elastisch	keine harte Grenze

Anwendungs- und Transportprotokolle

Anwendung	Anwendungsprotokoll	Transportprotokoll
E-Mail	SMTP [RFC 5321]	TCP
Web	HTTP [RFC 2616]	TCP
Dateitransfer	FTP [RFC 959]	TCP
Streaming Multimedia	RTP [RFC 3550], HTTP	UDP oder TCP
Internettelefonie	SIP [RFC 3261], RTP	UDP oder TCP

HTTP

Einfacher Ablauf:
→ URI = Uniform Resource Identifier
→ URI enthält Web-Server-Namen und Pfad zu Objekt
→ 1. Benutzer gibt URI in Web-Browser ein
→ 2. Web-Browser stellt Anfrage an Web-Server für Objekt
→ 3. Web-Server liefert Objekt an Web-Browser zurück
→ 4. Web-Browser stellt Objekt für den Benutzer dar

Antwortnachrichten:
→ 200 OK (Anfrage erfolgreich)
→ 301 Moved Permanently (Redirection)
→ 400 Bad Request (Anfragenachricht nicht verstanden)
→ 404 Not Found (Objekt nicht gefunden)
→ 505 HTTP Version Not Supported

nicht-persistentes HTTP:
→ Ablauf:
⇒ 1. Client initiiert TCP-Verbindung zu Server an Port 80
⇒ 2. Server auf Host wartet auf TCP-Verbindungen an Port 80, nimmt TCP-Verbindung an, benachrichtigt Client
⇒ 3. Client übergibt HTTP-Anfrage an TCP-Socket, enthält URL mit Verweis auf Objekt
⇒ 4. Server empfängt Anfrage, erstellt HTTP-Antwort mit Objekt und übergibt diese dem TCP-Socket
⇒ 5. Server schließt TCP-Verbindung
⇒ 6. Client erhält HTTP-Antwort mit dem Inhalt, analysiert ihn, stellt ihn auf dem Bildschirm dar.
⇒ 7. Schritte 1-6 werden für jedes weitere Objekt wiederholt
→ Antwortezeit:
⇒ TCP-Verbindungsaufbau braucht RoundTripTime (RTT)
⇒ Anfrage mit Antwort erfordert noch eine RTT
⇒ insgesamt: 2 RTT + Sendezeit + TCP-Wartezeit

persistentes HTTP:
→ Server lässt Verbindung bestehen
→ alle Objekte werden über eine TCP-Verbindung gesendet
→ ohne Pipelining: eine Anfrage für jedes Objekt
→ mit Pipelining: eine Anfrage für alle Objekte

File Transfer Protocol (FTP)

→ Übertragung von Dateien zwischen Hosts
→ Eine TCP-Verbindung (Port 21) zur Steuerung
→ Pro Datei eine TCP-Verbindung
→ mehr Einzelheiten in der Übung
Out-of-Band-Control:
Steuerbefehle gehen nicht über die normale Datenverbindung, sondern über extra aufgebaute Verbindungen
⇒ Vorteile: Steuerung während Datenübertragung möglich
Passive Mode:
⇒ C baut Steuerverbindung zu S auf Port 21
⇒ C schickt Command pasv
⇒ S antwortet beliebigen Port und wartet
⇒ C baut TCP Verbindung zum S auf
⇒ Vorteil: C braucht Firewall nicht verwenden

Simple Mail Transfer Protocol SMTP

→ Zur ÜBertragung von ASCII Nachrichten
→ Versenden mit SMTP über TCP
→ Abholen mit POP3, IMAP, HTTP
→ Drei Phasen zur Übertragung:
⇒ Handshaking, Nachrichtenübertragung, Abschlussphase
→ Vertaulichkeit durch Verschlüsselung
→ Datenintegrität durch Hashfunktion
→ Sicherer Emailstandart: Pretty Good Privay (PGP)

Netzwerkmanagement

Aufgaben (FCAPS-Modell nach ISO)
→ Allgemein: Überwachung &Verwaltung eines Netzwerkes
→ Fehlermanagement
→ Konfigurationsmanagement
→ Abrechnungsmanag.
→ Leistungsmanag.
→ Sicherheitsmanag.

Simple Network Management Protocol (SNMP)
→ Benötigt Client mit Manager-Prozess
→ Managed Device = Gerät im Netz
→ Managed Object = HW /SW im Managed Device
→ Management Agent = Prozess auf Managed Device
→ Anfrage/Anwort-Protokoll über UDP
→ Manager entscheidet regelbasiert, was zu tun ist

SNMP-Messages
→ Get-Request: M to A um Daten von A zu erhalten
→ Get-Next-Request: M to A für nächsten Datenzugriff
→ Get-Bulk-Request: M to A für mehr Daten auf einmal
→ Set-Request: M to A erstellt oder ändert Daten
→ Response: A to M, Antwort auf Get / Set- Request
→ Trap: A to M Nachricht über Fehler

Management Information Base
Anwendung:
→ MIB als ASCII-Datei in der SMI-Syntax
→ Verzweigungsfestlegung / Angaben zu Managed Objects
⇒Datentyp (SYNTAX) / Zugriffsrechte (ACCESS)
⇒Status (STATUS) und die Position in der MIB
⇒Beschreibung (DESCRIPTION) des Objektes

Basic Encoding Rule (BER):
→ Repräsentation zur Übertragung
→ Tag, Length, Value (TLV)
⇒ Tag = Nummer für Typ
⇒ Length = Länge der Bytes

POP-3

Beispiel Ablauf:
S: (wartet auf Verbindung)
C: (oeffnet eine Verbindung)
S: +OK example .com POP3-Server
C: USER username
S: +OK Please enter password
C: PASS passwort in klartext
S: +OK mailbox locked and ready
C: STAT
S: +OK 2 436
C: LIST
S: +OK mailbox has 2 messages
S: 1 232
S: 2 204
S: .
C: QUIT
S: +OK bye
S: (schliesst die Verbindung)

Programmierung:

```
class Pop3 extends Base{
 void init() throws Exception;{
  socket = new Socket(POP3_SERVER, 110);
  out = new BufferedWriter(
   new OutputStreamWriter(
    socket.getOutputStream()));
  in = new BufferedReader(
   new InputStreamReader(
    socket.getInputStream()));
  authOK = authentificate();
 }
 boolean authentificate() throws Exception{
  string s = in.readLine();
  if(!s.startsWith("+OK")) return false;
  out.write("user " + USER); out.newLine();
  s = in.readLine();
  if(!s.startsWith("+OK")) return false;
  out.write("pass "+Password); out.newLine();
  s = in.readLine();
  if(!s.beginsWIth("+OK")) return false;
  return true;
 }
 int countMails() throws Exception{
  out.write("list"); out.newLine();
  int counter = 0;
  while(!in.readline().equals(".")){
   counter++;
  }
 return counter;
 }
 void test() throws Exception;{
 if(authOK == true){
  print(countMails());
 }else{
  end();
}}}
// Verbindungen schliessen:
 socket.close();
```

Bemerkung:
Round-Trip-Time:
→ Verzögerungen addieren
→ Hin und Rückweg addieren
CW = Congestion Window
→ Versenden von CW vielen Paketen pro Schritt

SMTP

Beispiel Ablauf:
S: 220 hamburger . edu
C: HELO crepes . fr
S: 250 Hello crepes . fr , pleased to meet you
C: MAIL FROM: <alice@crepes . fr>
S: 250 alice@crepes . fr . . . Sender ok
C: RCPT TO: <bob@hamburger . edu>
S: 250 bob@hamburger . edu . . . Recipient ok
C: DATA
S: 354 Enter mail , end with "." on a line by i t s e l f
C: How are you?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger . edu closing connection

Programmierung:

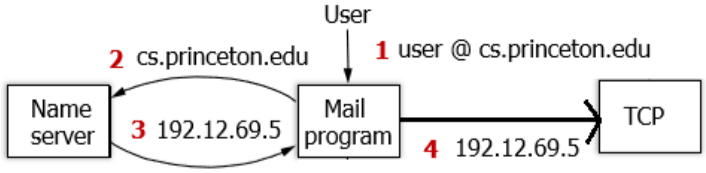
```
void sendEmail(int count) throws Exception {
 Socket socket = new Socket(
  "cipmail.cs.fau.de",25);
 final String CRLF = "\r\n";
 BufferedWriter writer =
  new BufferedWriter(
  new OutputSreamWriter(
  socket.getOutputSream(), "UTF-8"));
 BufferedReader reader =
  new BufferedReader(
  new InputStreamReader(
  socket.getInputStream(), "UTF-8"));

 String message = reader.readLine();
 // checkResponse Beispiel:
 if(!messageStartWith(220)){
 socket.close(); exit();}

 writer.write("HELO rk.org" + CRLF);
 message = reader.readLine();
 checkResponse( );
 writer.write("spam" + CRLF);
 message = reader.readLine();
 checkResponse( );
 writer.write("spam" + CRLF);
 message = reader.readLine();
 checkResponse( );
 writer.write("DATA" + CRLF);
 message = reader.readLine();
 checkResponse( );
 writer.write("spam" + CRLF);
 writer.write("." + CRLF);
 message = reader.readLine();
 checkResponse( );
 writer.write("QUIT" + CRLF);
 message = reader.readLine();
 checkResponse( );
 socket.close();
}
```

Domain Name System (DNS)

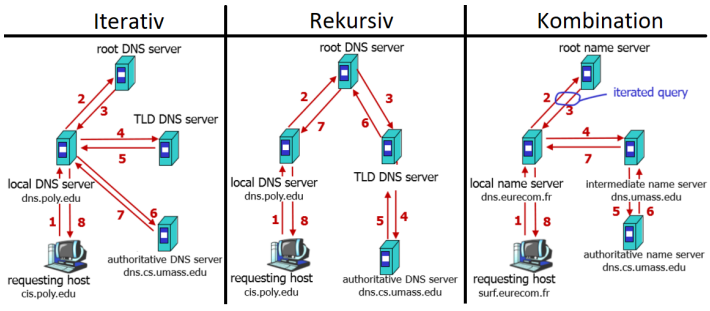
→ Host-Namen bzw. Domain-Name lesbar
→ DNS bildet Domain-Name auf Werte ab (z.b. IP-Adresse)
→ DNS ist verteilte Datenbank, die aus vielen Namen-Servern besteht, die über ein Anwendungsprotokoll kommunizieren



Domain-Struktur:
→ DNS implementiert Namenshierachie für InternetObjekte
→ von links nach rechts lesen, andersherum verarbeiten
→ eine Zone wird von einem Name-Server verwaltet
→ die Hierarchie wird durch die Name-Server implementiert
→ 13 Root Name Server weltweit

DNS Resource Records
→ Datensätze der Namensserver
→ Besteht aus Domainname, Wert, Typ, TTL
→ TTL = Time to Live / Dauer der Gültigkeit
→ Typ A : Wert = IPv4-Adresse
→ Typ NS : Wert = Host-Domain-Name, auf Namen-Server, der Namen in der Domain auflösen kann
→ Typ CNAME : kanonischer Hostname
→ Typ MX : Host-Domain-Name, auf dem Mail-Server läuft

DNS-Protokoll:
→ Anfrage- und Antwortnachrichten mit gleichem Format:
⇒ Kopf: Identification / Flags
⇒ Rumpf: Questions / Answer / Authority
→ Anfragearten: iterativ / rekursiv
→ iterativ: Antwort: anderer Server, der Namen auflösen kann / NS- und A- Datensatz / Antwort wird sofort geliefert / keine Informationen werden gespeichert
→ rekursiv: Antwort: Auflösung des namen, die u.U. von anderen Servern geholt wird / A-Datensatz / Information muss bei Anfragen auf andere Server gespeichert werden



Content Distribution Networks (CDNs):
→ Ziel: Reduktion der Wartezeit beim Surfen
→ Idee: Statt einem RZ, viele SpiegelServer weltweit verteilt
→ 3 Probleme vermeiden:
⇒ Engpässe auf Strecke zum Nutzer minimieren
⇒ Keine Mehrfachübertragung über eine Strecke
⇒ Verhindern eines Single point of failure
→ Verteilung der Anfragen:
⇒ Server-basierte HTTP Redirection: Server empfiehlt besseren Server, erfordert zusätzliche RTT
⇒ Client-nahe HTTP-Redirection: schwer umzusetzen
⇒ DNS-basierte Redirection: DNS-Server bilden den Domain-Namen des Services auf andere Domain-Namen ab
⇒ URL-Rewriting: Server liefert Basisseite, enthaltene Objekte-URLs werden mit Domain-Namen besserer Server umgeschrieben

Peer-to-Peer (P2P)

→ Idee: Inhalte von Peers, statt von zentralem Server
→ Eigenschaft: Hohe Konnektivität, Overlay-Netz
→ Andwendung: Filesharing / Beispiel: Bittorrent, Napster
→ Peers kommunizieren direkt mittels TCP oder UDP

Unstrukturiert
Zentralisiert (Architektur von Napster):
→ Eintritt eines Peers: informiert zentralen Server über seine IP-Adresse und seine Inhalte
→ Suche nach Inhalt: über zentralen Server
→ Dateiübertragung: direkt zwischen Peers
→ zentraler Server: Problemefaktor (bspw. juristisch)

Rein P2P:
→ kein zentraler Server benötigt
→ Fluten: wenn Anfrage nicht beantwortbar: Weiterleitung an mehrere Nachbarn (nicht an anfragenden Peer)
→ Fluten wird durch maximalen Hopcount begrenzt
→ Skalierbarkeit ist wegen des Flutens problematisch
→ Peers bilden Overlay-Netzwerk über TCP-Verbindungen
Eintritt in das Overlay-Netzwerk:
→ Nachricht an eine öffentliche Liste von möglichen Peers
Anfrage:
→ anfragender Peer sendet Anfrage an alle seine Nachbarn
→ Anfrage wird mit angebotenen Inhalten verglichen
→ wenn ein Peer den Inhalt anbieten kann, antwortet er dem anfragenden Peer, dieser leitet wiederum zurück (Identität des anfragenden Peers bleibt unbekannt)
→ Antwort gelangt zur Quelle. Peer der die Anfrage beantworten kann wird kontaktiert: Übertragung erfolgt

Hybrid:
→ Peers bilden Gruppen, einer ist Group Leader
→ Group Leader kennt Inhalte aller Peers aus Gruppe
→ Overlay-Netzwerk zwischen Group Leadern
→ Austausch zwischen Group Leadern
→ bessere Skalierbarkeit, ebenfalls keine zentrale Kontrolle

Strukturiert:
Routing:
→ Finden des gesuchten Eintrags in der DHT
→ Chord verwaltet Ring (Vorgänger-/Nachfolgerverweis)
→ Routing entlang des Rings (ineffizient) oder durch Sprungtabellen (logarithmischer Aufwand)
Neuen Knoten einfügen:
→ Voraussetzung: ein bekannter Knoten p in der DHT
→ neue ID wählen zwischen p und Nachfolger von p
→ Aktualisieren der Finger-Tabellen
Knoten entfernen:
→ Daten migrieren, ID entfernen, FingerTabellen ändern
Selbststabilisierung:
→ kontinuierliche Überprüfung aller Knoten

Bittorrent:
→ Torrent: Schwarm von Peers für gleiche Datei
→ Chunks: Teile der zu verteilenden Datei, z.B. 256 KB
→ Tracker: Server, bei dem sich Peers registrieren
→ .torrent-Datei mit Meta-Daten über Datei und Tracker
→ Rarest First: A fragt die seltensten Chunks der Peers zuerst nach, dadurch gleichmäßige Verteilung
→ Incentive Mechanismus: A misst Antwortrate der Peers und antwortet in entsprechendem Anteil der Upload-Rate
Neuer Peer A tritt Schwarm bei:
⇒ A registriert sich bei Tracker
⇒ A erhält IP-Adressen zufälliger anderer Peers
⇒ A baut TCP-Verbindung zu einigen dieser Peers auf

Transportschicht

UDP 32-Bit-Aufbau:
→ 16 Bit Quelltportnummer
→ 16 Bit Zielportnummer
→ 16 Bit Segmentlänge
→ 16 Bit Prüfsumme

UPD De-/Multiplexen:
→ Multiplexen: Segmentzusammenfürhung verschiedener Anwendungsprozesse durch Transportschicht auf Quellhost
→ Demultiplexen: Segmentaufteilung an die verschiedenen Anwendungsprozesse durch Transportschicht des Zielhosts
→ Anwendungsprozess vereinbart mit Transportschicht auf Quellhost Quellportnummer
→ UDP auf dem Zielhost erkennt nur an Zielportnummer, zu welcher Anwendung das Segment geliefert werden soll
→ ein Anwendungsprozess kann mehrere Sockets besitzen

UDP Prüfsummenrechnung:
Sender:
→ 1. Prüfsummenfeld wird mit 16 0er vorinizialisiert
→ 2. UDP-Paket-Daten werden in 16-Bit-Blöcke aufgeteilt
⇒ Falls letzter Block < 16 Bit, mit 0er auffüllen
→ 3. Addiere nun alle 16-Bit Blöcke mit Übertrag auf
⇒ Übertrag ergibt einfach ein Prüfsumme + 1
→ 4. Addiere dieses Ergebnis auf das Prüfsummenfeld
→ 5. Pseudeheader in 16-Bit-Blöcke und Summe bilden
→ 6. Pseudoheader Summe auf Prüfsummenfeld addieren
→ 7. Prüfsumme invertieren, falls nicht nur 1er bzw. 0er
Empfänger:
→ 1. Selben Algorithmus wie Sender durchführen
⇒ Keine Invertierung der Prüfsumme vornehmen!
→ 2. Empfänger und Sender Summen addieren
⇒ Falls Ergebnis der Summe 0: alle richtig
⇒ Ansonsten neues versenden anfordern (unbemerkt)

UDP Pseudo-Header:
→ enthält Quell- und Ziel-IP-Adresse, Protokollnummer (17 für UDP) und Segmentlänge
→ UDP des Senders schreibt zunächst 0 in das Checksum-Feld, erstellt einen Pseudo-Header und berechnet die Prüfsumme zusammen für das UDP-Segment und den Pseudo-Header
→ dann wird das Segment und der Pseudo-Header an IP weitergereicht
→ UDP des Empfängers erhält von IP das UDP-Segment und den PseudoHeader, schreibt 0 in das Checksum-Feld und berechnet die Prüfsumme für Segment und Pseudo-Header
→ Vorteil: die Kontrolle der Prüfsumme erkennt auch Fehler in den IP-Adressen, z.B. fehlgeleitete Segmente
→ Nachteil: Verletzung des Schichtenprinzips wg. IP-Adresse(aber nur auf Endsystem)

UDP Bitfehlerwahrscheinlichkeiten:
→ p = Wahrscheinlichkeit eines einzelnen Bitfehlers
→ L = Segmentlänge
⇒ p für min. einen Bitfehler: $1 - (1 - p)^L$
→ p für zwei Bitfehler: $(L^2 \cdot p^2)/2$

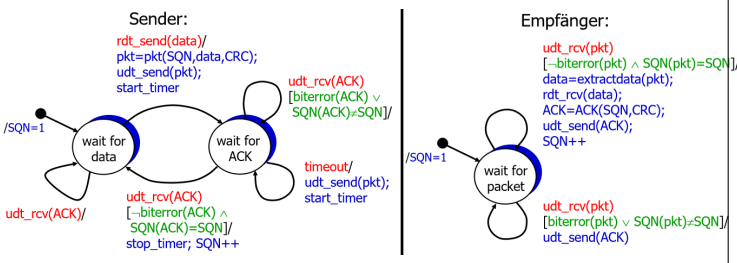
Fehlerkontrolle
Rauschen, Pufferüberläufe, Komponentenausfälle verursachen Bitfehler und Paketverluste. Dies Kann durch Protokolle mit Fehlererkennung, Bestätigung und Sendewiederholung ausgeglichen werden

```
// Klausur SS 2015
public class RKUDPScanner {
 public static boolean[] Busy;
 Busy = new boolean[1000];
 public static int n;
 public static int m = 0;
 public static void main(String[] args){
  DatagramSocket serverSocket;
  serverSocket = new DatagramSocket(1500);
  while (true) {
   byte[] recieveData = new byte[4];
   DatagramPacket recievePacket =
    new DatagramPacket(
    recieveData,recieveData.length);
    serverSocket.receive(recievePacket);
    String text = recieveData.toString();
    if (text.equals("exit") && m >= 10){
     serverSocket.close( ) ;
     System.exit(42);
    }else if(text.equals("scan")){
     m++;
     scan();
     sendEmail();
 }}}
 private static void scan(){
  n = 0;
  for(int i = 2000; i <= 3000; i++) {
   Busy[i - 2000] = !scanPort(i)
   if (Busy[i - 2000]) n++;
 }}

 private void sendEmail() throws IOE {
  DataOutputStream dOut;
  dOut = connectTo("cipmail.cs.fau.de", 25);
  dOut.writeBytes(
   "HELO" + "cipmail.cs.fau.de" + "\r\n");
  dOut.writeBytes(
   "MAIL FROM: status@rk.org" + "\r\n");
  dOut.writeBytes(
   "RCPT TO: mailChecks@cs.fau.de"+"\r\n");
  dOut.writeBytes("DATA" + "\r\n");
  dOut.writeBytes(
   "Subject: Belegte UDP Ports" + "\r\n");
  dOut.writeBytes(
   "Scan "+m+": "+n+
   " Ports sind besetzt"+"\r\n");
  for (int i = 0; i < Busy.length; i++) {
   if (Busy[i]) {
    int port = i + 2000;
    dOut.writeBytes(
     "Port " + (port) + " besetzt. \r\n");
  }}
  dOut.writeBytes(".\r\n");
  dOut.writeBytes("QUIT" + "\r\n");
  closeConnection(dOut);
}}
```

Quick UDP Internet Connections (Quic):
→ Transportprotokoll von Google
→ verschränkte Übertragung mehrerer Objekte
→ Gegenstück zu HTTP/2
→ UDP-basiert ⇒ keine Probleme mit Firewall/Middleboxes
→ Header weitestgehend verschlüsselt

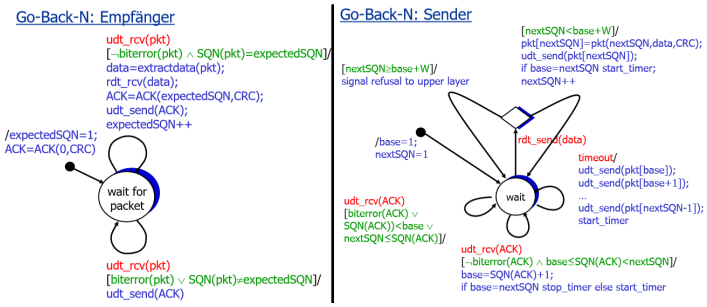
Stop-and-Wait:
→ wg. ACK-Verlust wird ACK zwischengespeichert
Verhalten des Senders:
→ 1. sende Paket mit aktueller SQN und starte Timer
→ 2. if ACK kommt ohne Bitfehler und mit aktueller SQN vor Ablauf des Timeouts zurück: SQN++
→ 3. else Timeout: sende das Paket erneut, starte den Timer erneut und goto 2.
Verhalten des Empfängers:
→ wenn Paket ohne Bitfehler und mit aktueller SQN ankommt, sende ACK mit dieser SQN und inkrementiere SQN, sonst sende das letzte ACK erneut



Stop-And-Wait Bemerkungen
Fehler:
⇒ 1. Paketverlust (Sender zu Empfänger)
⇒ 2. ACK-Verlust (Empfänger zu Sender)
⇒ 3. Duplikat
→ SQN wichtig zur Duplikatsüberprüfung
→ Alternating-Bit-Protokoll: SQN nur 0 oder 1
→ Durchsatzberechnung mit Fehler:

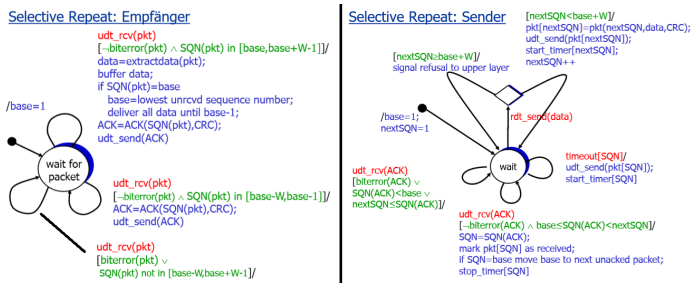
$$S = \frac{1-p}{1+2a} \; mit \; a = \frac{D}{L/R}$$

Go-Back-N
→ TCP ohne SACK
Verhalten des Senders:
→ 1. wenn Daten zum Senden und Platz im Fenster: sende Paket mit nextSQN und inkrementiere nextSQN; wenn es das erste Paket im Fenster ist, starte Timer
→ 2. wenn ein ACK ohne Bitfehler und mit SQN im Fenster zurückkommt, schiebe das Fenster bis zu dieser SQN; wenn das Fenster leer ist, stoppe den Timer, sonst starte den Timer neu
→ 3. wenn der Timeout abläuft, sende alle unbestätigten Pakete des Fensters erneut, starte den Timer erneut
Verhalten des Empfängers:
→ Verhalten wie bei Stop-and-Wait



Go-Back-N Bemerkungen:
Vorteile:
→ kumulative ACKs gleichen ACK-Verluste und -Verspätungen schnell aus, ohne dass die Pakete erneut gesendet werden müssen
→ der Sender benötigt nur einen Timer
→ der Empfänger benötigt keinen Puffer
→ Sender und Empfänger können einfacher realisiert werden, weil keine Lücken in den Fenstern beachtet werden müssen

Selective Repeat:
→ TCP mit SACK
Verhalten des Sender:
→ 1. Wenn Daten zum Senden und Platz im Fenster: sende Paket, starte Timer für dieses Paket und nextSQN++
→ 2. Wenn ein ACK ohne Bitfehler und mit SQN im Fenster zurückkommt: markiere das Paket mit SQN als bestätigt, schiebe das Fenster bis zur nächsten Lücke
→ 3. Wenn der Timeout für das Paket mit SQN: Paket senden und Timer erneut starten
Verhalten des Empfängers:
Wenn Paket ohne Bitfehler und mit SQN im Fenster ankommt: sende ACK mit dieser SQN, puffere das Paket und schiebe das Fenster bis zur nächsten Lücke
Wenn Paket mit SQN aus vorigem Fenster ankommt, sende das ACK hierfür erneut



Selective Repeat Bemerkungen:
Vorteil: weniger Sende-Wiederholungen, da nur wirklich fehlerhafte oder verschwundene Pakete erneut gesendet werden

TCP - Transmission Control Protocol:
→ Verbreiteste zuverlässige Transportprotokoll im Internet
→ Punkt-zu-Punkt: ein Sender, ein Empfänger
→ Reihenfolgenbewahrender Bytrestrom
→ Vollduplex und Verbindungsorientiert
→ Maximales Fenster: 2^{16}
⇒Erhöhbar durch ScaleFactor im Header

TCP - Segmentformat:
→ SQN-Number: Nummer des ersten Bytes des Segments
→ ACK-Number: Nummer des nächsten erwarteten Bytes
→ AdvertisedWindow: Fenstergröße für Flussstreuerung
→ checksum: Prüfsumme (wie bei UDP)
→ Flags mit Steuerinformationen:
⇒ URG (urgent pointer gültig)
⇒ ACK (ACK gültig)
⇒ PSH (Push Segment)
⇒ RST (Verbindung zurücksetzen)
⇒ SYN (synchronisiere Verbindung)
⇒ FIN (beende Verbindung)

TCP - Multiplexen und Demultiplexen:
Eindeutige Kennzeichnung durch:
→ Quell-IP-Adresse / -Portnummer
→ Ziel-IP-Adresse / -Portnummer

TCP - Fehlerkontrolle:
Verhalten des Senders:
→ 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

Verhalten des Empfängers (wenn Segment ankommt):
→ SQN = Fensteranfang und vorherige Segmente bestätigt: schiebe Fensteranfang bis zum nächsten erwarteten Byte und warte ein Timeout, wenn bis dahin kein neues Segment ankommt, schicke ein ACK mit dem Fensteranfang
→ SQN = Fensteranfang und Segmente teils nicht bestätigt: Schiebe Fensteranfang bis zum nächsten erwarteten Byte und schicke sofort ein kumulatives ACK mit dem Fensteranfang
→ SQN > Fensteranfang:
puffere die Daten und schicke sofort ein kumulatives ACK mit dem Fensteranfang

Fast Retransmit:
Verhalten Sender:
→ Sender schickt Pakete hintereinander
→ erhält der Empfänger mehrmals die Anfrage für das gleiche ACK, dann schickt er ohne den Timer zu beachten sofort das verlorene Paket erneut
Empfänger:
→ Schickt die ACK des letzten angekommenen Paketes +1
→ bei Paketverlust werden mehrmals die gleichen ACKs verschickt

TCP - Multipath
→ Erweiterung von TCP, um mehrere PFade für eine Verbindung nutzen zu können
Ansatz:
→ MPTCP als Zwischenschicht in Transportschicht
→ transparent, erscheint wie normales TCP
→ zusätzlich API zur Nutzung der Funktion von MPTCP

TCP - SACK
Grundidee:
→ Selective Acknowledgments informieren den Sender über einzelne Pakete, welche im Fenster liegen und nicht durch kumulative ACKs bestätigt werden können
→ Sender muss diese nicht erneut übertragen
Regeln:
→ normale ACKs werden unverändert verschickt
→ SACKs-Verschickung für das erste Paket außer der Reihe
→ Empfänger: verschickt so viele SACKs wie möglich
→ Sender: Neuübertragungen werden initiiert

TCP - Verbindungsaufbau:
3-Wege-Handshake:
→ 1. SYN-Segment: Client sendet Segment mit SYN-Flag=1, zufälliger initialer Client-SQN, ohne Daten
→ 2. SYNACK-Segment: Server sendet Segment mit SYN-Flag=ACK-Flag=1, zufälliger initialer Server-SQN, ACK=client_isn+1, ohne Daten; er legt Puffer und Variablen an
→ 3. ACK-Segment: Client sendet Segment mit ACK-Flag=1; SQN=client_isn+1, ACK=server_isn+1 und ggfs. Daten; er legt Puffer und Variablen an

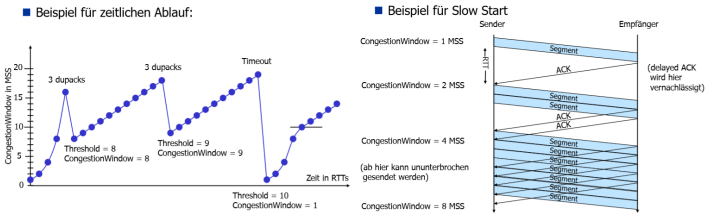
TCP-Verbindungsabbau:
→ jede Seite kann Verbindungsabbau durch Segment mit FIN-Flag=1 veranlasse
→ die andere Seite bestätigt mit ACK-Flag=1
→ beide Seiten müssen ihre Hälfte der Verbindung schließen
→ hat eine Seite geschlossen, sendet sie keine Daten mehr, nimmt aber noch welche an
⇒ 3-Wege-Handshake → Time Wait: Vebindungsabbau veranlasste Seite wartet, um bereit verschickte Segmente zu empfangen (schützt neue TCP-Verbindungen), übliche Werte: 30 s - 2 min., Verbindungsvariablen werden gehalte, Socket wird nicht neu vergeben

TCP - RTT-Schätzung:
→ Sender muss einen Timeout wählen
→ ACK kann frühestens nach einem RTT zurückkommen
→ Zu kleiner Timeout: unnötige Sendewiederholungen
→ Zu großer Timeout: späte Fehlererkennung
→ Timeout konfigurationsabhängig und dynamisch
→ Vorgehen: Zeitstempel für Segment und ACK ⇒ Differenz Messeb ⇒ Durchschnitt und Abweichung aus mehreren Messungen bestimmen ⇒ Timeout ableiten
Genauer:
→ Jede Messung ergibt SampleRTT
→ EstRTT = $(1-\alpha) \cdot EstedRTT + \alpha \cdot SplRTT$
⇒ großes α: starke Reaktion auf aktuelle Schwankung
⇒ kleine α: größere Stabilität, langsame Reaktion
⇒ typischer Wert: $\alpha = 0,125$
→ DevRTT = $(1-\beta) \cdot DevRTT + \beta \cdot |SplRTT - EstedRTT|$
⇒ typischer Wert: $\beta = 0,25$
→ TimeoutInterval = EstimatedRTT + 4 x DevRTT

TLS - Transport Layer Security:
Auf TCP-Verbindungsaufbau folgt:
→ Handshake: Aushandlung von Algorithmen (Verschlüsselung, MAC), Austausch von Nonces, Zertifikaten, Premaster Secret (PMS), Ableitung von Master Secret (MS) mit 2 symmetrischen SitzungsSchlüsseln, 2 MAC-Schlüsseln, 2 Initialisierungsvektoren für CBC
→ Record-Protokoll (Datenaustausch): Fragmentierung, Kompression, Hinzufügen von MAC, Verschlüsselung, Hinzufügen eines Headers
→ Verbindung schließen über spezielles Header-Feld (authentifiziert durch MAC)

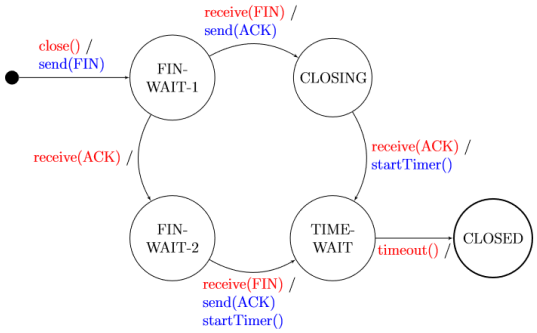
Flusskontrolle:
→ Empfänger wird geschützt, Sender wird kontrolliert
Prinzip der Flusskontrolle:
→ Empfänger besitzt Puffer, IP fügt neue empfangene Daten ein, die Anwendung liest Daten aus
→ der jeweils freie Pufferplatz wird dem Sender mitgeteilt
→ Sender besitzt Puffer, inn den die Anwendung neue Daten scheibt und mit IP soviel Daten entfernt werden, wie es der Puffer der Empfangsseite zulässt
→ die Anwendung des Senders blockiert, wenn Puffer voll
→ dadurch reguliert die Empfängeranwendung die Sendeanwendung

Überlastkontrolle:
→ Netzwerk wird geschützt, Sender wird kontrolliert
→ dynamische Festergröße wird versucht zu maximieren
→ CW = Congestion Window
→ MSS = Maximum Segment Size
→ Threshhold wird immer abgerundet
Slow Start:
⇒ Threshold ist am Anfang unendlich
⇒ setze CongestionWindow = MSS
⇒ nach Erhalt eines ACKs: CongestionWindow += MSS
⇒ bis Threshold erreicht ist, dann Additive Increase
nach 3 doppelten ACKs:
⇒ Multiplicative Decrease:
⇒Threshold = CongestionWindow/2
⇒CongestionWindow /= 2
⇒ Additive Increase: bei Erhalt eines ACKs: CongestionWindow += MSS x (MSS/CongestionWindow)
nach Timeout:
⇒ Threshold = CongestionWindow/2
⇒ CongestionWindow = 1 MSS
→ Durchsatz: im Mittel 3/4 w/RTT



TCP-Diagramm Aufgabe:
→ SYN, FIN hat 1 Byte imaginäre Größe
→ CW wird bei timeout zu 1, ansonsten halbiert
→ CW ab Threshold: CW *= MSS * (MSS / CW)
Handshake:
→ die ersten 3 Pfeile
→ ACK = Seq + 1, bei Pfeil 2 und 3
→ Bei drittem Pfeil dürfen Daten versendet werden
Allgemein:
→ ACK wird immer um erhaltene Daten erhöht
→ Seq wird immer auf zuletzt erhaltenes ACK gesetzt

TCP-Programmierung




```
public class TCPFIN extends Base{
 int state = -1;
 void close() throws IOE{
  if(state != -1) throw new IOE();
  send(FIN); state = 0;
 }
 void receive(int flag) throws IOE{
  if(state == 0){
   if(flag == FIN){
    send(ACK); state = 1;
   }else{ state = 2;}
  }else if(state == 1 && flag == ACK){
   startTimer();
  }else if(state == 2 && flag == FIN){
   send(ACK); startTimer(); state = 3;
  }else{ throw new IOE();}
 }
 void timeout() throws WSE{
  if(state == 3) state = 4;
  else throw new WSE();
}}}}
```

TCP-Latenzzeiten

TCP - Leistungsanalyse:
→ S : MSS in Bits
→ O : Objektgröße in Bits
→ R : Bitrate
→ W : Fenstergröße in MSSs
⇒ Handshake: 2 RTT
⇒ Objektübertragung: O / R

$$Verzoegerung = 2RTT + \frac{O}{R} + P[RTT + \frac{S}{R}] - (2^P - 1)\frac{S}{R}$$

Gesucht: Fester Q, bis zu dem Wartezeiten auftreten
→ Gegeben: unendlich großes Objekt O
⇒Slowstart mit * n ⇒ log_n

$$Q = \lfloor log_2(1 + [RTT/(L/R)]) \rfloor + 1$$

Gesucht: Anzahl Fenster bis Objekt übertragen:
⇒Slowstart mit *n ⇒ log_n und $O * n - 1$

$$K = \lceil log_2(O/S + 1) \rceil$$

Gesucht Anzahl Wartezeiten:

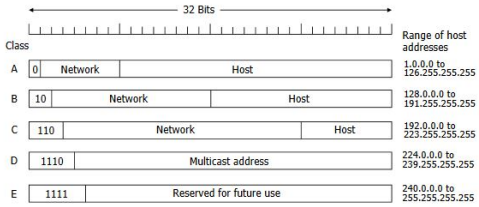
$$P = min\{Q, K - 1\}$$

Netzwerkschicht

Netzwerkschicht steuert den Austausch von Datenpaketen, da diese nicht direkt an das Ziel vermittelt werden können und deshalb mit Zwischenzielen versehen werden müssen.

IP - Datagrammformat:
→ Version (ver): 4 Bits (4 für IPv4)
→ Header Length (HL): 4 Bits
→ Differentiated Services (DS): 6 Bits
→ Explicit Congestion Notification: 2 Bits
→ length: Gesamtlänge in Bytes
→ identifier, flgs, fragment offset: für Fragmentierung
→ time to live: Hoplimit, jeder Router dekrementiert
→ upper layer: höheres Protokoll
→ Options: z.b. Zeitstempel, Weg aufzeichnen
→ ohne Options 20 Bytes

IP - Klassenbasierte Adressierung (IPv4):
→ kennzeichnet Schnittstelle von Host bzw. Router
→ mehreren Schnittstellen benötigen mehrere IP-Adressen
→ 32 Bits, 4 Bytes, Netzwerkteil und Hostteil
→ Kontrolle durch Internet Corporation for Assigned Names and Numbers
→ Dottet-Decimal-Schreibweise:
→ Bsp: $10000000.10000111_2 = 128.135$
→ Aufteilung der standardmäßigen IPv4-Adressen in die Netzwerk- und Hosträume:



IP - Weiterleitungstabelle:
Router:
→ Ziel: Alle Leitungsadressen unter Router
⇒default: Nächst höherer Router
⇒kein default, wenn höchster Router
→ nächster Hop: erster Router auf Weg zu Leitung
→ Flag: G, falls Router im Weg
→ Schnittstelle: Routerausgangsadresse
Hosts:
→ Äquivalent zu Router, Schnittstelle ist immer eth0

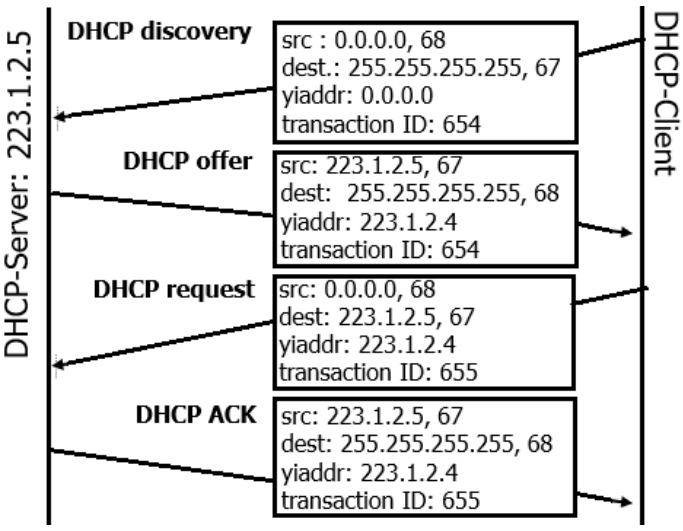
IP - Vorteile/Nachteile:
Vorteile:
→ selbstidentifizierende Adressen: an den ersten Bits wird erkannt, um welche Klasse es sich handelt
→ Weiterleitungstabelle benötigt nur Netzwerkteil der Adresse und kann klein gehalten werden
Nachteile:
→ Feste Netzwerkzuordnung: Rechner benötigen für Netzwerkwechsel eine Anpassung der IP-Adresse
→ C-Netz: wenige Hosts(8Bits), B-Netz: viele Hosts(16Bits)
→ Verschwendung: größere Organisationen bemühen sich um B-Netz und nutzen Adressen nur teilweise
→ 2011: Ausgabe letzter IPv4-Adressen durch ICANN

IP - ICMP(Internet Control Message Protocol)
→ Kontrollnachrichten von Routern an andere Teilnehmer
Format mit 32 bit Länge:
→ Type - Code - Checksum
→ Identifier - Sequence Number
→ Data ⇒ werden in IP-Datagrammen befördert

IP - Subnetze und klassenlose Adressierung:
Allgemein:
→ gleiches Subnetz: kein Router zur Kommunikation
→ Hostanteil wird weiter unterteilt in Subnetz und Host
→ Topologisch: Zusammenhängender Bereich ohne Router
Teilnehmeranzahl im Subnetz:
→ Nur 0er ist der eigene Host
→ Nur 1er bedeutet alle anderen ansprechen
⇒Schritt 1: x = Bit Hostanteil
⇒Schritt 2: $y = 2^x$
⇒Schritt 3: Anzahl möglicher Hosts = y-2
Berechnung der Subnetzadresse:
→ Gegeben: 32 bit Adresse a.b.c.d / n
⇒n gibt die Anzahl der wichtigen Bits von links an
⇒alle Bits hinter n werden zu 0
⇒168.128.20.4 / 20 ⇒ 168.128.16.0
Subnetzmaske:
→ Gegeben: 32 bit Adresse a.b.c.d / n
⇒die ersten n Bits werden zu 1, Rest wird zu 0
⇒168.128.20.176 / 25 ⇒ 255.255.255.128
⇒Ergebnis kommt meist so in Weiterleitungstabelle
CIDR (Classless Inter-Domain Routing):
→ neben Subnetting auch Supernetting (Zusammenfassen von aufeinanderfolgenden klassenbasierten Netzen)
⇒Die starre Zuordnung klassenbasierter Adressen ist aufgelöst ⇒ Größere Tabellen.
⇒Router führen Longest-Prefix-Match durch: Auswahl der Adressen, bei denen die meisten Bits übereinstimmen

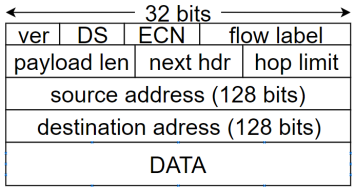
IP - Fragmentierung:
Wenn Verbindungen unterwegs eine kleinere MTU (Maximum Transmission Unit) erfodern: Datagramm wird in Fragmente zerlegt und einzeln weitergeleitet
→ IP-Header indentifiziert zusammengehörige Fragmente
→ Reassemblierung: Fragmente im Ziel zusammensetzen

Autokonfiguration mit DHCP:
→ Client-Server-Protokoll zur automatischen Einbindung der Netzwerkschnittstelle in ein bestehendes Netzwerk
⇒Notwendige Informationen wie IP-Adresse, Netzmaske, Gateway, DNS werden automatisch vergeben
→ Bei autokonfig wird am DHCO-Server ein Bereich von IP-Adressen definiert:



Network Adress Translation (NAT):
→ Umgehung der Adressknappheit
→ Netzwerk verwendet intern global ungültige Adressen
→ Netzwerk besitzt global gültige Adresse
→ Größe durch Anzahl von Portnummern begrenzt
Nachteile:
⇒ Verletzung des Schichtenprinzips
⇒ Eingriff in die Ende-zu-Ende-Verbindung
⇒ Hosts hinter NAT können nicht als Server auftreten
⇒ IPv6 wäre besser, ist aber weniger verbreitet
Vorteile:
⇒ interne Änderungen ohne externe Auswirkung
⇒ bessere Abschirmung
Ablauf:
→ 1. Host sendet Datagramm an Server
→ 2. Router ändert Quelladresse/Port/Nat-Tabellen-Eintrag
→ 3. Antwort mit Zieladresse und Port
→ 4. Router überschreibt Zieladresse anhand Tabelle
⇒ Nat-Tabelle: Speichert zu Request IP die IP des Internen Hosts. Somit sind mehrere Interne Hosts unterscheidbar

IPv6 (IPv5 war experimentell):
Gelöste Probleme:
⇒ Fester Headerlänge (schnelles Weiterleiten)
⇒ keine Fragmentierung
⇒ keine Prüfsumme (Fehlerkennung in höheren Schichten)
⇒ zustandslose Autokonfiguration
⇒ Informationssicherheit
Header Konzept:
→ Jede Aufgabe hat „Extension Header“ fester Größe
→ Schneller, da jedes System nur benötigten Header sieht
Übergang von IPv4 zu IPv6:
→ Dual Stack: Endsystem mit IPv6 und IPv4
→ Tunneling: IPv6-Data wird in IPv4-Data eingebettet
→ NAT: privat IPv6 öffentlich IPv4
Datagrammformat:



IPsec
→ Sicherheit auf Netzwerkschicht
→ bietet Authenzität, Datenintegrität, Vertraulichkeit
→ ermöglicht Virtuelle Private Netze (VPN)
Varianten:
→ Authentication Header (AH) Protokoll
→ Encapsulation Security Payload (ESP) Protokoll
→ Transport Mode und Tunnel Mode
Erforderliche Security Associations:
→ Security Association (SA)
→ Security Parameter Index (SPI)
→ Security Association Database (SAD)
→ Security Policy Database (SPD)
Schlüsselaustausch:
→ Internet Key Exchange Version 2 (IKEv2) Protokoll
→ beide Seiten besitzen Zertifikate für öffentliche Schlüssel
→ 1 Phase:
⇒Deffie-Hellman-Verfahren Schlüsselerzeugung
⇒Aufbau bidirektionale IKE SA
⇒Master-Secrest austausch
→ 2. Phase:
⇒benötigt nur symmetrische Verschlüsselung
⇒Algorithmen und Parameter für einzelne IPsec SAs
→ die Schlüssel werden mit Hilfe des Master-Secrets erzeugt

Netzvirtualisierung:
Private Netze:
→ Eigenes in sich geschlossenes Netz
→ Vorteile: Auslastung wählbar, hohe Sicherheit
→ Nachteile: hohe Kosten
Virtual Private Networks (VPNs)
→ Aufgesetzt auf öffentlichem Netz OVerlay
→ Erscheint wie ein privates Netz
→ Varianten:
⇒End-to-Site: virtuelle Integration Host in anderes Netz
⇒Verbindung von Netzen
⇒Verbindung von Hosts zu einem virtuellen Netz

VPN mittels IP-Tunneling
Tunneling: Einbettung von Paketen eines Protokolls in Pakete eines Protokolls der gleichen oder einer höheren Schicht
TSL-VPN (z.B.OpenVPN):
→ A baut Verbindung zu B mit TLS
→ Tunneling von DHCP durch TLS
→ Tunneling von IP-Paketen durch TLS
→ weitere Konfigurationsmöglichkeiten
IPsec-VPN:
→ A baut Security Association (SA) zu B auf
→ Tunneling von IP-Paketen durch ESP
→ weitere Konfigurationsmöglichkeiten

Multiprotocol Label Switching (MPLS):
→ virtuelle Leitungsvermittlung
→ schnelleres Weiterleiten mit Labeln statt mit IP-Adressen
→ Traffic Engineering
→ Dienstgüteeigenschaften (QoS) für LSPs möglich

Software-Defined Networking (SDN):
→ Trennung von Data-, Controle-, Application-Plane
→ Protokoll zwischen Control und Data Plane
→ Einsatz von preiswerter HW auf Data Plane
→ Vermeidet Protokollsuppe
→ flexibel programmierbare Netze
→ Offene Schnittstellen / Open Source in Netzen
→ schnelle Reaktion auf Netzsituationen

IP-Netzwerk:
→ Hubs gehören nicht zu einem Subnetz
→ Router gehören zu einem Subnetz
→ Rahmen kollidieren nur im gleichen Subnetz
⇒Subnetz bildet somit Kollisionsdomaine
→ Routing-Tabelle: Leitungsadresse = Subnetzadresse

Aufbau eines Routers:
Funktionen am Eingagnsport (EP):
→ Pufferung bzw. Paketverlust (Pufferüberlauf)
→ verteiltes Weiterleiten
→ effiziente Datenstrukturen für schnelle Suche
→ inhaltsadressierbarer Speicher
Switching Fabric (SF):
→ Speicher: früher: Router war einfacher Rechner
⇒heute: Direct Memory Access durch Eingangsport
→ Bus: verbindet alle Ports
⇒üblich für kleine bis mittlere Router
→ Verbindungsnetzwerk:
⇒Corssbar: Jeder Port mit jedem verbunden
⇒ermöglicht nebenläufige Weiterleitung
Funktionen des Ausgangsports (AP):
→ Pufferung (SF schneller als Leitung)
→ Paketverlust (Pufferüberlauf)
→ Active Queue Management: Entscheidung, wann Pakete verworfen werden, z.B. mit Random Early Detect (RED)
→ Scheduling: Entscheidet welches Paket als nächstes(FIFO)
Bemerkung:
Head-of-the-Line-Blocking: SF schneller als Ports und mehrere EP wollen auf gleichen AP, dann müssen manche Warten, die dahinter werden blockiert, trotz freiem AP

Routing Allgemein:
Routerverfahren zur Bestimmung der Paketwege
Unicast-Routing (Punkt-zu-Punkt):
proaktiv: Information über Netztopologie wird ausgetauscht, aktuell gehalten. Graphalgorithmen bestimmen Pfade zu allen Zielen. Bei Sendewunsch werden diese genutzt.
Multicast-Routing (Punkt-zu-Mehrpunkt):
→ Router und Links sollen effizient genutzt werden
→ Erweiterung von Unicast-Routing ⇒ proaktiv
Ad-Hoc-Routing:
→ dynamische Netztopologie: Pfade veralten schnell
→ erweiterung von proaktiv Verfahren
→ auch reaktiv: erst bei Sendewunsch Pfadbestimmung
Datenzentrische Verfahren:
→ adresslos, Nachrichtenweiterleitung wg. Inhalt

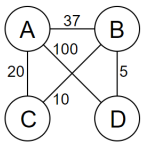
Interdomain-Routing:
→ innerhalb Routingdomäne: Intradomain Routing
→ zwischen Routingdomänen: Interdomain Routing
→ Routingdomänen: autonome Systeme (AS):
⇒Stub AS: nur Verbindungen zu anderen AS
⇒Multihomed AS: mehrere Verbindungen zu anderen AS
⇒Transit AS: Multihomed AS mit Durchgangsverkehr
⇒Transit AS hat zentral vergebene AS-Nummer(16 Bits)
Pfadbasiertes Routing:
→ Austausch von Pfaden ohne Kostenberücksichtigung
→ Router gibt nur zu benutzende Pfade bekannt
→ Zyklenerkennung und Pfade können annulliert werden
Pfad besteht aus:
⇒ Liste von Netzwerken in einem erreichbaren AS
⇒ Sequenz von AS-Nummern zu diesem AS
⇒ IP-Adresse des sendenden Gateways

Routing-Vergleich:
Link-State-Routing:
→ zentrales Verfahen mit beschränkter Skalierbarkeit
→ Komplexität Dijekstra: $O(n^2)$
→ effiziente Implementierung: $O(nlogn)$
→ Nachrichtenaustausch: $O(ne)$ mit e= Kanten
Distanzvektor-Routing:
→ verteilter Algorithmus mit beschränkter Skalierbarkeit
→ Konvergenzprobleme bei Zyklen
→ Fehlfunktion eines Routers wirkt sich auf andere aus

Link-State-Routing:
→ Knoten kennen komplettes Netzwerk (dank Fluten)
→ Knoten berechnet kürzesten Pfad zu anderen Knoten
→ Änderung des Netzwerks erfordert Neuberechnung
Fluten:
→ Link-State-Advertisements mit:
⇒Kennung des Knotens, der LSA erzeugt
⇒Kosten zu Nachbarn und dessen Kennung
⇒Sequenznummer und Lebensdauer
→ Jeder Knoten erzeugt LSAs
→ LSAs werden weitergeleitet an alle Nachbarn
→ Zuverlässigkeit: Bestätigung und Sendewiederholung

Dijekstra-Tabelle:

Schritt	V`	A	B	D
0	C	20,C	10,C	inf,-
1	C,B	20,C	"	15,B
2	C,B,D	20,C	"	"
3	C,B,D,A	"	"	"

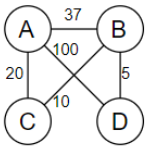


Forwarding-Search-Tabelle

Schritt	bestätigte Liste	vorläufige Liste
0	(C,0,-)	
1	(C,0,-)	(A,20,A), (B,10,B)
2	(C,0,-), (B,10,B)	(A,20,A), (D,15,B)
3	(C,0,-), (B,10,B), (D,15,B)	(A,20,A)
4	(C,0,-), (B,10,B), (D,15,B), (A,20,A)	

Aufbau der Tripel: (Knoten; Kosten; Erster Knoten über den man laufen muss, um zum Knoten zu gelangen)

Distanzvektor-Routing:
→ Wenn kein Weg: Distanz = infty, nH = -
→ Algorithmus endet nach zwei "gleichen" TAbellen
Count-to-infinity-Problem:
→ veraltete Routing-Tabellen enthält zyklischen Pfad
→ Iterationsende: Kosten des alternativen Pfads erreicht
Konvergenz:
→ Zeitpunkt, ab dem die Tabelle sich nicht mehr ändert



Schritt 2: Kürzester Weg, wie B die anderen über höchstens 2 Schritte erreichen kann

von B zu	D	nH	versendet
A	30	C	x
B	0	-	
C	10	C	
D	5	D	

Schritt 1: Kürzester Weg, wie B die anderen Knoten über einen Schritt erreichen kann

von B zu	D	nH	versendet
A	37	A	x
B	0	-	
C	10	C	x
D	5	D	x

Schritt 3: Kürzester Weg, wie B die anderen über höchstens 3 Schritte erreichen kann

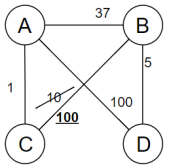
von B zu	D	nH	versendet
A	30	C	
B	0	-	
C	10	C	
D	5	D	

Peer-to-Peer Routing:
Peer to Peer Finger Table:

$$succ(p + 2^{(i-1)} \% m)$$

→ p = Aktueller Knoten / i = Zeilennummer
→ m =max. Knotenanzahl / succ =nächst größerer Knoten
Peer to Peer Lookup:
→ Solange weitergehen, bis man Knoten übersprungen hat
→ Weitergehen, indem man im Table den Eintrag sucht, welcher am nächsten unter dem zu Suchenden Peer ist

Poisoned-Reverse:
→ Vermeidung von Zyklen der Länge 2
→ Lösung des Count to infinity Problems
→ Tipp: Nur Knoten mit neuer Kante ändert Tabelle



Zeit	... zu D	cost	nh
t0	A	16	C
t1	C	17	A
t2	A	18	C
...	...	...	...
tn-2	A	42	C
tn-1	C	43	A
tn	A	42	B

Ohne Poisoned Reverse.

Zeit	... zu D	cost	nh
t0	A	16	C
t1	C	infty	A
t2	A	42	B

mit Poisoned Reverse

Sicherungsschicht

Datensicherung:
Fehlercharakteristika:
→ thermisches Rauschen, elektromagnetische Einstrahlung
→ Bitfehlerwahrscheinlichkeit: 10^{-3}(Funk)/10^{-12}(Glasfaser)
→ oft treten Fehler als Bursts auf
Ansätze zur Datensicherung:
→ Fehlererkennung und Fehlerkorrektur
→ Zyklische Redundanzprüfung:
⇒Nutzdaten D mit d Bits, Prüfdaten R mit r Bits
⇒Senden von (D,R)
⇒Generatorpolynom G, r+1 Bits
⇒Sender wählt R so, dass $G|(D,R)$
⇒R ist Rest bei $D \cdot 2^r/G$
⇒(D,R) entspricht $D \cdot 2^r + R$ und ist durch G teilbar
⇒Empfänger teilt (D,R) durch G, kein Fehler falls Rest=0

Medienzugriff:
→ Beispiel: CDMA, ALOHA
Punkt zu Punkt Verbindung:
→ nur die zwei Endpunkte greifen auf das Medium zu
→ keine komplizierte Koordination notwenig
Medien mit Mehrfachzugriff:
→ gemeinsamer Bus bzw. Internetzugang über Kabel
→ gemeinsamer Funktkanale
→ benötigt verteile Koordination des Medienzugriffs
Möglichkeiten für den Mehrfachzugriff:
→ feste Kanalaufteilung
→ Zufallszugriffverfahren
→ zyklische Zuteilung

CSMA / CA
CA minimiert die Wschlkeit für Kollision und sorgt für eine effiziente Nutzung des Mediums CA prüft vorm Senden, ob Medium benutzt wird, wenn ja, dann warte bis IDLE

CSMA / CD
CD terminiert Übertragung sobald Kollision

Medienzugriff

CSMA/CD mindest Framegröße:
→ R = Bitrate / l = Länge / v = Ausbreitungsgeschw.

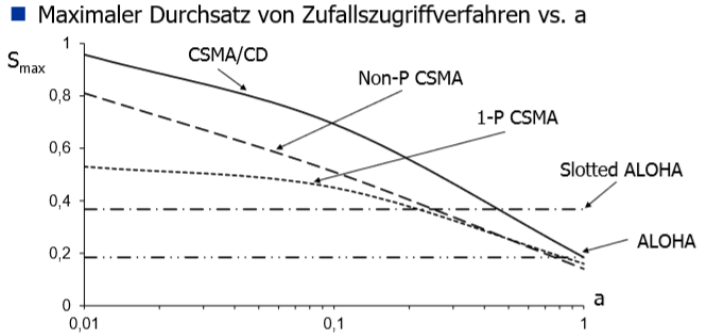
$$L/R > 2D \Rightarrow L > D \cdot R \Rightarrow L > R \cdot 2 \cdot (l/v)$$

CSMA/CD maximaler Durchsatz:
→ R = Bitrate / l = Länge / v = Ausbreitungsgeschw.
→ D = l / v und L = Framegröße

$$1/(1 + 4,4 \cdot (RD/L))$$

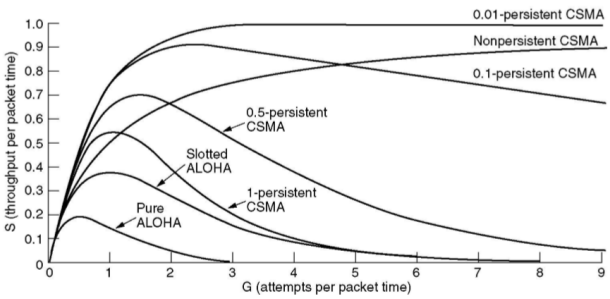
Welches Zugriffverfahren:
→ wenn $a > 0.5$ Slotted Aloha, sonst CSMA/CD

$$a = RD/L$$

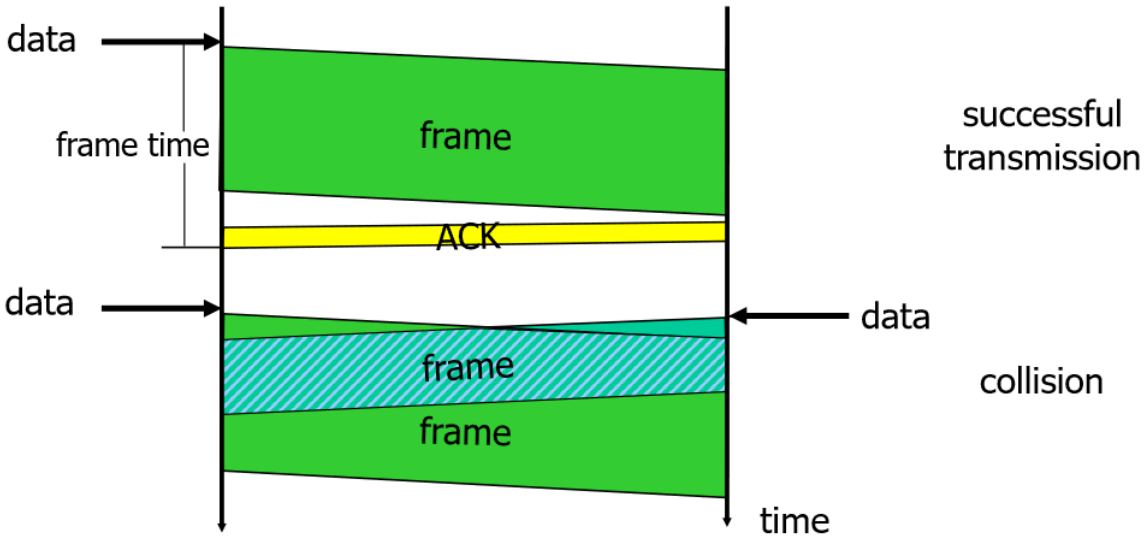


Zufallszugriffverfahren:
→ Bei schwacher Last sinnvoll
→ Wenn Knoten gleichzeitig Senden: Probleme
⇒Kollision von Signalen zerstört diese
⇒Problemlösung: Sendewiederholung
→ Verfahren zur Vermeidung von Kollisionen:
⇒ALOHA, slotted ALOHA
⇒Carrier Sense Multiple Access (CSMA)
⇒Collision Detection
⇒Collision Avoidance
ALOHA Verfahren:
→ MAC-Schicht eines Knoten erhält Datagramm
⇒Rahmen wird sofort gesendet
→ Empfänger erhält Rahmen fehlerlos
⇒Antwort mit positiver Bestätigung (ACK)
→ Wenn Timout ohne ACK
⇒ Sender wiederholt Senden nach random Zeit (Backoff)
ALOHA Durchsatz:
→ ein Knoten/ random Slot/ kollisionsfrei:⇒ $p(1-p)^{2(N-1)}$
→ Random Knoten/ Slot/ Kollisionsfrei: ⇒ $Np(1-p)^{2(N-1)}$
→ Maximal 18% Durchsatz können erreicht werden
Slotted ALOHA:
→ alle Knoten synchronisieren ihre Slots
→ Sendebeginn nur zu Beginn eines Slots
→ Kollisionsintervall verkürzt sich auf einen Slot
→ Random Knoten / Slot / Kollisionsfrei: ⇒ $Np(1-p)^{N-1}$
Carrier Sense Multiple Access (CSMA):
→ Knoten prüfen vor dem Senden, ob Medium belegt
→ reduziert Kollision
→ Bedingung: 2 Ausbreitungsverzögerung<Rahmensendezeit
→ Kollision immer noch möglich
Verfahren:
→ 1. MAC-Schicht eines Knotens von der Netzwerkschicht erhält ein Datagramm: Überprüft Medium; wenn es frei ist, wird der Rahmen gesendet, ansonsten gewartet
→ 2. Empfänger erhält fehlerlos: ACK zurück
→ 3. Timeout ohne ACK: Sender wartet Backoff und sendet erneut
Varianten:
→ 1-persistent:
⇒Wenn Medium belegt ist:
⇒Knoten wartet bis frei und sendet sofort
⇒geringe Wartezeit aber mögliche Kollisionen
⇒wenn z.b. mehrere Knoten warten
→ nicht persistent:
⇒Medium belegt: Knoten geht in Backoff
⇒weniger Kollisions aber längere Wartezeit
→ p-persistent:
⇒Wenn Medium frei wird:
⇒Knoten sendet mit Wahrscheinlichkeit p
Kollisionserkennung:
→ Knoten besitzen HW zur Kollisionserkennung
⇒Danach wird senden unterbrochen
⇒Jamming Signal wird gesender
⇒Andere Knoten erkennen Kollision

Durchsatz für verschiedene CSMA-Varianten



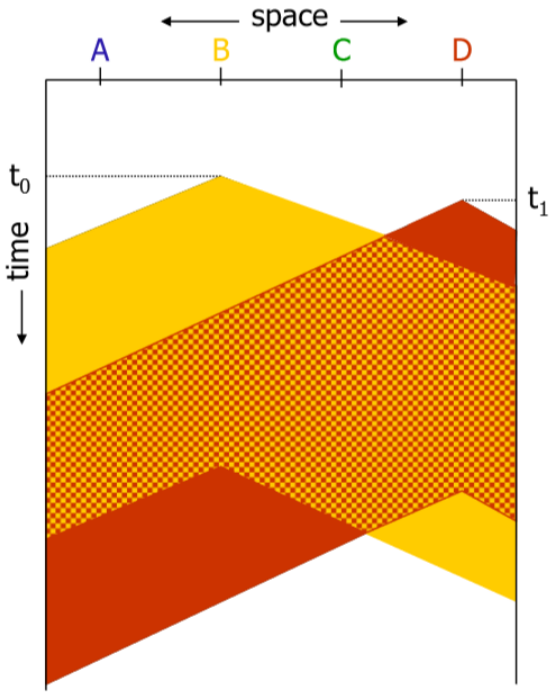
ALOHA Beispiel:



CSMA Beispiel:

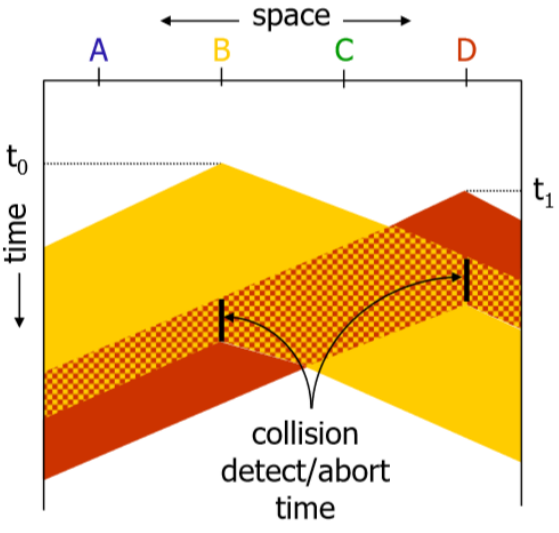
Medienzugriff

- Carrier Sense Multiple Access (CSMA)
  - Knoten prüfen vor dem Senden, ob Medium belegt (listen before talking)
  - reduziert Kollisionen
  - Voraussetzung: Ausbreitungsverzögerung < Rahmensendezeit (sonst ist die Information des belegten Kanals veraltet und das Verfahren sinnlos)
  - Kollisionen immer noch möglich: wenn anderer Knoten startet, bevor sich das Signal auf dem Medium zu ihm ausgebreitet hat



Medienzugriff

- Kollisionserkennung, CSMA/CD
  - Knoten besitzen HW, um während des Sendens Kollision zu erkennen (listen while talking)
  - nach Kollisionserkennung wird Senden abgebrochen (weniger Verschwendung), ein Jamming-Signal wird gesendet, damit alle Knoten Kollision sicher erkennen
  - keine ACKs
  - kombinierbar mit allen CSMA-Varianten



Feste Kanalaufteilung:
→ Anzahl Nutzer: MaxDatarate / MaxDatarateuser
→ Realisierung durch Multiplexverfahren
⇒Frequenz-/Zeit-/Code-multiplex
→ Nachteil für Datenkommunikation
⇒Daten werden sporadisch versendet
→ Effizienz fester Kanalaufteilung:
⇒Datenkommunikation ist bursty
⇒feste Kanalaufteilung ist ineffizient

Zyklische Zuteilung:
Polling:
→ Sendeerlaubnis wird den Knoten sukzessive zugewiesen
⇒durch zentralen / random Knoten / verteiltes Protocol
→ Zykluszeit: Zeit bis Sendeerlaubnis zurück bei Knoten
⇒Sendezeit für Sendeerlaubnis + Ausbreitunszeiten
⇒Verarbeitungszeiten + Sendezeit für Daten
→ Nachteile:
⇒Overhead/ zentraler Knoten ist Single Point of Failure
Token:
→ Knoten sind ringförmig vernetzt
→ Knotenadapter hat Ein- und Ausgang
⇒Listen Mode:Eingangsbits werden gepuffert
⇒Transmit Mode: Direkte weiterleitung der Bits
→ Ein Token zirkuliert auf Ring in 2 Zuständen
→ Knoten mit Sendewunsch empfängt freies Token:
⇒Token wird in belegt geändert / Daten werden gesendet
⇒Empfänger erhält die Daten
⇒nach Ringumlauf: Sender entfernt Token ⇒ Token frei

Ethernet:
→ dominierende Technologie für kabelgebundene LANs
→ durchgesetzt gegen Token Ring und anderen
Rahemformat:
→ Präambel (8 Byte) / Quell- und Zieladresse (6 Byte)
→ Type (2 Byte)/ Nutzdaten (46-1500 Byte)/ CRC (4 Byte)
⇒ Gesamtgrößte minimal 64 Byte
⇒Damit L/R > 2D für ursprüngliche Bustopologie gilt
Medienzugriff:
→ 1-persistentes CSMA/CD
→ Jam-Signal 32-48 Bit
→ binär expotentieller Backoff
→ verbindungslos: kein Hanshaking
→ unzuverlässig: kein ACK
Selbstlernen:
→ Switch erhält Rahmen:
⇒Entscheidet wohin Rahmen weitergeleitet wird
⇒physikalische Adresse und Portnummer wird gespeichert
→ Portadresse = Zieladresse Rhamen:
⇒Rahmen wird verworfen
→ Port der physikalischen Adresse unbekannt:
⇒Rahmen wird an alle Ports geflutet
Spanning Tree Protocol:
→ Switches mit zyklischer Struktur:
⇒selbstlernen funktioniert nicht mehr
→ Switches bilden mit Algorithmus einen Baum
→ Root Switch ist Switch mit kleinster MAC-Adresse
→ kürzste Wege zu Rootswitch werden genutzt

Adressierung (MAC / LAN - Adresse):
→ MAC-Adresse muss zur Kommunikation nicht bekannt sein, da man auch einfach Broadcasten kann
Physikalische Adresse:
→ 48 Bits, 6 Bytes, 12 Hedadezimalziffern
→ in ROM des Adapter eingebrannt und von IEEE verwaltet
→ global eindeutig, keine Strukturierung

Netzwerke:
→ Bride teilt Netz in zwei Kollissionsdomänen
→ Kollision bei Paketsendung, wenn Kollisionsdomäne gleich

Netzwerkgeräte:
Hubs:
→ Physikalische Schicht
Router:
→ Netzwerk- , Sicherungs- , Physikalische Schicht
Repeater:
→ Physikalische Schicht
→ Repeater mit vielen Ports, keine Pufferung
Switches:
→ Sicherungs- , Physikalische Schicht
Bridge:
→ Medienzugriffsschicht
→ Verbindung von Ethernet-Segmenten
→ Empfang eines Rahmens an einem Eingangsport:
⇒Entscheidung zu welchem Ausgangsport es kommt

Drahtlose LANs:
Charakteristika von Funkkommunikation:
→ Abnahme der Signalstärke mit Entfernung
→ Interferenz durch andere Sender
→ Mehrwegausbreitung und hohe Fehlerrate
Hidden-Terminal Problem:
→ 3 Kommunikationspartner
→ A hört B, B hört C, A hört C nicht
Architektur eines Infrastrukturnetzes:
→ Stations / Zugangspunkt (Acess Point)
→ Basic service Set / Portal / Distribution System
Medienzugriff gemäß Basic Access:
→ MAC-Schicht einer Station erhält Datagramm:
⇒Medium wird überprüft
⇒freie Zeitdauer wird erwartet
⇒Danach wird Rahmen gesendet
→ Empfänger erhält Datagramm fehlerlos: ⇒ Sendet ACK
→ Timeout ohne ACK:⇒ Sender geht in Backoff
RTS/CTS-Austausch:
→ vorher: Austausch von Reservierungsnachrichten
→ Sender sender Request-To-Send mit Länge des Rahmens
→ Empfänger anwortet mit Clear-To-Send (CTS)
→ Medienzugriff für RTS mit Basic Access
→ Nachteil: größerer Overhead
→ Vorteile:
⇒keine Kollision: Medium ist reserviert
⇒Kollision: nur kurz
⇒Hidden-Terminal-Problem teilweise gelöst

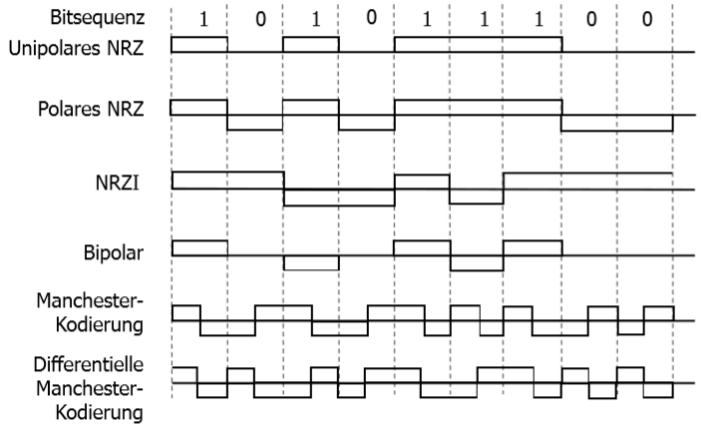
ARP - Adress Resolution Protocol
Ablauf:
→ Knoten A zu B: A kennt nur IP-Adresse von B
→ A sendet ARP-Anfrage als Broadcast
⇒mit physikalischer Adresse von A
⇒und IP-Adresse von B
→ B erkennt sich als Ziel an IP-Adresse in der ARP-Anfrage
⇒B sendet in ARP-Antwort seine physikalische Adresse
⇒ARP-Antwot von B an die physikalische Adresse von A
→ A speichert die Zuordnung der Adressen in ARP-Tabelle
Tabelle eintragen:
→ Betrachtet werden alle Hosts im gleichen Subnetz
→ Eingetragen werden IP-Adresse und MAC-Adresse
Allgemein
→ jeder Knoten besitzt ARP-Tabelle mit Zuordnungen
⇒IP-Adresse / physikalische Adresse / Time to Live

Physikalische Schicht:

Signale:
→ Signalewerte sind kontinuierlich oder diskret
→ Zeitverlauf ist kontinuierlich oder diskret
⇒analog (komplett kontinuierlich)
⇒digital (wert und zeit diskret)

Leitungskodierung binärer Signale:
→ Zuordnung von Signalwerten zu Nullen und Einsen einer Bitsequenz bei der Übertragung im Basisband
→ Basisbandübertragung:
⇒Signale werden direkt auf das Medium gesendet
Eigenschaften:
→ selbsttaktend: Sendetakt aus Signal erhaltbar
→ gleichstromfrei: kein Gleichanteil elektrischer Signale
→ Bandbreitenbedarf: Frequenzbandbreite
Mögliche Fehler:
→ Dämpfung
→ Rauschen
→ Verzögerungsverzerrung

Leitungscodierung:



→ Fehlerhaftes Signal ergibt X als Wert
Manchaster:
→ Manchester betrachtet erste Hälfte des Signals
→ Diff. Manchester invertiert bei 1 vorheriges Signal
→ Diff. Mamchster wiederholt bei 0 vorheriges Signal
→ Übertragungsrate = 1/2 Signalrate
→ Vorteil: Taktrückgewinnung
→ Nachteil: höhere Bandbreite (doppelt so hoch als NRZ)
NRZ:
→ Non-Return-to-Zero
→ ohne Taktrückgewinnung

Maximale Datenrate:
Nyquist-Theorem:
Abtasttheorem: Signal der Bandbreite B kann durch Abtastwerte der Frequenz 2B wiederhergestellt werden, mehr Abtastwerte sind unnötig
Umkehrung: Signal der Bandbreite B kann nur Abtastwerte der Frequenz 2B repräsentiere
Shannon-Theorem:
Theoretisch max. Datenrate nicht erreichbar (Rauschen)
Rauschabstand: Verhältnis von Signal- und Rauschleistung S/N
Für einen Kanal mit Bandbreite B [Hz] und Rauschabstand S/N [dB] ergibt sich:

$$maxDatarate = B \cdot log_2(1 + S/N)[Bit/s]$$

Modulation:
Modulation: Änderung von Parametern (z.B. Amplitude, Frequenz) eines Trägersignals durch ein modulierendes/aufgeprägtes Signal
Demodulation: Rückgewinnung des modulierenden Signals aus dem modulierten Informationsträger
Modem: Modulation und Demodulation in einer Einheit
Analog-Analog-Wandlung (TV-Signal):
→ Amplitudenmodulation
→ Frequenzmodulation
→ Phasenmodulation
Digital-Analog-Wandlung (Daten über Funkkanal):
→ Amplitudentastung
→ Binäre Frequenzumtastung
→ Binäre Phasenumtastung
→ Multiple FSK
→ Differential BPSK
→ Quadrature PSK
→ Multilevel PSK
→ Quadraturamplitudenmodulation
Analog-Digitalumwandlung (Sprache):
→ Pulscodemodulation
⇒Abtastung, Quantisierung, Codierung

Übertragung über elektrische Leiter:
Eigenschaften:
→ Bandbreite / Wellenwiderstand
→ Dämpfung / Nebensprechen
2 Arten:
→ Koaxialkabel und Twisted-Pair-Kabel

Signalübertragung über Lichtwellenleiter:
Eigenschaften:
→ hohe Datenübertragungsrate
→ geringe Dämpfung ⇒ hohe Entfernung
→ kein Nebensprechen, wenig Störung
→ geringe Bitfehlerrate
→ Einsatz überall außer im Anschlussbereich
Prinzip:
→ Sender wandelt Signal mit LED in optisches Signal
→ Empfänger wandelt optisches Signal wieder um

Funkübertragung:
Eigenschaften:
→ Kabellos durch elektromagnetische Wellen
→ Frequenz f, Wellenlänge l, Lichtgeschw. c
→ Zusammenhang c = l * f
Relevante Wellen:
→ Radiowellen f ca. 10kHz - 10 MHz
→ Mirkowellen f ca. 10MHz - 1 GHz
→ Infrarot f ca. $3 * 10^{11}$Hz - $2 * 10^{14}$
Funksysteme:
→ terrestrischer Funk und Satellitenfunk

Aufgabe Harmonische Schwingung:
Gegeben:
→ Periodisches Signal 2 B, Bitrate C [bit/s]
→ Anzahl harmonischer Schwingungen n
Gesucht:
→ Bandbreitenbeschränkung f
Lösung:
→ Normal: Periodisches Signal B = 8 Bit ⇒ 2 B = 16 Bit
→ Frequenformel: f = 1 / T = C / 2B
⇒C einsetzen und ausrechnen
→ Banbreitenbeschränkung: f = C * 20

Rechnungen

Paketverzögerungen:
1. Ausbreitungsverzögerung d_{prop}
2. Übertragungsverzögerung d_{trans}
3. Verarbeitungsverzögerung d_{proc}
   ⇒FehlerPrüfung / Bestimmung AusgangsLink
4. Warteschlangenverzögerung d_{queue}
   ⇒Wartezeit auf den ausgehenden Link zur Übertragung

Rechengrößen:
→ l = Länge
→ v = Ausbreitungsgeschwindigkeit
→ O = Daten
→ R = Datenrate
→ L = Paketgröße

Ausbreitungsverzögerung d_{prop}:

$$d_{prop} = \frac{l}{v}$$

Übertragungsverzögerung d_{trans}:

$$d_{trans} = \frac{L}{R}$$

Übertragungszeit d berechnen:

$$d = \frac{l}{v} + \frac{O}{R}$$

Kanalpuffergröße a:

$$a = v * R$$

Verzögerung an einem Knoten:

$$d_{prop} + d_{trans} + d_{proc} + d_{queue}$$

Sendezeit berechnen:
Gegeben:
→ Datei Größe O=N * L auf Pfad mit E sequentiellen Links
→ Bei Paketvermittlung N Paketen der Größe L
→ Ausbreitung/Warteschlangenverzögerung vernachlässigen
Sendezeit: h Bits header und d sec. Verbindungsaufbau
→ t = d + (N + E - 1) * (L + h)/ R
Sendezeit: Verbindungslos und 2h Header
→ t = d + (N + E - 1) * (L + 2h)/R
Sendezeit: verbindungslos, ohne segmentierung, 2h Header
→ t = (L * N + 2h)*E / R
Sendezeit: leitungsvermittelt,ohne segmentierung, h Header
→ t = d + (N * L + h) / R

Latenzzeitenberechnung:

→ d_{prop} = Ausbreitungsverzögerung
→ d_{trans} = Übertragungsverzögerung

Cut-Through:
→ Vorteil: Sehr schnell
→ Nachteil: Fehler können auftreten
→ Zeit s um ein Paket zu verschicken:

$$s = d_{prop} + d_{trans}$$

Store-And-Forward:
→ Vorteil: keine Fehler
→ Nachteil: Langsam
→ Zeit s um ein Paket zu verschicken (E = Links):

$$s = E \cdot d_{trans} + d_{prop}$$

Übertragungszeit minimieren/maximieren:
Gegeben:
→ Host A schickt Datei Größe O an Host B
→ Zwischen den Host sind E-1 unbelastete Hosts
→ Host a segmentiert in die Größe L und fügt h Bits an
→ Übertragungsrate der Links: R bps
→ Gesucht: L, damit Übertragungsverzögerung minimal
Lösung:
Übertragungszeit insgesamt:

$$U(L) = \frac{h \cdot O}{L \cdot R} + \frac{O}{R} + \frac{E \cdot L}{R} + \frac{E \cdot h}{R} - \frac{L - h}{R}$$

→ Ü insgesamt als Ü(L) betrachten und ableiten
→ Ü'(L) = 0 und nach L auflösen

→ Gesucht: Übertragungszeit maximieren
Lösung:
Ein Weg dauert wegen Store and Forward am längesten, wenn O = L ⇒ $d_{max} = E \cdot O/R$

HTTP-Verzögerungszeiten:

Gegeben:
→ M Grafik Dateien
→ Objekte haben Größe O Bits
→ Link zw. Client und Server mit RTT und Rate R
→ ausreichendes Überlastfenster, Segmentgröße L
→ keine Sendewiederholung

Fragen:
1. Wenn eine Webseite aus genau einem Objekt besteht, dann entsteht durch nichtpersistentes und persistentes HTTP unterschiedliche Antwortzeiten.
falsch, Persistent kann sofort ACK und Request nach erhalt der Basisseite senden, nicht persistent schickt zuerst ACK und SYN, bevor er requestet

2. Betrachten Sie die Übertragung eines Objekts der Größe O von einem Server zum Browser über TCP. Wenn $O > L$, wobei L die maximale Segmentgröße ist, muss der Server mindestens einmal warten.
richtig, da statt $O = L$, $L < O$ gilt

3. Nehmen Sie an, eine Webseite bestünde aus zehn Objekten mit jeweils der Größe O Bits. Für persistentes HTTP ist der RTT-Anteil an der Antwortzeit 20RTT.
falsch, RTT liegt unter 20 RTT

4. Nehmen Sie an, eine Webseite bestünde aus zehn Objekten mit jeweils der Größe O Bits. Für nichtpersistentes HTTP mit 5 parallelen Verbindungen ist der RTT-Anteil an der Antwortzeit 12RTT
falsch, RTT-Anteil der Antwortzeit liegt unter 12 RTT

Formeln:
1. Latenzformel:
→ nicht persistentes HTTP
→ sequentielle Verbindungen

$$delay = (M+2) \cdot (2RTT + \frac{O}{R} + P \cdot (RTT + \frac{L}{R}) - (2^P - 1) \cdot \frac{L}{R})$$

2. Latenzformel:
→ nicht persistentes HTTP
→ parallele Verbindungen
→ x parallele Verbindungen

$$delay = (M+2) \cdot \frac{O}{R} + 2(\frac{M+1}{x} + 1)RTT + P(RTT + \frac{L}{R})$$
$$-(2^P - 1)\frac{L}{R} + \frac{M+1}{x}(P'(RTT + \frac{xL}{R}) - (2^{P'} - 1)\frac{xl}{R})$$

Grundwissen

Größenordnungen:

Name	Byte	Name	Bit
Kilobyte	10^3 Byte	Kibibyte	2^{10}
Megabyte	10^6 Byte	Mebibyte	2^{20}
Gigabyte	10^9 Byte	Gibibyte	2^{30}
Terabyte	10^{12} Byte	Tebibyte	2^{40}
Petabyte	10^{15} Byte	Pebibyte	2^{50}
Exabyte	10^{18} Byte	Exbibyte	2^{60}

Prozentrechnung:
Prozentuale Änderung:
→ Gegeben: Ausgangswert x , Neuer Wert y
→ Frage: Um wie viel Prozent hat sich x verändert?
→ Rechnung: Veränderung = (x/100)*|x-y|

Potenzgesetze
→ $a^m \cdot a^n = a^{m+n}$
→ $(a^n)^m = a^{n \cdot m}$
→ $a^n \cdot b^n = (ab)^n$
→ $a^{\frac{b}{n}} = \sqrt[n]{a^b}$