Computer Netzwerke und verteilte Systeme

Zusammenfassung

Fabian Damken

6. März 2022

TECHNISCHE UNIVERSITÄT DARMSTADT

Inhaltsverzeichnis

1 Einführung

1.1 Properties of Communication

- Ausbreitungsverzögerung
- Datenrate
- Fehlerrate

1.2 Simplex

In einem Simplex Netz können Daten nur in eine Richtung gesendet werden.

1.3 Halb-Duplex

In einem Halb-Duplex Netz können Daten in beide Richtungen gesendet werden, aber nicht zur gleichen Zeit.

1.4 Voll-Duplex

In einem Voll-Duplex Netz können Daten zur gleichen Zeit in beide Richtungen gesendet werden.

1.5 Switching

1.5.1 Verbindungs Switching

Es wird eine Verbindung zwischen zwei Endpunkten hergestellt.

1.5.2 Paket Switching

Es werden Pakete versandt und keine Verbindung hergestellt.

1.6 Multiplexing

Mehrere Sender wollen gleichzeitig Daten über ein Medium senden.

1.6.1 Time Division Multiplexing

Die Sender senden nacheinander.

1.6.2 Frequency Division Multiplexing

Die Sender senden zeitgleich auf unterschiedlichen Frequenzen.

1.6.3 CDM, SDM

Code Division Multiplexing Die Sender senden zeitgleich mit verschiedenen Spreizcodes.

Space Division Multiplexing Die Sender senden zeitgleich über unterschiedliche Medien (räumlich).

1.7 Broadcast Medium

Medien, die nur von einem Sender zur gleichen Zeit genutzt werden können (bspw. Kupferleitungen).

1.8 Netzwerk Typen

1.8.1 Verbindunsorientiert

Die Erste Phase der Kommunikation ist die Herstellung einer Verbindung. Danach sind die Kommunikationspartner verbunden.

1.8.2 Verbindungslos

Es wird keine Verbindung hergestellt und übertragen direkt die Daten.

2 OSI Schichtenmodell

Das OSI-Schichtenmodell beschreibt ein Standard-Schichtenmodell für Netzwerke und besteht aus 7 Schichten, wobei die oberen drei Schichten nur schwach voneinander abgegrenzt sind und oftmals zusammengefasst werden. Im Internet umgesetzt ist oftmals nur das DoD-Schichtenmodell, welches dem OSI-Schichtenmodell in den unteren vier Schichten gleicht und die oberen drei in einer Schicht zusammenfasst.

Das Schichtenmodell besteht aus den folgenden Schichten, die weiter unten auch noch genauer erläutert werden:

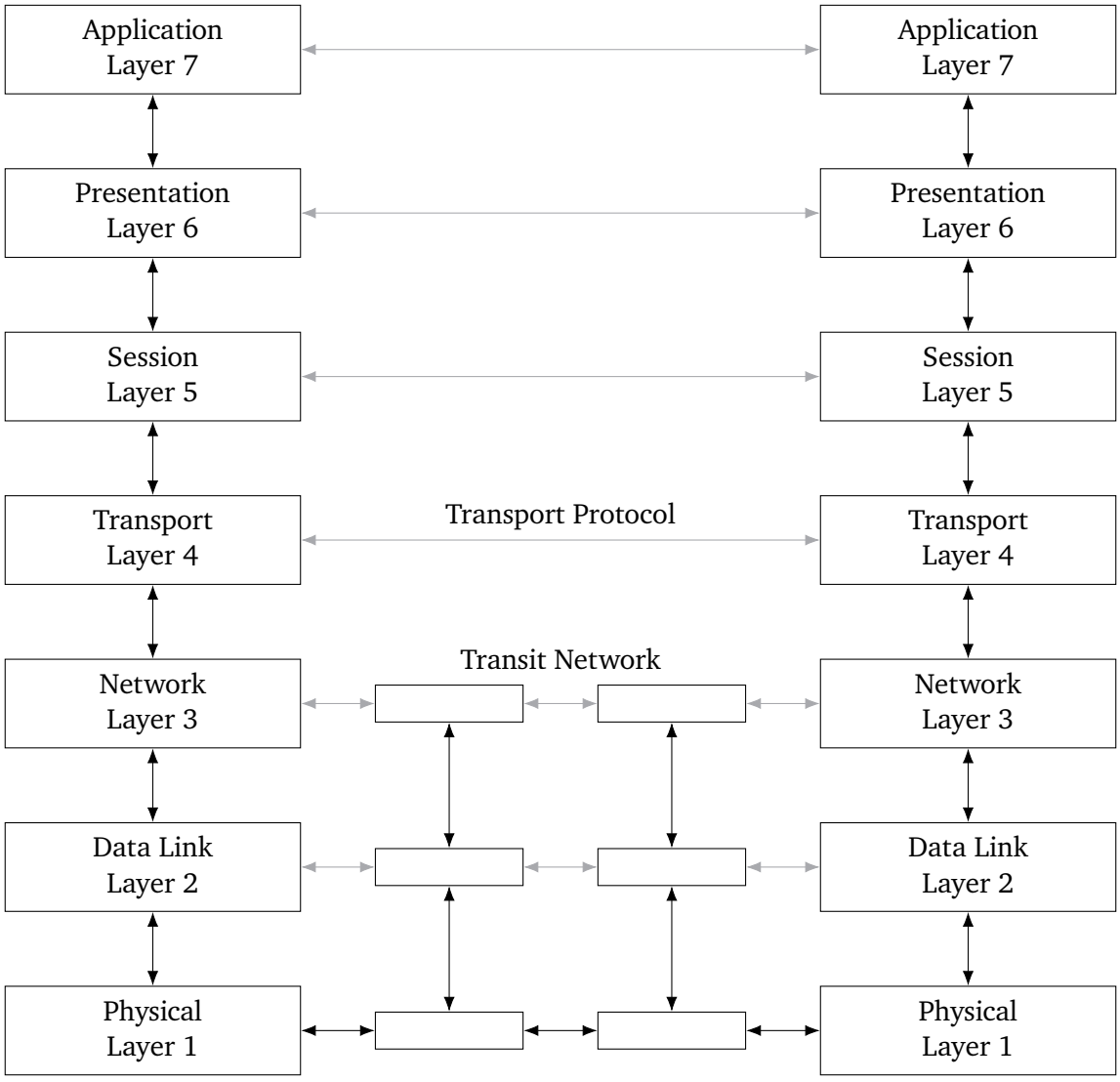


Abbildung 2.1: OSI Schichtenmodell

1. Physical Layer

- Schicht 1
- Fehleranfälliger (unsicherer) Bit-Strom zwischen physikalisch benachbarten Systemen
- Definitionen von Kabeln, Steckern, …
- Protokollbeispiele: RJ-45, RJ-21

2. Data Link Layer

- Schicht 2
- Fehlerbereinigter Strom von Frames zwischen physikalisch benachbarten Systemen
- Pakete sind „Frames“
- Maximale Framegröße des physikalischen Netzes wird beachtet
- Erkennung (und Bereinigung) von Übertragungsfehlern (Prüfsummen)
  Nicht alle Systeme bereinigen Fehler, einige Protokolle geben nur weiter, das Fehler passiert sind.
- *Kann* schon Flow Control enthalten (bspw. Sliding Window)
- Protokollbeispiele: Ethernet, WLAN

3. Network Layer

- Schicht 3
- Übertragung von Paketen zwischen entfernten Systemen
- Übernimmt Routing (Wegfindung und Weiterleitung)
- Congestion Control und Flow Control
- Protokollbeispiele: IP, X.25

4. Transport Layer

- Schicht 4
- Logische Verbindung von Prozessen auf entfernten Systemen
- Multiplexing, Paketaufteilung/-zusammenstellung für Schicht 3/5
- Fehlerkorrektur
- Congestion Control und Flow Control
- Abstrahiert Netzwerkdetails und Qualitätsunterschiede im Netzwerk
- Protokollbeispiele: TCP, UDP

5. Session Layer

- Schicht 5
- Logische Verbindung zwischen Prozessen
- Verbindungswiederherstellung nach Fehlern mit sogenannten Checkpoints zur Vermeidung von vollständigen Neuverbindungen
- Protokollbeispiele: X.215, X.225

6. Presentation Layer

- Schicht 6
- Systemunabhängige Darstellung von Daten
- Enthält Datenkompression und Verschlüsselung
- Gewährleistung der syntaktischen Korrektheit bei Sender und Empfänger (Nutzung von ASN.1)
- Protkollbeispiele: X.216, ISO 9576

7. Application Layer

- Schicht 7
- Anwendungen für Endnutzer
- Die Anwendungen gehören selbst nicht zur Schicht, sondern nur die Datenein/-ausgabe
- Anwendungsbeispiele: Browser, E-Mail Programm

3 Routing

Routing beschäftigt sich mit der Pfadfindung von Start zu Ziel und wie Pakete am schnellsten zugestellt werden können.

- Diese Aufgabe wird auf OSI-Schicht 2 (Network) übernommen, also von IP (o.ä.).
- Zur Pfadfindung werden *Routing Algorithmen* eingesetzt, die den Pfad finden.

3.1 Weiterleitung (Forwarding)

Als *Forwarding* wird die Weiterleitung von Paketen bezeichnet, die sich an die von Routing Algorithmen berechnete Route hält. Anders ausgedrückt: Forwarding bezeichnet das simple verschieben eines Paketes von einem Knoten zum nächsten.

Der Routing Algorithmus erstellt dabei eine Routing Tabelle, in der für jeden Knoten (bzw. Netz) steht, an welchen Knoten die Pakete weitergeleitet werden sollen.

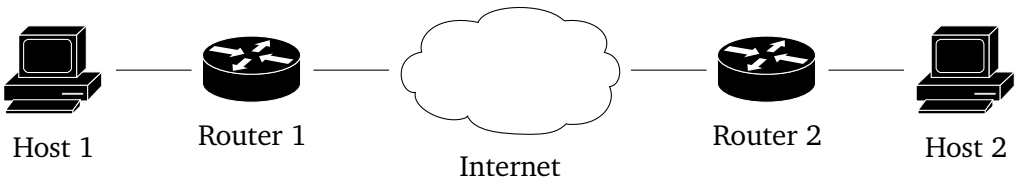


Abbildung 3.1: Routing Einleitung

3.2 Optimierungskriterien

- Korrektheit
- Simplizität
- Robustheit
- Fairness
- Optimalität

Diese Ziele konkurrieren oft miteinander und sind nicht alle zur gleichen Zeit umsetzbar.
Zur Bewertung der Algorithmen können folgende Metriken helfen:

- Durchschnittliche Paketverzögerung
- Gesamte Brandbreite

- Individuelle Verzögerungen

In der Praxis versuchen Routing Algorithmen die Anzahl an Hops pro Paket zu reduzieren. Dies ist nicht immer optimal, aber gut genug.

3.3 Routing Algorithmen

Routing Algorithmen werden klassifiziert in:

Adaptive Routing Algorithmen Der Algorithmus passt sich den aktuellen Netzwerkgegebenheiten an (bspw. DVR, LSR, ...).

Nicht-Adaptive Routing Algorithmen Der Algorithmus passt sich nicht dem Netzwerk an und funktioniert immer gleich (bspw. Flooding).

Adaptive Routing Algorithmen Adaptive Routing Algorithmen werden in folgende Unterkategorien eingeteilt:

Zentralisiert Ein zentraler Knoten steuert das Routing für das gesamte Netzwerk.

- Die Router teilen regelmäßig ihre Routing Tabellen mit dem zentralen Dienst.
- Der zentrale Dienst kann die besten Routen berechnen und wieder mit den Routern teilen.
- Problematiken:
  - Angreifbar: Wenn der zentrale Dienst stirbt, wird das Routing zu einem nicht-adaptivem Routing.
  - Skalierbarkeit: Der Dienst muss sehr viele Informationen bereit halten und viel Traffic behandeln.
  - Flaschenhals: Um das Zentrum herum ist viel Traffic unterwegs.
  - Alterung: Router, die nahe am Zentrum sind bekommen neue Routing Tabellen eher mit als entfernte Router.

Isoliert Jeder Router entscheidet sich selbst für Wege basierend auf den lokalen Informationen. Es findet kein Nachrichtenaustausch zwischen Routern statt.

Verteilt Die Router kommunizieren regelmäßig miteinander, um die Routing Tabellen zu teilen und zu updaten.
Auch diese Art von Routing kann nochmals in unterteilt werden:

- Dezentrale vs. Globale Informationen
  - Bei dezentraler Informationsverwaltung kennt jeder Knoten nur seine direkten Nachbarn und es werden Informationen mit diesen ausgetauscht.
    Diese Algorithmen heißen *Distance Vector* Algorithmen.
  - Bei globaler Informationsverwaltung kennt jeder Knoten alle anderen und die Kosten der Kanten, ...
    Diese Algorithmen heißten *Link State* Algorithmen.

- Statische vs. Dynamisch Routen
  - Bei statischen Routen ändern sich diese nur sehr langsam über die Zeit.
  - Bei dynamischen Routen können sich die Routen schnell ändern (bspw. durch periodische Updates oder als Reaktion auf Änderungen im Netzwerk).

3.3.1 Flooding [Nicht Adaptiv]

- Standard-Algorithmus für nicht-adaptive Algorithmen.
- Die Pakete werden nicht geroutet sondern einfach über das gesamte Netzwerk versendet (es wird geflutet).
- **Problematik:** Es werden sehr viel mehr Pakete versendet als nötig.
- Strategien zur Verringerung der duplizierten Pakete:
  1. Einen Time-To-Life Wert im Header mitführen, der bei jedem Schritt um 1 reduziert wird. Ist der Wert Null, so wird das Paket verworfen.
  2. Router behalten die Sequenznummer der bereits weitergeleiteten Pakete und leiten keine Pakete weiter, die bereits weitergeleitet wurden.
- Flooding ergibt Sinn, wenn:
  - das Netzwerk sich schnell ändert und adaptives Routing zu langsam wäre oder
  - viele oder alle Pakete Multicast Pakete sind und die Pakete somit ohnehin dupliziert werden.

3.3.2 Hot Potato [Nicht Adaptiv]

- Eingehende Pakete werden so schnell wie möglich weiter gesendet.
- **Problematik:** Manche Pakete kommen erst sehr spät oder nie an.
- Allerdings wird Hot Potato Routing dennoch oft eingesetzt (bspw. um Mitglieder im Netzwerk zu erkennen).

3.3.3 Statisches Routing [Nicht Adaptiv]

- Alle Router werden „von Hand“ vorkonfiguriert mit allen möglichen Zielen.
- Dies funktioniert sehr gut in statischen Umgebungen.
- **Problematik:** Im allgemeinen ändern sich Netzwerke regelmäßig, statisches Routing kann hiermit nicht arbeiten.
- **Problematik:** Wenn der Traffic sehr stark schwankt, kann mit nicht-adaptivem Routing keine bessere Route gefunden werden, da sich der Algorithmus nicht anpasst.

3.3.4 Backward Learning Routing [Adaptiv, Isoliert]

- Grundidee
  - Die Quelladresse und ein Hop Counter befinden sich im Header.
  - Anhand dieser Daten lernt der Algorithmus Dinge über das Netzwerk, indem Pakete verarbeitet werden.
- Algorithmus
  1. Initialisiere alle Router mit einer leeren Routing Tabelle.
  2. Starte mit zufälligem Routing (Hot Potato oder Flooding).
  3. Für jedes Paket tue:
     a) Wenn Hop Count = 1
        → Das Paket kommt direkt von einem Nachbarn.
        → Nachbarn werden über ihre Verbindung identifiziert.
     b) Wenn Hop Count > 1
        → Die Quelle ist n Hops entfernt.
  4. Wird ein Paket gefunden, dessen Hop Count in Kombination mit der Quelle kleiner ist als alle vorherigen, wurde eine bessere Route gefunden
     → Passe die Routing Tabelle entsprechend an.

```
1 Initialize all Routers with empty Routing Table;
2 Start with Random Routing (Hot Potato or Flooding);
```

Abbildung 3.2: Backward Learning Routing

3.3.5 Distance Vector Routing [Adaptiv, Verteilt, Dezentrale Info]

- Es werden Nachrichten mit den Nachbarn ausgetauscht und die Informationen aggregiert.
- Iterativ: Das Verfahren läuft so lange, bis keine Verbesserungen mehr auftreten. Dann stoppen die Knoten die Kommunikation (Selbstterminierend).
- Verteilt: Jeder Knoten kommuniziert nur mit seinen direkten Nachbarn und kennt nur deren Zustand.

Struktur der Distanztabelle

- Jeder Knoten hat eine Zeile für jedes mögliche Ziel.
- Jeder Knoten hat eine Spalte für jeden Nachbarn.
- Distanz (in Knoten X, mit Ziel Y über Nachbar Z): $D^X(Y,Z) = c(X,Z) + \min_w D^Z(Y,w)$
- Anschließend kann die Routing Tabelle einfach abgelesen werden, indem in jeder Zeile die Spalte mit dem geringsten Wert gewählt wird.

Algorithmus

- Jede *lokale Iteration* wird verursacht durch:
  - Eine Änderung der Kosten oder
  - Einer Nachricht von einem Nachbarn.
- Jeder Knoten benachrichtigt seine Nachbarn, wenn sich mindestens ein Pfad geändert hat.
  Die Nachbarknoten wiederum benachrichtigen ihre Nachbarknoten nur, wenn sich ihre Tabelle geändert hat.
- **Problematik:** Schlechte Nachrichten wandern langsam durch das Netz und können zum „Count-to-Infinity“-Problem führen (eine Verbindung zählt „bis“ unendlich).

Initialisierung Knoten X:

```
for all neighbours v:
    D^X(*, v) = ∞
    D^X(*, v) = c(X, v)
rof

for all destinations y:
    send min_w D^X(y, w) to each neighbour
rof
```

Abbildung 3.3: DVR: Initialisierung Knoten X

Schleife Knoten X:

```
forever:
    wait (link cost change to neighbour V or update from neighbour V)

    if (c(X, V) changed by d):
        for all destinations y:
            D^X(y, V) = D^X(y, V) + d
    else if (update received from V with destination Y):
        let newval = min_w D^V(Y, w) received from update
        D^X(Y, V) = c(X, V) + newval
    fi

    if (new min_w D^X(Y, w) for any destination Y):
        send min_w D^X(Y, w) to each neighbour
    fi
reverof
```

Abbildung 3.4: DVR: Schleife Knoten X

Poisened Reverse

- Posened Reverse ist eine Lösung des Count-to-Infinity Problems in kleinen Netzen.
- In großen Netzen kann das Problem dennoch auftreten.
- **Funktionsweise:** Erhöhen sich die Link-Kosten, so benachrichtigt ein Knoten die Nachbarn und tut dabei so, als sei er gestoben. Das heißt, er teilt den Nachbarknoten mit, eine Verbindung dauere unendlich lange.

- Da gute Nachrichten sich schnell durch das Netzwerk verbreiten, wird die Unerreichbarkeit sobald wie möglich aufgehoben.

Kommentar: Bei diesem Abschnitt bin ich mir unsicher, ob die Informationen so korrekt sind.

Split Horizon

- Split Horizon ist ähnlich zu Poisened Reverse und versucht, das gleiche Problem zu lösen.
- **Idee:** Geänderte Routen nicht an Nachbarn weitergeben, der das Update ausgelöst hat.
- Hierdurch wird das Count-to-Infinity Problem effektiv behoben.

3.3.6 Link-State Routing [Adaptiv, Verteilt, Globale Info]

- Die Routen werden effizient mittels Dijkstra gefunden.
- Jeder Knoten und jede Kante sind jedem Knoten bekannt.
- Diese Informationen werden mittels Broadcasts im Netzwerk bekannt gegeben.
- Dabei sendet jeder Router die Kosten zu seinen Nachbarn an seine Nachbarn (Flooding).
  Bereits bekannte Pakete oder abgelaufene werden verworfen, die anderen Pakete werden weitergeleitet.
- Sobald alle Pakete bekannt sind, wird die Topologie konstruiert und Dijkstra ausgeführt.

3.3.7 Vergleich DVR $\leftrightarrow$ LSR

- Nachrichtenkomplexität

  DVR Austausch nur zwischen den Nachbarn

  LSR Mit n Knoten und E Kanten: $\mathcal{O}(m \cdot E)$ Nachrichten pro Runde.

- Konvergenzgeschwindigkeit

  DVR Kann Schleifen enthalten, Count-to-Infinity Problem

  LSR $\mathcal{O}(n^2)$ Algorithmus benötigt $\mathcal{O}(n \cdot E)$ Nachrichten, kann Schwingungen enthalten

- Robustheit (Was passiert, wenn ein Router nicht funktioniert?)

  DVR Knoten können inkorrekte Pfad-Kosten verteilen, jede Tabelle wird von jedem anderen Knoten verwendet $\implies$ Der Fehler wird durch das gesamte Netz propagiert.

  LSR Knoten können inkorrekte Link-Kosten verteilen, jeder Knoten hat eine eigene Tabelle $\implies$ Der Fehler wird nicht propagiert.

3.3.8 Hierarchisches Routing

- Bisher wird von flachem Routing ausgegangen.
- In großen Netzwerken skaliert dies allerdings nicht, weshalb hierarchisches Routing eingesetzt werden sollte.
  - Aufgrund der schieren Anzahl an Knoten ist eine Routing Tabelle der Größe nicht möglich, sie würde die Router abschießen.
- Administrativ gesehen ist das Internet ein Netzwerk von Netzwerken.
- Jeder Administrator möchte das Routing in seinem Netz selbst kontrollieren können.

Autonome Systemen (AS)

- Das globale Internet besteht aus vielen *Autonomen Systemen* (AS), welche untereinander Verbunden sind.

  Stub AS Kleine Organisationen, nur ein Link ins Internet

  Multihomed AS Große Organisationen, mehrere Links ins Internet

  Transit AS : Provider
- Jedes AS hat eine eindeutige ID.
- Jedes AS muss eine Route in jedes andere AS kennen.
- Autonome Systeme aggregieren Router in Regionen.
- Router in einem AS laufen mit dem selben Routing Protokoll. In unterschiedliche AS können unterschiedliche Routing Protokolle verwendet werden.
- *Gateway Router* sind spezielle Router in einem AS, die das Routing zwischen den AS steuern.

Intra-AS/Inter-AS

- Intra-AS: Routing innerhalb eines AS und Routing von Paket nach außen (zum Inter-AS)
  - Routing Information Protocol (RIP): Distance Vector
  - Open Shortes Path First (OSPF): Link State
  - Interior Gateway Routing Protocol (IGRP): Distance Vector (Cisco-Proprietär)
- Inter-AS: Routing zwischen verschiedenen AS und Routing von Paketen nach innen (zu Intra-ASs)
  - Border Gateway Protocol (BGP): Path Vector
    Ähnlich wie Distance Vector, nur mit Schleifenvermeidung etc.

3.3.9 Border Gateway Protocol (BGP)

- Der Standard beim Inter-AS Routing
- Erhält die Information, wie welche Subnetze innerhalb eines AS erreicht werden können.
- Propagiert diese an andere AS weiter.
- Baut „gute“ Routen zu Subnetzen basierend auf den vorherigen Informationen.
- Erlaubt Subnetzen, sich anzukündigen („Halle, ich bin hier!“).

Grundlagen

- BGP Peers tauschen Routing Informationen aus über TCP Verbindungen (BGP Sessions).
- Prefixes können bei einer Bekanntmachung zusammengeführt werden.
- Die Prefixe werden den Gateway Routern registriert, welche diese dann im Inter-AS propagieren.

Attribute und Routen

- Wenn ein Prefix propagiert wird, so enthält die Nachricht den Prefix selbst und Attribute, was zusammen eine Route ergibt.
- Die zwei wichtigsten Attribute sind:

  AS-PATH Enthält das AS, durch welches die Nachricht gelaufen ist.

  NEXT-HOP Enthält den nächsten Intra-AS Router, um zum nächsten AS zu kommen (dies können mehrere Links sein).

- Ein Gateway nutzt *Import Policies*, um zu entscheiden, ob eine Route angenommen oder verworfen wird.

Skalierung

- Die steigende Anzahl an AS ist ein Problem.
- Die Anzahl klein zu halten und dafür die Größe der AS zu erhöhen ist aber auch keine Lösung.
- Dies würde zu hohen Routing Zeiten innerhalb von ASen führen.
- Auf großer Sich beeinflusst das Verhalten eines Routers alle anderen.
- Dies kann zu folgenden Problemen führen:
  - Suboptimales Routing
  - Instabilität der Routen
  - Schwingungen innerhalb einer Route

Security

Quellen Untergrabene Router oder Links.

Konsequenzen
- Falsche Routing Informationen
- Täuschung von normalen Routern
- Störung der normalen Routerfunktionen
- Kompromittierung von Routern (bspw. um Traffic mitzulesen)

Konsequenzbereiche Einzelne Knoten bis zum gesamten Internet

Konsequenzdauer Nur während der Attacke bis zu sehr langen Zeitperioden

Hier fehlen einige Inhalte der Vorlesung, siehe Kapitel 2: Routing, Folie 81 bis 85.

3.4 Mobile Routing

Siehe 4.

3.5 Overlay Routing

- Overlay Netzwerke sind sehr populär geworden.
  - Alle P2P-Netzwerke nutzen Overlay Netzwerke.
  - Die meisten Cloud-Speicher nutzen Overlay Netzwerke.
- Ein *Overlay Netzwerk* ist eine virtuelle Netzwerktopologie, die das darunterliegende Netzwerk versteckt (abstrahiert).
- Nachbarn in einem Overlay Netzwerk sind nicht unbedingt Nachbarn im darunterliegenden Netz.
- Im Prinzip funktioniert Routing in einem Overlay Netzwerk nicht anders als in einem normalen IP Netzwerk.
- Doch es müssen einige Metriken beachtet werden, ob das Overlay Netzwerk noch effizient ist:
  - Die Hops im Overlay Netzwerk.
  - Die Hops im Underlay Netzwerk.
  - Der sogenannte *Stretchfaktor*: $(\frac{\text{Hops im Underlay Netzwerk}}{HopsimOverlayNetzwerk})$

4 Mobile Netzwerke

In mobilen Netzen ist oftmals keine Infrastruktur vorhanden, sodass das Routing ein großes Problem wird (es kann nicht einfach alles über einen Knoten geleitet werden, also muss jeder Knoten Pakete weiterleiten können). In diesem Kontext geht es um Ad-Hoc Netzwerke, also nicht um UMTS oder ähnliches, wo eine zentrale Infrastruktur verfügbar ist. Außerdem ändert sich mobile Infrastruktur kontinuierlich, somit somit müssen sich die Routing-Algorithmen schnell an neue Netzwerke anpassen können. Außerdem ändert sich, bedingt durch das Übertragungsmedium der Radiowelle, andauernd die Verbindungsqualität zwischen den Knoten.

Proaktive Verfahren Die Route wird festgestellt, bevor ein Paket versendet wird.

Reaktive Verfahren Die Route wird während der Zustellung des Paketes festgestellt.

4.1 Routing

Traditionelle Algorithmen wie DVR oder LSR funktionieren nicht sehr gut in mobilen Netzwerken, da sie folgende Eigenschaften eines mobilen Netzes nicht gut behandeln:

- Mobile Netze sind sehr *dynamisch*,
- durch andauernde Routing-Nachrichten nimmt der *Stromverbrauch* zu,
- die *Bandbreite* zwischen den Knoten ist stark begrenzt,
- durch sehr unterschiedliche Knoten ist die Bandbreite *asymmetrisch* ($A \to B$ kann schneller sein als $A \leftarrow B$),
- die Verbindung kann durch andere Knoten *gestört* werden und
- mobile Netze haben meist eine hohe *Redundanz*, da ein Knoten mit vielen anderen Verbunden sein kann.

Im folgenden werden folgende mobile Routing-Algorithmen behandelt:

- Destination Sequenced Distance Vector (DSDV), ein proaktiver Algorithmus und
- Dynamic Source Routing (DSR), ein reaktiver Algorithmus.

4.1.1 Destination Sequenced Distance Vector

DSDV ist eine Erweiterung des DVR-Algorithmus um mit mobilen Netzen zu arbeiten (annahmen über die Vermeidung des Count-to-Infinity-Problems funktionieren nicht in mobilen Netzen!).

Folgende Erweiterungen sind möglich, damit DVR in mobilen Netzen funktioniert:

1. Sequenznummern für alle Routing Updates

- Diese sichern eine geordnete Abarbeitung der Updates
- Vermeidet Schleifen und Inkonsistenzen

2. Geringere Update-Frequenz
   - Speicherung der Zeit zwischen *erster* und *bester* Bekanntmachung eines Pfades
   - Sperrung von Updates wenn die gespeicherten Zeiten unsicher wirken

Ist DSDV noch immer proaktiv und leider auch keine wirkliche Verbesserung im Vergleich zu DVR in mobilen Netzen.

4.1.2 Dynamic Source Routing

DSR ist nach folgenden Grundsätzen designt worden:

- Spaltung von Routing in Pfadfindung und Pfadverwaltung
- Vermeidung von periodischen Updates
- Unterstützung von statischen und dynamischen Netzwerken mit ca. 200 Knoten (skalierbar)
- Der Sender findet den Weg eines Paketes

Pfadfindung

- Die Pfadfindung wird nur ausgeführt, wenn es wirklich benötigt wird und noch keine Route bekannt ist.
- Nutzung von Flooding und Broadcasts mit *Zieladresse* und *ID*.
  - Flooding ist im allgemeinen sehr Aufwändig, wird allerdings auch nur selten ausgeführt (bei dem ersten Verbindungsaufbau).
- Wenn ein Knoten ein solches Broadcast-Paket erhält, dann:
  1. Wurde bereits ein Paket mit der gleichen ID empfangen?
     $\Longrightarrow$ Paket verwerfen.
  2. Ist die eigene Adresse gleich der Zieladresse?
     $\Longrightarrow$ Paket zum Sender zurück senden (der Pfad wird im Paket gehalten).
  3. Ansonsten
     $\Longrightarrow$ Eigene Adresse an das Paket anhängen und Broadcasten.
- Am Ende erhält der Sender ein Paket, welches den Pfad zum Ziel enthält.
- Sind bidirektionale Verbindungen nicht garantiert, so muss das Paket mit Flooding zurück geschickt werden.
- **Optimierungen**
  1. Statt nur einer ID zusätzlich einen Counter mitsenden, der in jedem Schritt erhöht wird.
     - Ist der maximale Durchschnitt des Netzwerkes bekannt, kann ein Paket verworfen werden, wenn der Counter größer als der maximale Durchschnitt ist.
  2. Cachen der Pfade von vorbeikommenden Paketen.
     - Im Normalfall würden Knoten nur Pfade lernen, wenn sie Pakete senden.
     - Während dem Weiterleiten anderer Pakete können diese gecachet werden.

Pfadverwaltung

- Die Pfadverwaltung wird nur ausgeführt, wenn ein bestehender Pfad genutzt werden soll.
- Ungenutzte Pfade werden aus der Routing-Tabelle entfernt
- Nach Absenden eines Paketes führt ein Knoten einen der folgenden Schritte zur Instandhaltung aus:
  1. Warten auf 2 ACKs (wenn möglich, beispielsweise im normalen WLAN).
  2. Carrier-Sense: Abwarten, ob andere Knoten das Paket weiterleiten (nicht möglich bei bspw. Richtfunk)
  3. Explizites Anfragen eines ACKs
- Wenn ein Problem aufgetreten ist, wird entweder der Sender darüber informiert oder ein neuer Weg gesucht (Pfadfindung).

4.2 Clustering

- Auch in mobilen Netzen kann hierarchisches Routing sinnvoll sein.
- Hierbei werden benachbarte Knoten in einen Cluster zusammengefasst.
- Die Grundidee ist, in einem „intra“-Cluster proaktives Routing zu verwenden und in einem „inter“-Cluster reaktives Routing zu verwenden.
- Dies hat folgende Vorteile:
  - In einem kleinen Knoten ist es möglich, alle Knoten zu kennen $\Longrightarrow$ Proaktives Routing ist möglich.
  - Bei vielen Clustern gibt es unter diesen Clustern gibt es quasi keine Kommunikation $\Longrightarrow$ Reaktives Routing funktioniert besser.

> **Warning:** Clustering in dieser Form funktioniert nur, wenn die Knoten innerhalb der einzelnen Cluster die Cluster nicht andauernd wechseln.

5 Inter-Networking

Das *Internet Protocol* (IP) befindet sich auf OSI-Schicht 3 (Network) und stellt ein Verbindungsloses Protokoll dar, auf welchem höhere Protokolle wir TCP und UDP aufsetzen. Es beschäftigt sich mit dem Routing der Pakete, Subnetting, NATing und vielem mehr.

5.1 Datagramme

Bei IP wird von sogenannten *Datagrammen* gesprochen, welche die einzelnen Übertragungseinheiten darstellen (dies ist auf Ethernet-Ebene ein „Frame“ und auf TCP-Ebene in „Paket“). Allerdings wird, trotz der klaren Namensgebung, häufig von IP-Paketen geredet, weshalb in diesem Kontext „Datagramm“ und „Paket“ synonym aufgefasst wird.

5.2 IPv4 Paketformat

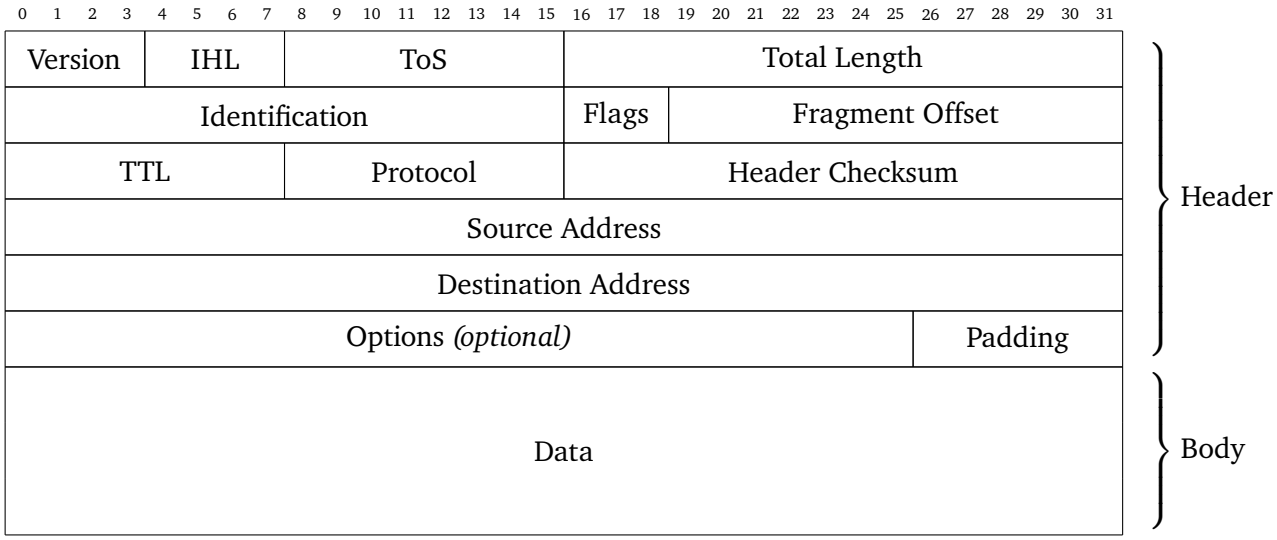


Abbildung 5.1: IPv4 Paketformat

Version 4 Bit breit. Die IP-Version. Kann 4 oder 6 sein (IPv4 bzw. IPv6).

IHL (Internet Header Length) 4 Bit breit. Die Gesamtlänge der Kopfdaten in 32 Bit Worten. Somit kann der Header maximal $(2^4 - 1) = 15$ Worte lang sein.

ToS (Type of Service) 8 Bit breit. Gibt die Priorisierung des Pakets an, wobei RFC 791, RFC 2474 und RFC 3168 zu teilen widersprüchlich sind und nur beide erfüllt sind, wenn die ersten 6 Bit 0 sind.

Total Length 16 Bit breit. Die Gesamtlänge des Paket (inklusive Kopfdaten) in Byte. Somit kann das Paket maximal $2^16 - 1 = 65535$ Byte ≈ 64KiB lang sein.

Identification 16 Bit breit. Eindeutige Identifikationsnummer, welche zur Zusammensetzung von zusammengehörigen Paket dient (in Verbindung mit den Feldern *Flags* und *Fragment Offset*).

Flags 3 Bit breit. Mit DF (Don't Fragment) und MF (More Fragments) wird angezeigt, ob ein Paket fragmentiert ist und weitere Fragmente folgen (MF) oder es nicht fragmentiert werden darf (DF).

Fragment Offset 13 Bit breit. Gibt (unabhängig von der Fragmentierung) an, welche Position innerhalb des fragmentierten Paketes durch das erste Byte in dem Datagramm dargestellt wird. Die Angabe erfolgt in 64 Bit Worten und ist unabhängig von der Fragmentierung, wodurch das Paket mehrmals hintereinander fragmentiert werden kann. Ist das Paket nicht fragmentiert, ist das Feld 0.

TTL (Time to Life) 8 Bit breit. Gibt die Lebensdauer des Paketes an, bevor es verworfen wird (sobald TTL = 0). Der Wert des Feldes wird bei jedem Hop um eins verringert.

Protocol 8 Bit breit. Gibt das darunterliegende Protokoll (bspw. TCP) an.

Header Checksum 16 Bit breit. Die Prüfsumme über die Kopfdaten. Zur Berechnung wird das Feld auf 0 gesetzt und der Header in 16 Bit Worte unterteilt. Für die weitere Berechnung siehe 10.1.

Source Address 32 Bit breit. Die Quelladresse.

Destination Address 32 Bit breit. Die Zieladresse.

Options and Padding *(optional)* 0 bis 40 Byte breit. Bis zu 40 Byte an Optionen, wobei verbleibende Bytes bis zum nächsten 32 Bit Wort mit Nullen aufgefüllt werden (Padding).

Data Die Nutzdaten des Paketes.

5.3 Adressierung

Da flaches Adressieren mittels MAC-Adressen nicht auf große Netzwerke skaliert (es gibt zu viele Geräte und es müsste für jedes Gerät eine Routing-Tabelle vorgehalten werden), hat IP eigene Adressen, welche hierarchisch (bzw. topologisch) strukturiert sind (in sogenannte Netze).

Eine IP Adresse ist eine Folge von 32 Bits, wobei jedes *Interface* eine solche zugewiesen bekommt. Ein Interface ist eine Netzwerkschnittstelle und ein Knoten kann mehrere Interfaces haben (folglich mehrere IP-Adressen), dies ist zum Beispiel bei Routern üblich, welche die Interfaces dann verbinden und die Pakete zwischen diesen weiterleiten.

Da es sehr anstrengend wäre, IP Adressen immer in binär (oder dezimal) anzugeben, werden sie in vier 8 Bit Blöcke unterteilt, welche mit einem Punkt getrennt Dezimal aufgeschrieben werden:

```
  10101010101000001001011010001100
⇝ 10101010.10100000.10010110.10001100
⇝ 170.160.150.140
```

5.3.1 Netzmaske

Eine IP Adresse enthält immer einen *Netzanteil* und einen *Hostanteil* und die Größe des Netzanteils wird mittels einer *Netzmaske* angegeben. Dabei ist der Netzanteil vorne in der IP Adresse und n Bits lang, wobei n durch die Netzmaske festgelegt wird. Die restlichen $32 - n$ Bits stellen dann den Hostanteil dar. Für die Netzmaske kann entweder das n angegeben werden (Kurzschreibweise) oder eine „IP Adresse“, welche nur vorne 1en hat. Die Anzahl der 1en ist das gleich dem n.

#	IP Adresse	Netzmaske	Netzmaske (kurz)	Netzanteil	Hostanteil
1	`192.168.178.2`	`255.255.255.0`	`24`	`192.168.178.0`	`0.0.0.2`
2	`130.83.40.233`	`255.255.255.0`	`24`	`130.83.40.0`	`0.0.0.233`

Tabelle 5.1: IP Adressen, Netzanteil, Hostanteil und Netzmaske

Um gleichermaßen eine vollständige IP Adresse als auch die Netzmaske anzugeben, wir diese mit einem Slash getrennt hinter die IP Adresse geschrieben: `192.168.178.2/24`.

5.3.2 Klassen

Um weltweit IP Adressen an Firmen oder kleinere Unternehmen zu vergeben, wurden die IP Adressen in die folgenden Klassen aufgeteilt:

Class A Für sehr große Organisationen, nur 255 Netze und 16.000.000 Hosts pro Netz.

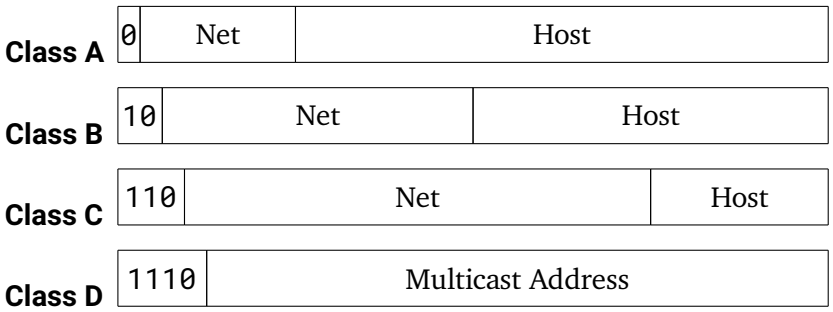
Class B Für große Organisationen, 65.000 Netze und 65.000 Hosts pro Netz.

Class C Für kleine Organisationen, 16.000.000 Netze und 255 Hosts pro Netz.

Class D Multicastadressen, keine Netz/Host Hierarchie.

Class E Reserviert.

Ferner wurden für die Netze folgende Eigenschaften festgelegt, damit die Netze von Routern und Menschen voneinander unterschieden werden können:



Class E	1111	Reserved

Werden Netze nach diesem Schema vergeben, spricht man von „Class-full Addressing“.

Classless InterDomain Routing (CLDR)

Da Class-full Adressing sehr verschwenderisch ist, wurde mittlerweile dazu übergegangen, IP Netze nicht mehr fest nach den Klassen einzuteilen sondern Netzmasken wie `/17` o.ä. zuzulassen.

In diesem Fall spricht man von „Classless InterDomain Routing“, kurz CIDR.

Dies kann zu effizientem Routing führen, beispielsweise weißt man allen IP Adressen in Europa den gleichen Prefix zu, wodurch in Amerika nur ein Routing-Eintrag für jeden Traffic nach Europa erstellt werden muss.

5.3.3 Spezialadressen

Netzwerkadresse Werden alle Bits des Hostanteils auf Null gesetzt, so ergibt dies die Netzadresse.

Broadcastadresse Werden alle Bits des Hostanteils auf Eins gesetzt, so ergibt dies die Netzadresse. Pakete, welche an diese Adresse gesendet werden, werden allen Hosts in dem Netzwerk zugestellt.

Loopback Außerdem gibt es das Loopback-Netz `127.0.0.0/8`, welches vollständig dem lokalen System zusteht. Die Pakete werden somit nicht geroutet und verbleiben auf dem PC.

5.3.4 Subnetze

Wird ein bestehendes Netz (bspw. `192.168.0.0/16`) in kleinere Netze (bspw. `192.168.0.0/24` bis `192.168.255.0/24`) aufgeteilt, so wird von *Subnetting* gesprochen.

Große Netzwerke werden somit in weitere, kleinere, Netzwerke aufteilt, wodurch eine hierarchische Struktur entsteht und effizientes Routing ermöglicht wird.

5.3.5 Variable Length Subnet Mask (VLNM)

Durch *Variable Length Subnet Mask* (VLNM) ist eine effiziente Nutzung des Adressbereiches möglich. VLNM bezeichnet das Nichtklassenstarre Systeme, in denen Variable Netzmasken vergeben werden können. VLNM stellt eine Implementierung von CLDR dar und wurde 1995 als dieses eingeführt.

5.4 Dynamic Host Configuration Protocol (DHCP)

Das *Dynamic Host Configuration Protocol* (DHCP) ist ein Protokoll, welches Plug’n’Play bei Netzwerken ermöglicht. DHCP weißt hierbei neuen Geräten automatisch eine IP zu, sodass eine statische Konfiguration dieser wegfällt[1].

[1] In der Realität tut DHCP noch weit mehr Dinge, bspw. Routing-Tabellen und DNS-Server konfigurieren

5.5 Network Address Translation (NAT)

Da es zu wenig IP Adressen für alle Geräte weltweit gibt, müssen IP Adressen eingespart werden.

Eine Möglichkeit hierfür ist NATing (*Network Address Translation*), wobei alle Hosts in einem lokalen Netzwerk über eine einzige IP Adresse nach außen kommunizieren:

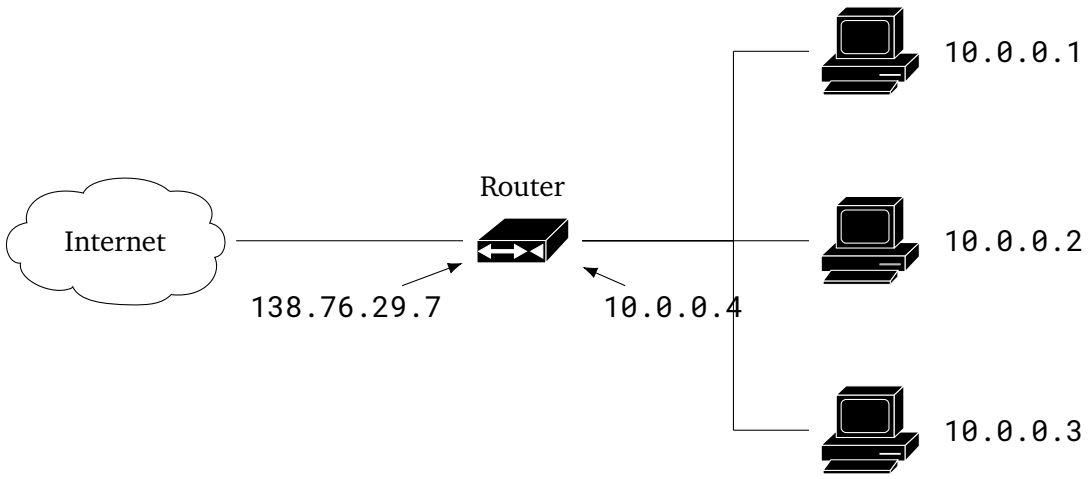


Abbildung 5.2: Network Address Translation

Zu den Ideen hinter NATing und den Vorteilen zählen:

- Das lokale Netzwerk nutzt nur eine IP Adresse zum „Rest der Welt“.
- Der ISP muss nur eine IP Adresse reservieren und keinen Adressbereich.
- Die IP Adressen innerhalb des Netzwerkes können einfach geändert werden, ohne das äußere Netz zu beeinflussen.
- Der ISP kann einfach gewechselt werden, ohne das lokale Netz zu ändern.
- Geräte innerhalb des Netzes sind nicht einfach von außen erreichbar/adressierbar ($\Longrightarrow$ mehr Sicherheit).

Allerdings hat NATing auch einige große Nachteile:

- Router sollten eigentlich nur auf OSI-Schicht 3 (Network) arbeiten und nicht in die höheren Schichten eingreifen.
- Das Prinzip von Ende-zu-Ende Verbindungen wird verletzt $\Longrightarrow$ der Router kann mitlesen.
- Server in NAT-Netzwerken können nicht von außen angesprochen werden $\Longrightarrow$ normalerweise befinden sich Server nicht in einem NAT-Netzwerk.
- Probleme wie Adressknappheit sollten durch andere Systeme wie bspw. IPv6 gelöst werden.

Implementation

- Ausgehende Datagramme: Der Router ersetzt die Quell-IP und den Port von jedem ausgehenden Datagramm durch seine IP Adresse und einen neuen Port. Die Zuweisung wird in einer *Translation Table* gespeichert.
- Entfernte Hosts antworten mit der IP von dem Router und dem entsprechendem Port.
- Eingehende Pakete: Router Router guckt in der Translation Table nach und ändert die IP Adresse und den Port entsprechend.
- Durch 16 Bit lange Ports ist es theoretisch möglich, ≈ 65.000 Hosts mit einer IP Adresse anzubinden.

5.6 (Reverse) Address Resolution Protocol (ARP/RARP)

Das *Address Resolution Protocol* (ARP) ist ein Protokoll zur Auflösung von IP Adressen zu MAC Adressen, um die Lücke zwischen OSI-Schicht 3 und 2 zu schließen. Mit RARP (*Reverse ARP*) existiert ferner eine Möglichkeit, MAC Adressen zu IP Adressen aufzulösen.

Funktionsweise (ARP)

- Nachfrage, wem eine bestimmte IP Adresse gehört mittels MAC Broadcast im LAN.
- Der Zielhost mit der IP Adresse antwortet auf das Paket.
- Der Quellhost empfängt die Antwort und kann nun das Paket senden.
- Die Zuordnung wird eine bestimmte Zeit (meist 10min) im Cache behalten.

> **Warning:** Der Cache kann gefälscht werden, indem falsche Antworten durchs Netzwerk gesendet werden.

5.7 IPv6

- Mit IPv4 gehen die IP Adressen aus $\rightarrow$ mehr Bits für die IP Adresse.
- Mit größeren IP Adressen ist mehr hierarchisches Routing möglich.
- Plug'n'Player Netzwerke ohne DHCP Server.
- Ende-zu-Ende Kommunikation, Authentifizierung und Verschlüsselung auf IP Ebene sind möglich.

5.7.1 IPv6 Paketformat

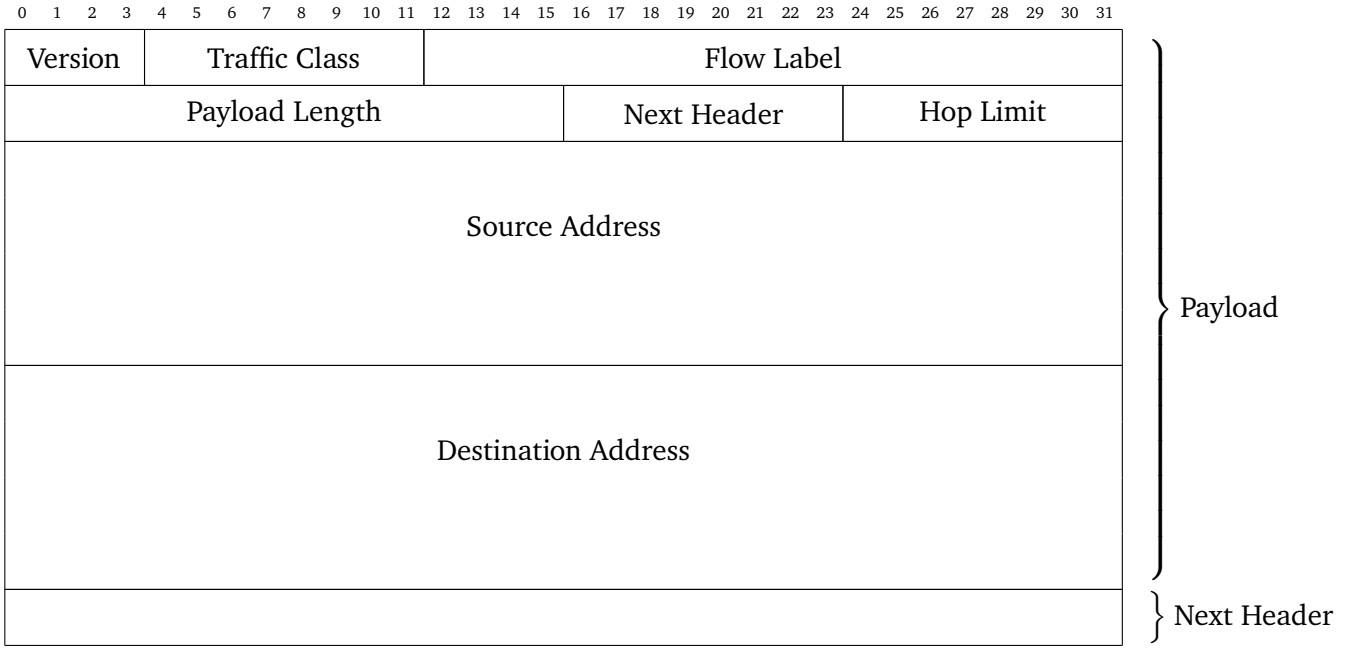


Abbildung 5.3: IPv6 Paketformat

Version 4 Bit breit. Die IP-Version. Kann 4 oder 6 sein (IPv4 bzw. IPv6).

Traffic Class 6+2 Bit breit. Die ersten 6 Bit klassifizieren das Paket, die letzten 2 Bit sind nötig für explizites Congestion Control.

Flow Label 20 Bit breit. Gibt an, dass Pakete nicht umsortiert werden sollen.

Payload Length 16 Bit breit. Die Gesamtlänge Der Kopfdaten inklusive Erweiterungsheader.

Next Header 8 Bit breit. Spezifiziert den Typ des nächsten Headers. Dies kann ein höheres Protokoll (bspw. TCP) sein oder auch Erweiterungsheader.

Hop Limit 8 Bit breit. Gibt die maximale Anzahl Hops des Paketes an, bevor es verworfen wird (sobald Hop Limit = 0). Der Wert des Feldes wird bei jedem Hop um eins verringert.

Source Address 128 Bit breit. Die Quelladresse.

Destination Address 128 Bit breit. Die Zieladresse.

Next Header Der nächste Header (Erweiterungsheader oder höheres Protokoll).

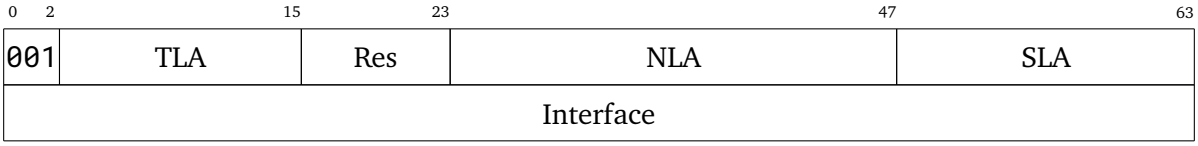
5.7.2 Adressierung

IPv6 Adressen sind 128 Bit breit und werden in 16 Bit Blöcken hexadezimal notiert (getrennt mit Doppelpunkten). vorangehende Nullen können weggelassen werden und an genau einer Stelle des Paketes kann ein

großer Block nullen weggelassen werden:

```
    0101111000110111101111000000000...
⇝   5f1b:df00:0000:0000:0020:0800:2078:e3e3
⇝   5f1b:df00:0:0:20:800:2078:e3e3
⇝   5f1b:df00::20:800:2078:e3e3
```

Eine IPv6 Adresse ist in folgende Blöcke unterteilt:



TLA 13 Bit; Top Level Aggregation; global von der IANA zugewiesene ISP-Nummer

Res 8 Bit; Reserved; Reserviert für Erweiterung von TLA und/oder NLA

NLA 24 Bit; Next Level Aggregation; Routing-Struktur vom ISP

SLA 16 Bit; Site Level Aggregation; Routing-Struktur von einer Organisation

Interface 64 Bit; Client-ID, basierend auf der MAC Adresse oder zufälliger Zahl

5.7.3 Übergang von IPv4 zu IPv6

Da es keinen festen Termin für die Umstellung von IPv4 auf IPv6 gibt/gab/geben wird, erfolgt der Übergang fließend. Hierzu wurden die folgenden Mechanismen geschaffen:

4in6 Tunneln von IPv4 in IPv6

6in4 Tunneln von IPv6 in TPv4

6over4 Transport von IPv6 Paketen zwischen zwei Dual-Stack Knoten über ein IPv4-Netzwerk

Dual-Stack Parallelbetrieb IPv4/IPv6

Dual-Stack Lite Wie Dual-Stack, aber mit globaler IPv6 und Carrier-NAT IPv4

Teredo Kapselung von IPv6 Paketen in IPv4-UDP Paketen

NAT64 Übersetzung von IPv4 Adressein in IPv6 Adressen

474XLAT Übersetzung von IPv4 Adressein in IPv6 Adressen in IPv4 Adressen

5.7.4 Mehr IPv6

- ARP wird nicht mehr genutzt und wurde durch das *Neighbour Discovery Protocol* (NDP) ersetzt.
- *Duplicate Address Detection* (DAD) wird genutzt, um Adressdopplungen zu vermeiden.
- Mit IPv6 sollten DHCP Server nur noch der Informationsverteilung dienen, die Adressen werden mittels SLAAC (*Stateless Address Autoconfiguration*) vergeben.

  - Der Router vergibt ein `/64` Netzwerk (*Router Advertisement* (RA) über NDP).
  - Jeder Knoten weißt sich selbst eine IP Adresse zu mittels EUI-64.
    Ethernet: Netzadresse + `FFFE:FFFE` + MAC Adresse

- Quad-A (AAAA) Records in DNS zur Auflösung von Domains zu IPv6 Adressen.

5.8 Was ist mit IPv1, IPv2, IPV3 und IPv5 passiert?

IPv1, IPv2 nad IPv3 Die ersten drei Version von IP haben versucht,die Probleme auf Schicht 3 und 4 zu vereinen und sind dadurch gescheitert.

Die ersten drei Versionen trugen dementsprechend noch den TCP Version 1, 2 bzw. 3.

IPv5 IPv5 wurde im Jahr 1995 in RFC 1190, RFC 1819 spezifiziert und wird auch *Internet Stream Protocol* (SR2) genannt.

Da der Kosten-Nutzen Faktor schlecht war und das Protokoll dementsprechend nicht angenommen wurde, wurde direkt mit Version 6 (IPv6) fortgefahren.

6 Transport

Transportdienste und Protokolle auf OSI-Schicht 4 bieten eine *logische Verbindung* zwischen Prozessen auf verschiedenen Maschinen.

- Der Sender bricht die zu sendende Nachricht in Segmente, welche an den Netzwerkschicht übergeben werden.
- Der Empfänger baut die Nachrichten wieder aus den Empfangen Segmente zusammen und gibt diese an die Transportschicht weiter.
- Die dominantesten Protokolle im Internet sind TCP und UDP.

6.1 Adressierung (Ports)

Bisher wurden ausschließlich IP Adressen zur Adressierung von Hosts genutzt. Allerdings muss auch innerhalb eines Gerätes unterschieden werden, an wen eine Nachricht gehen soll (die Kommunikation findet immer zwischen zwei Prozessen statt!). Da es viele Prozesse auf einem System gibt, müssen diese adressiert werden.

Hierzu gibt es sogenannte *Ports*, welche einen Prozess identifizieren und die Kommunikation zu diesem ermöglichen. Ein Port ist eine Zahl mit 16 Bit, es sind also maximal $(2^{16} = 65.536)$ Ports vergebbar.

> **Warning:** Ein Port ist nicht zu verwechseln mit einer PID (Process ID), welche einen *Systemprozess* eindeutig identifiziert. Ebenfalls ist eine PID nicht zur Adressierung geeignet, da sie sich ändert, wenn ein Prozess neu gestartet wird.

- Ports werden von dem Betriebssystem verwaltet, welches auch dafür sorgt, dass kein Port doppelt vergeben wird.
- Ein Prozess kann sich für einen Port registrieren (sich an diesen *binden*).
  Achtung: Für einige Ports (meist alle Ports ≤ 1024) muss ein Prozess Root-Rechte vorweisen.
- Das OS hält alle Pakete in einer Warteschlange, bis der Prozess diese abholt.
- Nachrichten, welche an nicht-gebundene Ports gesendet werden, werden abgewiesen.
- Ist ein Port gebunden, so spricht man von „der Port ist offen“.
- Zur Kommunikation muss ein Sender wissen, auf welchem Port ein bestimmter Dienst läuft.
  $\rightarrow$ Standardisierung von Ports.
- Der Verbindungspunkt, mit dem das OS spricht, wird *Socket* genannt.

6.1.1 Trivia

Service	Port/Protokoll
echo	7/udp
time	37/udp
name	42/udp
dns	53/udp
tftp	69/udp
sunrpc	111/udp
who	513/udp
talk	517/udp
ftp	20/tcp
ftp	21/tcp
ssh	22/tcp
smtp	25/tcp
dns	53/tcp
http	80/tcp
pop3	110/tcp
https	443/tcp

Tabelle 6.1: Well-Known Ports (TCP/UDP)

Oftmals muss ein Prozess Root-Rechte vorweisen, um sich an einen Well-Known Port zu binden.

6.2 Multiplexing/Demultiplexing

Da es viele Ports gibt, aber am Ende alles über eine IP Adresse fließen muss, wird Multiplexing (beim Sender) und Demultiplexing (beim Empfänger) betrieben. Dies funktioniert folgendermaßen:

- Der Sender nimmt die Daten von den Sockets entgegen und bastelt daraus IP Pakete, welche als Daten die Daten selbst und einen Header enthalten, der den Quell- und Zielport enthält.
- Die entstehenden Pakete werden anschließend versendet.
- Der Empfänger nimmt die Daten von IP entgegen und stellt sie, dem Header entsprechend, den korrekten Sockets zu.
- Somit enthält ein TCP/UDP Paket mindestens die Felder *Quellport* und *Zielport*.

> Es gibt hier noch kein Client-Server Prinzip. Auch der „Client“ öffnet einen Ports, um mit einem „Server“ zu kommunizieren. Nur wird dieser Port schnell wieder geschlossen.

6.3 Segmenting, Reassembling

Da die maximale Übertragungsgröße (*Service Data Unit* (SDU)) auf höheren Ebenen deutlich größer sein kann als auf tieferen Ebenen (Vergleich Ethernet vs. IP vs. TCP), müssen die Nachrichten aufgeteilt werden. Dies passiert in den unteren Ebenen transparent für die höheren Ebenen, sodass diese sich nicht darum sorgen müssen.

Beispiel:

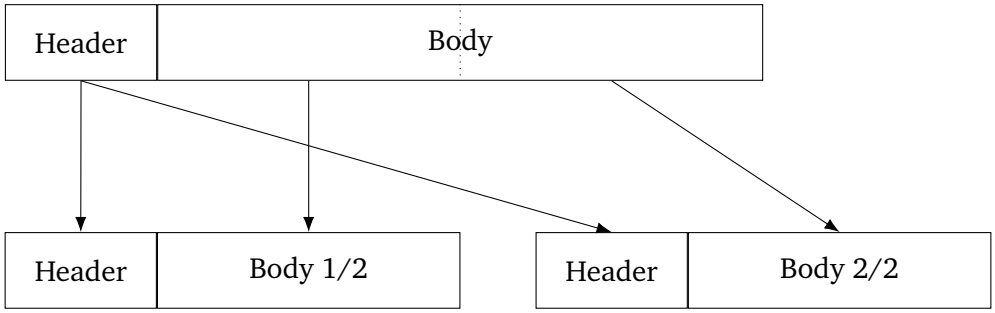


Abbildung 6.1: Transport Segmentierung

6.4 Verbindungssteuerung

Es wird nun um Verbindungsorientierte Protokolle, bzw. um die Steuerung der Verbindung gehen.

Im Grunde gibt es folgende Phasen einer Verbindung:

- Verbindungsaufbau (*Connection Establishment Phase*, *Connect*)
- Datentransfer (*Data Transfer Phase*, *Data*)
- Verbindungsabbau (*Connection Release Phase*, *Disconnect*)

6.4.1 Primitive Serviceeinheiten

Welche durch folgende primitive Ausgedrückt werden können:

- `T-Connect` (Bestätigt)
  - `T-Connect.req(Destination Address, Source Address)`
    Startet den Verbindungsaufbau.
  - `T-Connect.ind(Destination Address, Source Address)`
    Eingang der Verbindungsanfrage.
  - `T-Connect.rsp(Responding Address)`
    Bestätigung der Verbindungsanfrage.
  - `T-Connect.cnf(Responding Address)`
    Eingang der Bestätigung.
- `T-Data` (Unbestätigt)
  - `T-Data.req(userdata)`

  - `T-Data.ind(userdata)`

- `T-Disconnect` (meist Unbestätigt)
  - `T-Disconnect.req(userdata)`
  - `T-Disconnect.ind(cause, userdata)`

Parameter

- Verbindungsaufbau

  `Destination Address` Adresse des aufgerufenen Dienstes.

  `Source Address` Adresse des aufrufenden Nutzers.

  `Responding Address` Adresse des antwortenden Dienstes (meist die Adresse des aufrufenden Nutzers).

- Datentransfer

  `userdata` Zu übertragende Daten (beliebiger Länge).

- Verbindungsabbau

  `userdata` Zu übertragende Daten (Länge beschränkt).

  `cause` Grund des Verbindungsabbaus.

6.4.2 Drei-Wege Handschlag

Problematik: Geht ein `T-Connect.rsp` verloren, bekommt die der Sender dies nicht mit. Er weiß somit nicht, ob die Verbindung hergestellt ist und ob er Daten senden kann.
Lösung: *Drei-Wege Handschlag*

Bei einem Drei-Wege Handschlag gilt die Verbindung erst als aufgebaut, wenn sowohl `T-Connect.req` als auch `T-Connect.rsp` bestätigt wurden. Die letzte Bestätigung kann durch ein simples ACK oder durch Daten bestätigt werden.

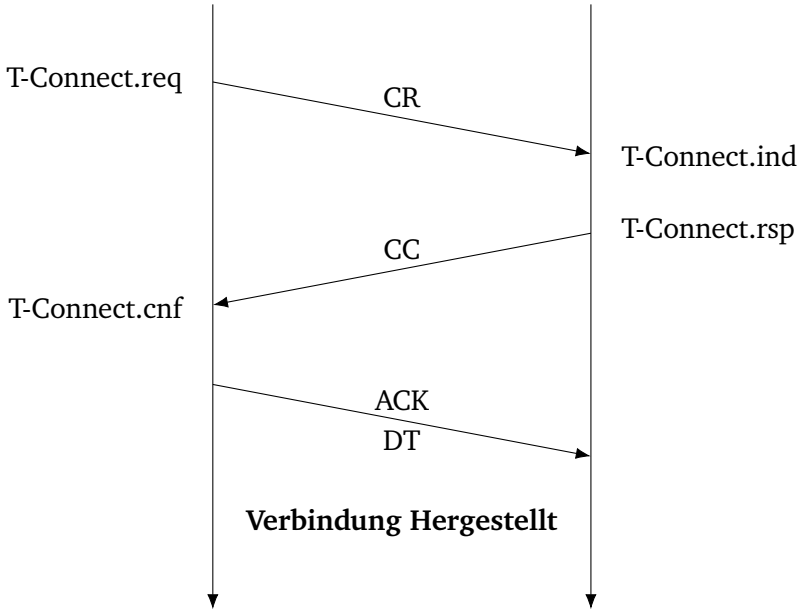


Abbildung 6.2: Drei-Wege Handschlag

Sequenznummern **Problematik:** Treffen duplizierte CCs oder CRs spät ein, so geht ein Host davon aus, dass die Verbindung hergestellt wurde, obwohl dies nicht der Fall ist oder sogar gar keine Anfrage nach einer Verbindung gestellt wurde.
Lösung: *Sequenznummern*

- Jedem Paket wird eine Sequenznummer angehängt.
- Der Empfänger kopiert diese in seine Antwort, sodass der jeweils andere sich sicher sein kann, dass das Paket eine Antwort auf die zuvor gestellte Frage ist.
- Die Verbindung wird nur hergestellt, wenn die korrekte Sequenznummer verwendet wurde.
- Sequenznummern sollten nicht zu schnell wiederverwendet werden.

6.4.3 Verbindungsabbau

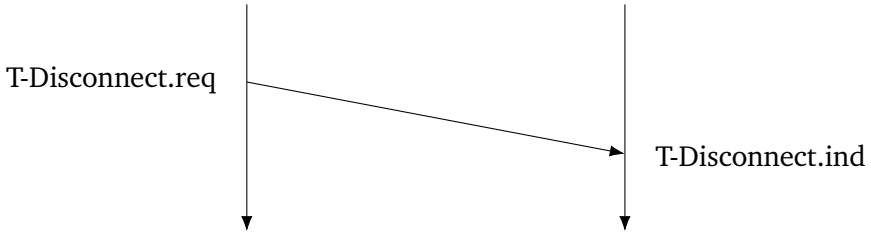


Abbildung 6.3: Verbindungsabbau durch Nutzer

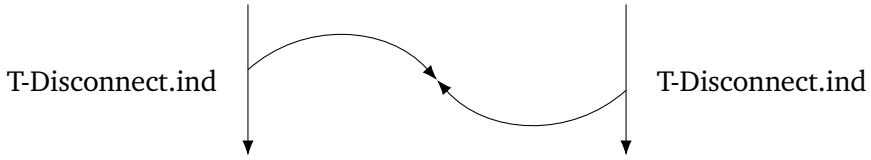


Abbildung 6.4: Verbindungsabbau durch Provider

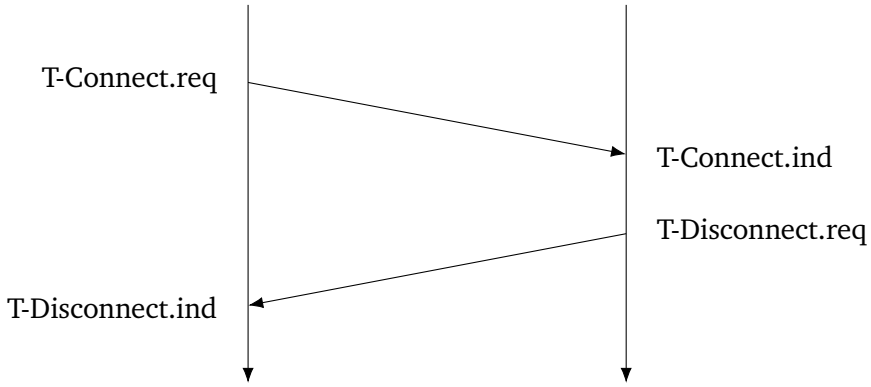


Abbildung 6.5: Ablehnung des Verbindungsaufbaus

- Die Verbindung wird regulär abgebaut.
- Dies kann zu einem Datenverlust führen, wenn die Verbindung zu früh abgebaut wird.
- Impliziter Abbau: Schließen des Sockets und nicht mehr antworten.
- Expliziter Abbau: Verbindungsfreigabe mittels `T-Disconnect`.
- **Problematik:** Es ist unmöglich sicherzustellen, dass keine Daten mehr unterwegs sind. Sämtliche Pakete können verloren gehen, ..., siehe *Two Army Problem*

6.4.4 Beispiellauf

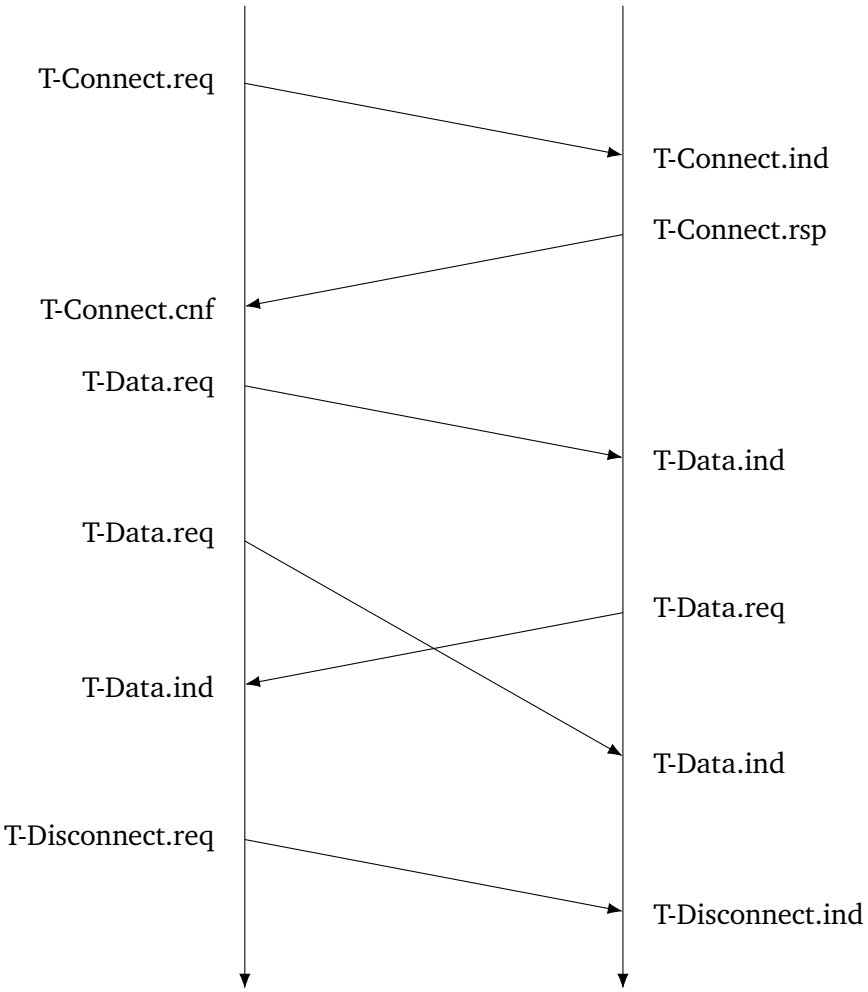


Abbildung 6.6: Ablehnung des Verbindungsaufbaus

6.5 Verlässliche Zustellung

- Bisher werden Pakete so gut es geht zugestellt.
- Allerdings können Pakete verloren gehen oder beschädigt werden (Bitfehler, Congestion, . . .)
- Somit sollte der Sender darüber Informiert werden, dass etwas schief gelaufen ist
  → Sende für jedes Paket ein Acknowledge (ACK)
  → Weitere Probleme (hohes Datenaufkommen, . . .)
- Was kann ein Sender tun, wenn ein Paket nicht bestätigt wurde?
  1. Das Paket erneut senden.
  2. Die Datenrate reduzieren.

6.5.1 Fehlersteuerung

> **Warning:** Nicht alle Fehler können erkannt werden. Mit einer geringen Chance kann es bspw. passieren, dass sich sowohl Daten als auch Prüfsumme ändern.

- Wenn ein Fehler erkannt wird (Seiten des Empfängers), so wird das Paket meist einfach gedroppt.
- In wenigen Fällen gibt der Empfänger dies auch über ein NACK (Negative ACK) bekannt (oftmals aber erst nach dem nächsten Paket, da sonst nicht geprüft werden könnte, dass ein Paket fehlt).

6.5.2 Automatic Repeat reQuest (ARQ)

ARQ Typ 1 Stop-and-Wait ARQ

- Entspricht dem Alternating-Bit Protocol (6.6.2)
- Schlechte Performanz bei schnellen oder breiten Netzwerken

ARQ Typ 2 Go-Back-N ARQ

- Angenommen, das aktuelle und das nächste Paket gehen verloren
  Bei alten Data Link Schichten kann der Empfangspuffer nur ein Element halten, d.h. out-of-order Pakete werden verworfen.
- Wird ein verlorenes Paket erkannt, so werden alle vorherigen Pakete erneut gesendet (bis zum letzten ACK).
- Bei einem Timeout wird nur das betroffene Paket erneut versendet.

6.6 Flow Control

- **Aufgabe:** Schützen des Empfängers vor zu vielen eingehenden Paketen.
- Link Layer: Schutz vor zu vielen weitergeleiteten Paketen.
- Höhere Schichten (Network/Transport): Schutz vor zu vielen Verbindungen.
- TCP implementiert Flow Control, dies ist auf Ebene 4 (Transport) aber kompliziert:
  - Die Paketgröße ist sehr variabel.
  - Die Antwort von Prozessen kann sehr lange dauern.
  - Mit mehreren parallelen Verbindungen müssen dynamische Puffer alloziert werden, auf unteren Schichten würde eine fixe Größe ausreichen.

6.6.1 Puffer Allokation

- Der Empfänger muss eingehende Pakete zwischenspeichern, bevor diese verarbeitet werden können.
- Um dies zu unterstützen, muss der Sender:
  - sich darauf verlassen, dass eingehende Pakete direkt bearbeitet werden (sehr unrealistisch!).

– erwarten, dass genug Speicherplatz im Puffer verfügbar ist.

Mögliche Implementierungen:

- Der Empfänger verlangsamt den Sender, wenn kein Speicher mehr verfügbar ist (implizit oder explizit).
- Der Sender fragt einen bestimmten Pufferplatz an.
- Der Empfänger teilt dem Sender mit, wie viel Platz er noch im Puffer hat (Beeinflussung der Datenrate oder „Credit“ des Senders).

6.6.2 Alternating-Bit Protocol

- Der Sender wird bei jedem Paket informiert, wenn es eingetroffen ist (ACK).
- Timeout beim Sender
  $\Longrightarrow$ Paket verschwunden, ACK verschwunden, ACK verspätet
  $\Longrightarrow$ Daten erneut senden.
- Dies ist allerdings sehr langsam, da der Sender andauernd warten muss und annähernd durchgehend im Idle ist. Ebenso der Empfänger.

6.6.3 Stop/Continue Nachrichten

- Es werden explizite Benachrichtigungen (*Stop*, Continue) ausgetauscht als Flow Control.
- Wenn der Empfänger nicht mit den eingehenden Daten mithalten kann, sendet er ein *Stop*.
- Nach Verarbeitung der Daten wird ein *Continue* gesendet.

6.6.4 Rate Basiert

- Eine andere Methode für Flow Control.
- Der Empfänger gibt an, welche Datenrate er maximal verarbeiten kann.
- Der Sender hält sich an diese Datenrate und überflutet den Empfänger somit nicht.
- Vorteile:
  – Es werden wenig Nachrichten ausgetauscht $\Longrightarrow$ geringer Overhead.
  – Netzwerk und Empfänger werden „sanft“ ausgelastet, es gibt keine einzelnen hohen Spikes.
- Nachteile:
  – Es werden keine Informationen über erfolgreichen Datenempfang gesendet.
  $\Longrightarrow$ Ergibt keinen Sinn für verlässlichen Transport über unzuverlässige Netze.
  $\Longrightarrow$ Es muss eine separate ACK/Repeat Methode geben.
  $\Longrightarrow$ Dies kann die Erwartung der Datenrate stören.

6.6.5 Credit Basiert

- Der Empfänger teilt dem Sender regelmäßige *Credits* zu, sodass dieser senden darf.
- Der Sender stoppt die Übertragung, wenn er keine Credits mehr hat.
- Explizites Fehlerkontrolle ist nötig, um Fehler zu erkennen.

Absolute Credits

- Der Empfänger weißt dem Sender eine absolute Anzahl Credits zu.

Sliding Window

- Credits werden in Relation zu den ankommenden ACKs vergeben.

6.7 Congestion Control

- In Netzwerken kann es zu Verstopfung (Congestion) kommen, da die Elemente eine begrenzte Bandbreite haben.
- Werden mehr Daten in das Netzwerk gegeben als verarbeitbar sind, so bricht das Netzwerk zusammen (*Congestion Collapse*) und es gehen Pakete verloren.
- Gehen Pakete verloren (bspw. durch volle Puffer), so sind die verbrauchten Netzwerkressourcen verschwendet worden.
  - Dies führt zu einem Schneeball-Effekt, da das Netzwerk mit neu gesendeten Paketen (aufgrund von Verlust) geflutet wird.
  - Dies führt wieder zu mehr Traffic.
  - Mehr Pakete gehen verloren.
  - ...
- **Lösung:** Die Senderate muss an das Netzwerk angepasst werden.
  Dies hängt von allen Komponenten im Netzwerk ab und ist damit sehr komplex.

Abgrenzung Flow Control ↔ Congestion Control *Flow Control* beschäftigt sich damit, dass *einzelne Knoten* nicht überlaufen,
Congestion Control beschäftigt sich damit, dass das *gesamte Netzwerk* nicht überläuft.

6.7.1 Design Properties/Options

Fairness Das System sollte Fair sein, d.h. jeder Knoten sollte fair viele Ressourcen zur Verfügung haben.
Dies heißt nicht, dass jeder Knoten gleich viele Ressourcen braucht (Bsp.: Telnet vs. Videokonferenz).

Router-Basiert vs. Host-Basiert Wo werden die Informationen gesammelt, Entscheidungen getroffen, Aktionen angewandt?

Window-Basiert vs. Rate-Basiert Wie wird beschrieben, wie viel Traffic ein Host in das Netzwerk senden darf?

- Rate - Eine bestimmte Anzahl Bytes pro Sekunde
- Congestion Window - Eine Anzahl Bytes, die gesendet werden darf, bevor neue Credits vergeben werden.

Open Loop Das System ist so gestaltet, dass es von vornherein funktioniert und zur Laufzeit keine Anpassungen notwendig sind.

Closed Loop Das System nutzt eine Art Feedback, um sich den Gegebenheiten anzupassen.

Implizites Feedback Congestion wird von dem Sender erkannt (bspw. durch ausbleibende ACKs).

Explizites Feedback Congestion informiert den Sender (auf irgendeine Art).

6.7.2 Pakete Verwerfen, Implizites Feedback, Mögliche Aktionen

Sei ein Router gegeben, dessen Puffer vollgelaufen ist.

- Welches Paket soll verworfen werden, wenn ein neues eintrifft?
- **Drop Tail:** Das neuste Paket (welches am Ende einsortiert wird) wird verworfen.
  Annahme: Alte Pakete sind wichtiger als neue.
- **Drop Any:** Um das neue Paket einfügen zu können, wird ein Paket verworfen, welches schon länger in der Warteschlange wartet.
  Annahme: Neue Pakete sind wichtiger als alte (bspw. bei Livestreams).

Feedback

- Der Verwurf eines Paketes stellt *implizites Feedback* dar, da das Paket verloren geht. Dies kann bspw. durch fehlende ACKs vernommen werden.
- Unter der Annahme, das Pakete nur verloren gehen, wenn Congestion auftritt, sollte der Sender seine Senderate verringern.
  In Kabelgebundenen Netzen mag dies zutreffen, nicht aber bei WLAN-Netzen.

> **Warning:** In Open Loop Systemen sollte eine voller Puffer nie auftreten, da sonst das System nicht korrekt gestaltet ist.

Erhalt von Feedback *Ein Sender hat (auf irgendeine Weise) mitbekommen, dass Congestion aufgetreten ist.*

- Bei **Rate Basierten** Systemen sollte der Sender seine Senderate reduzieren.
- Bei **Window Basierten** System sollte der Sender sein Window verkleinern.

Dies wird bei TCP näher betrachtet (6.9.7).

6.7.3 Proaktive Aktionen

- Da Congestion nicht gut ist, sollte dies möglichst früh vermieden werden.
- Hierzu sind *proaktive Aktionen* sinnvoll, durch die Sender frühzeitig Informiert werden, wenn ein Puffer volllaufen sollte (o.ä.).
- Ein solcher Zustand im Router wird *Warning State* genannt.

Choke Pakete

- Stellt ein Router fest, dass er congested ist oder dies bald sein wird, so sendet dieser *Choke Pakete*.
- Ein Choke Paket teilt dem Sender mit, dass dieser seine Senderate reduzieren sollte.
- **Problematik:** In einem bereits verstopften Netzwerk können Choke Pakete zu noch mehr Congestion führen.
- **Problematik 2:** Es kann lange dauern, bis der Sender merkt, dass Choke Pakete gesendet wurden.

Warning Bits

- Stellt ein Router fest, dass er congested ist oder dies bald sein wird, so setzt dieser ein *Warning Bit* in allen Paketen, die ausgesendet werden.
- Das Ziel kopiert dieses Bit in das ACK-Paket.
- Der Sender empfängt das Bit und reduziert seine Senderate.

Random Early Detection (RED)

- Verlorene Pakete werden als implizites Feedback gewertet.
- Es werden Pakete verworfen, selbst wenn der Puffer noch nicht voll ist.
- Ein eingehendes Paket wird mit einer gewissen Wahrscheinlichkeit verworfen, die bis zu einem gewissen Punkt mit der Anzahl Elemente im Puffer steigt.

6.8 UDP

Das *User Datagram Protocol* (UDP) ist verbindungslos, stateless, hat kein Congestion Control und keine Empfangsbestätigung. Die Pakete kommen nach bester Möglichkeit an, können aber verloren gehen. Dies ist für Multi-Media Anwendungen sinnvoll, bei denen Echtzeit wichtiger ist als Vollständigkeit (bspw. VoIP).

Dadurch ist UDP sehr einfach, es stellt sich allerdings die Frage, weshalb nicht einfach IP gesprochen wird? Der Grund liegt darin, dass UDP Multiplexing (Ports) unterstützt und somit mehrere Prozesse auf einem Host UDP sprechen können.

6.8.1 Datagramm-Format

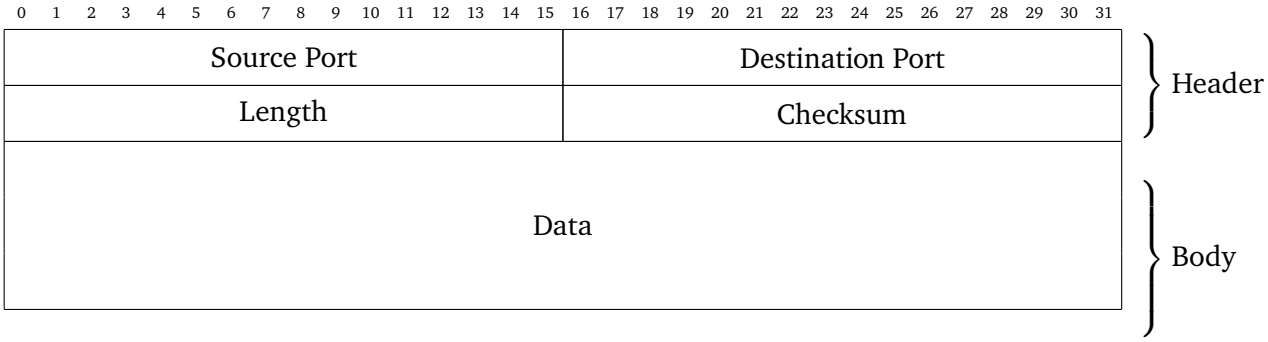


Abbildung 6.7: UDP Paketformat

Source Port 16 Bit breit. Der Quellport.

Destination Port 16 Bit breit. Der Zielport.

Length 16 Bit breit. Die Gesamtlänge des Datagramms in Byte. Somit kann ein Datagramm maximal $2^16 - 1 = 65535$ Byte lang sein.

Checksum 16 Bit breit. Die Prüfsumme über die Kopfdaten. Zur Berechnung wird das Feld auf 0 gesetzt und der Header in 16 Bit Worte unterteilt. Für die weitere Berechnung siehe 10.1.

Data Die Nutzdaten des Datagramms.

6.8.2 Demultiplexing

Siehe 6.2.

6.9 TCP

Das *Transport Control Protocol* (TCP) ist das Standard Protokoll im Internet und annähernd jedes andere Protokoll setzt auf TCP auf.

TCP implementiert die folgenden Features:

- Punkt-zu-Punkt
  Pro Verbindung gibt es einen Sender und einen Empfänger.
- Verlässlich und in-Order
  Alle Pakete kommen gesichert und in der richtigen Reihenfolge an oder es wird über einen Verlust berichtet.
- Vollduplex
  Daten können in beide Richtungen gesendet werden.
- Sende- und Empfangspuffer

- Verbindungsorientiert
  Handshaking (Drei-Wege Handschlag) initialisiert Sender und Empfänger bevor Daten gesendet werden können.
- Flow Control
  Der Sender kann den Empfänger nicht überlasten.
- Congestion Control
  Der Sender passt das Congestion Window an und übernimmt neue Senderaten.

6.9.1 Paket-Format

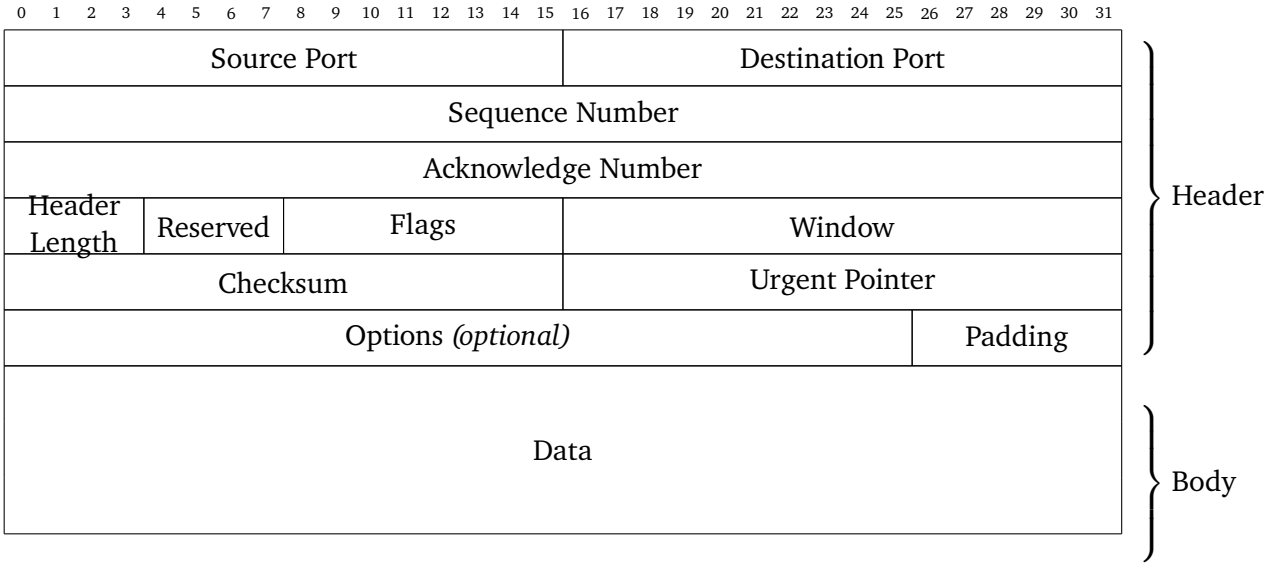


Abbildung 6.8: TCP Paketformat

Source Port 16 Bit breit. Der Quellport.

Destination Port 16 Bit breit. Der Zielport.

Sequence Number 32 Bit breit. Die Sequenznummer des Pakets.

Acknowledge Number 32 Bit breit. Die Sequenznummer des Paketes, welches bestätigt wird (plus die Länge des bestätigten Paketes).

Header Length 4 Bit breit. Die Länge der Kopfdaten in 32 Bit Worten.

Flags 6 Bit breit. Schalter zur Steuerung von bspw. dem Verbindungsaufbau in der folgenden Reihenfolge:

URG Urgent - Markiert alle Daten bis zu dem Byte Feld, auf das der *Urgend Pointer* zeigt, als dringend. Der Empfänger soll diese Daten bevorzugt behandeln. In der Regel wird das Feld aber nicht ausgewertet.

ACK Acknowledge - Zeigt an, dass das *Acknowledge*-Feld gültig ist und ein Paket bestätigt wird.

PSH Push - Weißt den Empfänger an, sowohl Eingangs- als auch Ausgangspuffer zu umgehen. In der Regel tut das Feld aber etwas anderes.

RST Reset - Bricht die Verbindung ab.

SYN Synchronize - Initiiert eine neue Verbindung. Der Server sollte mit *SYN+ACK* antworten, oder mit *RST*.

FIN Finish - Gibt die Verbindung frei und zeigt an, dass keine weiteren Daten kommen werden.

Window 16 Bit breit. Die Anzahl an Bytes, die der Sender dieses Pakets bereit ist zu Empfangen. Dient Flow Control.

Checksum 16 Bit breit. Die Prüfsumme über die Kopfdaten. Zur Berechnung wird das Feld auf 0 gesetzt und der Header in 16 Bit Worte unterteilt. Für die weitere Berechnung siehe 10.1.

Urgent Pointer 16 Bit breit. Zeigt, in Kombination mit dem Schalter *URG*, auf das erste nicht-dringende Byte in den Daten.

Options and Padding *(optional)* 0 bis 40 Byte breit. Bis zu 40 Byte an Optionen, wobei verbleibende Bytes bis zum nächsten 32 Bit Wort mit Nullen aufgefüllt werden (Padding).

Data Die Nutzdaten des Paketes.

6.9.2 Demultiplexing

Siehe 6.2.

In TCP wird ein *Socket* durch ein 4-Tupel (Quell IP, Quell Port, Ziel IP, Ziel Port) identifiziert. Ein Host nutzt alle vier Teile, um ein Paket dem korrekten Prozess zuzuordnen. Ein Server kann viele Sockets gleichzeitig unterstützen, jeder mit einem eigenen 4-Tupel.

6.9.3 Sequenznummern, ACKs

Die Sequenznummern werden wie folgt vergeben:

- Jedes Paket enthält die bis dahin versendeten Bytes.
- Damit ist es für den Empfänger möglich zu prüfen, ob alle Bytes erhalten wurden.

Die ACK-Nummern werden wie folgt vergeben:

- Jedes ACK-Paket enthält die bis dahin empfangenen Bytes. Vorherige ACKs werden nicht erneut gesendet.
- Damit ist es für den Sender möglich zu prüfen, ob alle Bytes angekommen sind.
- Ein ACK kann entweder „huckepack“ mit einem Datenpaket versendet werden oder einzeln.

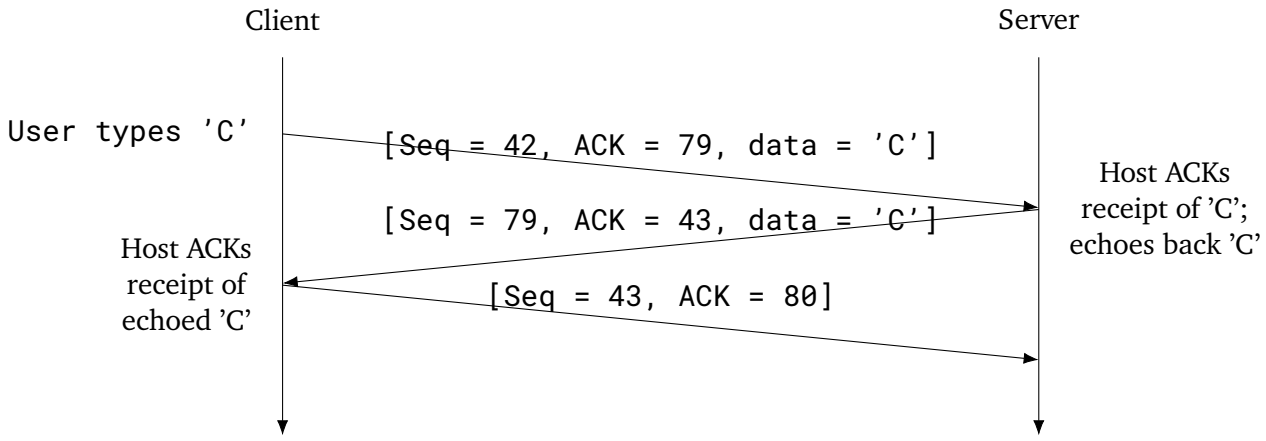


Abbildung 6.9: TCP: Sequenznummern + ACK; Szenario: Telnet

Beispiel

Retransmission

Der Sender überträgt ein Paket erneut, falls:

- Timeout: Ein ACK nicht innerhalb einer erwarteten Zeit eingetroffen ist (sende nur das Paket, dessen ACK fehlt erneut).

Der Empfänger überträgt ein ACK erneut, falls:

- Nach einem gesendeten ACK ein Paket eintrifft, dessen Sequenznummer kleiner ist als die zuletzt gesendete ACK-Nummer, wird das höchstmögliche ACK erneut gesendet.

Fast Retransmit

- Die Timeout Zeit ist schwer „gut“ zu wählen, denn:
  - eine lange Zeit führt zu Verzögerungen und
  - eine kurze Zeit führt zu viel Traffic.
- Verlorene Pakete können mit duplizierten ACKs erkannt werden.
  - Wenn ein Paket verloren wird, werden viele duplizierte ACKs versendet.

→ Wenn der Sender 3 ACKs für die selben Daten empfangen hat, wird davon ausgegangen, dass *alle* Segmente seit diesen Daten verloren gegangen sind → Alle Pakete erneut senden, bevor ein Timeout stattfindet.

6.9.4 Verbindungssteuerung

Verbindungen mit TCP können auf zwei Arten aufgebaut werden:

Aktiv Ein Prozess fragt explizit eine TCP-Verbindung an.

Passiv Ein Prozess informiert TCP, dass die Verbindungen annehmen kann.

- Dies kann ein spezifischer Socket sein oder
- alle eingehenden Verbindungen werden angenommen (Beispiel: Web Server). Sobald eine Anfrage eingeht, wird ein Socket erstellt.

Applikationen wissen nicht, ob eine TCP Verbindung aufgebaut wurde. TCP implementiert die primitive `T-Connect.rsp` nicht.

Verbindungsphasen

- Verbindungsaufbau
  - Drei-Wege Handschlag
  - Einigung auf Window Size und Sequenznummern
- Datentransfer
  - ACKs „huckepack“ auf Datenpaketen
- Verbindungsabbau
  - Bestätigt!
  - Dies verhindert den Verlust bereits gesendeter Daten

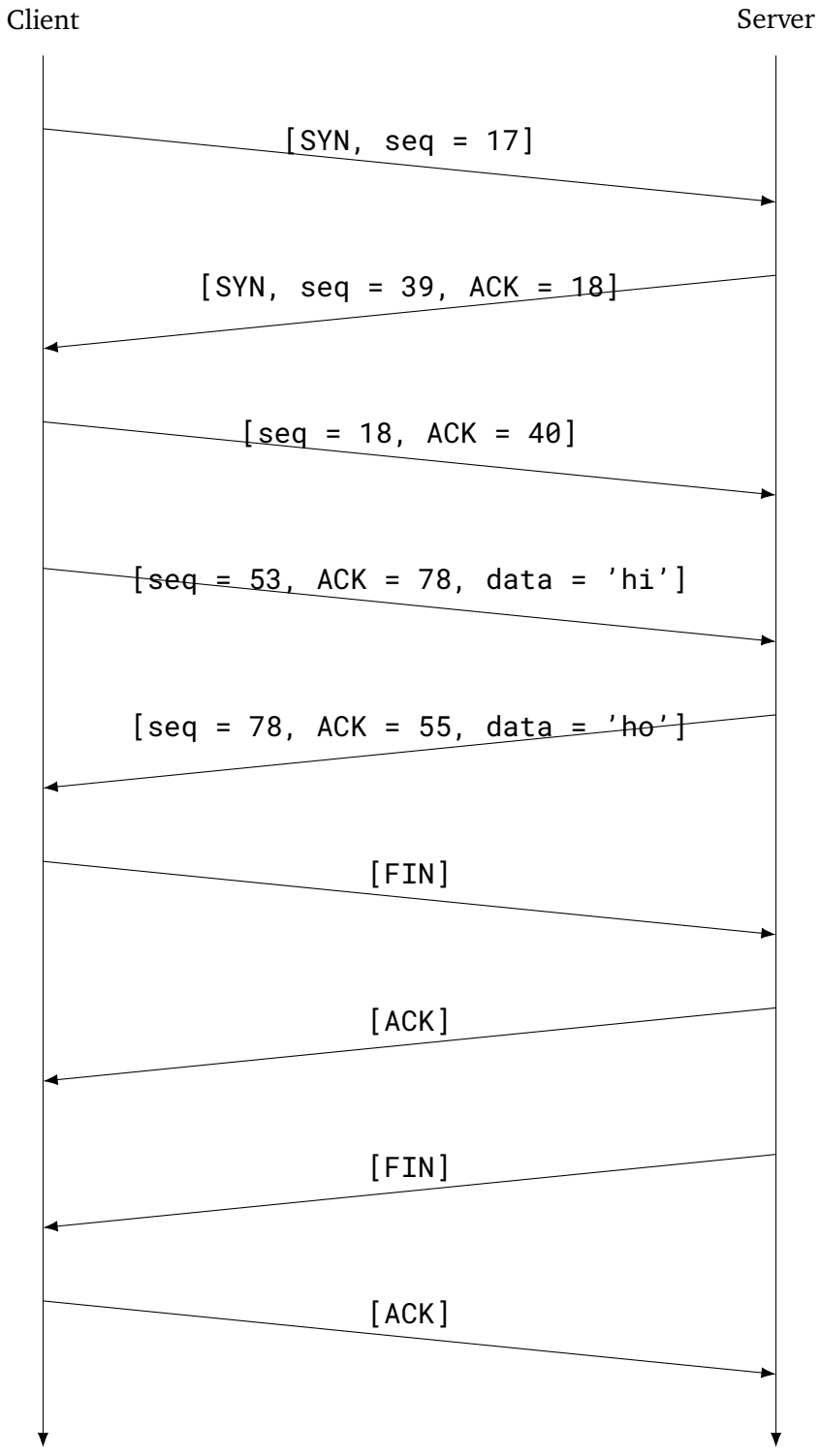


Abbildung 6.10: TCP Verbindungsphasen

6.9.5 Senden/Empfangen Puffer

TCP hält einen Puffer vor für:

- Ausgehende Pakete, um Fehlerkontrolle zu dienen.
- Eingehende Pakete, die noch nicht verarbeitet wurden oder out-of-order empfangen wurden.

6.9.6 Flow Control: Advertised Window

- In TCP teilt der Empfänger mit, wie viel Platz er im Empfangspuffer hat.
  - Position des nächsten erwarteten Bytes: *NextByteExpected*
  - Position des zuletzt gelesenen Bytes: *LastByteRead*
  - Platz im Puffer: $(\textit{NextByteExpected} - 1) - \textit{LastByteRead}$
  - Maximaler Pufferspeicherplatz: *MaxRecvdBuffer*
  - Ausgeschriebenes Pufferfenster: $\textit{MaxRcvdBuffer} - ((NextByteExpected - 1) - LastByteRead)$
- Das ausgeschriebene Pufferfenster (*Advertised Window*) limitiert die Anzahl Daten, die der Sender schicken wird.
  - Der TCP Sender sorgt dafür, dass gilt: $\textit{LastByteSent} - \textit{LastByteAcked} \leq \textit{AdvertisedWindow}$

$\iff$ $\textit{EffectiveWindow} = \textit{AdvertisedWindow} - (\textit{LastByteSend} - \textit{LastByteAcked})$

Nagle-Algorithmus (Self-Clocking)

- Das Eintreffen eines ACKs is ein Indikator dafür, dass neue Daten gesendet werden können.
- Trifft ein ACK für ein kleines Paket ein und es würde direkt ein neues, kleines, Paket versendet werden, würde das Netzwerk mit kleinen Paketen überlastet werden („silly window syndrome“).
- Dies wird durch den Nagle-Algorithmus verhindert.

```
when (Application produces Data) {
    if (AvailableData >= MaxSegmentSize && AdvertisedWindow >= MSS) {
        send full segment
    } else if (UnACKed Data in flight) {
        buffer data
    } else {
        send segment
    }
}
```

Abbildung 6.11: Nagle-Algorithmus

Nagle-Algorithmus

6.9.7 Congestion Control

- TCP nutzt implizites Feedback (bspw. durch den RED-Algorithmus) für Congestion Control.
- Es existieren einige Vorschläge für explizites Feedback, diese sind aber nicht Teil des TCP-Standards.
- Erwartung: Congestion ist der einzige wichtige GRund für verlorene Pakete.

- TCP nutzt ein Window Basiertes Verfahren für Congestion Control.
  - Der Sender passt auf, dass nicht mehr Daten als möglich versendet werden.
  - Es werden nur Daten gesendet, wenn $LastByteSent - LastByteAcked \leq CongestionWindow$ gilt.

ACK-Clocking

Annahme: TCP hat auf irgendeine Weise die korrekte Größe des Congestion Windows bekommen. Außerdem wurde bereits die maximale Anzahl Daten versendet.

- Neue Daten dürfen gesendet werden, wenn:
  - das Netzwerk wieder genug freie Ressourcen hat.
  - Dies ist der Fall, wenn andere Pakete das Netzwerk verlassen haben.
  - Durch eingehende ACKs wird der Sender hierüber informiert.
- Somit dient ein ACK nicht nur der Empfangsbestätigung, sondern auch der Erlaubnis, neue Pakete zu senden.
- Gute Nachricht: Eintreffendes ACK $\rightarrow$ Congestion Window vergrößern.
- Schlechte Nachricht: Kein ACK, Timeout $\rightarrow$ Congestion Window verkleinern.

Additive/Multiplikative Erhöhung/Verringerung

Da ein überlastetes Netzwerk sehr schlecht ist, muss darauf schnell reagiert werden. Außerdem sollte bei guten Übertragungen die Rate erhöht werden, damit sich das Verfahren auf eine gute Rate einpendeln kann.

Dies kann *multiplikativ* (die Last um 50% reduzieren) oder *additiv* (die Last um einen bestimmten Wert erhöht) stattfinden.

Folgende Kombinationen für Erhöhung/Verringerung wären möglich:

AIAD Additive Increase, Additive Decrease
- Reagiert nicht schnell genug auf Congestion im Netzwerk.

AIMD Additive Increase, Multiplicative Decrease
- Reagiert schnell auf Congestion und
- Erhöht die Senderate moderat.
- Somit ist **AIMD** die einzige gute Option!

MIAD Multiplicative Increase, Additive Decrease
- Reagiert nicht schnell genug auf Congestion im Netzwerk.

MIAM Multiplicative Increase, Multiplicative Decrease
- Reagiert schnell auf Congestion, aber:
- Überlastet das Netzwerk sehr schnell, da mit MI schnell über das Ziel hinaus geschossen wird.

Additive Increase erhöht das Congestion Window wie folgt:

$$\text{Increment} = \text{MSS} \cdot \frac{\text{MSS}}{\text{Congestion Window}}$$
$$\text{Congestion Window} = \text{Congestion Window} + \text{Increment}$$

Allerdings kann TCP hiermit schnell überreagieren, wenn Pakete aus anderen Gründen nicht ankommen (bspw. Verbindungsfehler). In Kabelnetzen passiert dies selten, kann aber zu großen Problem führen in Kabellosen Netzen.

Slow Start

Slow Start ist eine Methode, um TCP Verbindungen schnell auf ein vernünftiges Congestion Window zu heben, da dies mit AI sehr lange dauert.

- In der *Slow Start Phase* wird das Congestion Window bei jedem eintreffenden ACK verdoppelt.
- Geht auch nur ein ACK verloren (das Netzwerk ist überlastet), geht TCP in die *Congestion Avoidance Phase* über.
  Diese entspricht dem normalen TCP Congestion Control mit AIMD.

Zusammenfassung Congestion Control

- Ist das Congestion Window kleiner als ein bestimmter Threshold (Congestion) und der Sender ist in der Slow Start Phase, so wächst das Window exponentiell.
- Ist das Congestion Window größer als ein bestimmter Threshold (Congestion), wechselt der Sender in die Congestion Avoidance Phase und das Window wächst linear.
- Tritt ein dreifaches ACK auf, so wird der Threshold auf die Hälfte des Congestion Window gesetzt und das Congestion Window wird auf den Threshold gesetzt (das Window wird halbiert).
- Tritt ein Timeout auf, so wird der Threshold auf die Hälfte des Congestion Window gesetzt und das Congestion wird auf eine MSS gesetzt.

6.10 Stream Control Transmission Protocol (SCTP)

Das *Stream Control Transmission Protocol* (SCTP) ist:

- Nachrichten-Orientiert (wie UDP), bietet aber Flow Control, Congestion Control, Fehlerkontrolle und
- stellt die Zustellung und die korrekte Reihenfolge von Nachrichten sicher.

Bisher ist SCTP noch nicht in Betriebssystemen verfügbar, es existieren aber Libraries für den Userspace.
Features:

- Flexible Nachrichten-Orientierte Streams
  - Jeder Nachricht in einem Stream wird eine Sequenznummer zugewiesen, um die Reihenfolge sicherzustellen.

  - Der Empfänger kann Sortierung und Zustellungssicherung aktivieren/deaktivieren.
  - Der Sender kann Fragmentierung und Nagle aktivieren/deaktivieren.
- Multi-Streaming: Mehrere unabhängige Streams können mittels Multiplexing über eine Verbindung gesendet werden.
  - Unabhängige Sortierung.
  - Ein Browser kann bspw. Text in einem Stream und Bilder in einem anderen versenden.
- Multi-Homing: Sowohl Sender als auch Empfänger können beide mehrere IP Adressen haben.

6.11 Datagram Congestion Control Protocol (DCCP)

Das *Datagram Congestion Control Protocol* (DCCP) ist:

- für Applikationen, welche unzuverlässige, schnelle Übertragung benötigen wobei
- Congestion Control benötigt wird, was DCCP bietet.

Im Grunde ist DCCP:

- TCP ohne Streams und Verlässlichkeit oder
- UDP mit Congestion Control, Handshakes und ACKs (optional).

Features:

- Unzuverlässiger Nachrichtentransport
- Separate Header/Data Prüfsummen
- Nachrichten-Orientiert
- Verlässliche ACKs für Congestion Control
- Multi-Homing wie bei SCTP

6.12 Realtime Transfer (Control) Protocol (RTP/RTCP)

Das *Realtime Transfer Protocol* (RTP) ist:

- primär als Multimedia Protokoll gedacht, da es
- sehr gut auf große Netzwerke skaliert.

Features:

- Sequenznummern und Zeitstempel
- Eindeutige Quell/Sitzungs-ID (SSRC oder CSRC)
- Verschlüsselung

- Codec

Die Daten werden dabei über RTP versendet und die Kontrolle findet über das *Realtime Transfer Control Protocol* (RTCP) statt.

RTP baut auf UDP auf, somit gibt es außerdem alle Features von UDP:

- Ports, IP Adressen
- Paketnummerierung,
- ...

6.13 QUIC

Quick UDP Internet Connections (QUIC) ist ein von Google vorgestelltes Protokoll, welches aktuell in einem Standardisierungsprozess beim der IETF ist.

QUIC unterstützt sehr effiziente Verschlüsselung bei geringer Latenz und baut auf UDP auf.

Protokolle, welche auf UDP aufbauen haben den Vorteil, dass Firewalls mit diesen arbeiten können und keine speziellen Regeln gebaut werden müssen.

Da Verschlüsselung schnell ausgetauscht werden muss (z.B. bei Security-Updates), läuft QUIC im Userspace eines Systems und nicht im Kernel, wodurch nicht das OS geupdated werden muss, um QUIC zu updaten.

7 Warteschlangentheorie

Die Warteschlangentheorie beschäftigt sich mit folgenden Themen in einem Warteschlangen/Server-System (ein Server arbeitet die Anfragen ab):

1. *Ankunft* Wie viele Anfragen treffen ein?
   Üblicherweise als *Wahrscheinlichkeitsverteilung über Zeitintervalle* angegeben
2. *Service* Wie lange dauert es, Anfragen zu bearbeiten?
   Servicezeit, Wahrscheinlichkeit von einer Bearbeitungszeit x
3. Wie viel *Platz* ist in der *Schlange*?
4. Wie *viele Server* gibt es?
5. Was sind die *Abfertigungsstrategien* und wie werden die Schlangen bearbeitet?
   First Come, First Serve, Shortes Job First, Earliest Deadline First, Priority Queue, ...
6. Welche Metriken sind interessant?
   - Wartezeit pro Anfrage
   - Anzahl Anfragen im System
   - Anzahl unbearbeiteter Anfragen
   - Systemauslastung verteilt über den Tag/die Woche/das Jahr/...

7.1 Definitionen, Notationen

C_n (n-te Anfrage im System)

$\alpha(t)$ (Anzahl Anfragen im Interval $[0, t)$)

$\delta(t)$ (Anzahl Abfertigungen im Interval $[0, t)$)

$N(t) = \alpha(t) - \delta(t)$ (Anzahl Anfragem im System zum Zeitpunkt t)

τ_n (Ankunftszeit von Anfrage C_n)

$t_n = \tau_n - \tau_{n-1}$ (Zeit zwischen Ankunft von C_n und C_{n-1})

$x_n =$ (Servicezeit von Anfrage C_n)

$w_n =$ (Wartezeit von Anfrage C_n)

$s_n = x_n + w_n$ (Systemzeit (Gesamtzeit) von Anfrage C_n)

$\lambda_t = \frac{\alpha(t)}{t}$ (Durchschnittliche Ankunftsrate im Interval [0, t))

$\lambda = \lim_{t \to \infty} \lambda_t$ (Durchschnittliche Ankunftsrate)

$\bar{t} = \frac{1}{\lambda}$ (Durchschnittliche Zeit zwischen Ankünften)

$\gamma(t) = \sum_{i=1}^{\alpha(t)} T(i) = \int_0^t N(\tau)\, d\tau$ (Gesamte von Anfragen im System verbrachte Zeit)

$T_t = \frac{\gamma(t)}{\alpha(t)}$ (Durchschnittliche Systemzeit pro Anfrage im Interval [0, t))

$T = \lim_{t \to \infty} T_t$ (Durchschnittliche Systemzeit pro Anfrage)

$N_t = \frac{1}{t} \cdot \int_0^t N(\tau)\, d\tau = \frac{\gamma(t)}{t}$ (Durchschnittliche Anzahl Anfragen im Interval $[0, t)$)

$N, \bar{N} = \lim_{t \to \infty} N_t$ (Durchschnittliche Anzahl Anfragen)

μ_k (Durchschnittliche Abfertigungsrate pro Anfrage bei k Anfragen im System)

$P_k(t)$ (Wahrscheinlichkeit, dass sich zum Zeitpunkt t k Anfragen im System befinden)

$p_k = \lim_{t \to \infty} P_k(t)$ (Wahrscheinlichkeit, dass sich k Anfragen im System befinden)

δ (Wahrscheinlichkeit, dass sich Elemente in der Warteschlange befinden)

7.1.1 Diagramme

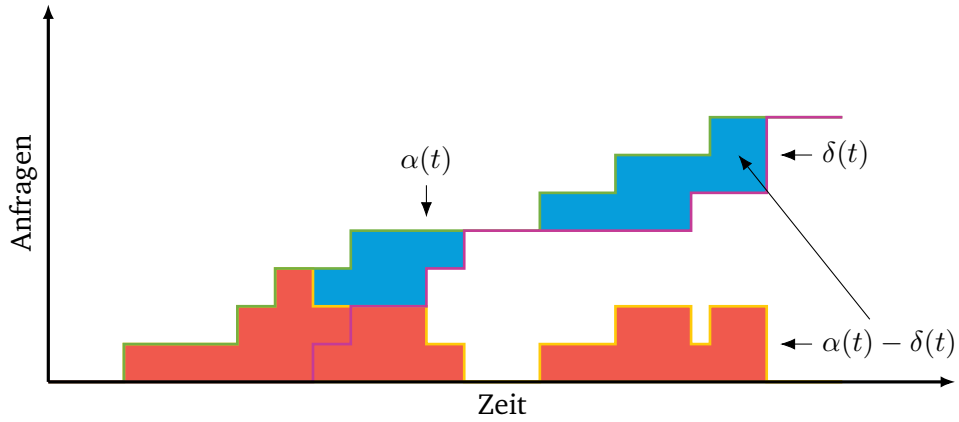


Abbildung 7.1: Warteschlangentheorie: Zeitdiagramm

Zeitdiagramme

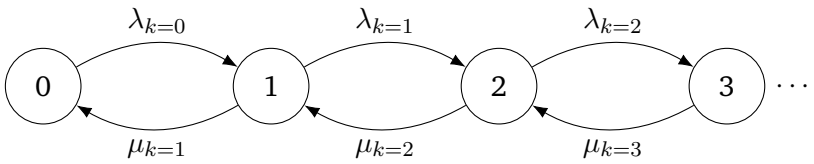


Abbildung 7.2: Warteschlangentheorie: Transitionsdiagramm

Transitionsdiagramme

7.1.2 Ankünfte/Ausgänge

Die Ein- und Ausgangsprozesse stellen stochastische Prozesse, weshalb diese stochastischen Wahrscheinlichkeitsverteilungen folgen. Hierdurch sind die Probleme deutlich einfacher zu untersuchen (in der Realität sind die Prozesse leider eher zufällig…).

7.1.3 Auslastung

- Der *Auslastungsfaktor* wird durch ρ angegeben.
- Er beschreibt das Verhältnis zwischen der Ankunftsrate und der Servicerate.
- In einem Einzel-Server-System gilt: $\rho = \frac{\lambda}{\mu} = \lambda\bar{x}$ (Achtung: Dies gilt nicht allgemein!)
- Es sollte $0 \leq \rho < 1$ gelten, da sonst mehr Anfragen eingehen als abgefertigt werden.

7.2 Littles Law

Es gilt (mit durchschnittlicher Anzahl Anfragen N, durchschnittlicher Ankunftsrate λ und durchschnittlicher Systemzeit T):

$$N = \lambda \cdot T$$

7.3 Stochastische Prozesse

Ein Stochastischer Prozess ist eine Gruppe von Zufallsvariablen $X(t)$, wobei die Zufallsvariablen mit der Zeit indiziert sind.

7.3.1 Markov Prozess

Ein stochastischer Prozess $\{X_n\}$ mit diskretem Zustandsraum ist ein *Markov Prozess*, wenn die Wahrscheinlichkeit, dass x_{n+1} der nächste Zustand ist nur von dem aktuellen Zustand x_n abhängt.

Außerdem fordert ein Markov Prozess:

- Die Prozesse sind *homogen*, das heißt die Transitionen hängen nicht von der Beobachtungszeit ab.
- Die Prozesse sind *speicherfrei*, das heißt der nächste Zustand hängt nicht von der Zeit ab, die im aktuellen Zustand verbracht wurde.
- Die Servicezeiten sind exponentiell Verteilt.

7.3.2 Birth-Death Prozess

Ein Birth-Death ist ein spezieller Markov Prozess, bei dem Zustandsübergänge nur zwischen benachbarten Zuständen stattfinden können.

Beispiel: Sei der Zustandsraum ein Ganzzahlraum, so sind in einem Zustand k Transitionen in die Zustände $k-1$, k und $k+1$ möglich.

Terminologie:

- Eine Transition $k \to (k+1)$ wird *Birth* genannt.
- Eine Transition $k \to (k-1)$ wird *Death* genannt.
- Birth-Rate λ_k: Birth-Rate bei einer Population von k.
- Death-Rate μ_k: Death-Rate bei einer Population von k.

Birth-Death Prozesse bilden die Grundlage der Warteschlangentheorie.

Ein stochastischer Prozess ist ein Birth-Death Prozess, wenn:

- der Prozess ein homogener Markov Prozess mit Zuständen $0, 1, 2, \cdots$ ist,
- Births und Deatchs unabhängig sind und
- die folgenden Bedingungen wahr sind ($\lim_{\Delta t \to 0} o(\Delta t) = 0$):

$$\begin{aligned} P(1 \text{ Birth in } (t, t+\Delta t) \mid k \text{ in pop.}) &= \lambda_k \Delta t + o(\Delta t) \\ P(1 \text{ Death in } (t, t+\Delta t) \mid k \text{ in pop.}) &= \mu_k \Delta t + o(\Delta t) \\ P(\text{kein Birth oder Death in } (t, t+\Delta t) \mid k \text{ in pop.}) &= 1 - (\lambda_k + \mu_k)\Delta t + o(\Delta t) \end{aligned}$$

Gleichgewicht (Equilibrium)

Ein Birth-Death Prozess ist im Gleichgewicht (engl. Equilibrium), wenn genau so viele Anfragen eingehen wie Ausgehen (d.h. es befinden sich keine Prozesse in der Warteschlange).

Kommentar: Bei diesem Abschnitt bin ich mir unsicher, ob die Informationen so korrekt sind.

Poisson Process

Sei ein Birth Prozess gegeben, d.h. $\mu_k = 0$ für alle k. Sei außerdem $\lambda_k = \lambda$ für alle k.

Damit ergibt sich für

$$P_k(t) = \frac{(\lambda t)^k}{k!} \cdot e^{-\lambda t}$$

eine Poisson-Verteilung.

7.4 Queueing Problems

Die Zentrale Frage der Warteschlangentheorie ist, **mit welcher Wahrscheinlichkeit $P_k(t)$ sich zu einem Zeitpunkt t k Anfragen im System befinden?**

Dies führt zu folgender Wahrscheinlichkeitsformel für $k \geq 1$:

$$\begin{aligned} P_k(t + \Delta t) = & P_k(t) p_{k,k}(\Delta t) \\ & + P_{k-1}(t) p_{k-1,k}(\Delta t) \\ & + P_{k+1}(t) p_{k+1,k}(\Delta t) \\ & + o(\Delta t) \end{aligned}$$

Für $k = 0$ ergibt sich:

$$\frac{dP_0(t)}{dt} = -\lambda_0 P_0(t) + \mu_1 P_1(t)$$

7.5 Kendall-Notation

Die von David George Kendall eingeführt Notation (*Kendall-Notation*) beschreibt bestimmte Parameter eines Warteschlangensystems.

Das System wird dabei durch ein Tupel A/S/m/N/K/SD beschrieben:

A Ankunftsprozess

S Serviceprozess

m Anzahl von Servern

N Anzahl Plätze im System (nicht angegeben $\implies$ N $= \infty$)

K Populationsgröße

SD Abfertigungsdisziplin

Dabei werden die folgenden Kurzbezeichnungen für Ankunfts- und Serviceprozess verwendet:

M Exponentialverteilung (Markovsche Verteilung)

D Konstante (Deterministische Verteilung)

G, GI Beliebige Verteilung

7.5.1 M/M/1 Warteschlangen

Eine M/M/1 Warteschlange ist das einfachste Warteschlangensystem mit einer Schlange und einem Server, wobei alle Birth- und Death-Raten gleich sind.

Für die Stabilität des Systems muss

$$\lambda < \mu$$

gelten.

Formeln

$$\begin{aligned}
\lambda_k &= \lambda \\
\mu_k &= \mu \\
\rho &= \frac{\lambda}{\mu} \\
p_k &= (1-\rho)\rho^k \\
\bar{N} &= \frac{\rho}{1-\rho} \\
T &= \frac{\frac{1}{\mu}}{1-\rho}
\end{aligned}$$

7.5.2 M/M/m Warteschlangen

Bei einer M/M/m Warteschlange gibt es eine Warteschlange und m Server, womit länger andauernde Jobs durch andere Server ausgeglichen werden können.

Für die Stabilität des Systems muss

$$\rho < 1$$

gelten.

Formeln

$$
\begin{aligned}
\lambda_k &= \lambda \\
\mu_k &= \min(k\mu, m\mu) \\
\rho &= \frac{\lambda}{m\mu} \\
p_k &= \begin{cases} p_0 \frac{(m\rho)^k}{k!} & k \leq m \\ p_0 \frac{m^k \rho^k}{m!} & k > m \end{cases} \\
\bar{N} &= \bar{N} = m\rho + \frac{\rho\delta}{1-\rho} \\
T &= T = \frac{1}{\mu}(1 + \frac{\delta}{m(1-\rho)}) \\
\delta &= p_0 \frac{(m\rho)^m}{m! \cdot (1-\rho)}
\end{aligned}
$$

7.5.3 M/M/1/N Warteschlangen

Eine M/M/m/N Warteschlange ist eine M/M/1 Warteschlange, wobei der Platz in der Warteschlange limitiert ist. Ist die Anzahl der Anfragen größer als N, werden neue Anfragen verworfen.

Formeln

$$
\begin{aligned}
\lambda_k &= \begin{cases} \lambda & 0 \leq k \leq N \\ 0 & k > N \end{cases} \\
\mu_k &= \begin{cases} \mu & k \leq N \\ \text{Fehler} & k > N \end{cases} \\
\rho &= \frac{\lambda}{\mu} \\
p_k &= \begin{cases} \frac{1-\rho}{1-\rho^{N+1}} \rho^k & 0 \leq k \leq N \\ 0 & \text{sonst} \end{cases} \\
\bar{N} &= \frac{\rho}{1-\rho} - \frac{(N+1)\rho^{N+1}}{1-\rho^{N+1}} \\
T &= \frac{\bar{N}}{\lambda'} \\
\lambda' &= \lambda \sum_{k=0}^{N-1} p_k
\end{aligned}
$$

7.5.4 m $\times$ M/M/1 vs. M/M/m

Wenn ein Wechsel der Schlange möglich ist, wird ein fünffaches M/M/1 System zu einem M/M/m System und es macht somit keinen Unterschied.

Ist ein Wechsel nicht möglich, so ist ein M/M/m System besser (dies ist in der Information meist der Fall).

8 Multicast

Manchmal ist es nötigt, die gleichen Daten an viele Empfänger zu versenden. Hierzu dient *Multicast*. Beispielhafte Anwendung sind IPTV, Videokonferenzen, ...

Je nach Implementierung von Multicast wird der Traffic und der Routingaufwand reduziert.

Es ist Möglich, Unicast und Broadcast als eine spezielle Form von Multicast aufzufassen.

8.1 Implementierungsarten

Zur Implementierung von Multicast gibt es verschiedene Möglichkeiten, wo die Pakete dupliziert werden:

- Quellduplikation (die Pakete werden beim Sender dupliziert)
- Netzwerkduplikation (die Pakete werden von den Routern dupliziert)
- Applikationsduplikation (die Pakete werden von den Applikationen selbst dupliziert)

8.1.1 Multicast mittels Unicast

- Der Sender dupliziert alle Pakete.
- Die Pakete werden über Unicast an alle Ziele verteilt.
- Problematik: Hohe Netzwerkauslastung, viel Routingaufwand

8.1.2 Netzwerk Multicast

- Der Sender sendet nur ein Paket.
- Die Router innerhalb des Netzwerkes duplizieren die Pakete Ad-Hoc (nur wenn Splitting benötigt wird).
- Problematik: Der Routingaufwand wird stark reduziert.

8.1.3 Applikations Multicats

- Der Sender sendet nur ein Paket.
- Die Empfangenden Applikationen verteilen die Pakete mittels Unicast zwischen sich.
- Problematik: Der Empfänger wird stark belastet.

8.2 Multicast Gruppen

- Im *Internet Multicast Service Model* werden die IP Adressen der Empfänger zu einer *Multicast Gruppe* zusammengefasst.
- Dieser Gruppe wird eine IP aus der Klasse D zugewiesen (Mutlicast Adressen).
- Die Router leiten Pakete weiter zu den Hosts, die einer Gruppe „beigetreten“ sind.

8.2.1 Addressing

- Auf Netzwerk-Ebene gibt es keine Möglichkeit zu erkennen, ob ein Host zu einer Multicast Gruppe gehört oder nicht.
- Es ist Schwierig, die korrekten Pakete den korrekten Empfängern zuzustellen.
- In IPv4 wurde der Adressbereich `224.0.0.0` bis `239.255.255.255` für Multicast Adressen reserviert.
- Diese Adressen können nur als Ziel genutzt werden (sie können nicht senden) und dienen als Ziel für Multicasts.
- Die Router müssen hierfür besondere Routing-Algorithmen implementieren.

8.2.2 Beitritt zu Multicast Gruppen

- Jeder kann einer Multicast Gruppe beitreten und diese wieder verlassen.
- Jeder kann Daten an eine Multicast Gruppe senden.
- Hierzu wird Infrastruktur benötigt, welche die Mitgliedschaften verwaltet.
- Lokal kommt hierfür kommt hier IGMP zum Einsatz, wobei der beitretende Host den lokale Router über das Vorhaben informiert.
- In weiten Netzen kommen hier verschiedene Protokolle zum Einsatz (DVMRP, MOSPF, PIM) und die Router kommunizieren untereinander.

8.2.3 IGMP

Das *Internet Group Management Protocol* (IGMP) dient der lokalen Verwaltung von Multicast Gruppen.

- Ein beitretender Host sendet ein `IP_ADD_MEMBERSHIP`-Report, wenn eine Applikation einer Multicast Gruppe beitritt.
- Es ist kein explizites Verlassen der Gruppe nötig.
- Ein IGMP-Router sendet in regelmäßigen Intervallen IGMP Queries um zu prüfen, ob noch alle Hosts da sind. Diese müssen auf die Pakete antworten.
- Antwortet ein Host nicht mehr, so wird dies als implizites Verlassen der Gruppe interpretiert.

8.3 Übertragung/Zustellung

- Lokale Zustellung
  - Im lokalen Netzwerk wird das Paket mittels Ethernet/WLAN/...-Broadcast durch das Netzwerk gesendet.
  - Empfangende Hosts teilen der IP-Schicht mit, dass Pakete an eine bestimmte Gruppe empfangen wollen.
- Zustellung über das Internet
  - Die Router müssen ein explizites Multicast Protokoll implementieren, welches die Konstruktion von Zustellungsbäumen erlaubt und welches Multicast Pakete weiterleitet.
  - Außerdem muss jeder Router ein IGMP implementieren, um Gruppen und Mitgliedschaften im lokalen Netz verwalten zu können.

8.4 Routing

Das Ziel von Multicast Routing ist, einen Baum zu finden, der alle Knoten verbindet. Dabei gibt es folgende Möglichkeiten:

- Quell-basierter Baum: Jeder Sender verwaltet seinen eigenen Baum.
- Geteilte Bäume: Alle Mitglieder nutzen den gleichen Baum (Minimaler Spannbaum (Steiner Baum), Core Based Trees (CBT))
- Gruppen-Geteilte Bäume: Alle Mitglieder einer Gruppe nutzen den gleichen Baum (Shortes Path Tree (STP), Reverse Path Forwarding (RPF))
- Flooding: Kein explizites Routing.

8.4.1 Flooding

- Das Netzwerk wird mit Multicast Paketen geflutet.
- Viel Netzwerkauslastung.
- Hoher Aufwand für die Router, da sich gemerkt werden muss, welche Pakete schon vorbei gekommen sind.

8.4.2 Spannbäume

- Zwischen zwei Routern sollte es nur einen aktiven Pfad geben.
- Die Pakete landen in keiner Schleife, werden aber jedem Knoten zugestellt.
- Es sollte der kürzeste/effizienteste Pfad genutzt werden.
- Ähnlich einem Minimalen Spannbaum, aber:
  - Nicht jeder Router ist ein Blatt.
  - Es nicht nicht von vornherein bekannt, über welchen Router der kürzeste Pfad geht.

8.4.3 Geteilter Baum: Steiner Baum

- Nutzt Minimale Spannbäume.
- Nachteile:
  - Jeder Knoten muss bekannt sein.
  - Das Problem ist NP-Schwer, aber es gibt sehr gute Heuristiken.
- Aufgrund der obigen Nachteile werden Steiner Bäume nicht in der Praxis eingesetzt.

8.4.4 Core Based Trees

- Ein Knoten im Netzwerk stellt den *Core* dar.
- Alle neuen Knoten registrieren sich bei dem Mittelpunkt, welche aus dem vom Paket gewählten Paket einen Spannbaum berechnet.
- Nach der Registrierung teilt der Core dem neuen Knoten den Spannbaum mit.
- Knoten, die auf dem Spannbaum liegen, schicken die Pakete einfach über den Spannbaum.
- Knoten, die nicht auf dem Spannbaum liegen, senden die Pakete an den Core, welcher die Pakete dann an den Spannbaum verteilt.
- Vorteile/Nachteile:
  - CBT nutzt Bandbreite und Ressourcen sehr effizient.
  - Im Vergleich zu MST ineffizient, aber deutlich einfacher.
  - Es kann zu Überlastung am Core kommen.
  - Ein einzelner, geteilter, Baum kann die Bäume nicht so effizient erstellen wie viele Bäume.

8.4.5 Baum mit kürzesten Pfaden

- Jeder Host berechnet einen eigenen Baum mittels Dijkstra.

8.4.6 Reverse Path Forwarding

- Verlässt sich darauf, das eingehende Pakete auf dem kürzesten Pfad gekommen sind.
- Flutet das Netz mit Paketen, die auf dem kürzesten Pfad gekommen sind.
- Alle anderen werden verworfen.

Pruning

- Der Baum enthält Unterbäume, welche keine Gruppenmitglieder sind.
- Datagramme müssen nicht zu diesen geleitet werden.
- bei Detektion eines solchen Falls werden Prune-Nachrichten an die übergeordneten Router versendet, damit diese keine Pakete mehr an den Unterbaum weiterleiten.

8.5 Verlässliches Multicast (NACK)

- Auch bei Multicast spielt es eine Rolle, das alle Pakete ankommen.
- Allerdings können viele ACKs zu einer „ACK Implosion“ führen, unter der das Netzwerk zusammenbricht.
- Eine andere Methode ist, negative ACKs (NACK) zu versenden, wenn ein Paket nicht angekommen ist.
- Ein Nachbarknoten hält mglw. eine Kopie, weshalb NACKs erst an diese verschickt werden, die die Nachricht dann, sollten sie das Paket nicht mehr vorhalten, weiter leiten.
- Nachteil: Geht das letzte Paket verloren, wird der Verlust nicht bemerkt, da kein NACK gesendet wird.

9 Applikationen/Verteilte Systeme

OSI-Schichten 5 bis 7.

9.1 Ausprägungen eines Applikations-Schicht Protokolls

Ein Protokoll auf Applikations-Schicht definiert die folgenden Dinge:

- Typ der ausgetauschten Nachrichten (wie Anfrage/Antwort, etc.)
- Syntax der Nachrichten
- Semantik der Felder in den Nachrichten
- Regeln wann und wie Nachrichten gesendet und beantwortet werden

9.2 Client-Server Modell

- Das Client-Server Modell ist das üblichste bei verteilten Systemen.
- Ein Nutzer (Client) fragt über Anfragen (Requests) Dinge vom Server an.
- Dieser Antwortet (Response) auf die Anfragen.
- Abhängig von dem untergeordneten Protokoll kann die Verbindung verlässlich (TCP) oder nicht (UDP) sein.
- Es wird zwischen zwei verschiedenen Serverarten unterschieden: Iterativ und Simultan
- Im Hintergrund kann ein Server auch als Client agieren und Daten von anderen Servern anfragen (erweitertes Client-Server Modell)

9.2.1 Iterative Server

- Es wird eine Anfrage zur gleichen Zeit bearbeitet.
- Andere Anfragen werden in eine Warteschlange einsortiert und später bearbeitet.

9.2.2 Simultane Server

- Für jede eingehende Anfrage wird ein eigener Thread gestartet.
- Die Anfragen werden simultan bearbeitet und können zeitgleich beantwortet werden.
- Der Hauptthread wartet auf neue Anfragen.

9.3 Peer-to-Peer (P2P)

- Besserer Name: Dezentralisiertes Verteiltes Netzwerk
- Jeder Knoten in einem Netzwerk kann sowohl Server als auch Client sein.
- Die Knoten kommunizieren untereinander und es ist kein Server nötig (Beispiel: SyncThing).
- Es ist keine spezielle Netzwerkstruktur nötig, diese kann aber über Algorithmen o.ä. simuliert werden.

9.4 Berkeley Socket-Interface API

Die Berkeley Socket-Interface API ist eine Schnittstelle, um verteilte System auf der Applikationsschicht zu implementieren und dient als Kommunikationsmittel mit der Transport-Schicht.

Entwickelt wurde die API für UNIX und ist auch dort integriert. Sie ist sehr simpel gestaltet, unterstützt sowohl verbindungsorientierte Protokolle als auch verbindungslose und macht Netzwerkzugriffe so einfach, als wären es Dateizugriffe. Allerdings ist Netzwerkzugriff natürlich komplexer, weshalb es noch einige Methoden mehr gibt als bei Dateizugriffen.

Eine Verbindung ist immer definiert als das 5-Tupel (Protokoll, Quelladresse, Quellprozess, Zieladresse, Zielprozess).

9.4.1 Verbindungslos

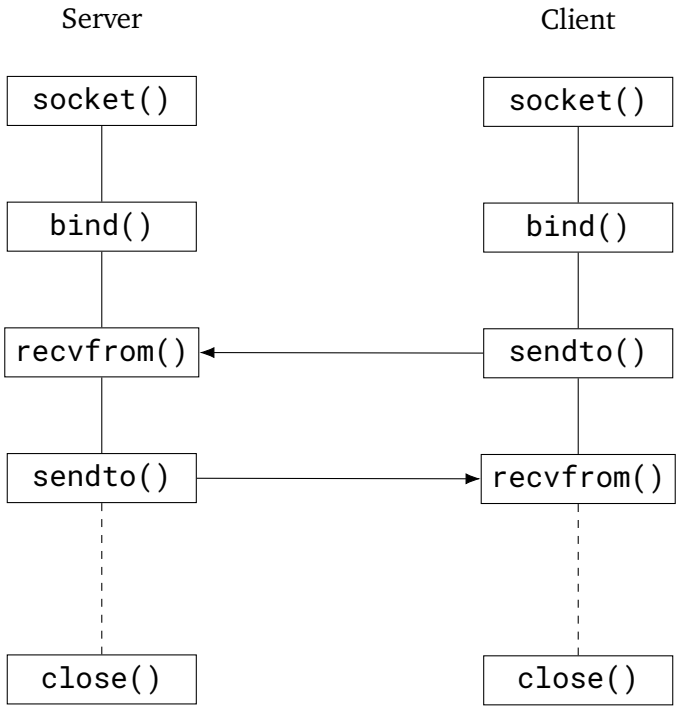


Abbildung 9.1: Berkeley Socket-Interface API: Verbindungslos

9.4.2 Verbindungslos: connect

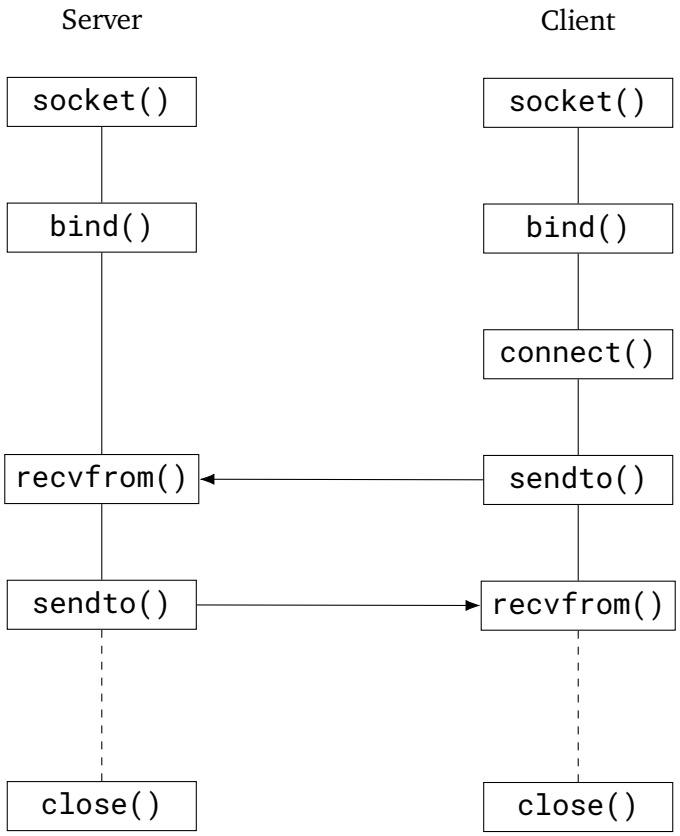


Abbildung 9.2: Berkeley Socket-Interface API: Verbindungslos: connect

9.4.3 Verbindungsorientiert

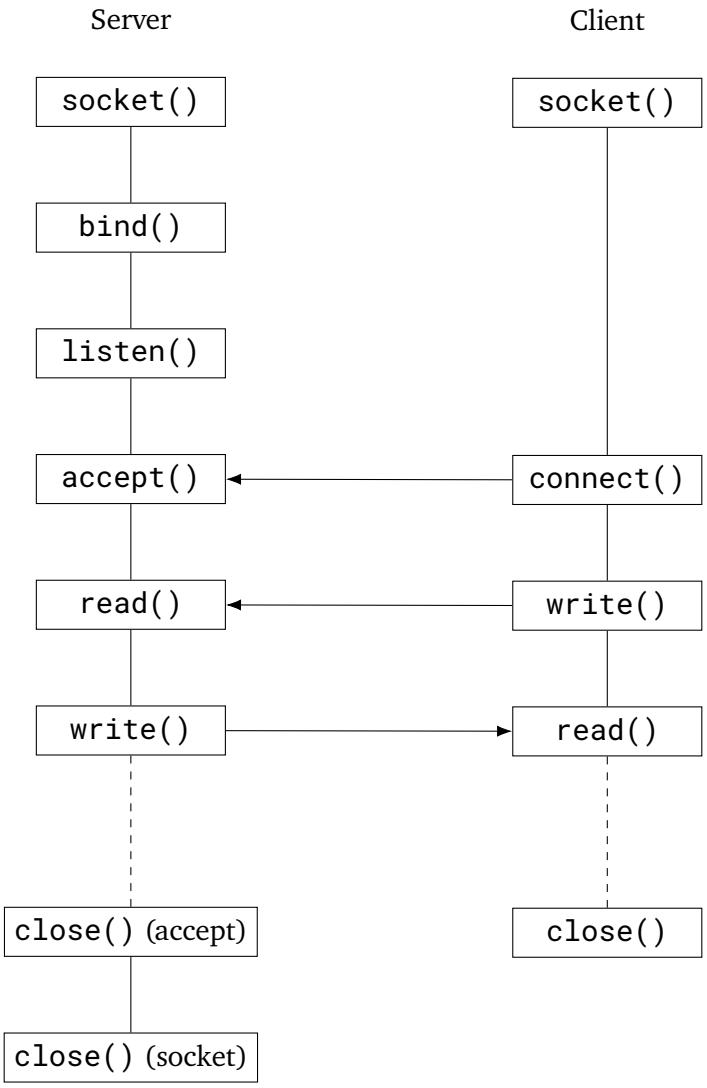


Abbildung 9.3: Berkeley Socket-Interface API: Verbindungsorientiert

9.5 DNS

- Das *Domain Name System* (DNS) wird genutzt, um Domains zu IP Adressen aufzulösen.
- Außerdem können Aliase, Mailserver, uvm. nachgeschaut werden.
- Ferner kann DNS auch zur Lastverteilung genutzt werden (Round-Robin).
- DNS ist eine verteilte Datenbank mit vielen Servern auf Applikations-Ebene.
- Ein zentralisierter DNS-Server wäre schlecht, denn:
  - Single Point of Failure

- Datenaufkommen
- Zentralisierte Datenbank, die nicht einfach editiert werden kann
- Verwaltungsaufwand

$\Longrightarrow$ Zentralisierung skaliert nicht!

9.5.1 Zonen

DNS ist in unterschiedliche *Zonen* aufgeteilt, die an der Root-Zone „befestigt“ sind. Die Root-Zone ist „.“, von dem aus alle Domains hierarchisch aufgebaut werden.

Die Hauptzonen zeigen nur auf die Unterzonen, welche Subdomains und ähnliche Dinge administrieren können.

Die Daten innerhalb einer DNS-Zone liegen als „Resource Records“ vor.

9.5.2 Top-Level Domain, Authorative Server

- Die Hierarchie wird durch verschiedene Komponenten hervorgerufen: Top-Level Domains (TLD), Authorative DNS Servers, ...
- Top-Level Domain (TLD) Server:
  - sind zuständig für `com`, `net`, `de`, ...
  - In Deutschland verwaltet die Denic die Zone `de.`.
- Authorative DNS-Server:
  - Verwalten die Adressen für eine komplette DNS Zone
  - TLD-Server sind beispielsweise Authorative Server, aber auch Server einer Organisation.
  - Die Parent-Zonen zeigen auf die Server, um die Anfragen weiterzuleiten.
  - Es muss mindestens ein primärer und ein sekundärer DNS-Server gesetzt werden (Lastverteilung).

9.5.3 Resolver

Ein Resolver läuft lokal und kümmert sich um die Auflösung an sich. Es werden ausschließlich rekursive Anfragen und keine iterativen Anfragen unterstützt.

Cache Server

- Ein Cache-Server hält sich nicht an die hierarchische Struktur.
- Lokaler Proxy für DNS-Anfragen.
- Cached die Antworten für eine bestimmte Zeit (TTL, Time To Life).
- Wenn die Daten nicht im Cache verfügbar sind, werden andere DNS-Server befragt.

9.5.4 Hierarchische Datenbank

Die Datenbank ist hierarchisch strukturiert, wobei die Domain Rückwärts aufgelöst wird.

Beispiel

- Ein Ein Nutzer möchte `www.amazon.com.` auflösen.
- Er fragt die Root-DNS nach dem DNS-Server für `com.`.
- Er fragt die Auth-DNS nach dem DNS-Server für `amazon.com.`.
- Er fragt die unteren DNS-Server nach der Adresse für `www.amazon.com.`.

Root Name Servers

Es gibt auf der Welt 13 Root DNS-Server, die die Anfragen an weitere Nameserver weiterleiten.
Diese übernehmen allein die Delegation an die Authorative Server von TLDs.

9.5.5 Nachrichtenformat

Eine DNS-Anfrage enthält, neben Quell-IP und Ziel-IP, die folgenden Daten:

Query/Response Flag Ob das Paket eine Antwort oder eine Anfrage darstellt.

Operation Code Spezifiziert den Typ der Nachricht (Query, Status, Notify, Update)

Authorative Answer Flag Gibt an, ob die Antwort von einem Authorative Server gekommen ist.

Truncation Flag Gibt an, ob die Nachricht zugeschnitten wurde, sodass die Antwort in ein UDP-Paket passt.

Recursion Desired Flag Gibt an, ob rekursive Anfragen gewünscht sind.

Recursion Available Flag Gibt an, ob rekursive Anfragen möglich sind.

9.5.6 Records

Ein Record ist ein 4-Tupel von folgenden Daten: (Name, Wert, Typ, TTL):

Name Der Name des Records.

Wert Der Wert des Records.

Typ Gibt an, wie der Name und der Wert des Records zu interpretieren ist:

A Der Name ist der Hostname, der Wert die IP Adresse.
Beispiel: `(dmken.com, 94.249.254.72, A, 300)`

MX Der Wert ist die Domain des Mailservers, der für die Domain zuständig ist, welche im Namen steht.
Beispiel: `(dmken.com, mail.dmken.com., MX, 300)`

NS Der Name ist eine Domain, der Wert die IP Adresse eines anderen Authorative DNS-Server, der für die Zone zuständig ist.
Beispiel: `(dmken.com, ines.ns.cloudflare.com., NS, 300)`

CNAME Der Wert ist ein Alias für den Namen.
Beispiel: `(git.dmken.com, falcon.srv.dmken.com., CNAME, 300)`

TTL Die Zeit, die ein Record im Cache bleiben soll in Sekunden.

9.5.7 Anfragen

Iterative Anfragen

Bei iterativen Anfragen sendet der angefragte Server die IP Adresse zurück, welche für die Domain zuständig ist.

Der Client muss nun iterativ alle Server anfragen, bis der zuständige gefunden wurde.

Rekursive Anfragen

Bei rekursiven Anfragen sendet der Server direkt die IP Adresse zurück und fragt im Hintergrund die zuständigen Server an.

Der Client muss nur eine Anfrage absetzen und erhält direkt eine Antwort.

9.6 HTTP

Das *Hyptertext Transport Protocol* (HTTP) wird primär für Webseiten genutzt und transportiert größtenteils Text.

- HTTP implementiert das Client-Server Modell, wobei der
- Client der Browser ist und der
- Server der Webserver, der auch Anfragen antwortet.
- Üblicherweise nutzt HTTP TCP und erstellt für jede Anfrage einen neuen Socket (Nicht-Persistent HTTP).
- HTTP ist zudem Zustandsfrei, was das Protokoll stark vereinfacht.

9.6.1 Persistent/Nicht-Persistent

- Bei nicht-persistentem HTTP wird für jede Anfrage eine neue TCP Verbindung aufgebaut $\implies$ sehr ineffizient. HTTP/1.0 nutzt dies.
  - Der Browser macht oftmals viele Anfragen gleichzeitig $\rightarrow$ viel Aufwand für das OS.
- Bei persistentem HTTP wird die gleiche TCP Verbindung für jede Anfrage genutzt $\implies$ deutlich effizienter. Ab HTTP/1.1 ist dies der Standard.
  - Effizient, aber nur mit Pipelining nur wenig unterstützt.
  - Ohne Pipelining müssen viele Anfragen gestellt werden, mit Pipelining kann alles in eine Anfrage gepackt werden.
  - Mit HTTP/2 wird dies durch Multiplexing ersetzt.

9.6.2 HTTP Anfragen

HTTP-Anfragen werden in menschenlesbarem ASCII versendet (sowohl Anfragen als auch Antworten) und bestehen aus einem Header, welcher Key-Value Daten enthält und dem Body, welcher die Anfrage/Antwort selbst enthält.

In HTTP werden sogenannte *Anfragemethoden* verwendet, die den Anfragen unterschiedliche Semantiken zuweisen:

- `GET` - Holt Daten vom Server.
- `POST` - Sendet Daten an den Server.
- `HEAD` - Weißt den Server an, nur den Header zu schicken.
- `PUT` (seit HTTP/1.1) - Sendet Daten an den Server.
- `DELETE` (seit HTTP/1.1) - Löscht eine Ressource auf dem Server.
- ...

```
GET /somedir/page.html HTTP/1.1
Host: www.example.org
User-Agent: Mozilla/4.0
Connection: close
Accept-Language: de
```

Abbildung 9.4: HTTP: Anfrage

Beispielanfrage

9.6.3 HTTP Response

HTTP-Antworten sind formatiert wie HTTP-Anfragen, wobei immer ein Status-Code mitgeschickt wird, der den Status der Anfrage angibt:

`200 OK`	Anfrage erfolgreich.
`301 Moved Permanently`	Die Ressource wurde verschoben, der neue Ort liegt im `Location`-Header.
`400 Bad Request`	Die Anfrage wurde nicht verstanden.
`404 Not Found`	Die Ressource wurde nicht gefunden.
`505 HTTP Version Not Supported`	Die HTTP-Version wird nicht unterstützt.

```
HTTP/1.1 200 OK
Connection close
Date: Thu, 19 Jul 2018 16:33:05 UTC+2
Server: Apache/1.3.0 (Unix)
Content-Length: 13
Content-Type: text/plain

Hello, World!
```

Abbildung 9.5: HTTP: Antwort

Beispielantwort

9.6.4 HTTP/2

- Unterstützung von Multiplexing: Mehrere Anfragen in einer Verbindung, wobei die Antworten out-of-order kommen.
- Kommunikationsrichtung Server → Client
- Stream Prioritäten
- Kompression

9.7 Nutzung von Dezentralisierung

- File Sharing
- Blockchain
- Video Streaming
- Lastverteilung (Load Balancing)
- Routing

9.7.1 P2P File Sharing

- Dateien werden in einem P2P-Netzwerk geteilt.
- Durch die Unabhängigkeit vom Internet können Antworten schneller kommen.
- Hochskalierbar

P2P Centralized Directory

Nutzen: Finden der Daten in einem P2P Netzwerk.

- Ein Peer informiert einen zentralen Server, dass er sich verbunden hat.
- Wenn Nutzer A eine Datei laden will, kann der Nutzer diese von dem PC des nächsten Users laden (Lastverteilung!).
- Nachteil: Single Point of Failure, Flaschenhals, Urheberrechtsverletzung

Gnutella

- Kein Zentraler Server
- Graph als Overlay Netzwerk
- Daten werden mittels Flooding gesucht
- Ein Peer versucht so lange, eine Verbindung zu einem anderen Peer aus einer Liste aufzubauen, bis eine Verbindung steht.

KaZaA

- Jeder Knoten ist entweder ein Gruppenleiter oder wird einem solchen zugewiesen.
- Der Gruppenleiter weiß, welche Daten wo liegen.
- Jede Datei hat einen Hashwert und einen Deskriptor.
- Ein Client sendet eine Suchanfrage an den Gruppenleiter, der in den Metadaten, der Hashsumme und der IP Adresse sucht.
- Der Gruppenleiter kann die Anfragen weiterleiten.
- Anschließend selektiert der Client die Dateien, die er herunterladen will.

Distributed Hash Table (DHT)

Ein anderer Ansatz ist eine *Distributed Hash Table* (DHT), bei keine Namen sondern Identifier genutzt werden und Hashwerte von allen Ressourcen vorliegen.

- Das Netzwerk wird in Namespaces aufgeteilt, in die die Daten einsortiert werden.
- Somit kann man Routing implementieren.

9.7.2 Chord

- Als Overlay-Netzwerk wird ein Ring verwendet.
- Jeder Knoten kennt seinen Vorgänger und Nachfolger.
- Zur Suche eines Paketes, wird eine Anfrage über den gesamten Ring gesendet, bis ein Knoten antwortet.
- **Problematik:** Bei großen Dateien werden die Knoten unnötig ausgelastet.

10 Allgemeines

10.1 Prüfsummen mittels Einerkomplement

Die Daten müssen in binäre Wörter mit einer fixen Länge aufgeteilt sein.

Bildung der Prüfsumme Zur Bildung einer Standard-Prüfsumme werden diese addiert, und sämtliche Überträge (Binär!) wieder addiert. Anschließend wird das Ergebnis negiert, außer es enthält ausschließlich Einsen.

Beispiel

- Über folgende Wörter soll die Prüfsumme gebildet werden: `1010`, `0101`, `1100`, `0010`.
- Zuerst werden die Wörter der Reihe nach addiert:
  - `1010 + 0101 = 1111`
  - `1111 + 1100 = 11011` $\rightarrow$ `1011 + 1 = 1100`
  - `1100 + 0010 = 1110`
- Nun wird das Ergebnis (`1110`) negiert und ergibt damit die Prüfsumme **`0001`**.

Validierung der Daten mit der Prüfsumme Um die Daten zu validieren, werden alle Wörter auf die Prüfsumme addiert, und sämtliche Überträge (Binär!) wieder addiert. Enthält das Ergebnis auch nur eine Null, so sind die Daten fehlerhaft.

Beispiel

- Folgende Wörter sollen Validiert werden: `1010`, `0101`, `1100`, `0010`.
- Für diese Wörter wurde die folgende Prüfsumme berechnet: **0001**.
- Nun werden alle Wörter der Reihe nach auf die Prüfsumme addiert:
  - `0001 + 1010 = 1011`
  - `1011 + 0101 = 10000` $\rightarrow$ `0000 + 1 = 0001`
  - `0001 + 1100 = 1101`
  - `1101 + 0010 = `**`1111`**
- Die Daten sind korrekt.

11 Abkürzungen

A Address Record

AA Authorative Answer (Flag)

AAA Triple A

AAAA Quad-A Record (A Record for IPv6)

ACK Acknowledge

ACM Association for Computing Machinery

ADSL2 Asymmetric Digital Subscriber Line 2

AIMD Additive Increase Multiplicative Decrease

AK Acknowledge

API Application Programming Interface

ARP Address Resolution Protocol

ARPANET Advanced Research Projects Agency Network

ARQ Automatic Repeat reQuest

AS Autonomous Systems

ASCII American Standard Code for Information Interchange

ASN Abstract Syntax Notation

BER Basis Encoding Rules

BGP Border Gateway Protocol

BSD Berkeley Software Distribution

BTSH BGP TTL Security Hack

CA Certificate Authority

CBT Core Based Tree

CC Congestion Control

CCID Congestion Control ID

CCR Commitment Concurrency and Recovery

CD Collision Detection

CDF Cumulative distribution function

CDM Code Division Multiplexing

CIDR Classless InterDomain Routing

CL Connectionless

CLNS Connectionless-mode Network Service

CN Computer Network

CNuvS Computer Netzwerke und verteilte Systeme

CO Connection-oriented

CONS Connection-oriented Network Service

COTS Connection-oriented Transport Service

CPU Central Processing Unit

CR Connection Request

CSCW Computer Supported Cooperative Work

CSMA Carrier Sense Multiple Access

CSRC Contributing Source

CSS Communication Subsystem

CW Cooperative Work

DAD Duplicate Address Detection

DAT Data Exchange

DB Database

DCCP Datagram Congestion Control Protocol

DF Don't Fragment (Flag)

DHCP Dynamic Host Configuration Protocol

DHCPv6 Dynamic Host Configuration Protocol for IPv6

DHT Distributed Hash Table

DIN Deutsches Institut für Normung

DIS Disconnect

DL Data Link

DNS Domain Name System

DoD Department of Defence

DP Destination Port

DR Disconnect Request

DS Distributed System

DSDV Destination Sequence Distance Vector

DSR Dynamic Source Routing

DT Data

DV Distance Vector Routing

DVMRP Distance Vector Multicast Routing Protocol

DVR Distance Vector Routing

EUI-64 64 Bit Extended Unique Identifier

FCFS First Come First Serve

FDD Frequency Division Duplex

FDDI Fiber Distributed Data Interface

FDM Frequency Division Multiplexing

FIFO First In First Out

FTP File Transfer Protocol

GB Gigabyte

GBit Gigabit

GE General Electric

GMT Greenwich Mean Time

GT Graphentheorie

HP Hewlett Packard

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

IANA Internet Assigned Numbers Authority

IBM International Business Machines Corporation

ICANN Internet Corporation for Assigned Names and Numbers

ICMP Internet Control Message Protocol

ICMPv6 Internet Control Message Protocol for IPv6

ID Identifier

IDL Interface Definition Language

IETF Internet Engineering Task Force

IGMP Internet Group Management Protocol

IGRP Interior Gateway Routing Protocol

IHL IP Header Length

IP Internet Protocol

IPC Inter Process Communication

IPSec IP Security

IPTV IP Television

IPv4 Internet Protocol Version 4

IPv6 Internet Protocol Version 6

ISO International Organization for Standardization

ISP Internet Service Provider

IT Information technology

ITU-T ITU Telecommunication Standardization Sector

JPEG Joint Photographic Experts Group

KaZaA KaZaA (Eigenname)

KB Kilobyte

KBit Kilobit

LAN Local Area Network

LLC Logical Link Control

LS Link State Routing

LSR Link State Routing

MAC Medium Access Control

MD5 Message Digest Algorithm 5

MF More Fragments (Flag)

MOSPF Multicast Open Shortes Path First

MS Microsoft

MSB Most Significant Bit/Byte

MSS Maximum Segment Size

MST Minimal Spanning Tree

MTA Mail Transfer Agent

MUA Mail User Agent

MX Mail Exchange Resource Record

NACK Negative Acknowledgement

NASA National Aeronautics and Space Administration

NAT Network Address Translation

NCP Network Control Program

ND Neighbour Discovery Protocol

NDP Neighbour Discovery Protocol

NIC Network Interface Card

NLA Next Level Address

NS Network Layer Services / Nameserver Record

NW Network

NY New York

OO Object Oriented

OPCode Operation Code

OS Operating System

OSI Open Systems Interconnection

OSN Online Social Network

OSPF Open Shortes Path First

P2P Peer-to-Peer

PC Personal Computer

PCCW Pacific Century Cyberworks

PCI Peripheral Component Interconnect

PDF Portable Document Format

PDU Protocol Data Unit

PH Physical Layer

PHY Physical Layer

PID Process ID

PIM Protocol Independent Multicast

PT Pakistan Telekom

QoS Quality of Service

QUIC Quick UDP Internet Connection

RA Router Advertisement

RARP Reverse ARP

RD Recusion Desired (Flag)

RED Random Early Detection

RFC Request For Comments

RIP Routing Information Protocol

RIPE Reseaux IP Europeens Network Coordination Centre

RIR Regional Internet Registry

ROSE Remote Operation Service Element

RPF Reverse Path Forwarding

RR Resource Record

RST Reset (Flag)

RTCP Real Time Control Protocol

RTP Real Time Transport Protocol

RTT Round Trip Time

SAP Systeme Anwendungen und Produkte in der Datenverarbeitung

SCTP Stream Control Transmission Protocol

SD Queue Discipline

SDM Space Division Multiplexing

SDU Service Data Unit

SE Service Element

SHA-1 Secure Hashing Algorithm 1

SIGCOMM Special Interest Group on Data Communications

SLA Site Level Address

SLAAC Stateless Address Autoconfiguration

SMTP Simple Mail Transfer Protocol

SNMP Simple Network Management Protocol

SP Source Port

SPT Shortes Path Trees

SRC Source

SSN Social Security Number

SSRC Synchronization Source

SW Software

SYN Synchronize

TC Truncation (Flag)

TCP Transmission Connection Protocol

TDD Time Division Duplex

TDM Time Division Multiplexing

TL Transport Layer

TLA Top Level Aggregation

TLD Top Level Domain

TLS Transport Layer Security

ToS Type of Service

TPDU Transport Protocol Data Unit

TS Transport Service

TSDU Transport Service Data Unit

TTL Time To Live

TV Television

UDP User Datagram Protocol

UI user Interface

UK United Kingdom

UMTS Universal Mobile Telecommunications System

URG Urgent (Flag)

URL Unified Resource Locator

US United States

USA United States of America

VLSM Variable Length Subnet Masks

VoIP Voice over IP

WAN Wide Area Network

WDM Wavelength Division Multiplexing

WLAN Wireless LAN

WSN Wireless Sensor Network

WWW World Wide Web