Zusammenfassung Computernetzwerke und verteilte Systeme

TECHNISCHE UNIVERSITÄT DARMSTADT

~~Alex Praus,~~ Tobi Kratz

21. Februar 2021

Inhaltsverzeichnis

1 Quick Tour

1.1 Making two devices communicate

Verschiedene Möglichkeiten, um Kommunitkation herzustellen: direkte physische Verbindung. Hier werden Daten als Bits übertragen. Eine 1 könnte als als steigende Taktflanke und eine 0 als fallende dargestellt werden, es gibt jedoch viele Probleme bei der Implementation (00, 11, wechselnde Taktfrequenz...).

low level properties of communication Es gibt verschiedene Werte, die bei low-level communication wichtig sind:

- Delay (Latenz) d = distance / Propagation speed v. (v ist im vakuum = c, in Kupfer etwa $\frac{2}{3}$ c)
- Data rate r (Datenrate) = Data size/ data rate (Bsp. bits/secound).
  Wichtig ist hier, dass nicht die Geschwindigkeit gemeint ist, mit der die Daten Transportiert werden (siehe v), sondern in welcher rate die Bits auf die Leitung gelegt werden.

Wenn der Sender eine Daten sendet, werden diese nicht auf Sender Seite gespeichert, sonder lediglich beim Empfänger. Allerhöchstens sind die Daten während der Übertragung im Kabel gespeichert.
Beispeil Latexberechnung (delay & data rate): Wir senden 1250 Bytes ($10^4 b$) über 6 Meter mit einer Geschwindigkeit von 10 Mbps ($10^7 \frac{b}{s}$). Der Delay d beträgt dabei $d = \frac{6m}{c} = \frac{6m}{3 \cdot 1 - 8 \frac{m}{s}} = 2 \cdot 1 - ^{-8} s = 20ns$. Die Data rate beträgt $r = \frac{10^4 b}{10^7 \frac{b}{s}} = 10^{-3} s = 1ms$.

Types of physical communication Die Typen lassen sich in 3 Fälle aufteilen:

- Simplex: Eine Seite kann nur senden, die andere nur Empfangen (one-way). Empfänger lassen sich jedoch beliebig skalieren.
- Half Duplex: Beide Seiten wechseln sich ab mit senden um Empfangen (vgl. Telefonat/Gesrpäch).
- Duplex: Beide Seiten können senden wie und wann sie lustig sind (vgl. Streitgesrpäch).

Während Simplex und Half Duplex einfach realisierbar sind, ist (Full) Duplex eine technische Herausforderung.

Realizing Half Duplex Denkbar wären 2 Kabel, eins je Host, das wäre jedoch eine Verschwendung, da diese Kabel nie Zeitgleich genutzt werden würden. Es gibt zwei Ansätze: Time divison duplex (TDD) und on-dmand duplex. Beim TDD hat jeder Host eine feste Zeit T zum senden, und es wird zwangsläufig abgewechselt. Beim on-demand duplex gibt jeder Host die Länge der nächsten Bit Sequenz am Anfang direkt bekannt (pre-announce).

Realizing Full Duplex Hier wären 2 Kabel eher anwendbar, bedeuten aber den doppelten Aufwand. Auf kurzer Distanz kann jedoch auch Full Duplex mit einem Kabel realisiert werden, in dem Jeder Host eine leicht andere Frequenz benutzt (z.B. bei WiFi). Auch TDD ist umsetzbar. Während A sendet, speicher B die zu sendenden Daten. Nach T sendet B dann den Stack während A speichert. Das ist jedoch nicht Analog umsetzbar, da z.B. Sprachdaten sich nicht in Portionen aufteilen lassen.

1.2 Connecting many computers

Jeden Computer direkt mit jedem anderen zu verbinden wäre zwar möglich, skaliert jedoch eher so semi (https://www.reddit.com/r/cablegore/top
Switches wäre eine Lösung, jedoch laufen dann mehrere Connections über ein Kabel, es kommt also zu Einbußung in der Data Rate. Außerdem besteht das Problem, wie man eine Connection über einen switch herstellt (vgl. Vermittlung beim Telefon). Das führt jedoch dazu, dass eine Connection nur einfach genutzt werden kann, ist ein Host also mit einem anderen verbunden, kann ein Host nicht mehr erreicht werden.

Packet switching Statt also ein Circut für eine Verbindung zu blocken, teilt der switch die Daten in packete und sendet diese. So wird die Leitung nur für die Länge eines Packets geblockt und es kann schneller gewechselt werden.
Probleme: Anfang und Ende bestimmen? Wie bestimmt man wo das Packet hin soll?
Beispiel Ablauf: »store-and-forward«switching:

1. recieve a complete packet
2. store the packet in a Buffer
3. Find out the packet's destination
4. decide where the packet should be sent next (benötigt Kenntnis über Netzwerk Topologie)
5. forward the packet to his next hop of its journey

Multiplexing

> »Organziging the forwarding of packets over such a single, shared connection is called multiplexing.«

Auch beim Multiplexing ist TDM (Time Division Multiplexing) (nur ein Packet gleichzeitig) und FDM (Frequenz Division Multiplexing) (mehrere Packete auf gleichzeitig auf unterschiedlichen Frequenzen) möglich. Bei optischen Verbindungen WDM (wavelength division multiplexing) statt FDM. Es gibt jedoch noch weitere Formen, die hauptsächlich für Wireless transmission geeignet sind: CDM (Code Divison Multiplexing) und SDM (Space Division Multiplexing).
Multiplexing lässt sich auch abstrahieren, wenn zum Beispiel mehrere connections eines höhrer Layers eine lower-level connection nutzen sollen (s.c. upward multiplexing). Allgemein lässt sich in dem Kontext von shared Recources sprechen.

Forwarding and next hop selection Bekanntes Problem: Wie weiß eine Host/router/switch, wo er Packete hinschicken soll? Wie kennt er die besste/schnellste Verbindung? Es gibt einfache Ansätze:

- Flooding: alle Packete an alle Nachbarn senden
- Hot-potato routing: So schnell wie möglich an einen/mehrere zufällige Nachbarn senden

Sinnvoller wäre jedoch, wenn sich der router die besten wege merkt, e.g. durch Routing Tabellen. Diese können durch 2 Arten gesammelt werden: aktiv und passiv.
Passiv: Aktiv traffic beobachten und daraus Schlüsse ziehen.
Aktiv: Informationen aktiv senden und empfangen unter den verschiedenen Routern (routing protocols).
Jedoch bei Netzen der größe unsere Internets können Routing Tabellen schnell sehr groß werden. Hier kann man das Netz in mehrere Teilnetze splitten (devide et impera).

1.3 Organizing the mess - and connecting 'Alien' computers

Der Schlüssel ist »Simplification by abstraction«.
Z.B. das Modell des DS (distributed Systems) hilft beim verstehen von CN (computer networks). ein DS besteht aus AS (autonomous systems) und CSS (communication subsystems). Oft referiert wird auch auf die Abstraktion des OSI (Open Systems Interconnection) Schichten Modell.

1.3.1 OSI

Part 1: Concepts and Terms

> »A protocol defines the format and the order of messages exhanged between two or more communicating entities, as well as the actions taken on transmission and/or reception of a message or other event«

Analog zum 'Computer' Protokoll kann man sich ein menschliches Protokoll vorstellen, z.B. der definierte Ablauf beim Telefonieren (Hallo, Hallo..... Tschüß, Tschau). Weiter ist das OSI Modell in schichten aufgeteilt. Analog wäre z.B. Layer 2 ein Manager, der einen Brief in Auftrag gibt (Layer 1), der dann von der Post (Layer 0) transportiert wird. Ein layer N hat bietet also services für den layer n+1 an. Zwei layer n kommunizieren nur mit den für diesen layer vorgesehenden Protokolle, diese können jedoch auf services von layer n-1 zugreifen (der unterste layer kann natürlich nur eigene services nutzen).
Services im OSI Modell lassen sich in 2 Gruppen aufteilen:

- **connection-orientated services:** Diese haben meist 3 Phasen:
  - CON (Connection Establishment)
  - DAT (Data Exchange)
  - DIS (Disconnect)
- **connectionless services:** Bei diesen fallen CON und DIS weg und es werden nur Daten Ausgetauscht.

Im Osi Modell werden Nachrichten höhrer Layer als Daten Einheiten (Data Units) tierfer Layer transportiert. Hierfür gibt es einigen common Notations:

- packet: Ist die Einheit, die Transportiert wird (kann aus Fragmenten bestehen)
- datagram: wird bei connectionsless services als ersatz für Packete verwendet
- frame: 'fertig und verpackt' um versendet zu werden.
- cell: kleinere packet mit einer definierten größe
- PDU (Protocol Data Unit): eine (N)-PDU ist definiert durch: (N)-PCI+(N)-SDU
- PCI (Protocol Control Information): wird nur von peers genuzt.
- SDU (Service Data Unit): Ist die zu versendete payload eines höheren layers. (N)SDU=(N+1)-PCI+(N+1)-SDU

Part 2: 7 Layer Model

Layer 1: physical Layer (PH) Senden von Bits durch (de-)aktivieren von Signalen auf Leitungen.
Beispiel: 1000 Base-T

Layer 2: data link layer (D) Sendet Packete als Frames um Fehler zu erkennen und zu beheben, um Fehleranfällige Hosts zu schützen. Kann auch Flow Controll verwenden, um langsame Hosts zu schützen.
Beispiel: Ethernet

Layer 3: network layer (N) Ziel ist es den Packet-Stream zwischen zwei Hosts zu ermöglichen. Koordinierung der Pfade von Host zu Host. Konkret: Routing Wege finden und Packete weiterleiten und Fehler beheben, z.B. durch 'flow controll'. Up- und downward multiplexing ist möglich. Außerdem kann congestion control auf diesem layer angewendet werden.
Beispiel: IP

Layer 4: transport layer (T) Logische Verbindungen zwischen zwei Prozessen (nicht nur zwischen zwei Computern), Fehlerkorrektur und Paketzusammensetzung für den N-Layer.
Beispiel: TCP

Layer 5: session layer (S) Koordiniert Session z.B. bei HTTP mit dem Session-Cookies und hilft Nutzer und Anwendung bei der Konstruktion und dem spannen von aufeinanderfolgenden Verbindungen.
Beispiel: HTTPS

Layer 6: presentation layer (P) Sellt die Daten in eine unabhängige Form um, um unahängig davon die Übermittlung - gg. mit Kompression und Verschlüsslung - zu ermöglichen.
Beispiel: LDAP

Layer 7: application layer (A) Stellt Funktionen wie Daten Ein- und Ausgabe zur Verfügung.
Beispiel: XMPP

1.3.2 5 Layer of the Internet

Im Internet verschmelzen Layer 5,6,7 oft und oft sind die übergänge nicht klar definiert.

Layer 1 übermittlung von frames als stream von bits

Layer 2 Daten von layer 3 in Frames verpacken und and direkte Nachbarn weiterleiten

Layer 3 Daten vom client zum web-server weiterleiten, router2router communication, außerdem können e2e Verbindungen durch hop-to-hop realisiert werden

Layer 4 Verlässliche Verbindung zum Web-server herstellen und sicherstellen, dass die Daten auch in der richtigen Reihenfolge ankommen, jedoch keine congestion control.

Layer 5,6,7 HTTP Anfragen erstellen, Ebene 4 aufrufen (TCP).

1.3.3 network types

Wie schon erwähnt gibt es CO und CL networks. Beispiel für CO ist z.B. Das Telefon, CL die Post. CO haben durch die durch Handshakes vorallem bei kurzen Verbindungen eine Hohe 'extra Last' durch die zusätzlichen Daten des Handshakes, lassen sich jedoch besser skalieren, da sie nicht einen Status gebunden sind (stateless). CO sind jedoch durch den Status der connection Zuverlässiger. Bei einem 'verstopften' Network können mit CL immernoch Daten versendet werden, es kann nur sein, dass diese verspätet ankommen, CO haben jedoch schwierigkeiten. Es ist möglich CO auf CL aufzubauen.

connection-oriented Networks Im CO network ist der erste Schritt mit einem handshake eine connection herzustellen. Nach dem handshake wissen beide Seiten von der connection und der Datenaustausch kann stattfinden. Die connection ist dabei nur ein loser status, auf dem Basis jedoch andere Eigentschaften (flow control, congestion control...) aufgebaut werden können.
Bei CO networks implementiert nicht direkt andere Eigentschaften der Verbindung. Dinge wie reliability, flow control und congestion control sind für CO networks sind nicht notwendig. Diese können z.B. mit TCP ermöglicht werden.

connectionless networks keine handshakes, direkt Faken (Daten Austausch). Wenn der Datenaustausch vorbei ist, kommt auch kein DIS mehr. CL networks brauchen wenig Aufwand, da keine connection gepflegt werden muss, es kann jedoch sein, dass der reciever nicht bereit zum Empfangen ist. CL implementieren keine reliability, flow control und congestion control.

2 Routing

2.1 introduction

Im Routing werden dafür gesort, dass Packete vom Empfänger an das richtige Ziehl kommen. Bei Direkt Verbindungen besteht das Problem nicht, jedoch ist das bei großen Netzwerk keine Option, wenn jeder Host mit jedem anderem Verbunden werden muss. Wenn Switches genutzt werden, muss diesen jedoch gesagt werden, wie das Netzwerk aufgebaut ist (Netzwerk Topologie). In diesem Kapitel geht es um das Problem, wie ein Host den besten Weg zu einem Ziehl findet.

Building a large network Bei großen Netzwerken ist flooding und Hot-potato routing keine Option, da mit jedem Host die Anzahl an Packeten steigt und so mit den beiden Routing uneffizienten Methoden das Netzwerk schnell an sein Limit kommt. Ziel ist es eine Effiziente Methode zu finden, die Packete möglichst schnell ans Ziel bringt, ohne dabei unnötig viel Traffci erzeugt.
Im folgenden werden zwei Begriffe genutzt:

- **Routing:** determine route taken by packets from source to destination. (Basis: Routing algorithms).
- **Forwarding:** move packets from router's input to appropriate router output.

2.1.1 Forwarding

Wenn Packete von einem Netzwerk in ein anderes Netzwerk geileitet werden sollen, wird ein router eingesetzt. (Heute haben uns bekannte Router mehrere Aufgaben, die früher von verschiedenen Geräten übernommen wurden: hub, bridge, switch, gateway.) Wenn also Packete von einem Netzwerk in ein anderes gesendet werden sollen, übernimmt der Router die Koordination und leitet das Packet (forwarded) in das entsprechende Ziel Netzwerk. Hängt das andere Netzwerk direkt am selben Router, handelt sich sich um ein single hop. Wenn mindestens 2 Router zwischen den Netzwerken sind, handelt es sich um ein Multi-Hop.

2.1.2 Routing

Routing findet für gewöhnlich auf Layer 3 statt, dort ist das Ziel Packete von Host A zu Host B möglichst effizient zu transportieren bzw. erstmal einen Pfad zu finden, auf dem das möglich ist. Dies wird i.d.R. von Routing Algorithmen durchgeführt. Das Internet besteht aus mehreren AS, die alle wieder aus Teilnetzen bestehen. Jedes AS führt dabei selbst routing Algorithmen aus, um die besten Wege zu finden.

- CONS (CO+NS) Nn CO Networks Routing Algorithmen werden meist in der CON Phase durchgeführt. Im COTS (CO+TS) wissen nur die Endsysteme, dass sie verbunden sind, in CONS hingegen wissen alle Systeme auf der Route, dass die Systeme verbunden sind.
- CLNS (CL+NS) IN CL Networks wird nicht bei jedem Packet der Algorithmus durchgeführt, das würde einen zu großen Overload bedeuten. Manchmal wird beim ersten Packet einer Verbidnung der Algorithmus durchgeführt, das ist allerdings für sich schnell änderte Netzwerke keine Option. Im Inetnet z.B. dies in regelmäßigen Abständen, oder wenn sich große Teile ändern.

optimizing Routing Algorithmen Routing Algorithmen haben oft unterschiedliche Kriterien, nach denen sie arbeiten:

- Average packet delay
- Total throughput
- individual delay (kann jedoch mit anderen Kriterien im Widerspruch stehen)

Am meisten jedoch wird nach dem Kriterium, des minimal-hop-count gearbeitet, da dieser oft einen Kompromiss aus allen Kriterien bedeutet, es gibt jedoch keine Garantie dafür.

2.2 Routing Algorithms

Routing Algorithmen werden meist in zwei Arten aufgeteilt:

- **Non-adaptive Routing Algorithms**
  Diese agieren unabhängig vom State des Netzwerks. Beispiele sind flooding oder preconfiguration.

- **Adaptive Routing Algorithms**
  Nehmen den aktuellen Status des Netzwerks mit in Betracht, wenn sie Routing Entscheidung treffen. Beispiel Hierfür wären distance-vector-routing oder link state routing. Das Problem hierbei ist, dass bei sich ändernden Neztwerken die Routen häufig neu entschieden werden. Algorithmen diesen Types sind trotzdem sinnvoll in eignen, fest bekannten Netzen. Bekommt ein Link z.B. so viel Traffic, das Packete verloren gehen, wird der Link als Broken makiert und es kommt zu noch größeren Ausfällen.
  Es gibt dort auch 3 Unterarten:
  - Centralized adaptive routing
  - Isolated (aka. local) adaptive routing
  - Distributed adaptive routing

2.2.1 Examples

Flooding (non-adaptive) Hier wird jedes einkommende Packet an alle bekannten Nachbarn weiterleitet. Das Problem dabei ist, dass so Nezte schnell überlastet werden. Gibt es zum Beispiel schleifen, kann es schnell zu einer Flut an nicht aufhörenden Packeten kommen. Eine Lösung für dieses Problem wäre z.B. das implementieren von TTL (Time to live) oder Sequence Number. TTL werden meist in Hops angegeben und werden bei jedem Hop um 1 dekrementiert. Hat ein Packet ein TTL von 0, wird es weggeworfen. Eine Sequence Number wird beim ersten Router initialisiert. Jeder Router führt eine Tabelle mit Sequence Numbers, die er schon einmal geroutet hat. Kommt ein Packet mit einer Sequence Number, die er schon kennt, wird dieses Packet weggeworfen.
Flooding macht jedoch durchaus Sinn in sich schnell ändernden Netzen, z.B. bei WLAN oder Mobilfunk oder wenn alle Packete Multicast sind und so wie so mehrere Ziele haben.

Static Routes (non-adaptive)Static Routes sind großartig für statische, vorhersehbare Umgebungen. Das Problem ist, dass sich das Internet regelmäßig ändert und statische routen dann viel Wartungsaufwand bedeuten.

Centralized Adaptive Routing (adaptive) Es gibt einen Zentralen Control Center (RCC), der regelmäßig Informationen über die Topologie von allen Routern bekommt und dann einen Idealen Routing Graph erzeugt (z.B. Dijkstra). Das Problem hierbei ist, dass das Netz zusammenbricht, wenn nur der RCC ausfällt. Außerdem werden Routen 'in der Nähe' des RCC bevorzugt, was dort zu einer hohen Last führt während 'abgelegene' Router meist wenig Aufgaben haben. Ebenso bekommen Router die näher am RCC sind schneller die neuen Routing Informationen, was zu unterschiedlichen States führen kann.

Isolated (aka. local) adaptive Routing (non-adaptive) Es werden Entscheidungen über Routen nur lokal getroffen. Beispiele sind Hot potato Routing und Backward learning.

- Hot Potato Routing
  Idee ist es, die Packete so schnell wie möglich loszuwerden, wobei nicht beachtet werden muss, zu welchen Host die Packete geschickt werden. Dieser Algorithmus ist nicht sehr effektiv, es gibt jedoch einige use-Cases in denen diese Art noch genutzt wird (peering/discovering).

- Backward Learning Routing
  Bei diesem Algorithmus werden im Packet Header Source Addresse und Hop Counter hinzugefügt, Router lernen also im laufenden Betrief über die Topologie und passen die Routen im Betrieb an. Jedoch müssen in jungen Netzwerken andere Algorithmen genuztzt werden (z.B. hot potato / flooding). Wenn der Hop-Count == 1 ist, kommt das Packet von einem direkten Nachbarn. Bei einem Hop-Count n>1 ist die source n hops away.

Distributed Adaptive Routing Durch Graph Abstraction die besten Routen finden. Knoten sind dabei Router und Kanten die physikalischen Links a.k.a. hops. Die Kosten eines links sind dabei z.B. delay, $, oder der congestion Layer. Die Kosten eines Pfades sind dann alle link Kosten vereint. Ein guter Pfad wird meist als der, mit den geringsten Kosten bezeichnet, es kann aber auch nach anderen Kriterien gesucht werden (e.g. min-hop-count).
Algorithmen hier lassen sich weiter klassifizieren:

- **Decentralized** Jeder Router kennt die Kosten zu seinen Nachbarn. Auch Distance Vector Routing gehört hierzu (z.B. BGP oder RIP)

- **Global** Alle Router kennen die komplette Topologie und alle link kosten. Hierzu gehören Link state Algorithmen z.B. Dijkstras oder OSPF.

- **Static** (nicht adaptiv) Routen ändern sich sehr selten

- **Dynamic** (adaptiv) Routen können sich oft ändern, Hier werden also regelmäßig updates in den Routen gemacht.

2.3 Distance Vecotr Routing

Beim Distance Vector Routing tauschen direkte Nachbarn Informationen über Routen mit ihren Nachbarn aus. Jeder Host pflegt eine Tabelle, in der jede mögliche Ziehladdresse eine Reihe und jeder Nachbar eine Spalte hat. In der Tabelle werden dann die "Kosten"der Route eingetragen und mit jeder Iteration verbessert. Konkret schreibt man dann für Route von X to Y via Z als nächsten Hop:

$$D^X(Y,Z) = c(X,Z) + min_w\{D^Z(Y,w)\}$$

Mit einem Routing Algorithmus wird dann eine "Distance Table/Matrix"gebaut, mit der dann Routing Tablelen aufgestellt werden, aus denen dann der Distance Vector an die Nachbarn announced werden kann. DVR hat jedoch einige Probleme (count to infinity), jedoch handelt es sich um einen sehr simplen Algorithms.
DVR Protokolle sind iterativ und Distributed:

- **Iterativ** Das heißt sie laufen nicht unendlich, sondern stoppen sobald keine weiteren Verbesserungen möglich sind. Außerdem sind sie ßelf-terminating"d.h. es gibt kein Stop Signal o.ä. Eine Iteration wird dabei ausgelöst indem entweder ein loakler link sich ändern z.B. in den Kosten oder wenn es eine Nachricht eines Nachbarn gibt, dass der Link dort sich geändert hat.
- **Distributed** Des weiteren tauschen sie Informationen nur mit direkten Nachbarn aus und kennen auch nur den State dieser. Eine Node informiert einen Nachbarn dann über neue Routen, wenn sich die Kosten zu einer Destination verringert haben.

2.3.1 Count to Infinity Problem

Gegebene Situation: Wir haben 3 Host A,B,C. A ist mit B mit einem Cost von 1 verbunden, und B ist mit C mit einem Cost von 2 verbunden. Daraus ergeben sich folgende 3 Routing Tabellen: Durch einen Ausfall verschwindet jetzt die Verbindung zwischen B und

A			B			C		
TO	COST	VIA	TO	COST	VIA	TO	COST	VIA
B	1	B	A	1	B	A	3	B
C	3	B	C	2	C	B	2	B

C. A announced an B jedoch, dass es eine Route zu C mit dem Cost von 3 gibt. B versucht nun also C via A zu erreichen: Nachdem B

A			B			C		
TO	COST	VIA	TO	COST	VIA	TO	COST	VIA
B	1	B	A	1	B	A	-	-
C	3	B	C	4	A	B	-	-

dann diese Information an A sendet, aktualisiert A dann seine Route zu C, da diese über B geht und sich die Kosten erhöht haben. die Route von A nach C via B ist dann wie gewohnt die Route von B nach C + die Kosten von A nach B. A announced das dann wieder an B, der ja nach C über A routet. Er addiert darauf also die Kosten von B nach A. Das läuft dann ungebremst so weiter, bis ins unendliche,
Möglichkeiten dieses Problem zu lösen:

Poisend Reverse Methode Wenn die Route von A nach C über B geht, sagt A dem Host B, dass seine Kosten nach C unendlich sind. In einem kleinen Netzwerk wird dann innerhalb weniger Iterationen ein stabiler State erreicht, in größeren Netzwerken besteht das Problem jedoch immernoch, z.B. in einem Netzwerk in dem A,B,C jeweils direkt verbunden sind, und C dann noch eine Verbindung zu D hat. Alle Kosten sind gleich. Fällt dann die Verbindung CD aus, bekommt A immernoch Falsche Routen zu D von B und umgekehrt.

Split Horizon Wenn Host B seine Routen updatet und das an A sendet und A daraufhin einige Änderungen übernimmt, sendet A diese Änderungen nicht wieder an B, sondern nur an seine anderen Nachbarn.

2.4 Link-State Routing

Beim Link State Routing sammelt für gewöhnlich ein Zentraler Knoten (RCC) Informationen und gibt diese dann an alle anderen Router im Netzwerk weiter. Die Netzwerktopologie ist somit dann allen Router im Netzwerk bekannt. Der RCC baut aus allen gesammelten Informationen einen Grapgen (V,E), wobei V ein set an vertices (nodes) ist und E für die Edges (links) steht. c(v,w) sind dann die Kosten der Kanten. Wenn eine Kante nicht in E ist, ist c unendlich. Das Ziel ist es dann den günstigsten Pfad von node s (source) zu node v zu finden. Hierfür wird meistens Dijkstras verwendet. Jeder Router versendet regelmäßig per flooding ”Link state packages” mit Informationen, die er zu seinen Nachbarn gesammelt hat (delay, hop count...) und verseht diese mit einer sequence Number und einem ”age flag”. Wenn ein Router so ein Packet bekommt, das er jedoch schon kennt (sequence Number) oder es abgelaufen ist, wirft er es weg.

Vergelich LSR und DVR Link State Routing und Distance Vecotr Routing im direkten Vergleich:

	LSR	DVR
Message complexity	mit n Knoten und E Kanten werden jedes mal $O(n \cdot E)$ nachrichten versendet	Austausch findet nur zwischen nur zwischen Nachbarn statt
Speed of Covergence	ein $O(n^2)$ Alogorithmus braucht $O(n \cdot E)$ Nachrichten	Variiert stark. Es kann zu Routing Schleifen kommen. Count-to-infinity Problem
Robustness	Es können verfälschte Link Kosten announced werden. Jede Router nutzt nur die eigenen Tabellen.	Es können verfälschte Path Kosten announced werden. Die eigenen Routen werden von anderen Routern genutzt.

Algorithmen wie LSR und DVR sind für Netze konzipiert, die sich selten ändern und physikalisch verbunden sind. Sie haben vorallem Schwächen bei Mobilen Netzen, bei z.B. folgenden Punkten:

- **High dynamics** z.B. durch stendig wechselnde Nachbarn und Links
- **Power conservation** regelmäßiges senden von Routing Packeten verbraucht Strom und Leistung
- **Low bandwidth links** wenn z.B. Routing Informationen nicht versendet werden können
- **Asymmetry** Links können in der Geschwindigkeit variieren
- **Interference** Störsignale
- **High redundancy** ein Gerät ist mit vielen anderen Verbunden / "meshed"

2.5 Hierarchical Routing

In der Realität sind große Netze nicht so ideal wie bisher beschrieben, sondern sind meist nicht flach wie wir sie beschrieben haben und Router unterscheiden sich oft fundamental. Im Internet heute gibt es über 1 Milliarde links, die alle zu erreichen würde Routing Tabellen explodieren lassen und der Austausch dieser Tabellen würde jeden Link sprengen. Deswegen ist das Internet in Teilnetze aufgeteilt: Autonomous Systems.

Autonomous Systems Jedes AS hat eine eigene Nummer (z.B. AS421220) und kennt eine Route zu jedem anderen AS. Es gibt im heutigen Internet etwa 60000 solcher Teilnetze und alle sind unterschiedlich groß. Verschiedene AS sind mit pysikalischen Links verbunden (peers), über die Daten ausgetauscht werden. Jeder Router innerhalb eines AS muss also nur die Routen zu anderen Routern im AS kennen, und die Route zu seinem Gateway. Innerhalb eines AS (inra-AS) hat der Administrator freie Wahl für die Nutzung von Routing Algorithmen, es kann z.B. RIP, OSPF oder IGRP verwendet werden. Für die Kommunikation ziwschen AS (Inter-AS) gibt es jedoch einen festen Standart: BGP (Border Gateway Protocoll).
Für die inter-AS Kommunikation wird ein Gateway Router benötigt, der den Transfer von Daten zu anderen AS koordiniert. Dieser Router pfelgt Routing Tabellen mit anderen Gateayrouter. Der Vorteil dieses Modells ist, dass es für das Reale Internet besser skaliert. Durch die reduzierung von Peers, die in Routing Tabellen gepflegt werden müssen kommt es zu selteren updates der Tabellen, was es leichter macht, schnelle Routen zu finden. Bei Inter-AS Routing gibt es jedoch Regulierungen, welcher Host über wen peeren darf, innerhalb eines AS wird es sowas nicht geben, da ein AS nur eine begrenzte Anzahl Admins hat. Inter-AS Communication kann deswegen eher Performance orientiert sein.

2.5.1 About BGB and Internet Routing

Zum Vegleich die Routing Protokolle RIP & OSPF:

- **RIP**
  - DVR
  - Erstmal 1983 aufgetaucht. RFC 1058 von 1988
  - Am minimalen Hop-Count orientiert
  - Poison Reverse
- **OSPF**
  - LSR
  - RFC 1131 (inzwischen Version 2 & 3)
  - am meisten verwendete IGP
  - verwendet TCP zum Übertragen von Routing Informationen
  - Multicast support

Border Gateway Protocoll BGP ist der heutige Standart für EGP. Durch regelmäßige "hello"Packete erfahren andere Netzteilnehmer von der existenz des AS. BGP peers bauen dann eine Session auf und tauschen via TCP Routing Informationen aus. Wenn AS1 z.B. AS2 sagt, dass es einen gewissen prefix routen kann, garantiert AS1 AS2 dann allen Traffic weiterzuleiten. BGP kann auch Innerhalb eines AS verwendet werden, man spricht dann von iBGP. Um den Unterschied dann zu inter-AS Communication zu spezifizieren, wird in dem Kontext dann von eBGP gesprochen.
Eine advertiste Route in BGP besteht immer aus einem Prefix und einem Attribut set. Die zwei wichtigsten Attribute sind

- AS-PATH: Beinhaltet den Pfad, der für die Route genutzt wird (z.B. AS5 AS8 AS10)
- NEXT-HOP: Der nächste AS Router für den nächsten HOP

Über die Jahre wurden immer mehr AS registriert und die Zahl steigt weiter (>6000). Das führt dazu, dass es wieder mehr Host gibt, für die alle Routing Einträge gepflegt werden müssen, was die Länge der Tabellen explodieren lässt. Durch die Hohe AS Zahl, kommt es auch regelmäßiger zu Änderungen im System, was Änderungen der Routing Tabellen beudeutet. Große Netze haben zudem eine höhere Fehleranfälligkeit.

BGP Security In BGP stellt jeder AS selbst ein, welches Subnetz er verwaltet. Kommt es bei dem einstellen zu Fehlern ein AS announced ein Subnetz, welches er nicht wirklich organisiert, kommt es zu falschen Routen. So geschehen z.B. 24.02.2008 der Pakistan Telekom (AS17557). Eine generelle Einschätzung zu Routing Security siehe RFC4593. Mit BGP können aber auch Security Operations implementiert werden, z.B. durch setzen einer IP TTL von 255 und es werden nur Routing Informationen mit einer TTL >= 254 verarbeitet. Außerdem können BGP Session mit MD5 Signaturen versehen werden.

2.6 Mobile Routing

Mobile Networking ist gut geeignet, um ein schnelles Netzwerk an Orten ohne gut Infrastruktur aufzubauen. Anders als im Internet sind beim Mobile Routen ganz andere Probleme zu bewältigen. Da zwischen Host keine physikalische Verbidnung besteht ist, wird Routing zwischen Host schwierig. Doch durch sich wechselnde Positionen o.ä. verändern sich links zwischen Host sehr schnell. Das und andere sind Faktoren, mit denen DVR und LSR nicht umgehen können, diese sind für statische Netze konzipiert. Routing Algorithmen können in zwei Kategorien aufgeteilt werden:

- **Proaktiv**
  - Routing Informationen werden unabhängig vom aktuellen Traffic regelmäßig und undabhängig generiert
  - Alle Ïnternet Routing Algorithmenßind Proaktiv, auch die oben kennengelernten Beispiele
  -
  - Beispiel für ein Reactive Mobile Routing Algorithmus: DSDV
- **Reactive**
  - Routen werden erst dann berechnet, wenn Daten übertragen werden sollen
  - Bei CO wird eine Route (wenn noch keine Vorhanden ist) beim connection Setup berechnet
  - Bei CL beim ersten Packet
  - Beispiel für ein Reactive Mobile Routing Algorithmus: DSR

Hierarchical Mobile Routing In Mobilen Netzen können Ideen des Mobile Routings verwendet werden. Teilnetze können in Cluster aufgeteilt werden. Innerhalb eines Clusters kann proaktives Routing implementiert werden, für die Kommunikation zwischen Clustern können reaktive Algorithmen verwendet werden. Bei kleinen Clustern können alle Nodes sich gegenseitig kennen, es ist also ideal für reaktive Routing. Zwischen Clustern findet seltener Kommunikation statt, und es können mit vielen kleinen Clustern hier am besten reaktive Algorithmen verwendet werden.

2.6.1 DSDV

DSDV ist eine Erweiterung des DVR spezialisiert für Mobile Netze. Es gibt 2 große Extensions:

1. Sequence Numbers for all Routing updates
   - Verbesserungen gegen Schleifen und Inkonsistenzen
2. Decrease update frequency
   - Zeit zwischen ersten und bestem announcemendd eines Pfades wird gespeichert

Trotz dessen ist DSDV noch ein Proaktiver Algorithmus und ähnelt sehr stark dem DVR.

2.6.2 DSR

DSR (RFC 4728) geht einen Schritt weiter Richtung Reaktivem Routing. Er baut auf zwei simplen Ideen auf:

1. Routing wird ind "Path discoveryünd "path maintenanceäufgeteilt
2. regelmäßiges Updaten wird vermieden

Er ist für statische und dynamische Netze geeignet und kann mit bis zu 200 Knoten arbeiten. SSource Routing"bedeutet dabei, dass der Sender für die Bestimmung des Pfades verantwortlich ist.

Path Discovery Wenn ein Sender ein Packet versenden will, jedoch noch keine Route für das Ziel hat, startet er die Path Discovery. Es wird dabei ein Packet per flooding und boradcast mit der Ziel Addresse und einer einmaligen ID versendet. Wenn ein Host ein solches Packet erhält und er ist das Ziel, sendet er das Packet mit dem Pfad über das es gekommen ist zurück. Das erste Packet dass diesen Host erreicht kam über dem schnellsten Pfad, jedes weitere Packet kann dann verworfen werden. Es sind noch weitere Optimierungen möglich: Wenn die Topologie (bzw. die maximale länge) des Netzes bekannt ist, kann statt einer ID eine conter/TTL hinzugefügt werden. Der Host kann dann am Counter sehen, wie viele Hops das Packet hinter sich hat und es wegwerfen, wenn der Counter größer als der maximale Diameter ist. Eine zweite Verbesserung wäre, wenn Hosts discovery Packete speichern, die sie weiterleiten sollen. Die Informationen dieser Packete können dann für die Suche einer route genutzt werden, wenn das Gerät selbst Packete versenden will. Ist ein Path gefunden, muss jedoch sichergestellt werden, dass dieser auch über die Länge der Verbindung offen bleibt.

Path Maintenance Nicht genutzte Paths werden nach einer Zeit aus der Routing Tabelle gelöscht. Es ist möglichkeit, ist z.B. als Sender auf eine Bestätigung des Empfängers auf layer 2 zu warten oder so eine explizit zu erfragen. Auch kann eine Station schauen, ob Nachbarn das Packet weiterleiten, wenn diese Technologie unterstützt ist. Falls es ein Problem gibt, können Pfade neu gesucht werden oder es kann versucht werden, den Empfänger zu erreichen und ihm mitzuteilen, dass es ein Problem gab.

2.7 Overlay Routing

Im Overlay Routing sitzt ein Virtuelles Neztwerk auf einem existierenden. Hier können die gleichen Algorithmen verwendet werden. Wird ein Packet in einem Overlay network an den Nachbarn versendet, wird das Packet über das darunter liegende Netz transportiert und kann dort auch über mehrere Hops geleitet werden, während Sender um Empfänger im Overlay Network denken, sie sind dierekte Nachbarn.

3 IP & Internetworking

Das Intenet ist anders implementiert als das ideale Netzwerk. Statt das OSI Modell wird das Internet mit dem TCP/IP Modell in 4 Schichten aufgeteilt:

- **http, ftp,...** Application Layer (OSI 5-7)
- **TCP/UDP** Transport Layer (OSI 4)
- **IP** Intenet Layer (OSI 3)
- **Physical** Network Interface (OSI 1-2)

Layer 1 bis 2 (Physical & IP) sind Hop by Hop oritentiert, während 2 bis 4 (IP, TCP/UDP & http,ftp) e2e oritentiert sind. Im weiteren Verlauf wird es konkret um Layer 2 IP (Network Layer, OSI 3) gehen. Beispeiel für Protokolle dieses Layers sind IGMP, ICMP, ARP, IP, RARP.
Sendet ein Host A ein Datenpacket via IP an Host B, versieht A dieses Packet mit einer einer Destionation Addresse. Dieses Packet wird dann von Routern weitergeleitet, bis es an ziehl kommt. Ein Beispiel um Routing mit IP zu erklären:
Host A (192.0.0.1) sendet ein Packet an 192.0.1.1. Host A hat jedoch keine direkte Verbindung sondern ist mit einem Router verbunden, der die Addresse 192.0.0.1 hat. In einer Routing Tabelle dieses Routers gibt es z.B. folgende Einträg: 192.0.0.0/24 über Link Interface 1 und 192.0.1.0/24 über Link Interface 2. Bekommt der Router also ein Packet mit der Destionation Addresse 192.0.1.1, so ist diese in der zweiten Range drin, und er sendet das Packet über Link 2 weiter.
Die Information über die Destionation IP steht im IP Header.

Tabelle 1: IP Header, length: <– 32 bits –>

VER[1]	IHL[2]	type of service	lenght[3]		
16-bit identifier[4]			flgs[4]	fragment offset[4]	
TTL [5]		protocol[6]	Internet Checksum[7]		
Source IP (32b format)					
Destination IP (32b format)					
Options (if any)					
Data (variable length,					
typically a TCP or UDP segment)					

3.1 IP Header

Der Type of Service referiert auf priority Informationen, dieser Teil des Headers wird eher selten benutzt. Die Total Length ist die Smme inklusive Header und TCP/UDP Segment. Der identifier kann genutzt werden, wenn Packete in mehrere Fragemente gespalten wurden, diese Packete wieder zusammenzusetzen. Wird ein Packet "fragmentiert"bekommen alle bis auf das letzte Fragment den MF (more Fragments) Flag. Ist ein Packet nicht fragmentiert, bekommt es das DF (Don't Fragment) Flag. Im Fragment Offset wird die Position des Fragments typischerweise in 8er Oktets angegeben.

3.2 Addressing

In den bisher kennengelernten Routing Algorithmen speicher der Router Addressen in einer Tablle, mit dem entsprechenden Pfad, den er dorthin routet. In großen Netzen kann dann so eine Tabelle sehr schnell sehr groß werden. Außerdem ist es nicht unüblich, dass eine Gerät mehrere (MAC) Addressen hat, da eine MAC-Addresse nicht Host spezifisich ist, sondern Interface speziefich.

	Biespiel	Verteilung
MAC Addresse	70:12:a5:42:93:10	Flach und Permanent (Hersteller)
IP Addresse	172.20.42.10	Topologisch (meistens)
Hostname	tu-darmstadt.de	hierarchich

3.2.1 Naming and DNS

Bei einem Flat namespacing bestehen Namen nur aus einfachen Strings, die von einer Zentralen Stelle verwaltet werden. Hierbei kann es schnell zu doppelungen kommen, da in einer langen Liste schenll der Überblick verloren werden kann.
Beim Hieracrchical Name Space hingegen wird die Vergabe von Namen dezentralisiert. Es gibt die ersten Namen TLD (Top Level Domain), z.B. de, die dann die Unteraddresse tu-darmstadt an einen anderen Host deligiert. Das kann rekursiev weiterlaufen, tu-darmstadt kann dann z.B. Informatik (informatik.tu-darmstadt.de) an einen weiteren Host deligieren, ohne de darüber zu Informieren. Wie beim Routing auch zeigt sich, dass flache Strukturen nicht funktionieren, in der Praxis wird deswegen beim namespacing auf das hierarchiche Modell gesetzt.

3.2.2 IPv4 Addresses

IPv4 Addressen sind in Blocks aufgeteilt, die für verschiedene Zwecke gedacht sind.

- CLASS A: Organisation <= 16 Millionen Hosts. First Bit 0, first 8 for Net, last 24 Bit Host.
- CLASS B: Organisation <= 65 Tausend Hosts. First Bits 10, first 16 for Net, last 16 Bit Host.
- CLASS C: Organisation <= 255 Hosts. First Bits 110, first 24 for Net, last 8 Bit Host.
- CLASS D: Multicast Addressen. First Bits 1110, last 28 Bits for Multicast Addresses

[1] IP protocol Verison Nummer (z.B. 4 oder 6)
[2] Header länge 32b Word
[3] Totale Länge in Bytes
[4] für fragmentation/Wiederherstellen
[5] Max Hop Anzahl. Wird bei jedem Hop um eins dekrementiert
[6] Protokoll des höheren Layers, dem das Packet zugestellt werden soll
[7] Checksum des Headers

- CLASS E: Reserviert (Privat, Dokumentation...) Firsts Bits 1111, last 28 reserved.

Eine Netzwerkaddresse wird angegeben durch die festen Werte (First Bits & Net) und die Host bits, wobei die alle 0 sind. Die letzte Addresse (Host bits sind alle 1) ist für Broadcast reserviert. 127.0.0.0/8 sind für loopback Anwendungen reserviert, Packete an diese Addresse werden am lokalen Host bearbeitet. Das /8 eben nennt man eine Netzmaske. Diese gibt an, wie groß das Teilnetz ist, also bei einem CLASS A z.B. /8, weil die ersten 8 Bits fest sind, CLASS B /16 und so weiter. Alternativ schreibt man auch: 11111111.00000000.00000000.00000000 (255.0.0.0, /8).
Einem Host Interface wird in der Regel ein /32, also eine genaue IP Addresse zugewiesen. Ein Host kann mehrere Interfaces, also auch mehrere Addressen haben. Einem Teilnetz wird dann ein Subnetz zugewiesen, in dem sich alle Hosts dieses Netzes befinden. Für lokale Netze werden in der Regel /24 verwendet. Ein Router, der mehrere Netze verwaltet, mehr sich dann nicht jede einzelende Addresse, sondern nur die Teilnetze je Interface. Ein Beispielrouter verwalter 3 Netze auf je einem Interface. Netz 1 läuft über Interface 1 mit 192.0.1.0/24, Netz 2 Interface 2 mit 192.0.2.0/24 und Netz 3 Interface 3 192.0.3.0/24.

Subnetting Ein Größerer IP Bereich kann auch in mehrere Subnetze geteilt werden, so kann von einem /16 z.B. ein /24 abgespalten werden. von den 16 Hosts bits werden die ersten 8 dann zu einer Subnet ID, die das Subnetz identifiziert. Die Netzmaske wird dann für Hosts innerhalb dieses Subnetzes ebenfalls angepasst. Aus einem Netz mit 256*256-2 (Broadcast & Network Address) Addressen kann so ein Netz mit 254 Subnetzen je 254 Hosts gebaut werden. Ein Subnetz ist jedoch nicht darauf beschränkt ein /24 zu sein, es können Variable Längen für Subnetze verwendet werden zwischen /17 und /32. In diesem Kontext spricht man von Variable Length Subnet Mask (VLSM). Damit lassen sich Wartungsarbeiten für ISPs z.B. verringern. ein ISP announced z.B., dass er sich um ein /20 kümmert, und leitet dann teile davon an unterschiedliche Organisationen weiter (z.B. /23 Subnets). Das erlaubt es, IP Addressen mehr effizient zu nutzen. Wenn eine Organisation z.B. 2k Addressen braucht, rein ein CLASS C Nezt nicht mehr aus. Es wird ein Class B Netz zugewiesen, auch wenn dann 63k Addressen ungenutzt bleiben. VLSM erlauben es, Addressen mehr effizient zu nutzen. So kann der Organisation z.B. aus einem CLASS B Netz eines ISPs ein /21 zugewiesen werden, dass dann $2^{32-21} = 2048$ Addressen ermöglicht zugewiesen werden.

Advertising Doch wie kommt ein Host am Ende an seine IP Addresse? Koordinierung durch Menschen kann sehr umständlich werden und schnell aus dem Ruder laufen. Es kann ein DHCP eingesetzt werden, der ein zugeteiltes Subnetz an verbundene Geräte verteilt. Ein solcher DHCP Server wird meistens auf dem default Gateway betrieben, und weißt IP Addressen anhand der MAC Addresse eines Interfaces zu.

3.2.3 NAT

In der Summe gibt es $2^{32} = 4$ Milliarden IPv4 Addressen. Was zu den 70er noch unfassbar viel war, wird heute knapp. IPv4 Addressen gehen heute für über 20$ das Stück über den Tisch, bei großen zusammenhängenden Netzen teils noch mehr. Eine Lösung wäre IPv6 (2^{128} Addressen), dafür sind einige Menschen aber zu faul. Eine[1] Lösung für das Problem ist NAT.
Bei einem NAT hat ein Gateway eine öffentliche Addresse einem Gateway zugewiesen. Hinter diesem Gateway können sich mehrere Hosts befinden, die mit Addressen aus einem local Network Bereich versehen werden. Packete, die aus dem lokalen Netzwerk versendet werden, bekommen dann die Source Addresse des Gateways. Für Hosts außerhalb des Netzes sieht es dann so aus, als käme der Gesamte Traffic nur von einem Host (Gateway).
NAT hat jedoch den Vorteil, dass nicht direkt auf Geräte innerhlab eines lokalen Netzes zugegriffen werden kann, außerdem kann sich die öffentliche IP Addresse ändern (z.B. durch wechseln des ISP), ohne dass das lokale Netzwerk neu konfiguriert werden muss. Jedoch greift der Router durch die Manipulation in des IP Headers in den Traffic ein. Ein Router sollte jedoch maximal 3 Layer bearbeiten. Außerdem funktionieren e2e oder p2p Anwendungen nicht mehr richtig, wenn ein Host innerhlab eines NATs ist. Außerdem ist jeder Host auf maximal 65k connections beschränkt (max anzahl Ports). Bei einem NAT sind alle Geräte jedoch auf insgesamt 65k Ports beschränkt, da das gateway nur soviel Ports hat.

NAT Translation Wenn ein Computer innerhalb eines lokalen Netzes hinter einem NAT Packete aus dem Netz verschicken will, versieht er jedoch den IP Header mit seiner lokalen Source Addresse. Damit z.B. ein Antwort Packet am Router wieder ankommt muss dieser jedoch seine öffentliche IP als Source IP in den Header eintragen. Außerdem wird der Port geändert, damit die connection am Router mit dem Port des Hosts aufrecht erhalten werden kann. Ankommende Packete werden dann wieder vom Router statt mit der NAT-Adresse mit der lokalen Addresse des Hosts versehen und innerhalb des Netzes geroutet.

3.2.4 ARP

Wie können Host innerhlab eines lokalen Netzes wissen, welche MAC Addresse hinter einer IP steht? ARP (Address Resolution Protocol) Anfragen! Bei einer ARP Anfrage sendet ein Host via Broadcast an alle Geräte im Netzwerk eine Anfrage: Wer hat IP X? Tell IP Y! Der Host mit IP X fühlt sich angesprochen und antwortet mit seiner MAC Addresse. Host X speichert dann die IP/MAC Kombination in einer Tabelle, bis diese Information abläuft. Es sind jedoch Probleme wie ARP cache poisoning denkbar.

[1] Nicht schöne!!!!

3.3 IPv6

Da IPv4 Addressen inzwischen eng werden, wurde ende der 90er IPv6 (RFC 2460) introduced. Neben dem Ziel, genug Addressen für alle Geräte zu haben und um z.B. NAT los zu werden, gibt es noch weiter Vorteile.

- 'traffic class' und 'flow labels' ermöglichen QoS
- Flexible Anzahl an Routing hierarchie Leveln
- Serverless Plug-and-Play
- Breite e2e und p2p, IP layer Authentication & encryption möglich
- Besserer Support für Mobile Devices

IPv6 Header Im Vergleich zum IPv4 Header fallen einige Feleder weg.

Tabelle 2: IPv6 Header, length: <– 32 bits –>

VER	Traffic Class	Flow Label
Payload Length	Next Header	Hop Limit
Source IP (32b format, 16 Bytes)		
Destination IP (32b format, 16 Bytes)		

Der v6 Header hat insgesamt die doppelte Länge (40 bytes), im Vergleich zum v4 Header. Im Vergleich zum v4 Header ist die größe hier fix, es können jedoch im Next Header viel TCP/UDP oder IPv6 extension Header genutzt werden. Traffic Class ist das Äquivalent zu Type of Service. Hop Limit ist Äquivalent zum alten TTL.

IPv6 Addresses v6 Addressen werden als 8x16 Bit Blöcke in hex Nummern geschrieben, wobei mit :: alles mit 00 gefüllt wird. Beispiel: fd42:4242:f31a:: = fd42:4242:f31a:0000:0000:0000:000:0000. Es kann auch genutzt werden um Blöcke aufzufüllen, z.B fd42:4242:f31a::abba = fd42:4242:f31a:0000:0000:0000:0000:abba. Die 128 Bits einer IPv6 Addresse sind wie folgt aufgeteilt:

3	13	8	24	16	64
001	TLA ID	Res	NLA ID	SLA ID	Interface ID

Wobei:

- TLA: Top Level Aggregation (IANA teilt diese ISPs zu)
- Res: Reserviert für vergrößerung des TLA oder NLA
- NLA: next-level (Kann zur Organisation von Routing Hierarchie verwendet werden)
- SLA: Site-level (Organisationen können den Bereich für Subnetting nutzen)
- Interface ID: Generierter Suffix für z.B. Endgräte. Hier kann auch ein v6 Suffix aus der MAC Addresse generiert werden:
  - in 2x24 Bits aufteilen, FFFE Einfügen und 7tes Bit invertieren. z.B. 9042:fe34:3413 → 9242:feff:fe34:3413
  - Diese Methode ist jedoch sehr umstritten, da Geräte so weltweit getrackt werden könne. stattdessen lassen sich auch zufällig generierte Suffixe verwenden.

Bisher ist jedoch kein Fester Stichtag für die Umstellung von IPv4 auf IPv6 vorgesehen, und der Übergang scheint such sehr schleppend zu sein. IPv6 kann jedoch auch zwischen Netzen übertragen werden, die eigentlich nur IPv4 betreiben. IPv6 Packete können dann als Payload in IPv4 Packeten versendet werden. IPv6 Routen sind i.d.R. auch in der Lage IPv4 Routen zu handeln.

Futher Improvements Die Umstellung auf IPv6 betrifft jedoch auch Protokolle wie DHCP. Das Ziel hierbei ist es jedoch, Adressing und Routing zu vereinfachen. Neighbor Discovery durch ARP Anfragen fällt auch weg und wird durch ICMPv6 ersetzt. Auch hilfreich sind Methoden wie Link-local, wo mit fe80::/10 Direkt übertragen werden können, befor sie global geroutet werden. Es gibt zwar eine v6 Alternative zu DHCP: DHCPv6 es wird jedoch empfohlen Stateless Address Autoconfiguration (SLAAC) zu verwenden.

4 Transport Layer

In Chapter 2 beschrieben, funktioniert Routing von Packeten global am besten Abstrackt. Jedoch gibt es Anwendungen und Fälle, in denen man eine direkte Verbindung zwischen 2 Applications auf je einem Host herstellen will. Der Sender teilt Nachrichten der Application in Segmente und schickt diese weiter an den Network Layer, um zum Ziel geroutet zu werden. Im Network Layer des Empfängers werden die Segmente dann zu Nachrichten zusammengebaut und an die App weitergereicht.
Im Transport Layer wird die logische Verbindung zwischen Prozessen hergestellt, während im Vergleich der Network Layer die logische Verbindung zwischen Hosts herstellt. Im Internet sind Beispiele für Protokolle des Transport Layers TCP & UDP.

- TCP
  - Congestion Control
  - Flow Control
  - connection Setup & teardown (COTS)
- UDP
  - connectionless
  - Best-effort implemtierung von IP auf dem Transport Layer

Keines dieser Protokolle bietet jedoch eine Lösung für Delay & Bandwidth guarantees.

4.1 Segmentation(+Blocking). Multiplexing, Addressing

Die theoretische beschränkung von IP Packeten sind 64k mit Header. TCP Streams sind zum Teil jedoch längern. Um Applications ein Verständnis zu geben, wie Packete verschickt und SDUs zusammengebaut werden, muss dieser Vorang transparent geschehen. Packete können über Multiplexing auf dem Host an alle Applications gesendet werden, und Applications holen sich nur die Packete, die sie brauchen. Das kommt jedoch mit einem großen Sicherheitsproblem, da nun Anwendungen wie Trojaner einfach den Traffic anderer Programme mithören.

Sockets and Ports Das OSI Modell sieht für die Kommunikation zwischen Programmen und zur identifizierung dieser die Nutzung von lokalen CEPs vor. In Systemen könnte der System PID zur identifizierung benutzt werden, der müsste jedoch für alle Hosts erreichbar sein, außerdem ändert sich dieser bei jedem Neustart des Prozesses oder des Hosts. Im Internet Modell hat man sich jedoch für die Einführung von Ports entschieden. Ein Prozess kann auf einen festen Port announcen oder nach einem zufälligen Port fragen. Eine Application ist dann durch einen socket (IP:Port Kombination) erreichbar, wobei die IP die des Empfänger Hosts und die Port Nummer die der Empfänger Application ist. Es gibt gewissen Ports, die für Anwendungen reserviert sind a.k.a. ”well-known” Ports (IANA: 0-1023). Beispiele sind 22/tcp für ssh, 25/tcp für smtp oder 443/tcp für https. Prozesse die solche Ports öffnen wollen brauchen in der Regel root Privilegien.
Um demultiplexing im Internet zu nutzen, empfängt der Host ein IP Datagram, bestehend aus source und destionation Socket. Diese Datagrams transportieren je ein Transport-Layer segment als Application Data. Das Problem hierbei: Wenn IP eines Tages ersetzt wird, funktionieren TCP/UDP nicht mehr, da die Nutzung von IP:Port hard in die implemtierung gecoded sind.

4.2 Connection Controll

Connection Control funktioniert natürlich nur bei CO Protokollen, in unserem Beispiel also nur TCP. Wie schon in Kapitel 1 angeschnitten, gibt es 3 Phasen der einer connection:

- CONNECT (herstellen der Verbindung)
- DATA (Transfer von Daten)
- DISCONNECT (Schließen der Verbindung)

Um zwischen Network und Transport Layer Phasen zu unterscheiden, schreibt man dort jeweils ein N– bzw. T- Prefix zu den Phasen also z.B. T-Connect oder N-Data.

4.2.1 Phases

Die verschiedenen Schritte können confirmed oder unconfirmed ablaufen. T-Connect wird immer confirmed, T-Data wird unconfirmed übertragen (in seltenden Fällen kann das auch confirmed laufen) und T-Disconnect können confirmed und unconfirmed ablaufen. Ein Beispiel für ein Ablauf ist auf 4:20 zu finden. Auch ein State-Diagramm, dass die Übergänge zu verschiedenen State beschreibt ist auf 4:24 enthalten.

Establishment Beim T-Connect gibt es mehrere Methoden, die aufgerufen werden (können)

- T-Connect.Request(Destination Address, Source Address)
- T-Connect.Indication(Destination Address, Source Address)
- T-Connect.Response(Responding Address)
- T-Connect.Confirmation(Responding Address)

Wobei die Destination Address die Addresse des Transport Service Users ist, also die Addresse die zum Transport gerufen wird. Die Source Address ist die Addresse des aufrufenden Service Users und die Responding Address die Addresse des Antwortenden Service Users.

Data Transfer

- T-Data.req(userdata)
- T-Data.ind(userdata)

Die userdata ist dabei die Payload, die von der Application transportiert werden soll.

Connection Release Der Grund für einen Disconnect kann verschieden sein. Es kann zu einem Release kommen durch abruptes verlieren der Verbding oder als Folge des Connects. Das verlieren von TSDUs (T + SDU) ist möglich.

- T-Disconnect.req(userdata)
- T-Disconnect.ind(cause, userdata)

Der cause für einen Disconnect kann z.B. ein request vom remote User, Quality of service below minimum, error oder auch unknowm sein. Beispiele für Abläufe verschiedener teardowns sind auf 4:23 zu finden.

4.2.2 Error Control

Error Handeling in CONNECT Bleibt ein T-connect.req (CR) unbeantwortet, es kommt also zu einem Timeout, sendet der Host einfach erneut ein CR. Kommt das erste CR jedoch doch nur verspätet beim Empfänger an, und dieser Antwortet auf das erste CR mit einem CC (Connection confirmation) und bekommt dann das zweite CR, sollte durch einbeziehung des Application Layers der zweite CR unbeantwortet bleiben. Was passiert jedoch, wenn das CC verloren geht und wie geht man damit um? Denn der Empfänger (sender of CC) erwartet jetzt einen Datenaustausch. Eine Lösung wäre hier der Three-Way-Handshake

Three-Way-Handshake Beim Three-Way-Handshake Wird das durch ein T-Connect.rsp ausgelöste CC nochmals bestätigt. Das kann entweder direkt ber Data oder per ACK passieren. Das schützt jedoch nicht davor, dass wenn CC und ACK verloren gehen, der Empfänger nicht sagen kann, ob es sich um eine neue oder alte Anfrage handelt. Eine Lösung wäre die Einführung von Sequence Number ins CR, ACK und CC. Host A sendet eine CR (seq=x) and Host B. B antwortet mit einem ACK(seq=y,ack=x) und A sendet dann z.B. DATA(seq=x,ACK=y). Nur wenn die richtigen Sequence Numbers angehängt sind, findet dann der Datenverkehr statt. Anderer Fall. Es kommt ein alter CR bei B an, dieser Antwortet mit der seq Number, da er nicht weiß wie alt er ist. A empfängt ein ACK zu einem alten CR und sendet jedoch ein REJECT, da die Anfrage nicht mehr relevant ist.

Connection Rejection & Release Eine angefragte Verbindung kann jedoch auch Rejected werden. Antwortet ein Empfänger z.B. auf ein CR mit einem DR (Disconnect Request), antwortet der Sender dann mit einem DC (Disconnect Confirm). Normalerweise sendet A ein DR an B, hört jedoch auf eventuelle Daten, die noch unterwegs sein könnten. Er wartet also auf das ACK (DC) von seinem DR und kann währenddessen noch Daten Empfangen. Empfängt er ein DC, kann er sich sicher sein, dass B keine Daten danach mehr sendet. So ein Teardown nennt sich explicit. Ein Implicit Teardown wäre ein Teardown der Network Layer Connection. Das Ziel eines Release ist es, dass beide Seiten alle Daten Empfangen haben und sich nichts mehr mitzuteilen haben. Doch was passiert wenn ein DR/DC/DT verloren geht und nue gesendet werden muss?
Wie kann man sicher gehen, dass A weiß was B weiß und B das weiß und A weiß dass B das weiß...

4.2.3 Two Army Problem

Wie koordinieren sich zwei Armeen, die sich nur Boten durch Tal voller Feinde schicken können und wie stimmen sie ab, dass sie gemeinsam angreifen? Man kann sich nie sicher sein, dass die eigene Nachricht angekommen ist. Das ist das gleiche Problem wie beim connection Release, bei dem ist das Risiko jedoch nicht so hoch, und es können Risiken in Kauf genommen werden. Die Lösung für den Connection Release ist es, einenen Timer parallel zum DR zu starten. falls ein DC/ACK verloren geht, timeoutet die connection automatisch und beide Hosts wissen von der geschlossenen Verbindung.

4.3 Flow Control

Bisher wurden Packete einfach per best Effort versendet. Problematisch wird es jedoch, wenn der Sender eine bessere Verbindung hat als der Empfänger. Wie kann der Sender sicher gehen ob und wenn ja welche Packete angekommen sind? Eine Bestätigung für jedes angekommene Packet vom Empfänger würde diesen eventuell noch mehr einschränken. Wird dieses Verfahren dennoch genutzt, kann der Sender einfach die Packete, die nicht geACKt wurden nach einer gewissen Zeit nochmal versenden, was den Empfänger jedoch nochmal mehr belastet. Eine anderere Lösung ist Flow Control. Das Ziel ist es, langsame Empfänger vor schnellen Sendern zu schützen. Flow Control kann auf zwei Layern implementiert werden: Im Link Layer, dort wird einem Überfluss an "forwading segments"vorgegriffen. Auf höhreren Layern, z.B. auf dem Network oder Transport Layer wird vor einem Überfluss an Conections geschützt. Auf dem Link Layer ist die implementierung simpel, da Segmente eine feste Größe haben, anders auf dem Transport Layer, da können PDUs stark in der Größe variieren.

Buffer Allocation Ein mit dem Flow Control starj zusammenhängendes Problem ist die Buffer Allocation. Ein Empfänger kann z.B. zwar nicht durch seinen Link eingeschränkt sein, sondern durch das Verarbeiten der Packete. Es wird dann ein Buffer geeigneter Größe erfordert, der eingehende Packete bis zu Bearbeitung/Weiterleitung zwischenspeichern kann. Um als Empfänger die Rate, in der Daten eingehen zu kontrollieren, gibt es verschiene Möglichkeiten. Der Empfänger verlangsamt den Sender, wenn es keinen freien Buffer Space mehr gibt (das kann explizit oder implizit geschehen). Dem kann man vorgreifen, indem der Sender initial Buffer Space anfragt und dieser dann allociert wird, oder der Empfänger announced Ïch habe noch X Buffer Space Verfügbar zur Zeit"

Alternating-Bit-Protocol Es wird angenommen, dass es genug freien Buffer Space nach dem Empfangen eines Packets gibt. Ein Empfänger sendet eine Bestätigung für jedes eingehende Packet. Der Sender warten nach jedem gesendeten Packet auf ein ACK des Empfängers. Der Empfänger sendet das ACK erst, wenn das Packet verarbeitet ist, er kann so also kontrollieren, wann und wie oft Packete eintreffen. Kommt ein ACK beim Sender nicht an, gibt es vier Fälle"

1. Packet loss
2. ACK-loss
3. ACK late (heavy Traffic)
4. Ack late (Flow control)

Der Sender kann jedoch nicht ziwschen den vier Fällen unterscheiden. Nach einer gewissen Zeit sendet er das DT PDU also nochmal. Das Alternating-Bit-Protocol ist in der Theorie eine gute Methode für Flow Control, aber eignet es sich auch in realen Umgebungen? In Einem Beispiel senden wir 8KBit von Frankfurt direkt nach NYC über einen 1GBit/s Link. Wir gehen davon aus, dass durch die Hosts kein Delay entsteht. Die Propagation durch die Verbindung ist

$$T_{prop} \approx \frac{6200km}{300.000\frac{km}{s}} \approx 20ms$$

Die Zeit, die das 8KBit Packet braucht, ist etwa

$$T_{trans} \approx \frac{8KBit}{10^9\frac{Bit}{s}} \approx 8\mu s$$

Weiteer nehmen wir an, das ein Ack etwa $1\mu s$ Transmission benötigt. Bis der Sender also 8Kbit übertragen hat und der Sender ein ACK PDU versendet hat und dieses beim Sender ankommt vergehen also $\approx 40.009ms$. Die Utilization des Links ist also etwa:

$$U_{sender} = \frac{0,008}{40,009} \approx 0.0002$$

Stop and Continue Eine alternative einfache Möglichkeit ist Stop-and-Continue Methode. Der Sender sendet Daten, bis er vom Empfänger ein Stop bekommt. Er pausiert das senden bis zu einem continue Signal. Das Problem ist jedoch, dass es es bei einem überlasteten Host schwer werden kann, die Stop nachricht zu versenden, wenn es schon zu einem Overflow kommt. Auch kann es sein, dass der Sender Packete versendet, obwohl der Empfänger schon eine Stop Nachricht versendet hat, da die Nachricht noch nicht angekommen ist (Delay). Der Empfänger muss das Stop also schon früh genug versenden, um einen Overflow proaktiv zu vermeiden. Diese Methode funktioniert nur bei Full-Duplex Links. Ein Beispiel für die implementierung ist das XON/XOFF Protokoll.

4.3.1 Rate and Credit based Flow Control

Beim Rate based Flow Control können im Vergleich zum Stop and Continue Änderungen in der Rate angekündigt werden. Wenn sich also der maximal Verfügbare Buffer ändert kann der Empfänger das mitteilen. Jedoch auch hier gibt es das Problem, dass der Sender erst zu spät davon mitbekommt. Der Übergang zwischen Phasen kann anders als bei Stop and Continue flüssiger ablaufen, da der Sender nicht direkt den Empfang stoppen muss, sondern ihn auch nur einschränken kann. Es gibt jedoch keine Möglichkeit für den Sender sicherzustellen, dass Packete ankommen.
Idee des Credit Based Flow Control ist es, dem Sender regelmäßig Credits zu geben, um Packete zu senden. Hat der Sender keine Credits mehr, pausiert er das senden, bis er wieder neue credits vom Empfänger bekommt. Alternativ kann der Sender auch einmalig eine absolute Anzahl bekanntgeben.
Auch lassen sich die Konzepte mit einer Error correction kombinieren. Der Sender kann dem Empfänger "permitten", daten zu senden und er kann das empfangen von Daten äcknowledgen". Eine ACK Nachricht beduetet dabei, die Nachricht ist korrept angekommen. Der Sender kann also seinen timoute-timer sowie die lokale Kopie des Packets löschen. Problem ist jedoch, wenn durch Flow Control das ACK Packet ausbleibt. Wenn der Sender timeoutet, sendet er das Packet dann nochmal.
Eine Protokoll des Credit Based Flow Control ist das Sliding Window Protokoll. Ein Beispiel ist auf 4:46.

4.4 Error Control

Es kann nie garantiert werden, dass alle Packete ankommen und noch weniger, dass das immer erkannt wird. Wird jedoch erkannt, dass ein Packet verloren gegangen ist, gibt es bei den meisten Protokollen keine Reaktion. Das Packet wird einfach gedroppt. Meistens wird das durch eine Fehlerhafte Checksumme im Header erkannt. Da die Checksumme für den IP Header jedoch die gleiche wie für die Payload ist, ensteht die meiste Fehlererkennung an der Stelle, kann jedoch falsch Interpretiert werden. Ein Empfänger kann auch beim feststellen dass ein Packet Fehlerhaft war ein negative-ACK versenden und die neusendung anfordern. Das hat den Vorteil, dass dann nicht erst der timeout beim Sender auslaufen muss, bis das Packet neu verschickt wird. Wird ein Fehler z.B. durch eine falsche Sequence Number erkannt und ein NACK versendet, kann es jedoch sein, dass ein Packet über einen anderen Weg geroutet wurde, weil ein schnellerer gefunden wurde, und das vorherige Packet noch z.B. im Router hängt. Einzig verlässlig in dem Punkt ist das System: timeout @sender → resend.

Automatic Repeat reQuest (ARQ) Der erste Typ ARQ war das Alternating Bit Protocol. Das lässt sich jedoch weiter verbessern: Go-back-N. Geht ein Packet z.B. mit der Sequence Number 2 verloren oder löst ein Error aus, sendet der Empfänger kein ACK für diesen Frame. Er sendet jedoch weiter die Folge Frames (z.B. 3-8), dessen Frames vom Empfänger Link jedoch weggeworfen werden, da dieser noch auf den Frame mit der Seq. Number 2 warten. Timeoutet dann der Frame2, wird dieser erneut vom Sender gesendet. der Empfänger empfängt diesen Frame korrekt und sendet dann ein ACK. währenddessen sendet der Sender die Folge Frames von 2 (3-8) ebenfalls nochmals da diese auch kein ACK bekommen haben. (Beispiel siehe 4:49).
Eine Adaption dieses Protokolls wäre, wenn der Empfänger nicht auf den Timeout wartet, sondern für den fehlerhaften Frame ein NACK sendet. Dieser Frame wird dann erneut gesendet, in der Zeit buffert der data link des Empfängers Frames mit dn folgende seq. Number.

4.5 Congestion Control

Bei Kommunikation zwischen zwei Hosts gibt es immer ein Bottleneck, dass die Geschwindigkeit beschränkt. Ein Netz ist dann auf genau dieses Bottleneck beschränkt. Werden mehr Daten gesendet, als das Bottleneck übertragen kann, kommt es zu einem congestion collapse. Bei solch einem collapse gehen viele Packete verloren und die Effizienz sinkt unter das Bottleneck. Je nachdem wo das Bottleneck ist, kann sich das ganze auch in sich selbst steigern, ist z.B. der Router durch den Buffer Space beschränkt und es kommt zu einem collapse wobei der Sender ein Protokoll nutzt, dass nicht angekommene Packete nochmal sendet, wird der Router mit einer noch größeren Flut an Packeten überschwemmt. Im folgenden sind λ_{in} die Daten die Host A versendet und λ_{out} die, die Host B empfängt. Im Idealfall ohne Packetverlust ist $\lambda_{in} = \lambda_{out}$

Scenario 1 Wir gehen von einem Router mit umbegränztem Buffer aus und Packete werden nicht neu versendet. Die Throughput steigt bis zum Maximum an, bleibt dann stabil, der Delay steigt exponentiell an, wenn es zu einer congestion kommt.

Scenario 2 Hat der Router jedoch einen beschränkten Buffer und Host A sendet nicht bestätigte Packete nach einiger Zeit nochmal. Host A sendet jetzt $\lambda'_{in} = \lambda_{in}+$ retransmitted packages. Perfekt wäre nur wirklich verloren gegangene Packete neu gesendet werden, λ'_{in} wäre $> \lambda_{out}$. Werden nicht wirklich verlorerene Packete neu gesendet, wird λ'_{in} hörher als nötig.
Wie bei den Scenarien gemerkt, wird congestion controll esentiell, um einen Schneeball Effekt zu verhindern.

4.5.1 solving congestion Control

Eine Globale Lösung wäre, die Sende Rate an den Bottleneck anzupassen. Das kann jedoch nur gloabl passieren und jeder Router müsste seine Maximal Kapazität bekanntgeben. Diese Lösung macht das Problem nur komplizierter und kleine Konfigurationsfehler können das gesamte System betreffen. Deswegen wird Congestion Control zu einem lokalen Problem gemacht. Meistens wird es zwischen Sender um Empfänger gelöst. Das Ziel ist es, so viele Packete wie möglich zu verschicken, ohne jedoch die Leitung zu verstopfen. Außerdem sollte Congestion Control Fair ablaufen, jedoch unter welchen Bedingungen? Ein 3-Hop Weg ist genausoviel Wert wie 3 1-Hop ways? Priorisierung in Anwendungen (Video Call > Downloads)?

Design Options for Congestion Control Es gibt 2 fundamentale Anstätze, Congestion Control zu gewährleisten:

- Closed-Loop - Der Sender bekommt Feedback über das aktuelle Verhalten und kann die Kapazität anpassen.
  - implementierung am Sender
  - implementierung am Empfänger
- Open-Loop - Das System von vorne herein funktioniert und keine Korrekturen im Betrieb notwenig sind.
  - explicit Feedback - Im Fall wenn es passiert, den Sender Informieren
  - implizit Feedback - Der Sender passt die Rate selbst an ohne direktes Feedback vom Empfänger sondern durch Daten aus dem Betrieb (z.B. ausbleibende ACKs). Beispeil: TCP

Possible Actions Um Netze vor Congestion zu schützen, gibt es andere alternativen als einfach den Traffic zu pausieren. Um häufige Congestions entgegen zu wirken, kann die Netz Kapazität z.B. durch mehr Router erhöht werden. Auch kann ein Router das allocieren von zusätzlichem Traffic einschränken, wenn das Netz schon fast ausgelastet ist, jedoch sind Informationen über den Netzwerk state selten Verfügbar. Ebenso können alle Session einen Teil der Verfügbaren Load reduzieren, um allgemein mehr Kapazität zu schaffen. Dafür wird jedoch Netzwerk Feedback benötigt, das geht also nur in closed-loops. Die Klassifizierung der Packete kann Zentral am Router oder am Host stattfinden. Im Internet geschieht das (außer beim droppen von Packeten) am End-Host. Die einzelnen Klassen bekommen entweder einen gewissen Byte Betrag pro Sekunde (rate based) oder eine gewisse Anzahl an Sequence Numbers/Bytes im Netz verarbeitet werden dürfen (Es gibt auch andere, jedoch unpopulärere Methoden z.B. Credit Based Congestion Control).

Package Dropping Ensteht ein collapse durch einen vollen Router Buffer, müssen Packete aus diesem gedroppt werden, doch welche? Eine Methode ist die drop-tail queue, bei dir das neuse Packet weggeworfen wird. Das spielt z.B. gut mit dem go-back-n Protokoll zusammen. Für andere Anwendungen, z.B. Telefon sind neue Packete wichtiger als alte, hier würden am besten Packete gedroppt, die schon lange in der Queue sind. Eine solche Aktion ist auch immer, ein implizites Feedback. Der Sender erkennt das nicht ankommen des Packets. Ensteht eine Congestion durch einen vollen Buffer, muss der Traffic verringert werden.

proactiv Actions Prinzipiell ist ein Router nicht mehr in der Lage, richtig zu funktionieren, wenn er einmal eine volle Queue hat, es sollte also um jeden Preis vermieden werden, z.B. indem man bereits bei z.B. 90% einen warning state Aufruft. Eine andere Methode ist das Chocke Packet. Ein Router mit vollen Buffer sendet so ein solches Packet an den Sender, dass diesem sagt, die Rate zu verringern. Das Problem ist jedoch, in einem congested Netz gehen mehr Packete verloren als sonst, es kann also sein, dass der Sender ein solches Packet nie erhält. Auch kann ein Router statt ein extra Packet zu senden, ein warning Bit in allen ausgehenden Packeten setzen, dass allen Hosts sagt, dass der Router überlastet ist, oder das kurz bevor steht. Eine vielleicht etwas radikalere Lösung ist die Random Early Detection (RED), die mit einer Rate zufällig Packete droppt, auch wenn der Router noch nicht überlastet ist, um diesen genau davor zu schützen. Diese Rate je nach Anzahl Packete im Buffer variieren.

Feedback Actions Sobald ein Sender Informationen über eine congestion erreicht hat, muss der Traffic reduziert werden. Doch wie das geschieht, hängt von den Protokollen des Transport Layers ab.

4.6 (Internet) UDP

RFC 768 UDP ist eine Best Effort Lösung für Packetübertragung im Internet. UDP ist connectionless, es gibt also keine Handshakes zwischen sender & Empfänger. UDP Packete können verloren gehen und ohne congestion Control einfach ”in die Welt” geblasen werden. UDP wird für z.B. Streaming benutzt, wo es nicht so schlimm ist, wenn ein Packet mal wegfällt, aber auch DNS Anfragen oder SNMP setzten auf UDP. In-Oder delivery oder reliability können auf dem Application Layer implementiert werden. Ein UDP Segment ist wie folgt aufgebaut:

source Port #	dest Port #
length	checksum
Application data (message)	

UDP Checksum Das Ziel ist es, Fehler in übertragenen Daten zu erkennen, z.B. geflippte Bits. Der Sender teilt für die UDP Checksum in 16 Bit Sequenzen. Er addiert dann jewils die Sequenzen zusammen und nimmt davon das 1er Komplement und schreibt das in das Checksum Feld. Der Empfänger muss dann nur die Segemnte wieder zusammen addieren und die Checksumme draufrechnen. Ist das Ergebnis 1 (0xFFFF), ist das Packet wahrscheinlich Fehlerfrei, ansonsten wird das Packet gedroppt. Ein Beispiel für eine solche Berechnung ist auf Folie 4:79.

4.7 (Internet) TCP

4.7.1 Overview

Im Gegensatz zu UDP hat TCP einige Eigentschaften, die es geeigneter machen für den gebrauchen in den meisten Netzen.

- **Point-to-Point** Es gibt einen Sender und einen Empfänger
- **Reliable, in-order byte stream** Es gibt keine direkten Nachrichten Grenzen
- **Pipelined** TCP Congestion und FLow Control setzt auf window-sized Version
- **Full duplex data** Bi-directional communication in der selben Verbindung möglich. Es gibt eine maximum segment size (MSS)
- **send & reciever Buffer** Es gibt im T-Layer (unter dem Socket zum A-Layer) einen TCP Buffer, der auch die zu sendenden Plakate buffert
- **Connection-Oriented Communication** Vor dem Übertragen von Daten werden Kontroll Nachrichten ausgetauscht (Handshake), die den State initiieren
- **Flow & congestion Control** In TCP sind implementierungen für diese beiden Ansätze vorhanden

Ein TCP Segment hat dabei den volgenden Aufbau:

source Port #								dest Port #
sequence Number[1]								
acknowledgement number[2]								
head len	not used	U[3]	A [4]	P [5]	R [6]	S [7]	F[8]	receive window [9]
checksum[10]								Urg data pnter
Options (variable length)								
application data variable length								

Der Socket eines Host, z.B. ein Webserver auf Port :443 kann dabei mehrere connections gleichzeitig haben. Beim TCP (CO) De-multiplexing wird dann zwischen dest und source Socket unterschieden. Dieser TCP hat dann genau 4-Tupel:

- Source IP Address

[1] Es wird nach anzahl der Bytes gezählt, nicht der Segment
[2] Es wird nach anzahl der Bytes gezählt, nicht der Segmente
[3] URG: urgend data. In der Regel nicht genutzt
[4] ACK: ACK # ist valid
[5] PSH: Push Data now. In der Regel nicht genutzt
[6] RST, connection Establishment/teardown commands
[7] SYN, connection Establishment/teardown commands
[8] FIN, connection Establishment/teardown commands
[9] # Bytes, die der Reciever bereit ist, zu akzeptieren
[10] Internet Checksumme (vgl. UDP)

- Source Port Number
- Dest IP Address
- Dest Port Number

Ein Empfänger nutzt diese 4 Tupel um ein Packet/Segment dem richtigen Socket zuzuweisen. WEbserver haben z.B. für jede Verbindung zu einem client einen eigenen Socket. Gewisse Webserver können auch für jede HTTP Anfrage einen eigenen Socket brauchen.

4.7.2 Error & Connection Control in TCP

Sequence Numbers werden bei TCP genauso wie ACKs genutzt. Ein ACK gibt dabei die Sequence Number des nächsten erwarteten Packets an. Ein Beispiel für einen Ablauf ist auf 4:87.
Um TCP timeouts richtig einzustellen, also die Zeit bis zu einem timeout, gibt es einige Faktoren zu beachten. Der Timeout darf nicht kleiner als die Round Trip Time (RTT) sein, sonst kommt es zu unnötigen retransmission, sie darf jedoch auch nicht zu lange sein, sonst kommt es zu unnötig langen verzögerungen beim neusenden von Packeten. Die RTT kann man z.B. berechnen, indem man die Zeit ziwschen senden eines Packets und ankommen des ACKs misst. Dieses verfahren nennt man SampleRTT. Da das von Packet zu Packet varrieren kann, kann man das Ergebnis durch runden mehrerer Rechnungen glätten.
TCP Connections können aktiv oder passiv hergestellt werden:

- **Active Mode:** Ein Host fragt eine Connection bei einem anderem Host an.
- **Passive Mode:** Eine Anwenung Informiert TCP, dass sie auf einem Socket connections annimmt. Es kann jedoch auch Port unspeziefisch sagen, dass alle eingehenden Verbindungen akzeptiert werden. Sobald eine Verbindung eingeht, wird ein neuen Socket aufgemacht, der als Endpunkt für die Verbindung gilt.

Der Ablauf einer TCP Connection ist auf 4:90 Beispielhaft aufgeführt, außerdem sind auf 4:91 & 4:92 Szenarien aufgeführt, wie sich TCP in einigen Fällen verhält, wenn ACKS ausbleiben, nach timeouts ankommen oder anderes.

Fast Retransmit Geht ein Packet verloren, kann es lange dauern, bis der timeout ausläuft, und das Packet neu gesendet wird. Eine Lösung für dieses Problem ist Fast Retransmit. Erhält ein Sneder aufeinanderfolgend 3 mal das gleiche ACK, da zwar nachfolgende Packete angekommen sind, der Empfänger jedoch noch das verlorene Packet erwartet sendet TCP schon vor einem timeout das Packet neu. Das ist guter Kompromiss ziwschen nicht unnötig lange warten und nicht unnötig Packete versenden.

Send and Receive Buffers in TCP TCP pflegt jeweils auf Sender und Empfänger Seite einen Buffer um beim Sender Packete zu speicher, die z.B. noch nicht geackt sind, um diese ggf. neu zu senden oder Daten der Application zu speichern, bevor diese versendet werden. Ältere Versionen von TCP haben nach dem Go-Back-N Prinzip gearbeitet und out-of-order Packete weggeworfen. Inzwischen werden auf Empfänger Seite Packete, die neuer als erwartet sind ziwschengespeichert, um nicht alle neu senden zu müssen.

4.7.3 Flow & congestion Control in TCP

In TCP kann der Empfänger den advertise size von seinem Buffer announcen. Der Buffer size ist dabei:

$$\text{Advertised window} = \text{MaxRcvdBuffer} - ((\text{NextByteExpected} - 1) - \text{LastByteRead})$$

der advertiste Speicher beschränkt dabei den Sender in der Anzahl an Daten, die dieser verschickt. Der Sender achtet dabei darauf, dass

$$\text{LastByteSent} - \text{LastByteAcked} \leq \text{AdvertisedWindow}$$

bzw.

$$\text{EffectiveWindow} = \text{AdvertisedWindow} - (\text{LastByteSent} - \text{LastByteAcked})$$

Self Clocking Winwo Bisher gingen wir davon aus, dass neue Packete diregt verschickt werden, wenn ein ACK eingetroffen ist. Problem jedoch, wenn ACKs schneller einkommen, als die Application Daten in den send Buffer schreibt, werden viele kleine Packete verschickt. Man spricht dabei vom »silly window syndrom«. Eine Lösung hierfür ist Nagle's Algorithmus.
Wenn die Verfügbaren Daten und das advertiste Window $\geq$ MSS sind, wird ein volles Segment verschickt. Ansonsten wird geschaut, ob bereits an Packet verschickt wurde, das noch ncith geACKt wurde, dann wird gewartet bis MSS voll ist. Falls kein Packet verschickt ist, werden die neuen Daten sofort verschickt.

Congestion Control TCP kontrollliert arbeitet mit implicit feedback um congestion zu verhindern, konkret mit der Information über dropped Packages. Es kann weiter nicht unterschiden werden, warum Packete gedroppt werden, deswegen wird davon ausgegangen, dass diese wegen congestion gedroppt wurden. TCP ermittelt mit diesen ein congestion window, dass im laufenden Betrieb reflektiert und aktualisiert wird. Zusätzlich zu den Limits der Flow Control beschränkt der Sender die Anzahl an Packeten durch

LastByteSent-LastByteAcked $\leq$ CongWin

Um die darstellung von congestion Control zu vereinfachen, wird in Zukunft das flow Control Window ignoriert.
Hat TCP jetzt genau so viele Packete in das Netzwerk gesendet, wie das congestion Window erlaubt, wird gewartet, bis diese das Netz wieder verlassen haben. Durch das erreichen von ACKs ist sichergestellt, dass die Packete nicht mehr im Netz sind und es können neue gesendet werden.

Congestion Window (AIMD) Bekommt TCP ein ACK, geht es nach Greedy von der Annahme aus, dass das Netz noch Kapazität hat und erhöht das congestion window. Bleiben jedoch ACKs aus und es kommt zu einem timeout, wird das congestion window verkleinert. Doch um wieviel? Um das Netz vor einem congestion collapse zu schützen, müssen drastische Maßnahmen getroffen werden, und das congestion window wird auf 50% gekürzt (halbiert)! Das Window kann jedoch nie kleiner als 1 sein.
Das kürzt jedoch die nehr als nötig, wenn Packete nicht wegen congestion, sondern z.B. durch Übertragungsfehler gedroppt werden. Das passiert in physischen Netzen selten, ist jedoch grade bei Mobilen Netzen ein Problem. Das ganze nennt sich mupliplicative decrease.
Wird jedoch das Congestion Window durch ACKs erhöhrt, wird es nicht so stark erhöht, da es sonst zu einem collapse kommt. TCP testet sich langsam an das Maximum heran und erhöht es immer nur um einen kleinen Teil. Wenn alle gesendeten Packete innerhalb der RTT (Round Trip time) ein ACK erhalten haben, sendet TCP in Zukunft ein Packet mehr pro RTT.
Das ganze kann auch per additive increase passieren, dabei wird das Congestion Window pro erreichtes ACK um folgenden Wert erhöht:

Congestion Window += MSS x (MSS / Congestion Window)

TCP setzt insgesamt auf ein AIMD (adaptive increase multiplicative decrease) ”Sägezahn” Pattern. (Beispiel siehe 4:105). AIMD start jedoch mit einem congestion Window von 1 oder 2 und brauch länger, um anzulaufen, was gerade bei schnellen Netzen unnötigt Zeit kostet. Es kann also anfangs das congestion Window z.B. je ACK verdoppelt werden, bis es einmal durch loss halbiert wird. Ab diesem Punkt wird dann auf linearen anstieg umgestellt.

Packet Burst Ein Problem jedoch: Ein Packet geht verloren. Nach einem timout wird das Packet neu gesendet und es kommt ein ACK zurück. Packete wurden seit dem timeout zurückgehalten und das congestion window zwar halbiert. jedoch wird jetzt auf einmal das gesamate (halbierte) Congestion Window auf einmal ausgeschöpft und ein ”Packet Sturm” losgeschickt. Dadurch wird es wahrscheinlich zu einem hohen Packetverlust kommen. Die Lösung ist, in dem Fall wieder slow zu starten mit einem congestion window von 1. Jedoch gibt es jetzt eine Idee von der maximalen Größe des Congestion Windows. Dieser wird kann also congestion threshold genutzt werden. Es wird also mit dem exponentiellen slow start bis zum threashold increased und dann auf additive increase gewechselt.
Ein Beispiel Ablauf ist auf 4:111.

4.8 Other Protocols

Da der Multimedia, vorallem Video, streaming in den letzten Jahren stark zugenommen hat, wird mehr auf Protokolle, die auf UDP Basis aufbauen gesetzt. Der meiste Traffic geschieht trotzdem größtenteils mit TCP, andere Protokolle z.B. QUIC sind auf dem vormarsch.

4.8.1 SCTP

Das Stream Control Transmission Protocol (SCTP) Protokoll ist message-Oriented (wie UDP), hat jedoch wie TCP flow, congestion und error control implementiert. Jede Nachticht in einem Stream erhält dabei eine sequence Number, um Packete zu ordern. Ein Empfänger kann dabei ordering ordering und reliability an und ausschalten, ebenso fragmentation und Nagle. Außerdem können mehrere Streams in einer einzelenen Verbindung gehandelt werden und beide Endpunkt können mehrere IP Addresses nutzen (failover).

4.8.2 DCCP

DCCP (DatagramCongestionControl Protocol) ist im Kern etwa wie TCP nur ohne bytestream semantik und reliability oder wie UDP mit congestion Control, Handshakes und optionalen ACKs. Es ist CO. Durch seperate Header und Data checksums ist es nicht zuverlässig und wird selten genutzt. Es besteht keine große Chance, dass es weitfläufig genutzt wird.

4.8.3 RTP/RTCP

RTP (Real-time Transfer Protocol) ist gedacht als realtime multimedia Protokoll. RTP Session Data unterstützt sequence Number und timestams, encryption und Informationen über den Payload inhalt. Ähnlich wie QUIC läuft RTP auf Basis von UDP. Dabei unterstützt die RTP library für den Application Layer Funktionen wie Port numbers and IP Addressen, Payload type identifizierung, Packet sequence numbering und time-stamping.

4.9 QUIC

QUIC (Quick UDP Internet Connections) wurde von Google entwickelt, ist inziwschen aber in IETF Standarts eingearbeitet. 2017 machte es bereits etwa 7% des globalen Traffics aus. Durch schnelles connection setup and secutiry mit TSL1.3 wird es besonders attraktiv gemacht. Es unterstützt außerdem error forwading.

5 Queueing Theory

Tabelle mit Zusammenfassung der Queue Koefizienten für diverese Queues ist auf 5:43. Beim Queueing gibt es 5 wesentliche Punkte:

1. Arival Process: How do customers/requests arrive?
   - Typically described as probability distribution of interarrival times
   - A(t) = P[time between arrivals $\leq$ t]
2. Service Process: Wie viel Nachfragen werden pro Anfrage generiert?
   - Service time B(x) = P[service time $\leq$ x]
3. Wie viel Platz ist in einer Queue
4. Wie viele Server stehen zur Verfühgung
5. Nach welcher policy werden Afragen verabreiter
   - FCFS (First come first servere), shortest job first, priority queue?

Außerdem werden in diesem Kapitel die folgenden Notationen genutzt:

- C_n ist der nte job/task (costumers), der dass System betritt
- $\alpha(t)$ = Anzahl der **arrivals** innerhalb $[0, t)$
- $\delta(t)$ = Anzahl der **departures** innerhalb $[0.t)$
- $N(t)$ = Anzahl der costumers im System zu der Zeit $N(t) : \alpha(t) - \delta(t)$
- τ_n = Ankunftszeit des costumers C_n
- t_n = Zeit zwischen C_n und $C_{n-1} = \tau_n - \tau_{n-1}$
  - $P[t_n \leq t] = A(t)$
  - Vergleiche **arrival rate =** λ wobei $\lambda = \lim_{t\to\infty} \frac{a(t)}{t}$
- x_n = service Zeit für C_n
  - $P[X_n \leq x] = B(x)$
  - vergleiche: service rate = μ
- w_n = waiting time in der queue für C_n
- s_n = system time für $C_n = w_n + x_n$

Neben diesen gibt es noch einige besodere Notationen. Wobei $\overline{t}$ die durchschnittliche Zeit zwischen costumers ist.

- $\overline{t} = \frac{1}{\lambda}$
- $\overline{x} = \frac{1}{\mu}$
- $\overline{w} = W$
- $\overline{s} = T$

5.0.1 Little's Law

Sei N_t die durchschnittliche Anzahl an costumers innerhalb $[0, t)$

$$N_t = \frac{1}{t}\int_0^t N(\tau)\, d\tau = \frac{\gamma(t)}{t}$$

Mit $T_t = \frac{\gamma(t)}{\alpha(t)}$ und $\lambda_t = \frac{\alpha(t)}{t}$ erhalten wir $N_t = \lambda_t T_t$. Wenn λ die durchschnittliche Ankunftsrate ist und wenn $\lim_{t\to\infty} T = T$ exisitiert, dann ist

$$N = \lambda T$$

Little's law. Die durchschnittliche Anzahl an costumers im System ist äquivalent zu der durchschnittlichen Rate an eingehenden costumers mal die durchschnittliche Zeit, die diese im System sind.

5.0.2 Utilazation

Der Utilazation Factor ρ ist die Rate zwischen eintreffenden jobs und den jobgs, die das System verarbeitet. Für Systeme bestehend aus einem server:

$$\rho = \lambda/\mu = \lambda\overline{x}$$

Normal ist $0 \leq /rho < 1$

5.1 Stochastic Process

(wahrscheinlich nicht relefant) Die bisherige Theorie des Queues ist determistisch, doch in der Theorie ist es weniger fest als mehr zufällig, da viele Faktoren mitspielen deren Berechnung schwierig ist. Stattdessen kann man Queues mit stochastischen Ansätzen berechnen. Ein Stochastic Process ist dabei eine Familie von zufälligen variablen $X(t)$, wobei die Variablen nach Zeit geindext werden. Die Processe lassen sich dabei folgendermaßen klassifizieren:

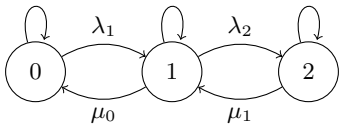
1. **State space:** Welche Werte kann $X(t)$ annehmen? Diskrete oder fortlaufende?
2. **Time parameter:** Ist die Zeit diskret oder fortlaufend? Falls Diskret können Änderungen nur an bestimmten Zeiten auftreten: $X(t) = X_n$
3. **Dependence between X(t)'s:** Konditionale wahrscheinlich (Abhängig vom vorherigen State). Order of dependence: $P[X(t_{n+1} \leq x_{n+1}|X(t_n) = x_n, ..., X(t_1) = x_1)]$

5.2 Birth-Death Process

Beim Birht-Death Process können state Änderungen nur nach Nachbar Sates führen. Der aktuelle Status sei k, dann sind die möglichen Sati in der nächsten Iteration k-1, k oder k+1. Eine Transition von k zu k+1 nennt man birth, eine von k zu k-1 Death. In einem leeren System kann es zu einem birth, jedoch nicht zu einem death kommen. Die Birth-rate sei λ_k und die Death-rate μ_k mit einer Populationgröße von k.
Ein Process gilt als Birth-Death Process, wenn die folgenden 3 Eigentschaften erfüllt sind:

1. Der Process ist eine homogene Markov chain X(t) auf dem State 0,1,2...
2. Births and Death sind unabhängig
3. folgende Eigentschaften sind wahr
   - P[1 birth in$(t, t+\delta t)$|k in pop.$] = \lambda_k \delta t + o(\delta t)$
   - P[1 death in$(t, t+\delta t)$|k in pop.$] = \mu_k \delta t + o(\delta t)$
   - P[no birht nor death in $(t, t+\delta t)$|k in pop.$] = 1 - (\lambda_k + \mu_k)\delta t + o(\delta t)$

Alternativ lässt sich ein brith-death Process aus folgend darstellen:



5.2.1 Poison Process

Nehmen wir an, dass wir für einen Brith-Process ist $\mu_k = 0 \forall k$. Außerdem nehmen wir an, dass $\lambda_k = \lambda \forall k$. Setzen wir das in unsere Gleich ein, erhlaten wir:

$$\frac{dP_k(t)}{dt} = -\lambda P_k(t) + \lambda P_{k-1}(t) \forall k \geq 1$$

$$\frac{dP_0(t)}{dt} = -\lambda P_0(t) \forall k = 0$$

Nehmen wir weiter an, dass wir im State $0, P_k(0) = 1$ für $k = 0$,0 für $k \neq 0$. Für $P_0(t)$ erhalten wir $P_0(t) = e^{-\lambda t}$. Mit $e^{-kx}dx = -ke^{-kx}$ erhalten wir $P_1(t) = \lambda t e^{-\lambda t}$. Mit Induktion erhalten wir dann die Poisson distribution

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

Poison Process beschreibt dabei eine Sequenz von brith Epochen, die hauptsächlich vom Parameter λ abhängig sind. Hier beschreibt Poison Process das eintreffen von costumers. DIe durchschnittliche Ankunftsrate in t ist dabei $E[k] = \lambda t$

5.2.2 Equilibrium in Birht-Death Systems

TODO

5.3 The M/M/1 Queueing System

5.3.1 Kendall's Notation

Mit der Kendall Notation werden die verschiedenen Parameter des Systems angegeben. Die Notation hat das Format

A/S/m/N/K/SD

wobei

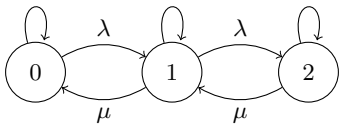
- **A = Arrival process**
- **S = Service Process**
- **m = Number of servers**
- N = Platz im System. Wenn nicht angegeben idR. ∞
- K = Population Size (in CNuvS wird nicht näher darauf eingegangen)
- SD = Queue discipline. default = FCFS

Für A und S werden die folgenden Notationen verwendet:

- M = Exponentioneller Prozess (Markovian)
- D = Determistisch
- G = Generell
- außerdem noch einige spezielle Fälle

5.3.2 M/M/1

M/M/1 ist das simplelste Queueing System. Konkret bedeutet das, Poisson Arrivals, exponential service auf 1 Server. Dabei ist $\lambda_k = \lambda_{k1} = \lambda_{k2} = \lambda_{k3} = \lambda$. Genauso für μ. Das State Diagramm sieht dann wie folgt aus:



Analog wäre dann der Brith-Death-Process die Anzahl Costumers im System, Birht das ankommen eines neues costumers (entweder in der Queue oder im System) und Death dass der Service geliefert hat und der costumer das System verlassen hat.

M/M/1 Solutions Die wahrscheinlichkeit für ein Steady State in M/M/1 Systemen ist

$$p_k = p_0 \prod_{i=0}^{k-1} \rho = p_o \cdot \rho^k$$

wenn die Vorraussetzung $\lambda < \mu$ erfüllt ist mit $\rho \frac{\lambda}{\mu} < 1$. Daraus lässt sich schließen, dass

$$p_k = (1-\rho)\rho^k$$

. Die durchschnittliche Zeit von costumers im System ist bei M/M/1

$$\overline{N} = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu - \lambda}$$

Mit Little's Law lässt sich außerdem die durchschnittliche Zeit im System berechnen

$$T = \frac{1/\mu}{1-\rho} = \frac{1}{\mu - \lambda}$$

Wie sieht an den Formen ablesen lässt, steigen N und T für ρ nah 1 exponentiell an.

M/M/1/N Auch diese Art ist mit Birth-Death Modellierbar, hier ist einfach die Länge der queue auf N beschränkt. Kommen costumer an, die Queue ist jedoch voll, werden diese verworfen. Wie Im Telefonnetz versuchen Costumer es dann zu einem Späteren Zeitpuntk erneut.

5.3.3 M/M/m Queues

Wie im Titel schon zu sehen, handelt es sich hier um M/M Queues auf m (oder auch n) Systemen. Idelt ein Server, bekommt dieser direkt eine neue Anfrage, sind alle Server beschäftigt, müssen Anfragen warten. M/M/m Queues können weiterhin mit Birth-Death modelliert werden. Die Koefizienten sind

$$\lambda_k = \lambda$$

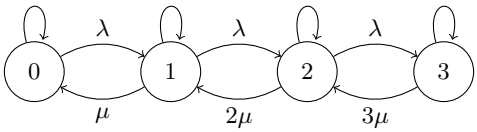
$$\mu_k = min(k\mu, m\mu)$$

wobei

$$k\mu \forall 0 \leq k \leq m$$

$$m\mu \forall m \leq k$$

Due State Machine sieht wie folgt aus:



Die probability of Queueing δ ist

$$\delta = P[Queueing] = p_0 \frac{(m\rho)^m}{m!(1-\rho)}$$

Die durchschnittliche Number of Costumers ist

$$\overline{N} = m\rho + \frac{\rho\delta}{1-\rho}$$

Mit Little's Law ist die durchschnittliche Zeit im System

$$T = \frac{1}{\mu}\left(1 + \frac{\delta}{m(1-\rho)}\right)$$

Andere Beispiele für Queues mit mehr als 1 Systemen auf 5:44-5:49

6 Multicast

Bisher haben wir mit folgenden Modellen gearbeitet:

- Unicast
  - Ein Sender und ein Empfänger
- Broadcast
  - Ein Sender, alle anderen Netzwerkteilnehmer Empfangen.

Multicast hingegen ist eine Lösung, Daten an mehrere Empfänger zu verschicken, ohne Traffic mehrmals zu versenden. Packete werden dann an definierte Multicast Gruppen versendet.

6.1 Implementing Multicast

Multicast via unicast Ein Sender sendet hierbei je Empfänger ein Packet konkret an den Empfänger. Der Host sendet also für N Empfänger N Packete. Hierbei fallen die Vorteile von veringerung des Traffics weg. Das Packet wird direkt beim Sender dupliziert.

Network Multicast Ein Host sendet hierbei nur ein Packet ins Netz, das von Routern dupliziert wird, falls dieses an verschiedene Routen weitergeleitet werden soll.

Application-Layer multicast Packete werden hier beim Host im Application Layer dupliziert. Dieser sendet die Packete dann erneut durch das Netzwerk an andere Empfänger.

6.2 IP Multiplexing

Die wohl beste und weit verbreiteste Lösung ist die Implementierung des IP Multiplexing. Es gibt definierte IP Blöcke, die für Multicast vorgesehen sind. Host können einer Gruppe (= eine Addresse) beliebig beitreten oder diese verlassen. Host speichern dabei keine Liste In IPv4 sind die Addressen 224.0.0.0-239.255.255.255 für Multicast reserviert, wobei 224.0.0.0/24 für Multicast Routing und Group Management gedacht sind. Die Addressen können nur als Destination genutzt werden und für Multicast datagrams können keine ICMP error messages generiert werden.
In IPv6 sind die Addressen FF00::/8 für Multicast reserviert. Nach den ersten 8 Bit sind die 4 weitern Bits für falgs reserviert, z.B. well known oder transient. Die nächsten 4 Bits (13-16) sind für den scope reserviert, z.B. loopback (1), link-local, site-local (5) oder global (E).
Um Multicast zum Ziel zu bringen, müssen Routern multicast routing protokolle sprechen, um delivery trees zu bauen und packete weiterleiten zu können. Außerdem muss ein Router ein Group membership protocol (IGMP) sprechen, um zu welchen, ob Geräte ”seines” Netzwerks in multicast Gruppen sind.

Joining a multicast Group Der Host informiert via IGMP den Router, dass er gruppe jounen will. Zwischen Routern gibt es andere Protokolle, z.B. DVMRP, MOSPF oder PIM.

6.2.1 IGMP

Will eine Application einer Multicast Gruppe beitreten, sendet Host eine IP_ADD_MEMBERSHIP request. Das ganze ist ein soft state, das heißt, das der Host nicht explizit leaven muss. Ein router leitet Packete via boradcast in ”sein” subnet weiter und alle Host die Teil dieser Gruppe sind, müssen regelmäßig replien.

6.3 Multicast Routing

Um den Tranfer des Packets vom Sender zum Router zu organisieren, gibt es verschiedene Algorithmen oder Methoden. Das Ziel ist es, einen Tree mit Routen zu bauen, dabei wird jedoch entschieden Zwischen

- Shared-tree (alle Hosts nutzen den gleichen Tree)
  - Group Shared Trees (Jede Gruppe nutzt ein Tree)
    - Minimal Spanning Tree (MST)
    - Core based trees (CBT)
- und Source-Based Tree (Ein Baum von jedem Sender zu jedem Empfänger)

- Shortest path trees (SPT)
- Reverse path forwarding (RPF)

Flooding Statt der nutzung von dedizierten Algorithmen kann ein Router auch per flooding Packete an andere Router weiterleiten, es muss sich allerdings merken, welche Packete er schon weitergeleitet hat, sonst kommt es zu einem flodding storm. Es wird also viel Leistung benötigt, um den state zu dokumentieren. Außerdem wird traffic sehr ineffizient genutzt, da Router auch Packete bekommen, die sich intern nicht weiterleiten.

6.3.1 Spanning Trees

Das Ziel ist es, einen Graphen zu bauen, der zwei Host nur über einen Pfad verbindet, um so loops zu verhindern.

Steiner Trees Bei Steiner Trees wird aus einem (Netzwerk-) Graphen G=(B,E,cost) und terminals $S \subseteq V$ ein Graph $T \subseteq G$ über S mit den leichtesten Pfaden gebaut. Da jedoch Informtionen über die Netzwerk Topoligie sowie viel rechenleistung je Änderung (z.B Ausfall oder hinzufügen eines Routers) benötigt wird, ist diese Methode nicht im Internet vertreten.

Shorstest Path Tree Hier wird in einem gewichteten Graphen mit Dijkstras der kürzeste Pfad zu jedem Host mit einem Group Member gesucht. Durch link state Routing kann Dijkstras Informtionen zur Topoligiebekommen.

Reverse Path Forwarding Wenn ein Packet von einem Host über einen Pfad kommt, von dem der Router weiß, dass es der kürzeste Pfad ist, sendet er das Packet per Flooding an alle outgoing links weiter. Ein Beispiel ist auf 6:29. Eine weiterentwicklung wäre, Prune Nachricht zu schicken, damit Subtrees die nicht Teil der Gruppe sind, keine unnötigen Packete per flooding bekommen.

6.3.2 Multicast Routing Protocols

Distance vector multicast routing protocol (DVMRP) Mit RFC1075 eingeführt, Flood and prune: RPF source-based trees. Keine Annahme von underlay unicast Verbindungen.

Protocol Independent Multicast PIM lässt sich in 2 arten implementieren:

- MODE 1: Dense Mode (PIM-DM)
  - Annahme dass Group Mitglieder sich nah beieinander befinden
  - Router sind explizit Mitglied einer Gruppe, außer sie prunen
  - ”Data-driven construction of mcasttree”
- MODE 2: Sparse Mode (PIM-SM)
  - Annahme, dass Mitglieder weit verbreitet sind (z.B. über mehrere AS)
  - explizites joinen
  - erst wird ein geteilter core-based Tree erzeugt, der dann im laufenden Betrieb in einen Source Based Tree umgebaut wird.

Ein Konkreter Ablauf ist auf 6:33

6.3.3 Reliable Multicast

Für ein konkreten Usage in Multicast auf AS Ebene 6:35-6:39

7 Application Layer

Application sind prinzipiell Distributed. Innerhalb eines Host kommunizieren Applications mit dem OS-Kernel, der die Kommunikation organisiert. So eine Inter Process Communication (IPC) kann auch über einen anderen Host laufen, dann kommen jedoch Faktoren wie Delay, Jitter oder Packet Lost wieder ins Spiel. Was ist/kann bei diesem Process Distributed (sein)?

- Data
  - Ort der Daten gebunden an den Ort des Users, der Updates oder der Quelle der Daten
  - Aus administrationsgründen können Daten Distributed sein: central Access, database...

- Computation
  - Paralleles berechnen auf mehreren Hosts
  - Auf spezialisierter Hardware
  - Access zu restriktierten Daten
- User
  - Physische an verschiedenen Orten
  - User haben verschiedene Rollen

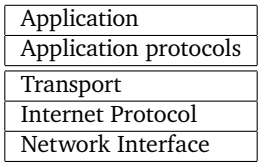
Wie auch auf anderen Layern gibt es beim A-Layer in RFC's deffinierte Protokolle, wie z.B. HTTPS, SFTP, SMTP. Diese beschreiben die z.B. Syntax und Semantik und sind für alle Quelloffen einsehbar. Es gibt jedoch auch proprietäre Protokolle, wie Skype oder MS Exchange, die nicht offen sind.

Client-Server Model Während in den jungen Jahren des Internets noch eher mit P2P Modellen gearbeitet wurde, ist heute das Client-Server Modell verbreiteter. Hierbei stellen Clients Anfragen bei zentralen, öffentlich erreichbaren Server, die den Clients dann Antworten senden. Mit diesem Modell fällt die ehemals starke dezentralität weg. Bei diesem Modell gibt es zwei Arten:

- Iterative Server
  - Der Server (A-Layer) handelt nur eine Anfrage gleichzeitig.
  - Kommen mehrere Anfragen, werden diese vom OS gequeued und nacheinander der Application serviert.
- Concurrent Servers
  - Die Hauptaufgabe ist es, Anfragen anzunehmen
  - Sobald eine Anfrage ankommt, wird ein Child-Process erschaffen, dessen Aufgabe es ist, die Anfrage zu beantworten
  - Der Main-Process kann in der Zeit, in der der Child-Process die Anfrage bearbeitet, neue Anfragen annehmen und zu jeder einen neuene Child-Process erschaffen
  - Hat der Child-Process die Anfrage bearbeitet und geantwortet, terminiert er

7.1 Socket Programming

Ein Modell zur Socket Programmierung ist die Berkeley Socket API, die es ermöglicht, Distributed Systems auf dem Application Layer zu implementieren. Die API wurde dabei hauptsächlich für UNIX Systeme entwickelt und wird dort eingesetzt. Die Socket API befindet sich zwischen Application protocols und transport: Ein Socket ist dabei die Kombination aus IP und Port. Eine Connection wird dabei



immer als Socket Paar (local/remote) identifiziert, wie schon im T-Layer beschrieben. Beispiel Socket Calls sind auf 7:14-7:16. Ein Prozess fragt dabei einen Socket beim OS an. der Port ist dabei meistens ein unpreviligierter, also größer 1024. Eine implementierung eines Python Echo Servers ist auf 7:17-7:19, eine Java Beispiel auf 7:20-7:28.

7.2 DNS

Wie schon erwähnt, werden Host mit Addressen angesteuert (32 Bit für IPv4, 128 für IPv6), doch möchte ein Menschlicher User einen Server erreichen, möchte er etwas leichter merkbares verwenden, eine domain aka. "Name". Ähnlich wie mit Telefon, braucht es dafür eine Art "Telefonbuch".
Im Internet werden Namen Hierarchich organisiert. Die wurzel aka. "root zone" bildet dabei ".", von der sich weiter Zonen sog. Top-Level-Domains abspalten, e.g. ".edu, .org, .com...". Diese können dann weiter deligiert werden. Es gibt einige wenige Root Server, die ihre IP Addressen per anycast teilen. Diesen sind die die Name-Server für die verschiedenen TLDs bekannt, in denen dann wieder Zone weiter deligiert werden. TLDs werden von verschiedenen Authoritäten maintained, denic z.B. ".de.", Educause ".edu.".
Um die Last zu verteilen, lassen sich auch mehrere Domain-Server parallel bereiben (primary/secondary).

Chaching and Resolver Damit nicht für jede Anfrage ein Domain Server höherer Hierarchie angefragt werden muss, können auch rekursive Chaching Server lokal betrieben werden. Diese sind der default Server für Geräte in lokalen Netzen und speichern aufgelöste Addressen für eine gewisse Zeit. Der Resolver ist dabei die Software auf den Hosts der Clients, die den Nameserver nach Addressen anfragen. Eine Nachricht hat dabei den folgenden Aufbau:

4	8	12	16	20	24	28	32
Identification				Flags and Codes			
Question Count				Answer Record Count			
Name Server (Auth Record) Count				Additional Record Count			
Questions							
Answers							
Authority							
Additional Information							

Wobei

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Q/R	OPCode				AA	TC	RD	RA	Zero			RespCode			

Dabei ist

- **Q/R** Query/Response Flag
- **OPCode** Operation Code
- **AA** Auth. Answer Flag
- **TC** Truncation Flag
- **RD** Recursion Desired Flag
- **RA** Recursion Available Flag
- **Zero** 3 reservierte Bits
- **Response Code**

Die verschiedenen Header fields übernehmen dabei folgende Aufgaben:

- **Identifier** 16 Bit Feld. Der Sender generiert einen Code, der dann vom DNS Server in den Response Teil kopiert wird, damit der Sender verifizieren kann, dass die Antwort vom richtigen Server kam.
- **Query/Response Flag** 0=Oqery 1=Response
- **Operation Code** Nür für Querys. Spezifiziert den Typ der Nachricht.
- **Response Code** Success=1 ansonsten Fehler Code
- **Authoritative Answer Flag (AA)** 1, wenn die Antwort Authoritative
- **Truncation Flag** Wenn 1, dann wurde die Nachricht getrunct
- **Recursion Desired** 1, wenn rekursiver Prozess gewünscht ist
- **Recursion Available** 1, wenn rekursion möglich ist

Es gibt diverse DNS Records, die meist verwendeten sind z.B. A/AAAA für Hostname, der Wert ist dabei die IPv4/v6 Addresse. NS Record ist die IP Des Authoritative Name Servers. MX ist der Mailserver für diese Domain und CNAME ist ein Alias für einen anderen Record.

7.3 HTTP

HTTP (Hypertext Transfer Protocol) ist ein Web Application Layer Protocol. Mit ihm können HTML, JPEG, Audio... Files übermittelt werden. HTTP 1.0 ist in RFC1945 festgehalten, HTTP2 in RFC7540 und wird inziwschen von fast 50% aller Webservern gesprochen und wird von den meisten Browsern untersützt. HTTP läuft idr. auf 80/TCP und ist stateless, d.h. der Server speichert keine Informationen über den Client. Während HTTP1.0 noch mit nonpersistent HTTP gearbeitet hat, d.h. nur ein Objekt über TCP versendet hat, arbeitet HTTP1.1 mit persistent HTTP, d.h. mehrere Objekte können über eine TCP Verbindung gesendet werden. Ein Beispielablauf für eine non-persistant HTTP Session mit einer Seite mit einem HTML File, dass 10 JPEG Files referiert ist auf 7:50-7:51.

Persistant vs. non-persistant HTTP Während nonpersistent HTTP für jedes Objekt eine TCP Connection aufbaut, was 2RTTs pro Connection sind, und der Browser oft mehrere Connections parallel aufmacht, wird die Connection nach dem Senden der Response erstmal aufrecht erhalten. Persistant HTTP kann außerdem mit oder ohne Pipeline implementiert werden. Ohne pipelining werden neue request erst gesendet, wenn die Antwort der vorherigen Request angekommen ist. In HTTP1.1 ist jedoch Persistant mit pipelining default Einstellung. Hier wird eine request direkt beim auftreffen des Objekts erstellt. In HTTP2 wurde pipelining durch multiplexing abgelöst.

HTTP Messages Es gibt zwei Arten von HTTP Messages, request und Response. Die Response Message ist Human-readable in ASCII formartiert:

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

und die Response Mesage:

```
HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 ......
Content-Length: 6821
Content-Type: text/htmldata

data data data data ...
```

Method Types Um Anfragen an den Server zu stellen, kann z.B. die Post Method verwendet werden, dort werden Anfragen im entity Body übermittelt. Alternativ ist die URL Methode, dort werden mittels GET die Anfragen in die URL geschrieben. Das kann z.B. so aussehen:

```
www.somesite.com/animalsearch?monkeys&banana
```

In HTTP1.0 gibt es die Mehtoden:

- GET
- POST
- HEAD
  - Asks server to omit the requested object in its response →only the http header (html header) is sent

Neben diesen gibt es in HTTP1.1 & HTTP2 noch

- PUT
  - Upload eines Files in der URL angebenen Addresse
- DELETE
  - Löschen des in der URL spezifizierten Files

HTTP Response Cat Es gibt diverse Status Codes für HTTP, die am häufigsten auftretenden sind z.B.

- 200: OK
- 301: Moved Permanently
- 400: Bad Request
- 404: Not Found
- 505: HTTP Version not Supported

Eine Liste mit Erklärung gibt es hier:

https://http.cat/

Cookies Cookies werden verwendet, um den State eines Users zu speichern, beispielweise mit einem Session-Cookie. Damit lassen sich Dinge wie Warenkörbe, Logins... ermöglichen. Jedoch lassen sich Nutzer über Cookies tracken.

7.4 Peer 2 Peer

Während das Internet mehr und mehr zentraler wird und Computation mehr auf weniger werdenden Server geschieht, setzen sich einige dafür ein, zum anarschistischen Kerngedanken eines dezentralen Internets zurück zu kehren. Eine möglichkeit dies zu implementieren sind peer2peer Verbindungen. Beispielsweise biem File Sharing angewendet wird eine Verbindung direkt zum anderen Host aufgebaut und Daten werden direkt ausgetauscht. Statt eines Zentralen Providers werden Files dort von mehreren anderen Hosts zusammengetragen. Andere Anwendungen bei denen Peer2Peer verwendet wird sind z.B. Blockchain, Video Streaming...

8 Referenzen

8.1 Quicktour

- TDD (Time divison duplex)
- TDM (Time Division Multiplexing)
- FDM (Frequenz Division Multiplexing)
- WDM (wavelength divisionmultiplexing)
- CDM (Code Divison Multiplexing)
- SDM (Code Divison Multiplexing)
- DS (Distributed Systems)
- AS (autonomous System
- CSS (communication subsystem)
- CN (computer networks)
- OSI (Open Systems Interconnection)
- CON (Connection Establishment)
- DAT (Data Exchange)
- DIS (Disconnect)
- PDU (Protocol Data Unit)
- PCI (Protocol Control Information)
- SDU (Service Data Unit)
- LAN (Local area network)

- LLC (Logical Link Control)
- MAC (Medium Access Control)
- CO Network (connection-orientated network)
- CL Network (connectionless network)
- e2e (End-to-end)

8.2 Routing

- NS (network layer services
- TS (transport service)
- TTL (Time to live)
- RCC (Routing Control Center)
- DVR (Distance Vector Routing)
- LSR (Link-State Routing)
- AS (autonomous System)
- RIP (Routing Information Protocol)
- OSPF (Open Shortest Path First)
- IGRP (Interior Gateway Routing Protocol)
- BGP (Border Gateway Protocol)
- IGP (Interior Gateway Protocol
- EGP (Exterior Gateway Protocol)
- DSDV (Destionation Sequenced Distance Vector)
- DSR (Dynamic Source Routing)

8.3 IP & Addressing

- VLSM (Variable Length Subnet Mask)
- DHCP (Dynamic Host Configuration Protocol)
- NAT (Network Address Translation)
- SLAAC (StatelessAddress Autoconfiguration)

8.4 Transport

- CEP (Sockets)
- COTS (connection-oriented Transport services)
- ARQ (Automatic Repeat reQuest)
- MSS (Maximum Segment Size)
- RTT (Round Trip time)
- AIMD (adaptive increase multiplicative decrease)