

Zusammenfassung Grundlagen der Rechnernetze WS18/19

Jonas Otto

edwin.otto@uni-ulm.de

Luca Krüger

luca.krüger@uni-ulm.de

Marco Deuscher

marco.deuscher@uni-ulm.de

Dezember 2018 - März 2019

Inhaltsverzeichnis

1	Einführung	4
1.1	OSI-Schichtenmodell	4
1.1.1	Schichtenmodell	4
1.1.2	OSI-Schichtenmodell und protocol stack	5
1.1.3	Einführung in die einzelnen Schichten	5
2	Anwendungsschicht	7
2.1	Grundlagen der Netzerkanwendungen	7
2.1.1	Architekturen für Netzerkanwendungen	7
2.1.2	Kommunizierende Prozesse	8
2.1.3	Sockets	8
2.1.4	Anwendungsprotokolle	8
2.2	HyperText Transfer Protocol	9
2.2.1	Überblick	9
2.2.2	HTTP Verbindungen	10
2.2.3	HTTP Nachrichtenformat	10
2.2.4	User-Server Interaktion: Cookies	12
2.2.5	Web-Caching	13
2.3	Simple Mail Transfer Protocol	15
2.3.1	Eigenschaften von SMTP	15
2.3.2	SMTP am Beispiel	16
2.3.3	SMTP Nachrichtenformat	17
2.3.4	Mail Access Protocol	17
2.4	Domain Name System (DNS)	17
2.4.1	DNS-Hierarchie	18
2.4.2	DNS Namensauflösung	19
2.4.3	DNS Records und Messages	20
2.4.4	Angriffe auf DNS	22
2.5	Peer-to-Peer Architekturen	23
2.5.1	Allgemeine Charakteristika	23
2.5.2	Vorteile von P2P	23
3	Transportschicht	25
3.1	Allgemeines zur Transportschicht	25
3.1.1	Transport- vs. Networklayer	25
3.1.2	Multiplexing und Demultiplexing	25
3.2	User Datagram Protocol (UDP)	25
3.2.1	Allgemeines zu UDP	25

3.2.2	UDP-Segment	26
3.2.3	Vorteile von UDP	27
3.3	Anforderungen an ein zuverlässiges Transportschicht Protokoll	27
3.3.1	Pipelining	28
3.4	Transmission Control Protocol (TCP)	30
3.4.1	Überblick	30
3.4.2	Seq.- and ACK-numbers	32
3.4.3	TCP Round Trip Time & Timeout	32
3.4.4	Grundlegende TCP-Mechanismen	33
3.4.5	Pipelining bei TCP - GBN oder Selective Repeat	34
3.4.6	TCP Flusskontrolle (Flowcontrol)	34
3.4.7	TCP 3-way Handshake	36
3.4.8	Prinzipien der Überlastkontrolle	37
3.4.9	TCP Überlastkontrolle (Congestion Control)	37
3.4.10	TCP Fairness	41
3.4.11	Multipath TCP	41
4	Netzwerkschicht	43
4.1	Überblick: Netzwerkschicht	43
4.1.1	Service Modell der Netzwerkschicht	44
4.2	Routerarchitektur	44
4.2.1	Switching	46
4.2.2	Outputport processing	48
4.3	Internet Protokoll (IP)	49
4.3.1	IPv4	50
4.3.2	Zuweisung von IP-Adressen	52
4.3.3	Network Address Translation (NAT)	53
4.3.4	Internet Control Message Protocol (ICMP)	54
4.3.5	IPv6	54
4.4	Routing	55
4.4.1	Intra-AS Routing	55
4.4.2	Inter-AS Routing	56
4.4.3	Routing Algorithmen	56
5	Linklayer	57
5.1	Überblick Linklayer	57
5.2	Fehlererkennung	57
5.2.1	Parity Check	57
5.2.2	Cyclic Redundancy Check (CRC)	57
5.3	MAC Protokolle	58
5.3.1	TDMA, FDMA	58
5.3.2	Random Access Protokolle	58
5.3.3	Taking Turns Protokolle	59
5.4	MAC Adressen	59
5.5	Address Resolution Protocol ARP	59

6	Drahtlose-Kommunikation	60
6.1	Eigenschaften Drahtloser Links im Vergleich zu drahtgebundenen Links	60
6.1.1	Hidden Terminal Problem	60
6.2	802.11	60
6.2.1	CSMA/CA	60
7	Netzwerksicherheit	61
7.1	Grundlegende Prinzipien der Netzwerksicherheit	61
7.1.1	Prinzipien	61
7.1.2	Angriffsformen	61
7.2	Symmetrische Verschlüsselungsverfahren	61
7.2.1	Moderne symmetrische Chiffren	62
7.3	Public Key Kryptographie	62
7.3.1	Praktische Anwendung	62
7.4	Digitale Signaturen	62
7.5	Certification authorities	63
7.6	SSL und TLS	63
7.6.1	TLS im Schichtenmodell	63

Kapitel 1

Einführung

1.1 OSI-Schichtenmodell

1.1.1 Schichtenmodell

Ein Schichtenmodell basiert darauf, dass eine Schicht Dienste an eine darüberliegende Schicht anbietet. Verschiedene Schichten kommunizieren durch Messages miteinander, während Schichten auf der gleichen Ebene über Protokolle miteinander kommunizieren.

1.1.2 OSI-Schichtenmodell und protocol stack



Abbildung 1.1: Five-layer Internet protocol stack and Seven layer ISO OSI reference modell

Ein Schichtenmodell ermöglicht die Abstraktion von der Komplexität der Aufgabe, so dass sich ein Anwendungsentwickler darauf verlassen kann, dass die unteren Schichten die er nutzt ihm gewisse Dienste zur Verfügung stellen.

1.1.3 Einführung in die einzelnen Schichten

- **Application Layer:** ein Anwendungsschicht Protokoll ist auf viele Endsysteme verteilt. Die Anwendung tauscht Informationen zwischen den Endsystemen aus. Man spricht auch von Messages die ausgetauscht werden. Einige Beispiele für Anwendungsschichtprotokolle sind HTTP, SMTP, FTP
- **Transport Layer:** Austausch von Segmenten zwischen Prozessen in getrennten Endsystemen. Bspw. TCP, UDP, QUIC
- **Network Layer:** Für das Routen von Datagramen verantwortlich. Enthält als Entitäten die Router im Netzkern. Bsp: IP und diverse Routingprotokolle
- **Link Layer:** Bringt Frames vom einen Hop zum nächsten. Bsp: Ethernet, WiFi

- **Physical Layer:** bewegt die einzelnen Bits vom einen Knoten zum nächsten.
Hängt vom Übertragungsmedium ab

Das ISO/OSI-Referenzmodell führt zusätzlich noch folgende weitere Schichten:

- **Presentation Layer:** zuständig für die Repräsentation der Daten, dies schließt Komprimierung und Verschlüsselung ein
- **Session Layer:** zuständig für Synchronisation, Checkpoints oder auch die Wiederaufnahme einer Sitzung

Kapitel 2

Anwendungsschicht

2.1 Grundlagen der Netzerkanwendungen

2.1.1 Architekturen für Netzerkanwendungen

Client-Server Architektur

Server:

- Host dauerhaft erreichbar
- permanente IP-Adresse

Clients:

- initiieren Verbindung zum Client
- im Allgemeinen nur zeitweise online
- dynamische wechselnde IP-Adressen
- kommunizieren nicht direkt miteinander

Peer-2-Peer Architektur

- kein ständig verfügbarer Server
- Endsysteme kommunizieren direkt miteinander
- Peers erbringen Dienste und nutzen Dienste von anderen Peers
- Peers sind nur zeitweise verbunden und wechseln IP-Adressen dynamisch

Eine Peer-to-Peer Architektur hat in manchen Fällen Vorteile gegenüber einer Client-Server Architektur: Zum Beispiel wächst die Verteilzeit einer Datei bei Client-Server linear mit der Anzahl der Clients an, bei Peer-to-Peer nur etwa logarithmisch.

2.1.2 Kommunizierende Prozesse

Def: Prozess: ein Programm, das auf einem Host abläuft
Hosts in verschiedenen Endsystemen kommunizieren über Messages miteinander.

Im Zusammenhang von kommunizierenden Prozessen ist der Prozess, welcher die Verbindung initiiert der Client. Der auf eine Verbindung wartende Prozess ist der Server.

2.1.3 Sockets

Ein **Socket** stellt eine Schnittstelle zum Betriebssystem dar, in welche Nachrichten geschrieben und aus welcher Nachrichten gelesen werden können. Das Konzept der Sockets bildet eine **Abstraktion** zu den darunterliegenden
Um einen Prozess identifizieren zu können, wird eine IP-Adresse und eine Portnummer benötigt. Anwendungsprotokolle nutzen oft Well-known Portnummern wie HTTP: 80 oder SMTP: 25

2.1.4 Anwendungsprotokolle

Anwendungsprotokolle definieren

- Typ der Nachricht (bspw. Response oder Request)
- Nachrichtensyntax
- Nachrichtensemantik
- Regeln wann und wie Prozesse Nachrichten austauschen oder auf diese reagieren

Im Allgemeinen können **Anwendungsprotokolle** in zwei Kategorien aufgeteilt werden:

Offene Protokolle:

- definiert in RFC
- ermöglicht Interoperabilität

Proprietäre Protokolle



Abbildung 2.1: RMS Disapproves.

2.2 HyperText Transfer Protocol

2.2.1 Überblick

- nutzt *Client-Server Modell*
- Nutzt TCP
 - Client initiiert TCP Verbindung zum Server auf Port 80
 - Server akzeptiert Verbindungsaufbau
 - HTTP Nachrichten werden ausgetauscht
 - TCP Verbindung wird geschlossen
- HTTP ist zustandslos (Server behält keine Information über Client (Workaround: Cookies))

2.2.2 HTTP Verbindungen

Es kann zwischen zwei Arten von HTTP Verbindungen unterschieden werden, *nicht-persistentes* und *persistentes* HTTP.

Nicht-persistentes HTTP

Client verbindet sich mit Server und fordert HTML file an welches 10 Bilder enthält.

1. HTTP Client initiiert TCP Verbindung mit Server auf Port 80
2. HTTP Client sendet Request-Nachricht über Socket an Server (enthält Pfad)
3. HTTP Server erhält Request, holt gewünschtes Dokument und sendet HTTP Response-Nachricht über Socket an Client
4. HTTP Server beendet Verbindung (TCP wartet bis Client Nachricht erhalten hat)
5. HTTP Client erhält Response. TCP Verbindung wird beendet. Das erhaltene Dokument enthält den Pfad zu 10 Bildern
6. Schritt 1-4 werden für jedes der Bilder durchgeführt

Persistentes HTTP

Mit einer persistenten HTTP Verbindung kann eine einzelne Verbindung zum Austausch des HTML Dokuments und der 10 Bilder genutzt werden, ohne eine neue TCP Verbindung aufbauen zu müssen.

Der Abbau der Verbindung erfolgt durch Client oder durch einen Timeout. Die Effizienz kann weiter verbessert werden durch Pipelining.

(Rechenbeispiel etwa B2Ü3.)

2.2.3 HTTP Nachrichtenformat

Es gibt zwei verschiedene Arten von HTTP Nachrichten, *Response*- und *Requestmessages*.

Aufbau einer HTTP-Request



Abbildung 2.2: Allgemeiner Aufbau einer HTTP Request

HTTP Methoden:

- **GET:** Anfragen von Information durch Spezifizieren der URI(uniform resource identifier)
- **POST:** Senden von Daten zum Server
- **HEAD:** Anfragen des Headers (bspw. für Debug Zwecke)
- **PUT:** Upload von Dateien an den Server, an die durch URL kodierte Stelle
- **DELETE:** Löschen von Dateien auf dem Webserver

Aufbau einer HTTP-Response



Abbildung 2.3: Allgemeiner Aufbau einer HTTP Response

HTTP-Status Codes:

- **200 OK:** Request erfolgreich. Webobjekt im Body
- **301 Moved Permanently:** Object dauerhaft verschoben, neue URL im Location Header
- **400 Bad Request:** Server versteht Request nicht
- **404 Not Found:** Angefragtes Objekt existiert nicht
- **505 HTTP Version Not Supported:** Unbekannte HTTP Version

und mehr. For Reference: [https://http.cat/\[status_code\]](https://http.cat/[status_code])

2.2.4 User-Server Interaktion: Cookies

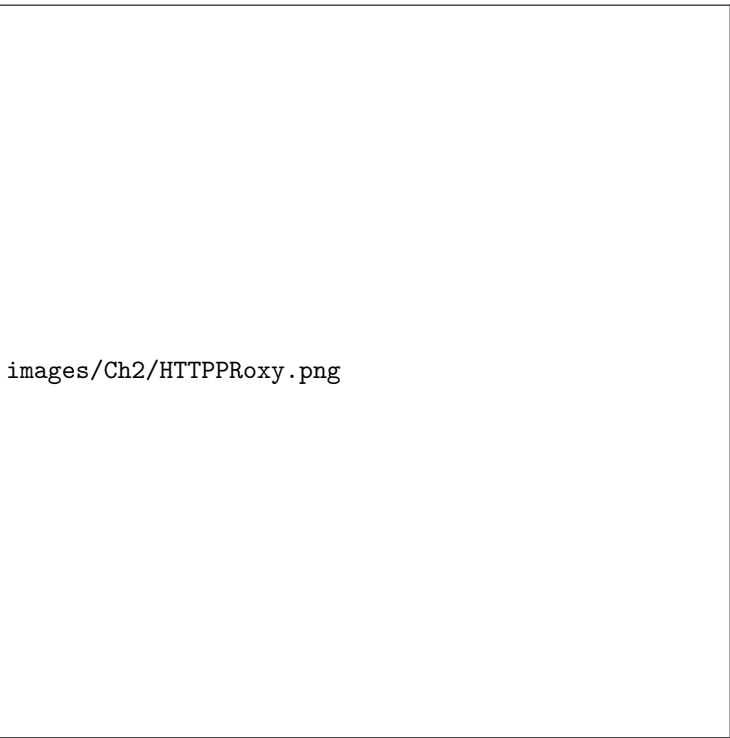
Wie bereits zuvor erwähnt ist HTTP *zustandslos*, allerdings ist es oftmals interessant Informationen über den Client zu erhalten. Um die Effizienz eines zustandslosen Protokolls zu erhalten und trotzdem Nutzerinformation verwenden zu können, werden *Cookies* verwendet.



Abbildung 2.4: Zustandsinformation mit der Hilfe von Cookies

2.2.5 Web-Caching

Ein Web-Cache oder Proxy-Server ist eine Netzwerk-Entität welche die Bearbeitung von HTTP Requests für den ursprünglichen Webserver übernimmt. Der Proxy-Server hat seinen eigenen Speicher auf welchem er kürzlich angeforderte Objekte speichert.



images/Ch2/HTTProxy.png

Abbildung 2.5: Clients requesten Objekte über Proxy-Server

Ist ein solcher Proxy-Server im Netzwerk vorhanden läuft eine HTTP-Request folgendermaßen ab:

1. Browser erstellt TCP Verbindung zum Web Cache und sendet HTTP-Request an diesen
2. Web Cache prüft ob er eine Kopie des Objekts lokal gespeichert hat. Wenn ja, gibt der Web Cache das gewünschte Objekt zurück in einer HTTP Response
3. Hat der Web Cache das Objekt nicht lokal gespeichert öffnet der Web Cache eine TCP Verbindung zum ursprünglichen Webserver. Er sendet eine HTTP-Request und ruft das gewünschte Objekt ab
4. Hat der Web Cache das Objekte vom ursprünglichen Webserver erhalten, speichert er dieses und sendet eine Kopie in einer HTTP-Response an den Client

Ein Web Cache ist also **gleichzeitig** Server und Client. Ein solcher Web Cache bietet mehrere Vorteile. Unter anderem eine **reduzierte Responsezeit** auf eine Client Request und zum anderen **vermindert** ein solcher Web Cache den **Traffic** auf anderen Links.

Der Proxy-Server ist an dieser Stelle verantwortlich, dass die gespeicherten Seiten aktuell sind. Der Proxy-Server speichert dazu das last-modified Datum. Für den Fall, dass sich etwas geändert haben könnte fragt der Proxy-Server mit einem sogenannten conditional GET den Server nach einer aktuelleren Version.

Dafür wird im Header die Zeile If-modified-since hinzugefügt. Die Antwort des Servers ist dann 304 Not Modified, oder eine gewöhnliche Response mit dem neuen Inhalt.

2.3 Simple Mail Transfer Protocol

Abbildung 2.6 ermöglicht einen vereinfachten Überblick über die beteiligten Entitäten (User Agents, Mail Server und SMTP Protokoll)

Hierbei sind **User Agents** bspw. MailClients wie Outlook oder WebMail. Der User Agent ermöglicht dem Benutzer das Lesen und Schreiben von E-Mails.

Der Mailserver speichert in der Mailbox die eingehenden Nachrichten der Benutzer und ermöglicht Nutzern das Abrufen von Nachrichten.

Das SMTP Protokoll definiert den Austausch zwischen zwei Servern bzw. zwischen Server und Mail Agent.



Abbildung 2.6: Überblick über SMTP

2.3.1 Eigenschaften von SMTP

- SMTP ist **TCP** basiert und nutzt well-known port 25
- SMTP verwendet ***persistente Verbindungen***
- drei Protokollphasen (Handshake, Nachrichtentransfer, Verbindungsabbau)

- SMTP ist **7bit-ASCII** basiert (→ MIME-Encoding für zusätzliches)
- SMTP ist ein *push-Protokoll*

2.3.2 SMTP am Beispiel

In Abbildung 2.7 ist mögliches Szenario in welchem SMTP eingesetzt wird zu sehen.

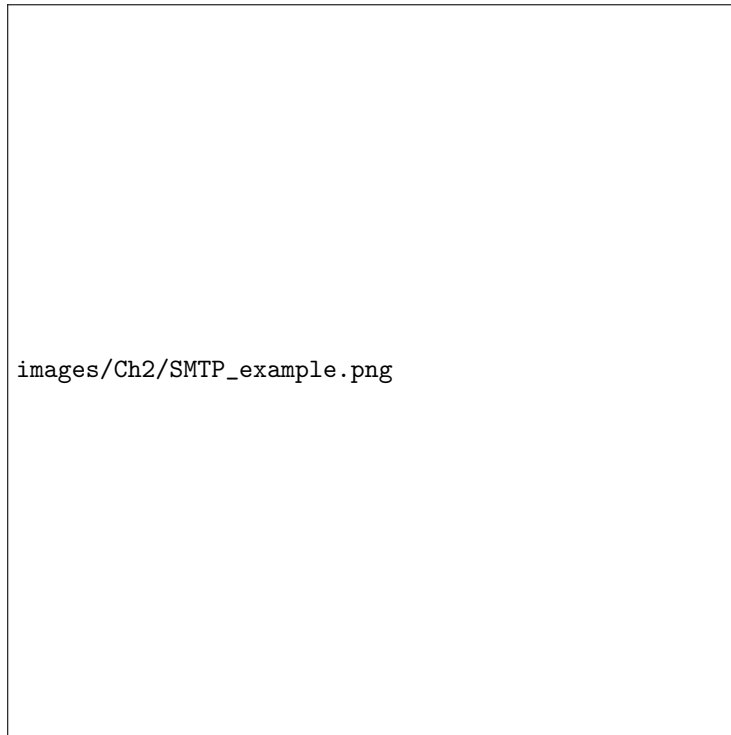


Abbildung 2.7: Alice sendet E-Mail an Bob

1. Alice erstellt Mail im UA adressiert an Bob
2. Alice UA sendet die Nachricht an ihren Mail Server. Dort wird die Nachricht in der Nachrichtenqueue gespeichert
3. Alice(Client) Mail Server initiiert TCP Verbindungs zu Bob Mail Server
4. SMTP Client sendet E-Mail über TCP-Verbindung
5. Bob Mail Server speichert die Nachricht in der MailBox ab
6. Bob kann jetzt UA verwenden um Mail abzurufen

```

220 mail.uni-ulm.de ESMTP Sendmail 8.15.2/8.15.2; \
Tue, 13 Nov 2018 21:02:51 +0100 (CET)
HELO mail.uni-ulm.de
250 mail.uni-ulm.de Hello whm-hms-nat4.rz.uni-ulm.de \
[193.197.66.16], pleased to meet you
MAIL FROM: <edwin.otto@uni-ulm.de>
250 2.1.0 <edwin.otto@uni-ulm.de>... Sender ok
RCPT TO: <grn-smtp@lists.uni-ulm.de>
250 2.1.5 <grn-smtp@lists.uni-ulm.de>... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
From: edwin.otto@uni-ulm.de
To: grn-smtp@lists.uni-ulm.de
Subject: GRN-SMTP
X-GRN-Semester: WS 2018/19
X-GRN-Group: 101

Hallo GRN!
.
250 2.0.0 wADK2p4P027754 Message accepted for delivery
QUIT
221 2.0.0 mail.uni-ulm.de closing connection

```

Abbildung 2.8: Manueller Versand einer Mail mit Hilfe von SMTP

2.3.3 SMTP Nachrichtenformat

- Encoding: 7-bit ASCII (MIME Encoding)
- Header: Zeilen im Format Key: Value
- Body: Eigentliche Nachricht
- End of Message: CRLF . CRLF

2.3.4 Mail Access Protocol

Das SMTP Protokoll kümmert sich nur darum, dass eine gesendet Nachricht in der MailBox des Empfängers landet. Für das Abrufen einer Nachricht sind Mail Access Protokolle verantwortlich. Die zwei bekanntesten sind **POP** und **IMAP**.

Mail Access Protokolle ermöglichen es Nutzern, Mails vom Server abzurufen. Bei POP werden die Mails auf dem Gerät des Nutzers gespeichert, bei IMAP verbleiben die Mails auf dem Server. Außerdem bietet IMAP zusätzliche Funktionalität, wie das Verwalten von Nachrichten in Ordnern oder das Speichern des Zustands einer Nachricht (gelesen, markiert, archiviert).

2.4 Domain Name System (DNS)

Die Hauptaufgabe von DNS ist die Auflösung von Domain Namen zu IP-Adressen.

- DNS ist eine *verteilte* Datenbank mit Hierarchie von vielen DNS Name-servern
- DNS ist Anwendungsprotokoll:
 - Zentrale Funktionalität des Internets, nicht in TCP/IP enthalten
 - KISS Prinzip (Keep it simple and stupid): Komplexität nur am Rand

Welche Dienste bietet DNS an?

- Auflösung von Hostnamen zu IP-Adressen und umgekehrt
- Host-Aliase
- zuständige Mail-Server
- Lastverteilung

Es ist sinnvoll, dass DNS ein verteiltes System ist. So hat man *keinen Single Point of failure*, das Datenvolumen ist verteilt und das System ist leicht zu warten und sehr zuverlässig.

2.4.1 DNS-Hierarchie

In Abbildung 2.9 ist der hierarchische Aufbau der DNS-Server zu sehen.



Abbildung 2.9: Aufbau der DNS-Hierarchie

DNS Root-Server

Im gesamten Internet gibt es 13 DNS Root Server. Jeder dieser 13 ist aber tatsächlich ein Verbund von weiteren Servern um die Zuverlässigkeit und Sicherheit zu erhöhen.

Toplevel Domain-Server (TLD Server)

Diese Server sind für Toplevel Domains wie .com, .space, .lol, .ninja, .porn usw. zuständig ebenso wie für die Landes spezifischen Domains wie .de, .fr und so weiter.

Autoritative DNS-Server

Jede Organisation mit öffentlich erreichbaren Hosts muss die DNS-Records dieser Hosts zur Verfügung stellen. Dies kann entweder durch einen eigenen DNS-Server oder durch Server eines Providers geschehen.

Dies ist der Nameserver der die Originaldaten einer Domain besitzt. Es gibt in der Regel einen Primary Nameserver welcher die Originaldaten verwaltet und einen Secondary Nameserver welcher aus Redundanzgründen Kopien der Daten hält.

Local Resolving DNS-Nameserver

- gehört nicht notwendigerweise zur DNS-Hierarchie
- ISPs betreiben resolving Nameserver für ihre Kunden
- Hosts senden anfragen in der Regel an resolving Nameserver. Dieser kümmert sich dann um die weitere Auflösung der Anfrage. Ein solcher resolving Nameserver betreibt auch Caching von früherern Anfragen

2.4.2 DNS Namensauflösung

Beachte, dass sobald der authoritative DNS-Server erreicht wird, dieser die Anfrage auflöst und es keine weiteren Anfragen mehr gibt (kam in Altklausur aufgabe dran, musste dort beachtet werden da dann eine unnötige Verbindung eingezeichnet war)

Iterative Auflösung

In Abbildung 2.10 ist die iterative Auflösung einer DNS Anfrage zu sehen. Dabei sendet jeder Server einen Verweis auf den nächsten Server, so lange bis der authoritative Nameserver erreicht ist. Nur local Resolver löst aus Client Sicht rekursiv auf.

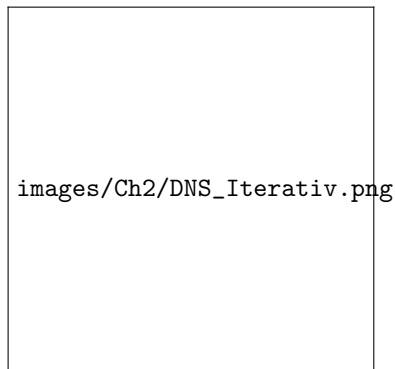


Abbildung 2.10: Iterative Auflösung einer DNS Anfrage

Rekursive Auflösung

In Abbildung 2.11 ist die rekursive Auflösung einer DNS Anfrage zu sehen. Dabei löst der angefragte Nameserver den Namen vollständig auf und liefert nur das Ergebnis zurück.

Der Nachteil hiervon ist eine sehr hohe Last für Server höher in der Hierarchie (Root- & TLD-Server), daher verweigern Root- und TLD-Nameserver rekursive Auflösung.

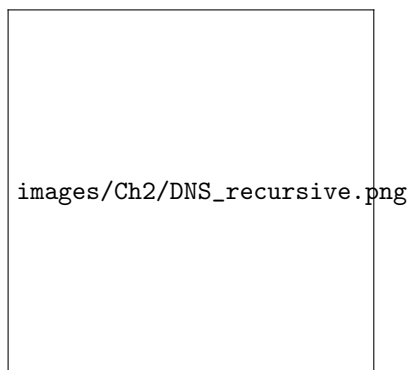


Abbildung 2.11: Rekursive Auflösung einer DNS Anfrage

2.4.3 DNS Records und Messages

DNS Records (RR: resource records) sind ein viertupel bestehend aus

(name,value,type ttl)

Type=A

- Name: Hostname
- Value: IPv4-Adresse

Type=NS

- Name: Domain (bspw. uni-ulm.de)
- Value: Hostname des autoritativen Nameservers
- mehrere Rekordeinträge für gleichen Namen möglich

Type=CNAME

- Name: Alias
- Value: „canonical“Name (echter Name)

Type=MX

- Name: Host- oder Domainname
- Value: Name des zuständigen Mailservers
- Bsp: uni-ulm.de → mail.uni-ulm.de

Type=AAAA

- Name: Hostname
- Value: IPv6-Adresse

DNS-Message Format

In Abbildung 2.12 ist das Format einer DNS Message zu sehen.

DNS nutzt **UDP** oder **TCP**. Query und Reply haben das gleiche Format.

Identification im Header, es werden 16Bit zur Zuordnung von Query und Reply verwendet. Außerdem gibt es die folgenden Flags: *query orreply, recursion desired, recursion available, reply is authoritative*.



images/Ch2/DNS_message.png

Abbildung 2.12: Aufbau einer DNS-Message

2.4.4 Angriffe auf DNS

- DDoS-Angriffe
 - Überlastung von Root oder TLD Servern heute kaum noch erfolgreich
 - Traffic Filter
 - Root und TLD Server vielfach gecached
- DNS Redirection Angriffe
 - Man-in-the-Middle: Angreifer fängt Datenverkehr ab und gibt sich als DNS Server aus
 - DNS Cache Poisoning: Caches in Middleboxes werden mit gefälschten Informationen vergiftet
- DNS Amplification Angriffe
 - DNS Server werden für DDoS misbraucht
 - Angreifer sendet Queries mit gefälschten Absenderadressen an DNS Server
 - DNS Server schicken (deutlich größere) Antworten an Angriffsziel

2.5 Peer-to-Peer Architekturen

2.5.1 Allgemeine Charakteristika

- keine always on Server
- beliebige Endsysteme kommunizieren direkt miteinander
- Peers sind nur sporadisch miteinander verbunden und wechseln ihre IP-Adressen

Einige Beispiele für P2P-Verbindungen sind **File Sharing Applikationen** wie BitTorrent, Streaming (KanKan), **VoIP** (früher Skype) und update Mechanismen in **online Spielen**.

2.5.2 Vorteile von P2P

Der hauptsächliche Vorteil von P2P ist, dass es deutlich besser **skaliert** als eine Client-Server Architektur (siehe Übungsaufgabe B5.Ü1).

In einem P2P-Netz ist jeder Peer gleichzeitig Client und Server und trägt etwas seiner Upload-Kapazität bei. Bei einer Client-Server Architektur nutzt der Client nur seine Downloadverbindung und lastet den Server stärker aus.

File Distribution Time: Client-Server

Server Übertragung:

- sequentielles Senden von N File Kopien
- Zeit pro Kopie F/u_s wobei F die Dateigröße ist
- Zeit für N Kopien somit NF/u_s

Client Übertragung: jeder Client muss eine Kopie übertragen

- d_{max} =maximale Client Download Rate
- Minimale Client Download Zeit: F/d_{max}

Damit ergibt sich also für die Übertragung von F an N Clients bei einer Client-Server Architektur eine Zeit

$$D_{CS} \geq \max[NF/u_s, F/d_{min}] \quad (2.1)$$

Es ist leicht zu erkennen, dass für eine große Anzahl an Clients die Zeit linear mit N zunimmt und es somit sehr schnell zu einer Überlastung des Servers kommen kann.

File Distribution Time: P2P

Server Übertragung: muss mindestens eine Kopie in den Umlauf bringen (Zeit für eine Kopie beträgt F/u_s)

Client: jeder Client muss eine Kopie übertragen (minimale Downloadzeit: F/d_{min}
Max. Upload Rate: $u_s + \sum u_i$ wobei die Summe der Uploadraten der Peers der

des Servers aufaddiert wird.

Damit ergibt sich für eine Peer2Peer Architektur also die folgende Übertragungszeit

$$D_{P2P} \geq \max[F/u_s, F/d_{min}, NF/(u_s + \sum u_i)] \quad (2.2)$$

Es ist leicht zu erkennen dass mit steigender Zahl Clients nicht nur die Menge an Bytes die gesendet werden muss linear wächst sondern für große N auch die Uploadrate. Die Verteilzeit ist also für Client-Server Architekturen linear während P2P Architekturen einen eher logarithmischen Verlauf liefern.

Kapitel 3

Transportschicht

3.1 Allgemeines zur Transportschicht

- Logische Kommunikation zwischen Anwendungen auf verschiedenen Hosts
- Transportschichtprotokolle laufen nur auf Endsystemen
- Typische Internet Transportschichtprotokolle sind TCP und UDP

3.1.1 Transport- vs. Networklayer

Netzwerkschicht stellt logische Kommunikation zwischen Hosts zur Verfügung.
Transportschicht stellt logische Kommunikation zwischen Prozessen zur Verfügung.

3.1.2 Multiplexing und Demultiplexing

Transportschicht des Senders verwaltet die Daten mehrerer Sockets und fügt diesen einen Transporthead hinzu.

Beim Demultiplexing werden die Headerinformationen verwendet um die Segmente an den richtigen Socket zu liefern

Was wird mindestens benötigt damit der Empfänger weiß an welchen Socket das entsprechende Segment zugestellt werden soll?

3.2 User Datagram Protocol (UDP)

3.2.1 Allgemeines zu UDP

- sehr simples Transportschichtprotokoll
- best effort Dienst, d.h. UDP-Segmente können
 - verloren gehen
 - in der falschen Reihenfolge ankommen
- Verbindungslos
 - kein initialer Handshake zwischen Sender und Empfänger

- jedes UDP-Segment wird unabhängig behandelt
- Anwendung:
 - Streaming multimedia (Verluste werden toleriert)
 - DNS
 - SNMP
- möchte man einen zuverlässigen Datentransport mit UDP erreichen, muss dies in der Anwendungsschicht implementiert werden

3.2.2 UDP-Segment

Es werden sowohl Source- als auch Destinationport benötigt damit das UDP-Segment von der Transportschicht eindeutig zugeordnet werden kann. Das Feld Length enthält die Länge des UDP-Segments *inklusive* Header in Bytes.

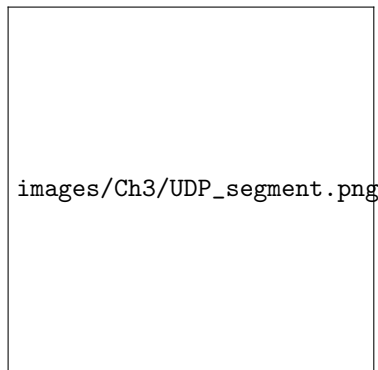


Abbildung 3.1: Aufbau eines UDP-Segments

UDP-Checksumme

Zur Erkennung von Fehlern bspw. flipped Bits wird eine Checksumme eingesetzt.
Sender:

- Segmentinhalt als Sequenz von 16-Bit Integers
- Checksumme: Addition (Einerkomplement) der Werte
- Sender trägt Ergebnis in Checksummen Feld ein

Empfänger:

- Berechnet ebenfalls die Checksumme des Segments
- vergleicht ob die beiden Summen übereinstimmen

Bemerkung: mit einer einfachen Checksumme können Fehler erkannt werden, es kann aber trotzdem noch zu Fehlern kommen

```

x 1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
x 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 Wraparound
x 1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0 Sum
x 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1 Checksum

```

Hierbei steht x jeweils nur als Platzhalter. Bei der binären Addition kommen Carry Bits zum Einsatz. Beim Wraparound wird das höchste Bit hinten aufaddiert und anschließend das Komplement der Zahl gebildet.

3.2.3 Vorteile von UDP

- Kein Verbindungsaufbau → weniger Delay
- sehr simples Protokoll
- kleiner Header → weniger Overhead
- keine Überlastkontrolle, UDP kann mit maximaler Leistung senden

Damit ist UDP vor allem für Anwendungen interessant die einen gewissen Paketverlust tolerieren können. Zu solchen Anwendungen gehören Streamingdienste, online Spiele und Internettelefonie.

3.3 Anforderungen an ein zuverlässiges Transportschicht Protokoll

- Fehler die durch unterliegenden Channel zustande kommen sollten erkannt werden. Wenn ein Fehler erkannt wird Korrektur:
 - versenden von ACK als Bestätigung
 - versenden von NAK als negative Bestätigung
 - Sender retransmittet bei ausbleibendem ACK oder NAK
- Behandlung von Duplikaten
 - Sender fügt jedem Paket Sequenznummer hinzu
 - Empfänger filtert so Duplikate
- Sender verwendet Timer
 - Sender wartet bestimmte Zeit auf ein ACK, kommt keines kann er retransmitten

Ein solches Protokoll hat aber ein großes Performanceproblem, da es immer im Stop-and-wait Betrieb ist. Um das zu lösen wird **Pipelining** eingesetzt.

3.3.1 Pipelining

Sender kann mehrere Pakete senden, bevor er ein ACK erwartet

- es muss mehr Sequenznummern geben
- nicht bestätigte Pakete müssen beim Sender gebuffert werden

Go-back-N

- Sender kann bis zu N nicht bestätigte Pakete in der Pipeline haben
- Empfänger sendet kumulative ACKs → bestätigt immer nur das letzte Paket im Datenstrom, vor dem es keine Unterbrechung gab
- Sender hat *einen* Timer für das älteste nicht bestätigte Paket
 - läuft der Timer ab, werden alle nicht-bestätigten Pakete erneut übertragen
- durch den retransmit aller N Pakete kann es beim Empfänger zu Duplikaten kommen
- auf Empfängerseite ist kein Puffer vorhanden → nicht erwartete / in falscher Reihenfolge angekommene Pakete werden daher verworfen

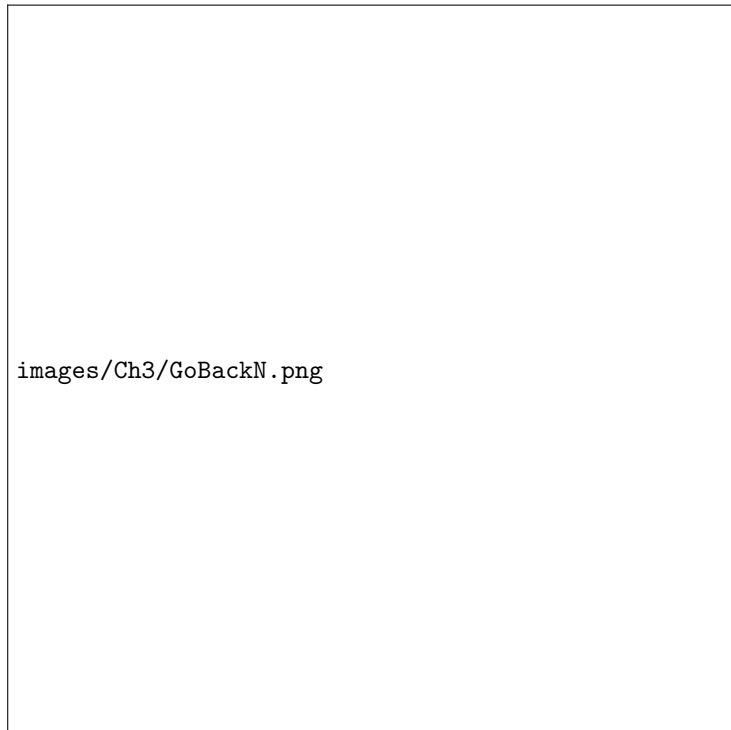


Abbildung 3.2: Sicht des Senders auf GBN

Bemerkung: siehe Moodle bzw. Buch für Applet/Visualisierung von GBN

Selective Repeat

- Empfänger bestätigt individuell alle korrekt empfangenen Pakete
 - buffert alle empfangene Pakete, die noch nicht an höhere Schicht geliefert werden können, da noch Lücken bestehen
- Sender schickt nur die Pakete erneut, für die ein ACK fehlt
 - Separater Timer für jedes nicht-bestätigte Paket
- Sender Window
 - N konsekutive Seq.-Nummern
 - beschränkt Sequenznummern der gesendeten und nicht bestätigten Pakete
- besitzt *mehrere* Timer
- Fenstergröße sollte kleiner gleich dem halben Sequenznummerbereichs sein, damit neue Pakete von Retransmits unterschieden werden können



Abbildung 3.3: Selective Repeat und Sender- und Empfängersicht

Bemerkung: siehe Buch bzw. Moodle für Applet/Visualisierung von Selective Repeat

3.4 Transmission Control Protocol (TCP)

3.4.1 Überblick

- **full-duplex** Datenverbindung:
 - bi-direktionaler Datenfluss innerhalb einer Verbindung
 - MTU: Maximum Transmission Unit ist die maximale Übertragungseinheit im IP Paket und Link-spezifisch. Das schließt den Header von TCP ein.
 - MSS: Maximum Segment Size gibt maximale Payload Größe vor und hängt von MTU ab
- **Verbindungsorientiert:**
 - Three-Way-Handshake: Austausch von Kontrollnachrichten zum Initialisieren des Zustandes von Sender und Empfänger beim Verbindungsaufbau
- **Flusskontrolle:** Empfänger kann Datenrate des Senders drosseln
- Punkt-zu-Punkte Verbindung: unicast (Ein Sender, ein Empfänger)
- zuverlässig geordneter Bytestream
- Pipelined: TCP **Überlast-** und **Flusskontrolle** setzen Windowgrenzen
- **Überlastkontrolle:** TCP drosselt eigene Sendeleistung bei Congestion im Netzwerk

TCP-Segment Struktur



Abbildung 3.4: Aufbau eines TCP-Segments

- *Sourceport* und *Destinationport*, wie bei UDP zur eindeutigen Zuordnung benötigt
- *Seq. number* und *ACK number* zählen Datenbytes und nicht Segmente
- *headerlength field*, da durch Optionen verschiedene Länge möglich ist
- *URG*: urgent data (normal nicht genutzt)
- *PSH*: push data now (normal nicht genutzt)
- *RST*, *SYN*, *FIN*: Verbindungsauf- und Abbau commands
- *Internetchecksum* analog zu UDP
- *receive Window*: anzahl an Bytes die Empfänger akzeptiert

Meist ist das Optionfeld leer, sodass der TCP-Header eine Größe von **20 Bytes** hat. Das Optionfeld kommt bspw. zum Einsatz wenn die MSS ausgetauscht wird.

3.4.2 Seq.- and ACK-numbers

- Sequenznummern:
 - Nummer des ersten Bytes im Segment im Bytestream
- ACKs:
 - Seq.-nummer des nächsten Bytes, welches die empfangende Seite erwartet
 - kumulative ACKs
- getrennte Zähler für jede Richtung



Abbildung 3.5: Triviales Beispiel einer TCP-Verbindung zwischen A & B

3.4.3 TCP Round Trip Time & Timeout

Wie soll der TCP Timeout gesetzt werden? Problem ist die RTT variiert sehr stark.

Idee: es wird die RTT gemessen und über mehrere solcher SampleRTT der Mittelwert gebildet.

Ein typische Möglichkeit für einen gewichteten Mittelwert wäre

$$\text{EstimatedRTT} = (1 - \alpha) \cdot \text{EstimatedRTT} + \alpha * \text{SampleRTT} \quad (3.1)$$

Damit nimmt der Einfluss vergangener Messungen exponentiell ab. Ein typischer Wert ist $\alpha = 0,125$

Das Timeoutintervall wird jetzt durch die EstimatedRTT plus eine Sicherheitsmarge berechnet

$$\text{DevRTT} = (1 - \beta) \cdot \text{DevRTT} + \beta |\text{SampleRTT} - \text{EstimatedRTT}| \quad (3.2)$$

Wobei $\beta = 0,25$ ein üblicher Wert ist.

$$\text{TimeoutIntervall} = \text{EstimatedRTT} + 4 \cdot \text{DevRTT} \quad (3.3)$$

3.4.4 Grundlegende TCP-Mechanismen

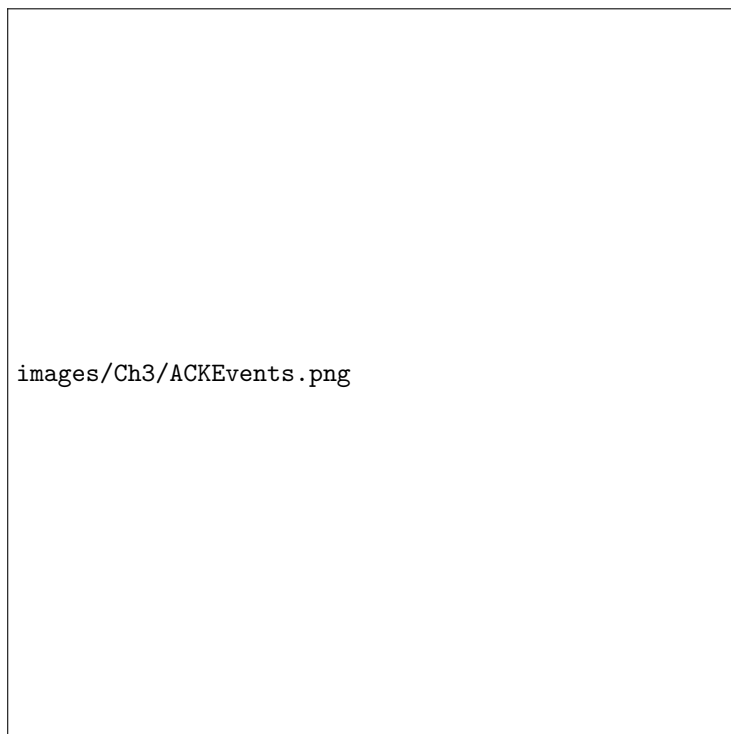


Abbildung 3.6: Erzeugen von ACKs

Fast Retransmit

Erhält der Sender 4ACKs mit gleicher Seq.-Nummber (**3 duplicate ACK**) erneutes Senden des nicht-bestätigten Segments mit kleinster Seq.-Nummer. Übertrage das Segment direkt, ohne auf den Timeout zu warten.



Abbildung 3.7: TCP Fast Retransmit bevor der Timer ausläuft

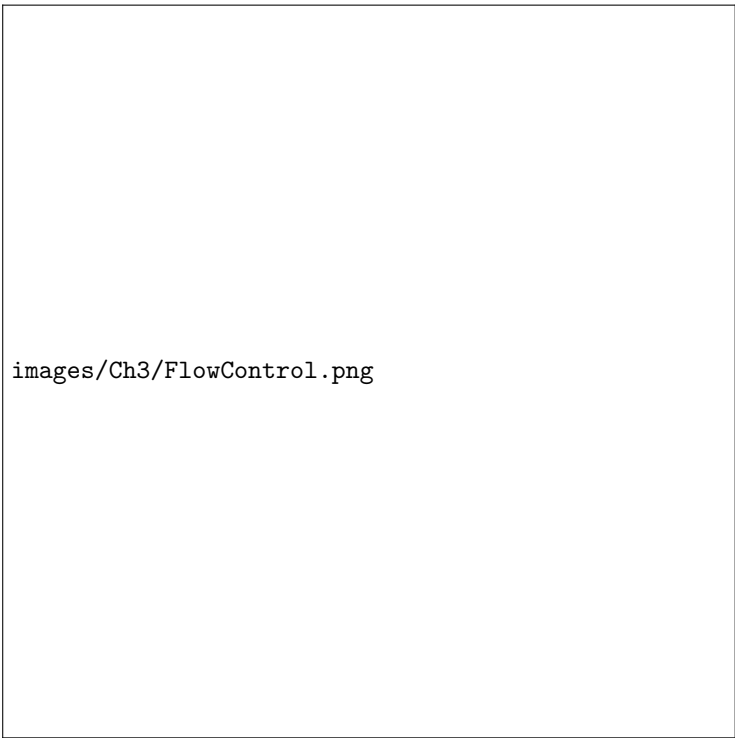
3.4.5 Pipelining bei TCP - GBN oder Selective Repeat

- TCP ACKs sind kumulativ
- out-of-order Segmente werden nicht individuell ACK
 - TCP-Sender merkt sich nur die niedrigste Seq.-Nummer des noch nicht bestätigten Segments (SendBase) und die Seq.-Nummer des nächsten noch nicht gesendeten Segments (NextSeqNum)
- es gibt TCP Implementationen die korrekt erhaltene out-of-order Segmente bufferen

TCPs Fehlerbehebungsmechanismus ist also ein Hybrid aus GBN und selective Repeat

3.4.6 TCP Flusskontrolle (Flowcontrol)

Prinzip: Empfänger kontrolliert Sender, so dass dieser den Buffer nicht zum Überlaufen bringt.



images/Ch3/FlowControl.png

Abbildung 3.8: Receive Window und receive Buffer

Empfänger teilt Sender die Größe seines receive Windows im Header des Segments.

Sender schickt niemals mehr als *receive Window* unbestätigte Daten

Damit ist garantiert, dass der Empfangsbuffer niemals überläuft.

Für den Fall, dass der Empfangsbuffer die Größe Null hat, werden als keepalive 1 Byte Segmente gesendet.

3.4.7 TCP 3-way Handshake



Abbildung 3.9: Ablauf eines TCP 3-way Handshakes

1. Der Client sendet ein spezielles TCP Segment an die Serverseite. Dieses Segment enthält keine Applicationlayerdaten. Das SYN-BIT ist gesetzt. Außerdem wählt der Client eine zufällige Seq.-Nummer (Sicherheitrelevant). Das Paket wird dann in ein IP-Datagramm verpackt und versendet
2. Kommt das vom Client gesendete Segment beim Server an, allokiert dieser den TCP-Buffer. Außerdem sendet er an den Client ein SYN-ACK Segment zurück. Er bestätigt damit das SYN-Segment (erhöht Seq.-Nummer im ACK wie gewohnt). Die Serverseite wählt ihre eigene Seq.-Nummer und sendet das Paket an den Client
3. Bei Empfang des SYN-ACK-Segments allokiert der Client seinen TCP-Buffer. Der Client sendet ein ACK an den Server in welchen er den Erhalt des SYN-ACK bestätigt

TCP Verbindungsabbau

Der TCP-Verbindungsabbau läuft analog zum Aufbau ab, siehe Abbildung 3.10.

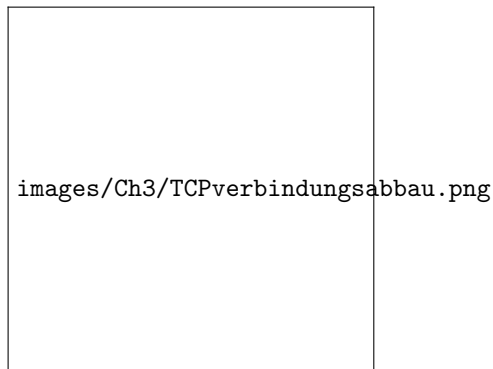


Abbildung 3.10: Abbau einer TCP-Verbindung

3.4.8 Prinzipien der Überlastkontrolle

Die Übertragungsrate einer Verbindung ist oft nicht durch die maximale Sendeleistung, sondern durch Links zwischen den Hosts beschränkt. Es macht also Sinn, die Senderate an die maximale Übertragungsrate anzupassen, um Paketverlust und Überlastung von Routern zwischen den Hosts zu vermeiden. Dies ist die Aufgabe der Überlastkontrolle (Congestion Control).

Ansätze der Überlastkontrolle

Ende-zu-Ende Überlastkontrolle:

- kein explizites Feedback vom Netzwerk
- Überlast wird von Endsystemen aus Verlust und Verzögerungen abgeschätzt

Netzwerk-unterstützte Überlastkontrolle

- Router liefern Feedback an Endsysteme

3.4.9 TCP Überlastkontrolle (Congestion Control)

Eine TCP-Verbindung hat ein Congestion- und ein Receive-Window. Das eine wird durch die Überlast- und das andere durch die Flusskontrolle bestimmt. Allgemein gilt

$$\text{LastByteSent} - \text{LastByteReceived} \leq \min[\text{cwnd}, \text{rwnd}] \quad (3.4)$$

TCP geht bei der Congestion Control nach den folgenden Prinzipien vor:

- Ein verlorenes Segment ist ein Indiz für Congestion und der Sender sollte seine Rate verringern

- Ein ACK Segment ist ein Indiz dafür, dass der Sender seine Rate weiter erhöhen kann
- Bandwidth probing: TCP erhöht bei erfolgreicher Übertragung die Rate so lange bis eine Übertragung schlussendlich fehlschlägt

Slow Start

Wenn die Verbindung beginnt wächst das cwnd exponentiell an, bis der erste Verlust auftritt.

- initiales cwnd = 1MSS
- verdopple cwnd in jeder RTT (cwnd++ bei jedem ACK)
- initiale Senderate ist langsam, steigt aber exponentiell

Aus einem Timeout wird auf Verlust geschlossen. cwnd wird auf 1 MSS gesetzt. Das cwnd wächst nun erneut wie beim Slow Start exponentiell, bis es den Threshold (Hälfte des vorherigen Paketverlusts) erreicht, dannach lineares Wachstum.

Congestion Avoidance

Im Congestion Avoidance Modus wird pro RTT das cwnd nur um eins erhöht.



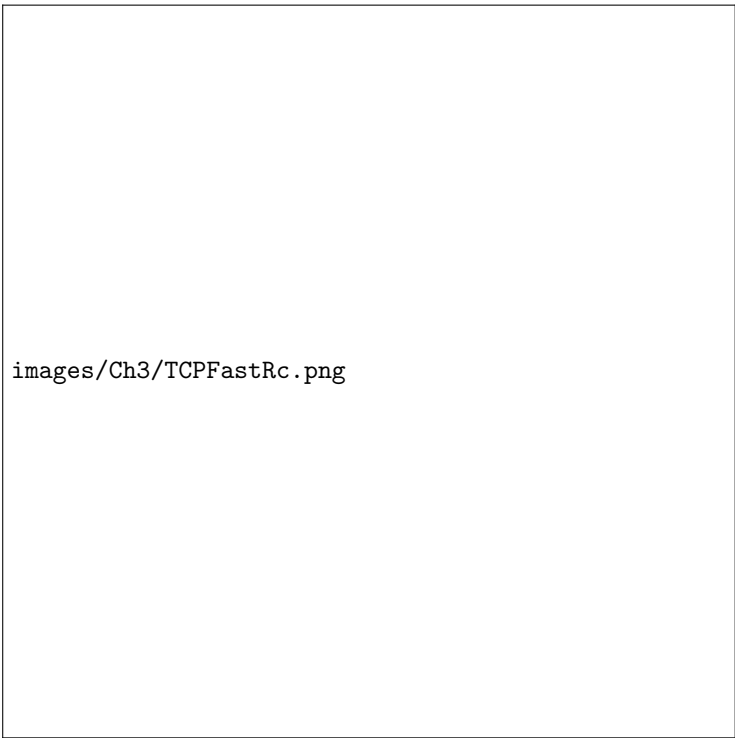
Abbildung 3.11: Automat der TCP Congestion Control

Fastrecovery

Bei 3 duplicate ACK geht TCP vom Slow Start/Congestion Avoidance Modus in den fast recovery Modus über. In diesem wird pro duplicate ACK das cwnd um eins erhöht (sofern möglich).

Kommt ein neues ACK an, geht die Verbindung in den Congestion Avoidance Modus über. Kommt es zu einem Timeout wird die MSS auf 1 gesetzt und es kommt zu einem erneuten Slow Start.

Fast recovery ist optional und nicht in jeder TCP-Implementierung enthalten. Siehe hier bspw. TCP Tahoe und TCP Reno.



images/Ch3/TCPFastRc.png

Abbildung 3.12: Vergleich von TCP Tahoe und TCP Reno

Additive Increase & multiplicative decrease

Additive Increase und multiplicative Decrease beschreibt den Mechanismus, dass das `cwnd` sich immer nur additiv erhöht (im Slow Start `cwnd++` für jedes erhaltene ACK, im Congestion Avoidance Mode `cwnd++` pro RTT). Im Gegensatz dazu schrumpft das `cwnd` im Fall eines Paketverlusts multiplikativ (abhängig von der Implementierung zurück auf `cwnd = 1 MSS` oder auf `cwnd = 0.5 cwnd`).

Bemerkung zur Congestion Control

In welcher Form die Congestion Control umgesetzt wird, hängt stark von der TCP Implementierung ab.

Es gibt auch explicit congestion notification, hierbei handelt es sich um Router unterstützte Congestion Control.

- Zwei bits im IP Header (ToS Feld) können vom Router gesetzt werden, um Congestion zu markieren
- Congestion Indication werden bis zum Empfänger durchgestellt
- Empfänger reflektiert diese Meldung im ECE Bit von ACK Segmenten um Sender zu benachrichtigen

3.4.10 TCP Fairness

Das Ziel ist, wenn sich K TCP Verbindungen einen gemeinsamen Bottleneck-link mit Bandbreite R teilen, sollte jeder Verbindung eine Bandbreite von R/K zugewiesen bekommen.

TCP ist fair, da AIMD angewendet wird und sich so ein Gleichgewicht zwischen allen Verbindungen einstellt.

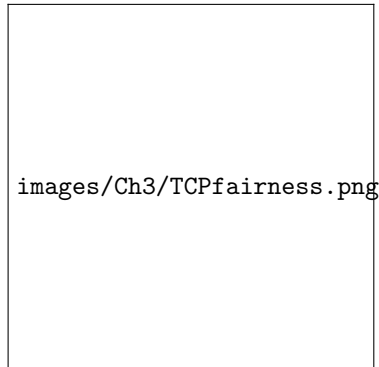


Abbildung 3.13: TCP Fairness durch AIMD

Fairness und UDP

Wie bereits erwähnt nutzen viele Multimedia Anwendungen oft kein TCP und werden daher auch nicht durch die Congestion Control gedrosselt. UDP entspricht nicht dem Fairness-Gedanken von TCP und benachteiligt diese Verbindungen.

Fairness und parallele TCP-Verbindungen

Anwendungen können mehrere parallele TCP Verbindungen zwischen zwei Hosts öffnen, dies wird oft von Webbrowsern eingesetzt. Auch wenn dies technisch gesehen der Fairnessbedingung entspricht erhält ein Host mit mehreren TCP Verbindungen auch mehr Bandbreite.

3.4.11 Multipath TCP

TCP Variante mit Unterstützung von multi-homing/multipath.

- Multi-homing: ein Host hat mehrere Endpunkte/Interfaces
- Multipath: zwischen zwei Hosts werden mehrere Pfade für eine Verbindung genutzt

Herausforderungen bei der Umsetzung:

- Komplexe Congestion Control

- Rückwärtskompatibilität: Fallback auf klassisches TCP
- Verbindungsabbrüche auf einzelnen Pfaden

Kapitel 4

Netzwerkschicht

4.1 Überblick: Netzwerkschicht

- Transport von Segmenten vom Sender zum Empfänger
- Einkapseln von Segmenten in Datagramme
- Empfänger: weiterleiten der eingegangenen Segmente an die Transportschicht
- Netzwerkschichtprotokolle in jedem Host und Router implementiert (erste Schicht die auch im Netzwerk-Core vorhandene ist)
- Router parsen Netzwerkheader aller IP-Datagramme

Aufgaben der Netzwerkschicht

- **Forwarding:** Weiterleiten von Paketen im Router. Findet vollständig auf der Data-Plane statt
- **Routing:** Routing ist der Vorgang, Pakete durch ein Netzwerk von einem Host zu einem Anderen zu transportieren. Dafür werden Pakete in Routern forwarded. Routing wird in die control-plane von Routern eingeordnet.

Data plane

- Lokale Funktionalität im Router
- Forwarding

Control plane

- Netzwerkweite Funktion
- Ansätze für Control plane:
 - herkömmliches routing im Router implementiert
 - Software-defined networking

4.1.1 Service Modell der Netzwerkschicht

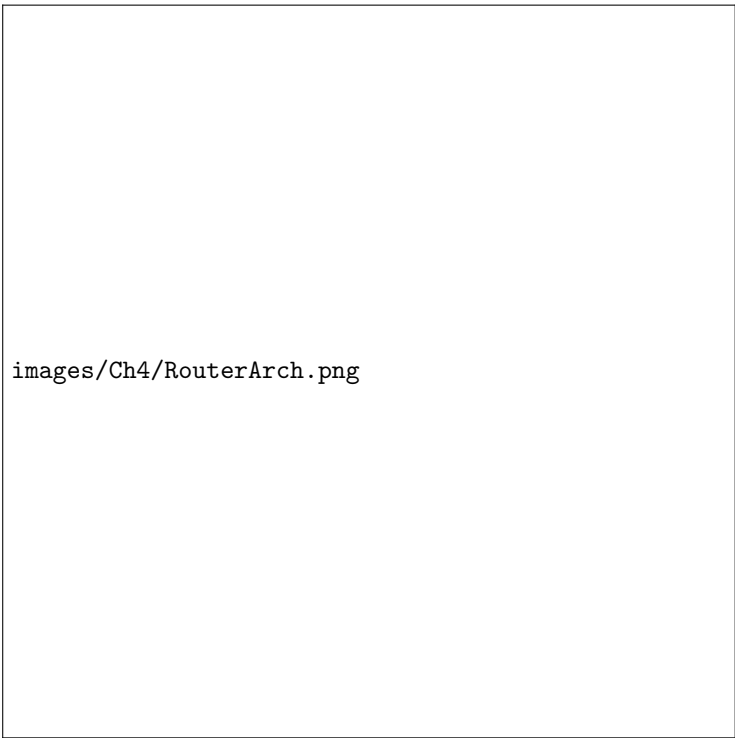
Die Netzwerkschicht kann ebenfalls eine bestimmte Dienstgüte anbieten. Einige Dienste wären

- garantierte Zustellung
- garantierte Zustellung mit weniger als 40ms delay
- geordnete Zustellung
- minimale Datenrate

Tatsächlich bietet das IP keinen dieser Dienste an. Das IP operiert nur nach dem *best effort* Modell.

4.2 Routerarchitektur

- **Inputports:** terminiert eingehenden Link. Hier geschieht der Lookup d.h. der Outputport wird anhand der Forwardingtable festgelegt
- **Switching fabric:** verbindet die Input- mit den Outputports
- **Outputport:** führt die notwendigen Link- & Physicallayer Funktionen aus um das Datagram zu senden
- **Routing processor:** der Routingprozessor führt die Routingprotokolle aus und enthält die Routingtables, aus dieser berechnet er die Forwardingtable

The image area is mostly blank, with the text 'images/Ch4/RouterArch.png' located on the left side. This text likely refers to a diagram illustrating the internal architecture of a router, which typically shows components like the CPU, memory, routing table, and various interfaces.

images/Ch4/RouterArch.png

Abbildung 4.1: Aufbau eines Routers

Inputport processing

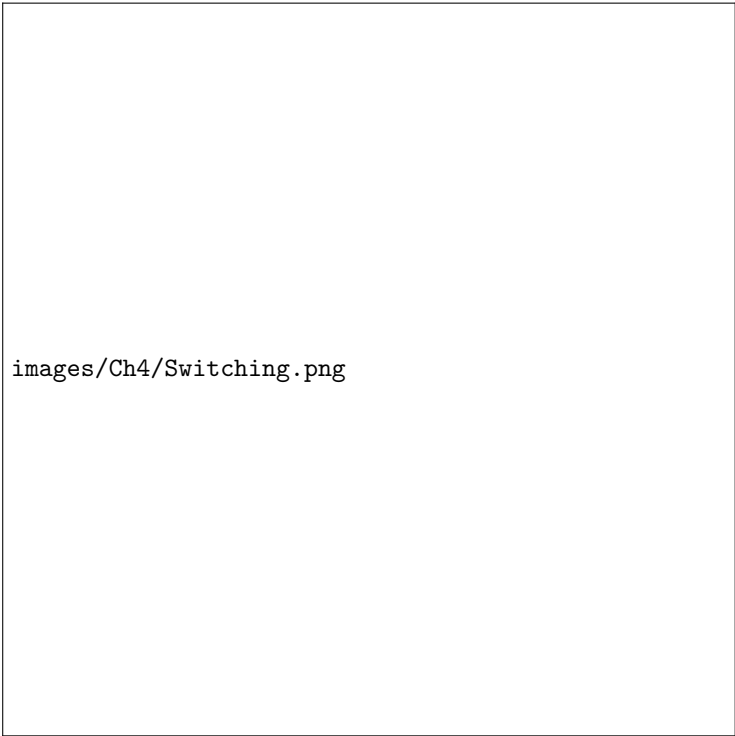


Abbildung 4.2: Inputport processing

Zur Bestimmung des Outputports wird *prefix matching* verwendet. Beim Durchsuchen der Routingtable nach einer Zieladresse wird der längste Adresspräfix verwendet, der mit der Zieladresse übereinstimmt. Im Bereich von Gigabit Übertragungsraten muss dieser Lookup binnen Nanosekunden geschehen. Folglich ist das ganze in Hardware implementiert.

4.2.1 Switching

Die Switching fabric verbindet die Input- mit den Outputports. Es ist daher essentiell dass der Delay hierbei so gering wie möglich ist.



images/Ch4/Switching.png

Abbildung 4.3: Verschiedene Implementierungen von Switching

Switching via memory

CPU übernimmt forwarding. Es kann immer nur ein ein Paket zur gleichen Zeit forwarded werden.

Switching via a bus

Paket wird direkt von Inputport zum Outputport transportiert ohne dass der Routingprozessor interferiert. Dies geschieht mit Hilfe eines Switch internen Labels, sodass der korrekte Outputport das Paket akzeptiert und verarbeitet. Die maximale Geschwindigkeit wird hier also durch die Geschwindigkeit des Buses limitiert.

Switching via an interconnection network

Hiermit können mehrere Pakete auf einmal forwarded werden, falls diese für verschiedene Outputports bestimmt sind.

4.2.2 Outputport processing



Abbildung 4.4: Outputport processing

- Buffern notwendig, wenn Datagramme schneller ankommen als Datenrate des Ausgangsport eine Weiterleitung zulässt
- Scheduling Strategie bestimmt, welche Datagramme verschickt werden (z.B. Fifo oder Priorität)

Nach RFC 3439 ist eine Daumenregel für die Größe des Pufferspeichers

$$\text{Buffer} = \frac{\text{RTT} \cdot C}{\sqrt{N}}$$

,wobei C die Linkkapazität und N die Anzahl der Flows ist.

4.3 Internet Protokoll (IP)



Abbildung 4.5: Überblick über das IP

4.3.1 IPv4

IPv4 Datagram Format



Abbildung 4.6: Aufbau eines IPv4 Datagrams

- **Version:** spezifiziert welche Protokollversion verwendet wird
- **Headerlength:** Header kann Optionen enthalten und hat dann beliebige Länge, sind die Optionen leer ist der Header 20 Byte groß
- **Type of Service:** Unterscheidung von Echtzeitdatagrammen (Telefonie etc.) von anderen
- **Datagram length:** totale Länge des IP- Datagrams **inkl.** Header
- **Fragmentation:** wird genutzt um Fragmentierung zu signalisieren und enthält zusätzliche Informationen (Offset etc.)
- **Time-to-live:** sorgt dafür, dass Datagramme nicht unendliche Länge im Netzwerk zirkulieren können. TTL wird dekrementiert wenn es von einem Router verarbeitet wird. Erreicht die TTL 0 wird das Datagram verworfen
- **Protocol:** zeigt an welches Transportschichtprotokoll verwendet wird
- **Header Checksum:** für Fehlererkennung. Berechnung ähnlich der Internet Checksum. Beachte: Summe muss an jedem Router neu berechnet werden, da sich die TTL ändert

- **Source and Destination IP-Adresse:**
- **Options:**
- **Data:** payload

Sowohl der TCP als auch der IPv4 Header haben ohne Optionen eine Länge von 20 Bytes.

IPv4 Fragmentierung

Wenn ein Paket mit einer Größe größer als die MTU des ausgehenden Links den Router erreicht, wird dieses in großen Fragmentiert, die der ausgehende Link senden kann.

Erklärung der Fragmentierung von IPv4 Paketen an einem Beispiel:

Aufgabenstellung: Ein IPv4-Paket mit 20 Bytes Header und 700 Bytes Payload (Nutzlast) erreicht einen Router. Der vom Router gewählte ausgehende Link besitzt eine MTU (Maximum Transmission Unit) von 340 Bytes. Führen Sie die notwendige IP-Fragmentierung durch, in dem Sie alle Fragmente inklusive relevanter Header-Felder auflisten. Jeder Eintrag soll hierbei ein einzelnes IP-Paket darstellen und die Header Felder ID, More Fragments Flag und Offset auflisten. Geben Sie ebenfalls jeweils die Größe von Header, Payload und Gesamtpaketgröße in Byte an. Erstellen Sie dabei genau so viele Fragmente wie nötig. Verwenden Sie als initiale ID 461.

Lösung: Eingangspaket:

20Bytes Header + 700 Bytes Payload = 720 Byte Paket
ID=461

fragflag = 0 und offset = 0

Da der Ausgangslink des Routers nur eine maxiamle MTU von 340 Bytes hat, muss das Paket fragmentiert werden.

Ausgang: Paket 1

20Bytes Header + 320 Bytes Payload = 340 Byte Paket
ID=461

fragflag = 1 und offset = 0

Paket 2

20 Bytes Header + 320 Bytes Payload = 340 Byte Paket
ID = 461

fragflag = 1 und offset = $320/8 = 40$

Paket 3

20 Bytes Header + 60 Bytes Payload = 80 Byte Paket
ID = 461

fragflag = 0 und offset = $640/8 = 80$

IPv4 Adressierung

Eine IP-Adresse ist aufgebaut aus Subnet-Teil und dem Host-Teil.
CDIR-Adressierung:

- CIDR: Classless InterDomain Routing
- erlaubt Subnetzne mit Netzwerkteil beliebiger Länge
- 200.23.16.0/23 ist in CIDR-Adressdarstellung

4.3.2 Zuweisung von IP-Adressen

Ein Interface kann seine IP-Adresse entweder fest durch einen Systemadmin zugewiesen bekommen oder dynamisch durch Verwendung von bspw. DHCP.

Dynamic Host Configuration Protocol (DHCP)

- dynamische Zuweisung von IP-Adressen and Interfaces eines Hosts durch DHCP-Server
- Lease Konzept: nur temporäre Vergabe
- Überblick:
 - Host broadcastet *DHCP discover*-Nachricht
 - DHCP-Server antwortet mit *DHCP offer*
 - Host fragt IP-Adresse an *DHCP request*
 - DHCP-Server bestätigt Adresse *DHCP ack*

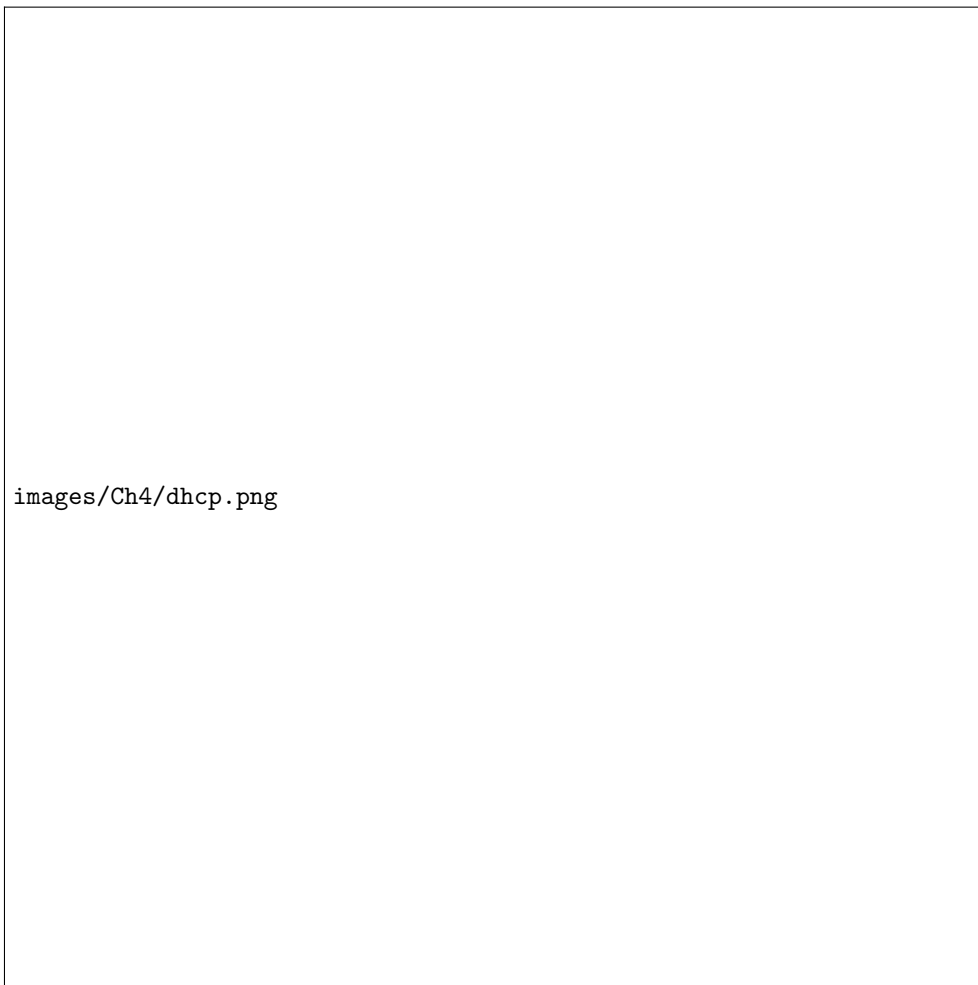


Abbildung 4.7: DHCP

4.3.3 Network Address Translation (NAT)

- Mittel gegen IPv4 Adressknappheit
- Lokales Netzwerk belegt nur eine globale IP-Adresse
- Keine Änderungen der internen Adressen bei ISP Wechsel
- Geräte im internen Netz nicht von außen adressierbar
- Umstritten, da Router die Header der anderen Schichten verändern

Verwendung von UPnP um Hosts in einem NAT zu adressieren (erfahren der öffentlichen IP-Adresse und Port-Mappings (temporär)).

Eine andere Möglichkeit ist Relaying. NATed Client erstellt Verbindung zu Relayserver, dieser leitet Pakete an anderen Client weiter.

4.3.4 Internet Control Message Protocol (ICMP)

- Kommunikation von Netzwerkschichtinformationen zwischen Hosts & Routern
- ICMP-Message besteht aus Type und Code
- einige der verfügbaren Nachrichten
 - echo reply (ping)
 - dest network/host/protocol/port unreachable
 - dest network/host unknown
 - route advertisement
 - router discovery
 - TTL expired
 - bad IP header

ICMP und Traceroute (D3Ü7)

Bei Traceroute werden nach dem die URL per DNS aufgelöst wurde je 3 UDP Datagramme mit einer TTL von i versendet werden. i beginnt hierbei bei 1 und wird so lange inkrementiert, bis der gesuchte Hop erreicht wurde oder die maximale Anzahl an Hops erreicht wurde (standardmäßig 30).

Kommt nun also bei Router i ein solches Paket an bemerkt dieser dass er die TTL nicht mehr dekrementieren kann, da die TTL dann 0 ist. Er sendet ein ICMP Paket vom Type 11, welches für Time-to-live exceeded steht, an den Sender zurück. Traceroute misst die Zeit zwischen senden des UDP Datagrams und erhalten des ICMP Pakets und gibt diese als RTT zum i -ten Router an.

Da im UDP Datagram eine Portnummer gewählt wird, welche sehr wahrscheinlich nicht in Benutzung ist, kann mit einem ICMP Paket vom Type 3 (dest. port unreachable) bestätigt werden, dass der Host erreicht wurde.

4.3.5 IPv6

- IPv4 Adressraum zu klein
- Header soll feste Länge haben → schnellere Verarbeitung
- bessere Unterstützung für Quality of Service

IPv6 Header

- Vergrößerung des Adressraums von 32 auf 128bit
- **Traffic-Class:** analog zu Type of Service bei IPv4
- **Flow Label:** Pakete können einem Flow zugeordnet werden und dann bevorzugt behandelt werden (bspw. bei VoIP oder Media-Paketen, nicht bei Filetransfer o.ä.)

- **Payload length:** gibt Länge des Pakets an, das *nach* dem Header kommt
- **Next Header:** gibt an welches Protokoll der Payload verwendet (TCP,UDP)
- **Hop limit:** klar, wird von jedem Router, der das Paket forwarded, dekrementiert bis 0 erreicht wird

Die folgenden Felder finden in IPv6 keine Verwendung mehr

- Fragmentation: IPv6 sieht keine Fragmentierung von Paketen mehr vor. Wird ein zu großes Paket versendet, sendet der Router eine ICMP zurück
- Checksum: schnelleres forwarden ohne Neuberechnung der Summe bei jedem Router und TCP/UDP führen ebenfalls eine Checksum durch
- Options: IPv6 Header hat eine feste Größe von 40 Byte

Kürzen von IPv6 Adressen

- weglassen von führenden Nullen
- einmaliges ersetzen von Nullenblock durch ::

Beispiele:

2001:0DB8:AC10:0000:0000:0000:FE01 --> 2001:DB8:AC10::FE01
 0000:000D:0000:0000:000A:0000:0000:000D --> 0:D::A:0:0:D
 2001:0047:0011:0000:0000:AFFE:0000:0000 --> 2001:47:11:0:0:AFFE::

4.4 Routing

Router sind Hosts mit mehreren Netzwerkinterfaces und damit mehreren IP Adressen. Aufgabe des Routers ist das Routing, d.h. das Weiterleiten von Paketen und das Entscheiden, an welches Interface ein eingehendes Paket weitergeleitet wird. Der Router besitzt dafür eine **Control Plane**, wo ein Routing Algorithmus Pfade für Pakete bestimmt und eine **Routingtabelle** (Forwarding Table) befüllt, anhand derer auf der **Data Plane** eingehende Pakete weitergeleitet werden. Oft ist dies ein **Destination-based Forwarding**, das Ausgangsinterface wird in der Routingtabelle mit Hilfe von **Longest Prefix Matching** mit der Zieladresse ermittelt.

Für bessere Skalierung und administrative Autonomie werden Netzwerke üblicherweise in viele **Autonomous Systems (AS)** unterteilt, deren innere Struktur nach außen nicht bekannt sein muss, und das im Inneren andere Routingprotokolle verwenden kann, als außerhalb, zwischen mehreren AS. Die Router eines AS, die nach außen hin sichtbar sind und mit Routern anderer AS kommunizieren, werden **Border Router** genannt.

4.4.1 Intra-AS Routing

Intra-AS Routing bezeichnet das Routen von Paketen im eigenen Netzwerk / innerhalb eines AS. Ein gängiges Intra-AS Routing Protokoll ist **Open Shortest Path First (OSPF)**. OSPF ist ein Link-State Algorithmus, zum Ausführen

des Algorithmus muss also die gesamte Netzwerktopologie bekannt sein. Dazu wird das Netzwerk mit Paketen mit Link State Informationen geflutet und dann auf jedem Knoten die Routen mit Dijkstra's Algorithmus berechnet. OSPF verfügt über fortgeschrittenen Funktionen wie das Authentisieren von Nachrichten, mehreren Pfaden mit gleichen Kosten oder mehreren Kosten pro Link (je nach Type of Service). In großen Netzen können mittels **hierarchischem OSPF** Teile des Netzes zu Areas zusammengefasst werden. Diese Areas sind mittels Area Border Routers mit dem Backbone verbunden, wo Backbone Router verschiedene Areas verbinden.

4.4.2 Inter-AS Routing

Für Inter-AS Routing wird üblicherweise das **Border Gateway Protocol (BGP)** verwendet. AS geben mittels eBGP bekannt, welche Routen ihnen bekannt sind. Diese Routen bestehen aus dem Präfix des entsprechenden Netzes und Attributen, wichtige Attribute sind

- **AS-PATH** Pfad, den das Advertisement genommen hat, jedes AS, das ein Advertisement weiterleitet, fügt diesem Attribut die eigene AS Nummer an.
- **NEXT-HOP** IP Adresse des Routers, der Pakete an das Ziel Netzwerk weiterleiten kann. Bei eBGP typischerweise die Absender IP-Adresse

Mittels iBGP können Border Router empfangene Informationen an andere Border Router im gleichen AS weiterleiten. Die eigentliche Routenauswahl kann dann anhand mehrerer Kriterien geschehen, wie bspw. kürzester AS-PATH, nächster NEXT-HOP, eigener Policies und mehr.

4.4.3 Routing Algorithmen

Dijkstra's Algorithmus

Übung machen \implies lernt man mehr als das hier zu lesen.

Distance Vector

Kapitel 5

Linklayer

5.1 Überblick Linklayer

Der Link Layer ist im Schichtenmodell zwischen Netzwerk- und Physical Layer einzuordnen, seine Aufgabe ist es, (IP-) Datagramme des Netzwerklayers verpackt als Frame von einem Host an einen anderen, direkt verbundenen Host zu senden. Wichtiger Aspekt dieses Layers ist **MAC** (Medium Access Control).

5.2 Fehlererkennung

5.2.1 Parity Check

Einzelbitparität

Hier wird den Daten ein einzelnes Bit angefügt, welches 0 ist, falls die Anzahl von 1-Bits in den Daten gerade ist, und 1 sonst. Durch überprüfen der Parity Information kann ein einzelner Bitfehler erkannt werden.

Zweidimensionale Parität

Hier werden die Daten in Matrixform angeordnet, und dann die Parität jeder Zeile und jeder Spalte berechnet und angefügt. Bei einem Bitfehler tritt ein Parity Error sowohl in der entsprechenden Zeile als auch der Spalte auf, der Fehler kann dadurch lokalisiert und korrigiert werden.

5.2.2 Cyclic Redundancy Check (CRC)

Die Datenbits der Länge d werden als Binärzahl D interpretiert. Um eine CRC Checksumme von Länge r zu berechnen, wird ein Generator G der Länge $r + 1$ benötigt. R wird dann so gewählt, dass $\langle D, R \rangle$ durch G teilbar ist. Für das Berechnen der CRC können $\langle D, R \rangle$ und G als Polynome vom Grad $d + r$ bzw. $r + 1$ angesehen werden. Für das Berechnen der CRC wird dann eine Polynomdivision von $\langle D, R \rangle$ und G durchgeführt. Der Rest, den diese Polynomdivision ergibt, stellt dann gerade R dar.

$$\langle D, R \rangle : G = \text{Polynom} + \frac{R}{G}$$

5.3 MAC Protokolle

Die Aufgabe eines MAC Protokolls ist es, einen gemeinsamen Kanal für mehrere Clients nutzbar zu machen. Diese können in folgende Kategorien eingeordnet werden:

- **Channel Partitioning**
Aufteilen des Kanals in Teile (Frequenzen, Zeitslots, etc)
- **Random Access**
Direktes Senden und eventuelles Handling von Kollisionen
- **Taking Turns**
Abwechselnder Kanalzugriff von jeweils einem Client

5.3.1 TDMA, FDMA

TDMA (Time Division Multiple Access) und **FDMA** (Frequency Division Multiple Access) sind einfache Beispiele für Channel Partitioning: Jedem Client wird ein fester Zeitslot bzw eine feste Frequenz zugeteilt. Auf diese Weise kann der Kanal aber nur sehr ineffizient genutzt werden und die Anzahl der Clients ist begrenzt.

5.3.2 Random Access Protokolle

Random Access Protokolle versprechen bei gering genutztem Kanal deutlich bessere Latenz und höhere Bandbreitennutzung als Channel Partitioning Protokolle. Zentrale Problemstellung ist hier der Umgang mit Kollisionen, wenn mehrere Clients gleichzeitig den Kanal nutzen wollen.

ALOHA

Beim **ALOHA** Protokoll wird ein Frame sofort gesendet. Wenn während der Übertragung eine Kollision auftritt, muss der Frame zu späterem Zeitpunkt erneut gesendet werden.

Eine verbesserte Version des ALOHA Protokolles ist **Slotted ALOHA**. Hier sind alle Frames gleich groß, und der Kanal wird in Zeitslots unterteilt, die jeweils der Übertragungsdauer eines Frames entsprechen. Die Clients sind nun Zeitsynchronisiert, und beginnen die Übertragung eines Paketes nur am Anfang eines Zeitslots. Kollisionen können hier nur im jeweils gleichen Zeitslot auftreten, und nicht wie bei reinem **ALOHA** auch mit Frames, die früher oder später gesendet werden. Die Effizienz (Anteil der Sendezeit, der zur effektiven Übertragung genutzt wird) von **Slotted ALOHA** beträgt 37%, die von reinem **ALOHA** nur 18%.

CSMA

Bei **Carrier Sense Multiple Access** wird vor dem Übertragen eines Frames sichergestellt, dass der Kanal momentan ungenutzt ist. Dies eliminiert jedoch nicht alle Kollisionen: Aufgrund des Propagation Delay kann es vorkommen, dass dennoch Kollisionen auftreten, welche erkannt werden müssen und eine Retransmission erforderlich machen.

Verbessert wird dies durch **CSMA/CD** (CSMA mit *Collision Detection*). Hier wird eine Kollision bereits während der Übertragung erkannt, und diese daraufhin abgebrochen. So wird weniger Zeit damit verbracht, Daten zu senden, die aufgrund einer Kollision aber unbrauchbar werden. Die Effizienz von **CSMA/CD** geht im best case gegen 1. Eine weitere Variante ist **CDMA/CA** (CSMA mit *Collision Avoidance*).

5.3.3 Taking Turns Protokolle

Polling

Für Polling fordert ein Client (**Master**) die anderen Clients (**Slaves**) einzeln zum Senden auf. Dies führt zu erheblichem **Polling Overhead** und einem **Single Point of Failure**, ermöglicht aber sehr einfache Clients, wenn diese nur die Rolle des **Slave** einnehmen. (Genutzt z.B. bei **Bluetooth**).

Token Passing

Die Clients werden in einer ringförmigen Topologie angeordnet, und es wird ein virtueller **Token** generiert, den immer nur genau ein Client besitzt. Nur der Besitzer des Tokens ist zur Nutzung des Kanals berechtigt. Auch dieser Ansatz führt zu erheblicher Latenz, und es werden weitere Mechanismen z.B. für Token (Re-)Generation benötigt.

5.4 MAC Adressen

tl;dr 48bit, globally unique für jedes NIC, kann man kaufen: 16777214 für 2905\$, 1052676 für 1745\$ oder 4096 für 730\$.

5.5 Address Resolution Protocol ARP

ARP ermöglicht es Teilnehmern in einem LAN, einer IP Adresse die passende **MAC Adresse** zuzuordnen. Dafür werden Broadcasts mit Zieladresse FF–FF–FF–FF–FF–FF gesendet, die von jedem Client im LAN empfangen werden. Der Client mit passender IP-Adresse antwortet dem Anfragenden mit seiner MAC-Adresse. Diese Informationen werden von Clients gecached, besitzen aber einen Timeout.

Kapitel 6

Drahtlose-Kommunikation

6.1 Eigenschaften Drahtloser Links im Vergleich zu drahtgebundenen Links

- Starke Dämpfung (*Path Loss*)
- Interferenz mit anderen Sendern
- Multipath Propagation
- Hohe Bitfehlerrate
- Variabler Signal-Noise-Ratio *SNR*

6.1.1 Hidden Terminal Problem

Der hohe Path Loss kann dazu führen, dass zwei Clients drahtlos mit einer Basisstation kommunizieren wollen, den jeweils anderen Client aber nicht detektieren. Die Clients können so keine Collision Detection betreiben und müssen auf Informationen der Basisstation zurückgreifen um kollisionsfreie Kommunikation zu gewährleisten. So deuten z.B. empfangene ACKs von der Basisstation darauf hin, dass diese gerade mit einem anderen Client kommuniziert.

6.2 802.11

Bisschen wie Ethernet, aber Funk:

6.2.1 CSMA/CA

802.11 Wireless LAN benutzt *CSMA/CA* für Mehrfachzugriff. Dabei sendet ein Client den ganzen Frame, wenn der Kanal für gewisse Zeit unbelegt ist. Wenn der Kanal belegt ist, wird nach einem zufälligen Backoff Intervall gesendet. Falls kein ACK empfangen wird, findet nach zufälligem Zeitintervall eine Retransmission statt.

Bemerkung 802.11 implementiert keine Collision Detection, da die Sendeleistung oft ein Vielfaches der empfangenen Signalstärke ist.

Kapitel 7

Netzwerksicherheit

7.1 Grundlegende Prinzipien der Netzwerksicherheit

7.1.1 Prinzipien

- **Vertraulichkeit:** nur Sender und rechtmäßiger Empfänger können auf Inhalt zugreifen
- **Integrität:** Sender und Empfänger stellen sicher, dass die Nachricht während der Übertragung nicht verändert wird
- **Verfügbarkeit:** Dienste müssen für den Benutzer zugreifbar und verfügbar sein

7.1.2 Angriffsformen

- **Eavesdropping:** abhören von Nachrichten
- **Man-in-the-middle:** Nachrichten abfangen und modifizieren
- **Spoofing:** sich als eine andere Partei ausgeben
- **Hijacking:** bereits bestehende Verbindung übernehmen
- **Dos** denial of service attacks

7.2 Symmetrische Verschlüsselungsverfahren

In symmetrischen Verschlüsselungsverfahren teilen sich Sender und Empfänger einen **gemeinsamen** symmetrischen Schlüssel K_S . Dieser gemeinsame Schlüssel wird sowohl zum Verschlüsseln als auch zum Entschlüsseln verwendet. Es gilt also $K_S(K_S(m)) = m$ wobei m die Nachricht ist.

Beispiel für ein einfaches symmetrisches Verschlüsselungsverfahren ist die Cäsarchiffre, dieses Verfahren gehört zur Kategorie der **Monoalphabetischen** Substitutionsschiffren. Komplexere Verfahren sind bspw. **polyalphabetische** Substitutionsschiffren.

7.2.1 Moderne symmetrische Chiffren

DES steht für *Data Encryption Standard* mit einer Schlüssellänge von 56bit und mit 64bit Klartextblöcken. Bei DES handelt es sich um eine Blockchiffre. Einfaches DES kann per brute force innerhalb eines Tages entschlüsselt werden. Eine sicherere Variante ist **3DES**, bei welchem drei mal mit drei verschiedenen Schlüsseln verschlüsselt wird.

Ein weiteres symmetrisches Verschlüsselungsverfahren ist **AES** (Advanced Encryption Standard). Dieser nutzt eine Blocklänge von 128 Bit und Schlüssellängen von 128, 192 oder 256 Bit. Durch die erhöhte Schlüssellänge kann die Nachricht nicht mehr per brute force entschlüsselt werden.

Bei symmetrischen Chiffren müssen der Sender und der Empfänger einen gemeinsamen Schlüssel kennen. Damit entsteht das Problem, des sicheren Transports dieses Schlüssels über einen unzuverlässigen Kanal. Eine mögliche Lösung für dieses Problem ist **asymmetrische Kryptographie** (public key cryptography). Hierbei besitzt der Empfänger/Sender sowohl einen öffentlichen als auch einen privaten Schlüssel.

7.3 Public Key Kryptographie

Jeder Empfänger/Sender hat einen privaten und einen öffentlichen Schlüssel. Eine Nachricht wird mit dem öffentlichen Schlüssel des Empfängers verschlüsselt, damit dieser die Nachricht mit seinem privaten Schlüssel entschlüsseln kann. Es gilt also $K^-(K^+(m)) = m$ wobei K^- der private und K^+ der öffentliche Schlüssel ist.

Es ist quasi unmöglich aus dem öffentlichen Schlüssel den privaten Schlüssel zu berechnen.

7.3.1 Praktische Anwendung

Operationen mit RSA sind sehr rechenaufwendig und man möchte dies so gut es geht vermeiden. Vor allem sind Verfahren wie DES oder AES mindestens 100x schneller als RSA.

Ein Lösung hierfür ist public key encryption zu verwenden um einen symmetrischen Schlüssel zwischen Sender und Empfänger austauschen. Der eigentliche Datenfluss wird dann mit einem symmetrischen Verschlüsselungsverfahren verschlüsselt. Ein Beispiel hierfür ist der **Diffie-Hellman key exchange**.

7.4 Digitale Signaturen

Es wird eine Möglichkeit gesucht, wie der Empfänger sicher gehen kann, dass das von ihm empfangene Paket tatsächlich vom Sender stammt. Eine Möglichkeit dies sicherzustellen sind digitale Signaturen.

Beispiel: einfache digitale Signatur

Es soll eine einfache digitale Signatur für eine Nachricht m gefunden werden. Es wird die Nachricht m in eine kryptographische hash-Funktion gegeben. Diese

erzeugt aus der Nachricht m von beliebiger Länge einen **Message Digest** fester Länge. Es ist quasi unmöglich aus diesem $h(m)$ die ursprüngliche Nachricht m zu berechnen.

Beispiele für solche Funktionen sind die unsichere MD5 oder SHA-2/3.

7.5 Certification authorities

Ein weiteres Problem sind MitM Angriffe, bei denen ein Angreifer falsche public keys in den Umlauf bringt. CA bindet den öffentlichen Schlüssel von Alice an ihren Namen. Dafür muss Alice einen Identitätsbeweis und ihren öffentlichen Schlüssel an CA liefern. CA erzeugt dann ein Zertifikat $\text{cert}(\text{PK}, \text{Alice}, \text{Gültigkeitszeitraum})$. Dieses Zertifikat ist von der CA digital signiert und somit von anderen überprüfbar.

7.6 SSL und TLS

Secure Socket Layer und **Transport Layer Security** ist ein weit verbreitetes security Protokoll. Es findet Anwendung im Browser und Webservern (https). SSL/TLS ermöglicht einige der Prinzipien der Netzwerksicherheit, Vertraulichkeit, Integrität und Authentizität. Ein Vorteil ist, dass es auf beliebige TCP Verbindungen angewendet werden kann.

7.6.1 TLS im Schichtenmodell

SSL/TLS stellt Anwendungen eine API zur Verfügung und ist also zwischen der Transport- und der Anwendungsschicht einzuordnen.

SSL/TLS Protokoll

Ist aus zwei Phasen aufgebaut, dem Handshake und dem Record Protokoll

- Handshake Protokoll
 - Vereinbarung der Kryptoalgorithmen
 - Server Authentisierung
 - Client Authentisierung
 - Vereinbarung von Sitzungsschlüsseln
- Record Protokoll
 - Datenübertragung
 - Verschlüsselung
 - Integritätssicherung
 - Session Verwaltung und Wiederaufnahme
 - Optional Komprimierung und Fragmentierung