Samenvatting Infrastructure

Noeël Moeskops

November 1, 2017

Contents

1 Computer Networks and the Internet

1.1 What is the Internet

1.1.1 A Nuts-and-Bolts description

Het internet bestaat uit verschillende lagen de **kern** en de **hosts/end systems** Eind systemen zijn apparaten die de **application layer** protocol verstaan. Hier uit kan je verstaan dat het laptops, mobiels, computers, tv's, game consoles etc. Deze zijn met elkaar verbonden door middel van **communicaton links en packets switches** (coax cable, radio etc.) Al deze dingen hebben verschillende **transmission rate**. Deze worden altijd afgebeelt in *bits/s. Meestal zie je mb/s of kb/s. Om van bit naar byte te gaan moet je het getal gedeelt door 8 doen.
De meest voorkomende vorm van packet switchers zijn **routers en link-layer switches**. routers worden meer in de netwerk core gebruikt. Het pad wat een pakketje neemt van van host A naar host B te komen word een **route of path** genoemd.
Een **Internet Service Provider (ISP)** geeft je een ip adres en verbind je met het internet. Je hebt hier verschillende lagen in. Je hebt lokale ISP (laag 3) laag 2 ISPs en in de core zitten laag 1 ISP.
Alle applicaties op je computer die verbindeng willen maken met een andere eind-systeem maken gebruik van een **application protocol** voorbeelden hiervan zijn http, https, ftp, skype, nfs etc. De belangrijkte protocolen voor het internet zijn **Transmission Control Protocol (TCP)** en **Internet Protocol (IP)** Het internet model wat nu in gebruik is heet daarom ook het **TCP/IP** model.
Internet standaarden zijn super belangrijk om te kunnen communiseren tussen systemen. Deze worden vaak gemaakt door het **Internet Engineering Task Force (IETF)**. Openbare vrijstandaarden noem je **requests for comments (RFCs)** daar mag iedereen aan werken; implementeren en wijzigen. Of het word goed gekeurd is een ander verhaal.

1.1.2 A service Description

distributed applications zijn programma's die verdeelt zijn onder meer mensen dus games, sociale media, streaming services omdat ze meer system ding doen ofzo (pag 33.).
Een eindsysteem bind zichzelf aan een socket. een webserver bijvoorbeeld poort 80. Dit noem je een **socket interface**.

1.1.3 What is a protocol

Een protocol is een van te voren afgesproken manier om met elkaar te praten. Dit hebben computers nodig om succesvol met elkaar te kunnen communiseren. Elk protocol heeft zo haar eigen doel.

1.2 The Network Edge

Het internet bestaat uit **clients** en **host** een client is een systeem wat iets opvraagt van een host. En een host is een systeem wat informatie geeft aan een client op aanvraag. Een **data center** kan bestaan uit meerdere **end-systems/host** kan oplopen in de tonnen.

1.2.1 Access Network

Om met het internet te verbinden heb je natuurlijk ook een kabel nodig. Een **Digital subscriber line (DSL)** is zon soort kabel. ookal word DSL voor meer dan 85% gebruikt kan je veel sneller gaan met **fiber to the home (FTTH)**

1.2.2 Physical Media

fysieke media vallen in twee categorieen: **guided media en unguided media**. guided media is bedraad en unguided media in draad-loos.
Unshielded twisted pair (UTP) word vaak gebruikt binnen een gebouw. transfer rates gaan van 10 Mbps tot 10Gbps. Coax cabels zijn een guided **shared medium**, op een shared medium kunnen meerdere systemen aangesloten zijn met maar 1 kabel. terwijl de data niet shared is.
In communicatie worden er twee verschillende soorten satalite gebruikt. **Geostationary satellites** deze blijven op een plek boven de aarde zweven op 36.000 KM. En **low-eath orbiting (LEO) satallites** deze zijn een stuk dichterbij en staan ook nooit stil.

1.3 The Network Core

1.3.1 Packet Switching

Wanneer je een bericht stuurt over het internet word hij in kleine stukjes gehakt, dit noem je **packets** dit pakketje gaat door allemaal **packet switches** (routers en link-layer switches).
De meeste packetswitchers zijn **store-and-forward transmission** dit betekent dat ze eerst wachten totdat alle stukjes van een paket binnen is voordat hij het door verstuurt. Voor iedere link aan een router heb je ook een **output buffer/output queue** dit is een stukje ruimte waar de router pakketjes stopt voordat hij ze versuurt. Een pakketje kan dus een queuing delay oplopen. Wanneer een queue vol zit kan het pakketje worden gedropt en ontstaat er dus **packet loss**.
Er zijn ook verschillende **routing protocols** Voor routers om met elkaar te praten.

1.3.2 Circuit Switching

word niet behandeld. (pag. 55)

1.3.3 A Network of Networks

Omdat er meerdere lagen van ISPs bestaan word degene waar jij in contact mee komt de access ISP genoemd en de bovenste laag de global ISP. Je zou ook kunnen stellen dat de access ISP een **customer** is van global ISP en dat hij weer de **provider** is.
Een **PoP** is een verzameling van routers op een lokatie (op een niet access ISP level) die word gebruikt om meerdere access ISP te verbinden naar local of global ISP. Een super switch eigenlijk maar dan van routers. ISPs kunnen ook direct met elkaar contact leggen zonder hulp van een local of global ISP dit bespaart kosten en noem je **peer**.
Een random bedrijf kan ook een **Internet Exchange Point (IXP)** opzetten. Dit is wel letterlijk een mega switch waar iedereen verbinding mee kan maken om zo internet te delen. Er bestaan ook **content-provider networks** Dit zijn mega bedrijven zoals Google, facebook, Amazon etc. Die als tier 1 (global) ISP spelen.

1.4 Delay, Loss And Throughput in Packet-Switched Networks

Wanneer een pakketje bij een node is kan het vertraging oplopen. Dit kan **nodal processing delay, gueeuing dealy, transmission delay of propagation delay** zijn. Al deze dingen samen noemen we **total nodal delay**.

processing delay is de tijd dat het duurt voor een node om naar de header van een pakketje te kijken. en om te kijken waar het naartoe moet.

queuing delay is de tijd dat het pakketje in de rij moet wachten tot dat het verstuurt kan worden.

transmission delay Dit is de tijd dat het duurt om alle pakketjes op de link te pushen. Dit is niet de tijd van een node naar de andere.

propagation delay Dit is de tijd dat het duurt om een pakketje van een node naar de anderen te sturen.
traffic intensity word bepaald door $paketgrote * aantalpakketjes/transmissionrate$ (La/R). Als een queue van een node is kan hij het pakketje **droppen** dan is de data verloren, hij kan er ook voor kiezen om een ander pakketje te droppen.
Instantaneous throughput is de throughput in een moment. **average throughput** is de gemiddelde throughput.

1.5 Protocol Layers and Their Service Models

1.5.1 Layerd Architecture

Protocol Layering Het idee van protocol layering is dat je al je dingen in andere lagen zet. Iedere laag heeft dan end-points waarmee het een service bied aan de laag erboven. en een end-point waarmee het service vraagt aan de laag eronder. De inhoud van de layer kan veranderen als de end-points maar hetzelfde blijven. Dit zorgt voor een schaalbaal systeem.

Applicatie layer Dit is de layer die drijd op een eind-systeem protocolen zoals HTTP, FPS, Skype, IMAP etc. dit gaat oneindig door. data word hier een **message** genoemd

Transport Layer deze laag zorgt ervoor dat de data van de applicatie laag uit je computer gaat en dat inkomende data naar de juiste applicatie worden geleid. Deze laag kent maar twee protocollen: **TCP & UDP**. Data word hiet een **segment** genoemd

Network Layer Deze laagt zorgt voor de navigatie van node naar node. Dit kent ook maar een aantal protocollen: **IP, en not iets?** //TODO: protocollen. Data word hier een **datagram** genoemd.

Link layer dit is de laag wat letterlijk de data van node A naar node B brengt. Het gebruikt hiervoor de physical layer. Dit doet het met het **PPP** protocol. Data op deze laag worden **frames** genoemd.

physical layer dit is letterlijk een ethernet kabel of wi-fi verbinding.

1.5.2 Encapsulation

Wanneer je data verstuurt over het internet doe je aan encapsulation. Een beetje als een russiche pop met meerdere lagen. Een Message wordt in een segment gestopt. waarbij de transport layer alleen kijkt naar de header van de message. Hij voegt dan wat nuttige data toe aan de segment header en geeft het dan aan de network laag. die maakt er een datagram van en kijkt weer naar de header van de segment. Zo gaat het door tot aan de Link laag. Wanneer de Frame bij een switch komt word hij niet uitgepakt omdat deze ook alleen gebruik maakt van de link layer. Een router werkt echter op de netwerk laag, het pakketje word dan uitgepakt naar een datagram wanneer het bij een router is en weer ingepakt als de router het weer verstuurt. Wanneer het bij zijn eindbestemming aankomt word het weer helemaal uitgepakt tot een Message.

1.6 Network Under Attack

slechte software word **malware** genoemd. Je hebt hierin meerde varianten: **Virusesen** die je zelf op je computer werkt en **worms** die een via een lek in je systeem komen. Veel malware in **self-replicating** het versprijd zichzelf.
deniel-of-service (DoS) is dat je heel veel verkeer stuurt naar een host om zo zijn toegang tot het internet te blokeren. Dit kan op drie manieren *vulnerability attack* is een aanval speciviek gemaakt om een lek te gebruiken om toegang te blokeren. *bandwidth flooding* Met deze aanval stuur/vraag je mega veel en vaak data naar/van een host. *connection flooding* is dat je alle sockets van een host bezet houd door een (half) open connectie te behouden. Je kan ook een **distributed DoS (DDos)** uitvoeren doormiddel van een **botnet**.
Je kan ook op een lokaal netwerk **packets sniffen** dan inspecteer je alle pakketjes op een netwerk. Je kan ook je **ip spoofen** dan doe je net alsof je een andere IP hebt.

2 Application Layer

Dit hoofdstuk gaat over de Application Layer. waar eigenlijk al je netwerk programma's opdraaien.

2.1 Principles of Network Applications

Het principe van netwerk layers is dat je je maar meestal zorgen hoeft te maken over 1 of 2 lagen van het netwerk systeem. Wanneer je een applicatie schijft voor een eind-systeem hoef je dus alleen maar zorgen te maken over hoe de applicatie layer jou programma implementeert.

2.1.1 Network Application Architectures

Er zijn twee verschillende application netwerk architecturen; **P2P** (peer-to-peer) en **client-server** architectuur. Je bent niet gelimiteerd aan deze twee architecturen maar het zijn de enige die op dit moment bestaan. (je kan zelf iets nieuws verzinnen als je genoeg vrije tijd hebt).

client-server

Een client-server is de meest simpele en meest traditionele architectuur. Het gebruikt twee system een *client* en *server* De server moet op een statische plek staan en 24/7 beschikbaar zijn. De client daarin tegen kan dynamiche zijn (van IP verwisselen) en uit en aan gaan wanneer gewenst. De server neemt geen dienst af van de client.

Peer-to-peer

Een peer-to-peer verbinding word onder anderen gebruikt bij Torrents en video gesprekken. De verbinding word onderling gedaan zonder centrale server. Dit systeem schaalt ook veel beter omdat wanneer iedereen een bestand wilt downloaden het ook door meer mensen word geupload.

2.1.2 Processes Communicating

Wanneer een **process** wilt communiceren met een ander process op het Internet heeft het 2 dingen nodig, het adres van de het anderen end-systeem en **Sockits** of **ports** zij

2.2 The Web and HTTP

Tot 1990 werd het internet grotendeels door onderzoekers en scholieren gebruikt. Weinig mensen wisten er nog van. Later in 1994 werd de eerste webbrowser ontwikkeld en daardoor werd het internet onder de het normale volk ook ontamd.

2.2.1 Overview of HTTP

Hyper Text Transer Protocol (HTTP) word gebruikt om documenten (objecten) over het internet te versturen. Wanneer een object (index.html) verwijst naar een ander object (favicon.ico) word deze ook opgehaald. HTTP maakt gebruik van TCP en word beschouwd als een **stateless protocol** dat houd in dat de server geen data bijhoud van haar clients. Wanneer een client 4x hetzelfde bestand zou aanvragen zal de http server de 4e aanvraag hetzelfde behandelen als de eerste.

2.2.2 Non-Persistent and Persistent Connections

HTTP kent twee soorten verbindingen: Non-Persistent en Persistent.

Non-Persistent

Wanneer een client een object aanvraagt van een HTTP server gaan ze eerste een **three-way-handshake** doen. De client vraagt aan de server om een TCP verbinden optestellen en de server antwoord vervolgends met *OK*. Daarna vraagt de client een object aan (bijv. */mydir/index.html*) De server antwoord dan met het bestand *index.html* en de verbinden word verbroken.

De client leest het bestand en ziet dat hij nog 10 plaatjes moet laden. Hij gaat dan weer een three-way-handshake doen en het hele verhaal begint weer opnieuw.

Round-Trip Time (RTT)

RTT is het het tijd wat het duurt voor een pakketje heen en terug te sturen van client naar een server. Propegation delay heen en terug. In het vorige voorbeeld zijn er twee RTT. De eerste is wanneer de client connectie aanvraagt aan de server en antwoord krijgt. De twee is het bestand wat hij vraagt en ook krijgt van de server. Voor elke extra object krijg je bij een non-persisten verbinding **2** RTT's (een om verbinden vast te leggen, en een om het bestand te krijgen.)

Persistent

Een persistent verbinden is veel zuiniger met data dan een Non-persistent verbinden. Want het houd de TCP connectie open tot een bepaalde timeout tijd. Dus in ons voorbeeld met 1 html pagina en 10 plaatjes heb je dus maar 12 RTT's op een persistent verbinding (1 RTT voor TCP verbinden + html bestand + (plaatje * 10)) In plaatjs van 11 * 2 = 22 RTT's bij een Non-persistent verbinding.

2.2.3 HTTP Message Format

Er zijn twee verschillende HTTP bericht formaten, een request en een response. Hieronder volgt een voorbeeld van een request:

```
GET /tools/index.html HTTP/1.1
Host:  www.noeel.nl
Connection:  close
User-agent:  Mozilla/5.0
Accept-language:  nl
```

De request is geschreven in ASCII. De eerste zin is de **request line** de lijnen die daarop volgen zijn de **header lines**. Daarna komt er een wit-regel gevolgt door de **Entity body** maar die is nu leeg omdat het een request is en geen response of `POST`

Request line

De request kan bestaan uit een `GET, POST, HEAD, PUT` of `DELETE`

`GET` request een bestand, `POST` stuur data naar een server (de content staat dan in de body) `HEAD` voor debugging, vraag alleen de head van een response op. `PUT` plaats een bestand en `DELETE` verwijder een bestand. Het word daarna gevolgt door de locatie en de HTTP versie.

Header lines

`Host:` verklaard de host waar de request naar toe moet gaan, is nodig voor proxy's `Connection:` of het een persistant (`open`) of non-persistant (`close`). Data zoals `User-agent` en `Accept-language` worden gebruikt om verschillende data weer te geven per taal/browser.

HTTP Response

```
HTTP/1.1 200 OK
Connection:  close
Date:  Tue, 18 Aug 2015 15:44:04 GMT
Server:  Apache/2.2.3 (CentOS)
Last-Modified:  Tue, 18 Aug 2015 05:44:04 GMT
Content-Lenght:  6532
Content-Type:  text/html

data data data ....
```

In de Status line staat de HTTP versie gevolgt door de statis code en statis zin. (200 betekent dat alles goed is) `Connection:  close` betekent dat de server de TCP verbinding zal sluiten hierna. `Date` De datum en tijd dat de server het bestand van het file system heeft gekregen. `Server:` Allemaal server data, welke http server hij draaid, welke versie en welk OS. `Last-Modified:` handig voor caching enzo. `Content-Lenght` hoelang is de data en `Content-Type` wat voor soort data het is.

2.2.4 User-Server Interaction: Cookies

Omdat HTTP stateless is weet de server niet of de client vaker gebruikt maakt van zijn diensten. Daarom bestaan er Cookies, een Cookie is een string die de server kan zetten met de HTTP header: `Set-cookie: myCookieValue` Wanneer de client een nieuwe request maakt naar dezelfde server stuurt hij altijd in zijn HTTP header (`Cookie: myCookieValue`) zodat de server dit kan processen en client speceviek diensten aan kan bieden.

2.2.5 Web Caching

Je kan ook een caching server hebben. Dit is een server dat een kopie maakt van een bestaande server en dit op een andre plek host. Dit kan handig zijn als de normale server een hoge ping heeft en dus een lage response tijd heeft. Om de caching server te berijken moet je zelf in je browser (of andere applicatie) instellen dat je met de caching server verbinding wilt maken inplaats van de bestaande server.
Wanneer je een HTTP GET request stuurt naar een caching server kijkt hij of hij het bestand heeft en stuurt deze dan naar jou toe. Wanneer het bestand ontbreekt vraagt hij het zelf op bij de orginele server en stuurt het dan naar jou toe en slaat het ook meteen op.

Conditional GET

Het grote nadeel van een caching server is dat hij een out-of-date bestand kan hosten. Om dat te voorkomen checked een caching server eens in de paar weken of er geen nieuwer bestand bestaat. Dat gaat omgeveer zo: /newline Eerst vraagt de caching server het orginele bestand aan

```
GET /img/logo.png HTTP/1.1
Host: www.mylogoImages.com
```

De orginele server geeft een respone:

```
HTTP/1.1 200 OK
Date: Sat, 3 Oct 2015 15:39:29
Server: Apache/1.3.0 (Unix)
Last-Modified: Wed, 9 Sep 2015 09:23:24
Content-Type: image/png

data data data ...
```

Wanneer er een bepaalde tijd voorbij is en de caching server is bang dat zijn bestand out-of-date is checkt hij bij de server of er een nieuwere versie beschikbaar is:

```
GET /img/logo.png HTTP/1.1
Host: www.mylogoImages.com
If-modified-since: Wed, 9 Sep 2015 09:23:24
```

Wanneer er geen nieuwe versie is beantwoord de server met een 304:

```
HTTP/1.1 304 Not Modified
Date:  Sat, 10 Oct 2015 15:39:29
Server:  Apache/1.3.0 (Unix)
```

2.3 Electronic Mail in the Internet

E-mail bestaat uit drie delen: **user agents** (het client programma op het eindsysteem van de gebruiker), **mail servers** (de servers die mails ontvangen en versturen) deze hebben zowel een client als server rol en het protocol om mails te versturen **Simple Mail Transfer Protocol (SMTP)**

2.3.1 SMTP

//TODO

2.3.2 Comparison with HTTP

2.3.3 Mail Message Formats

2.3.4 Mail Access Protocols

2.4 DNS-The Internet's Directory Service

Servers hebben vaak twee adresssen waarmee je met ze kan verbinden: hostname *www.noeel.nl* en het ip adress *185.182.56.20*. Computers willen met het ip addres verbinden omdat dat ook echt iets betekent. Maar voor mensen is dat moeilijk te onthouden. Daarom bestaan er hostnamen die verwijzen naar het ip-adres zodat je die in kan typen in je webbrowser.

2.4.1 Services Provided by DNS

Wanneer jij verbinding wil maken met een hostname verstuurd je computer een verzoek naar een DNS server om het ip addres op te zoeken van de hostname. De DNS server geeft dan het ipaddres en je kan verbinding maken.
Een DNS server bied een aantal services:

Host aliasing Een sub domain (*sub1.sub2.domain.org*) word een **canonical hostname** genoemd deze verwijzen naar *www.domain.org* of *domain.org*

Mail server aliasing Je kan van elk domain naam ook een MX-record opvragen, dit is het addres van de mail server.

load distribution Je kan ook meerdere records van een soort (MX,A etc.) opgeven bij een DNS server. Hij kiest er dan per request een random uit zodat niet een server van je word belast.

2.4.2 Overview of How DNS Works

Er bestaan 4 verschillende DNS servers:

Root DNS servers Dit zijn servers die alleen een lijst hebben met TLD DNS servers. Wanneer jij een request maakt naar *www.google.nl* verwijst deze server je door naar een NL TLD DNS server

Top-Level domain server elk top-level domein naam (nl, com, org, de etc.) heeft haar eigen TLD DNS server. In deze server staan alle **Authoritative DNS servers** van elk domain naam onder dat TLD. Wanneer je een request doet voor het ip van google.nl geeft hij je de Authoritative DNS servers van google.nl.

Authoritative DNS servers Dit zijn de DNS servers die ook daatwerkelijk het antwoord voor je hebben. deze hebben alle records van een domain naam en geven je ook het ipaddres van je bestemming.

local DNS server Dit zijn vaak DNS servers van je ISP (Ziggo bijv.) Deze hebben een lijst van alle Root DNS servers en vragen aan hen het ip addres van je request. Hier maak jij dus ook verbinding mee.

DNS request

Wanneer jij een vraag naar een DNS server doet voor het ophalen van een ip address kan dat op twee verschillende manier gaan.

Various DNS servers

- Een eind-systeem maakt doet een DNS request voor *www.noeel.nl* naar een locale DNS server
- locale server doet een request naar de root server
- de root server leest het *.nl* gedeelte en returnt een adres van een NL TDL DNS server
- de locale server ontvangt de request en stuurt een DNS request voor *www.noeel.nl* naar de NL TDL DNS server
- de TLD server leest het bericht en returnt de authoritative DNS server voor noeel.nl
- de local server stuurt een bericht naar de authoritative DNS server en krijgt de record die hij verwachte

- de local server stuurt het ip-adres door naar het eind-systeem dat het eerder aanvroeg.
- profit!

Recursive queries in DNS Dit werkt op een anderen manier als de varios DNS server variant. inplaats van dat de server steeds een adres returned van een ander DNS server gaat hij zelf vragen aan die server vor het adres. Maar dit komt minder vaak voor.

DNS servers cashen ook al hun resultaten om verkeer te verminderen, en om sneller te zijn. Meestal word iets voor 2 dagen gecached. Maar als web-host kan je dit ook korter of langer instellen.

2.4.3 DNS Records and Messages

2.5 Peer-to-Peer File Distribution

3 Transport Layer

3.1 introduction and Transport-Layer Services

Een transport layer protocol zorgt voor logiche communicatie tussen applicaties van verschillende hosts.

3.1.1 Relationship between Transport and Network Layers

3.1.2 Overview of the Transport Layer in the Internet

Op de transport layer heb je maar twee protocollen **UDP (User Datagram Protocol) en TCP (Transmission Control Protocol)**. UDP levert geen vertrouwlijke data overdracht. Wanneer dat niet aankomt word het niet gecontroleerd. TCP doet dat wel. UDP heeft daardoor wel een kleinere header en kan data versturen zonder een handshake waardoor het bij sommige gevallen sneller kan zijn.
IP/TCP word als een **best-effort delivery service en als unreliable service** gezien. Dit houd in dat het zijn best doet maar geen beloftes doet, je data kan dus niet aankomen of vermist raken. TCP bied wel **reliable data transfer** door te controleren of aan pakketje wel in aangekomen. TCP maakt ook gebruikt van flow control, acknowledgments en onderanderen **congestion control**. Congestion control is service die TCP bied aan het internet om er voor te zorgen dat haar pakketjes niet het hele internet overspoelen. Wanneer TCP merkt dat een node maar 10 pakketjes per seconde kan handelen en hij er zelf 100 per seconde stuurt, zal TCP dit terug schroeven om zo het netwerk niet onnodig te belasten.

3.2 Multiplexing And Demultiplexing

Een applicatie bind zich aan een socket, dit is waarmee hij communiseert met de transport laag en dus verbonden is met het internet.
Wanneer de transport laag een segment van de netwerk laag ontvangt inspecteerd hij deze en stuurt het door naar de bijbehoorende poort. Dit proces noemen we **demultiplexing**. Andersom van applicatie naar network heet **multiplexing**.
TCP en UDP hebben beide twee header fields gemeen: **source port, destination port**. Een port is 16bit en kan dus van 0 tm 65535 zijn. van 0 tot 1023 worden **well-known port numbers** gemoend. Deze zijn eigenelijk al bezet door ouroude internet applicaties. TCP laat zichzelf identifiseren door middel van een four-tuple: source ip, source port, destination ip, dest. port.

3.3 Connectionless Transport: UDP

voordelen van UDP:

Finer application-level control over what data is sent, and when Omdat de header van udp klein is stuur je geen dingen mee waarvan je niet weet

wat het is. Omdat udp ook bijna geen controle uitvoert op het pakketje word het bijna rouw doorgestuurt naar de netwerk laag.

no connection establishment Met UDP kan je meteen beginnen met versturen van data, het is niet nodig om eerst een **three-way hand-shake** te doen en/of om connecties open te houden. Wat bij TCP wel het geval is.

No connection state UDP is stateless, je hoeft niet voor het verzenden connectie te maken.

Small packet header overhead hier hebben we het al over gehad.

3.3.1 UDP segment structure

3.3.2 UDP checksum

udp heeft net als tcp ook checksums maar ze zijn een heel stuk toleranter. Wanneer udp merkt dat een pakketje niet klopt kan hij hem alsnog doorsturen (met flag) naar de applicatie laag.

3.4 Principles of Reliable Data Transfer

word niet behandelt. (pag 234)

3.5 Connection-Oriented Transport TCP

3.5.1 The TCP Connection

TCP is **connection-oriented** dat wil zeggen dat het eerst een actieve connectie moet hebben met een andere host voordat het pakketjes kan ontvangen of versturen. Dit noemen we een hand-shake. De client vraagt aan de host of hij verbinding mag maken, de host geeft antwoord dat hij het goed vind en de verbinding opent en de client reageert weer dat hij het bericht van de host heeft ontvangen.
TCP is een **full-duplex service** dat houd in dat hij dingen kan ontvangen en versturen tegelijkertijd, in tegenstelling to een walkie-talkie die maar half-duplex is, over. TCP is ook **point-to-point** dat wil zeggen dat er maar 1 ontvangen kan zijn en maar 1 verzender.
Wanneer TCP data krijgt van de applicatie laag doet hij dat in een **send-buffer**. En van tijd tot tijd stuurt hij dit door naar de netwerk-laag. De groote van de hoeveelheid dat hij pakt word bepaald door de **maximum segment size (MSS)** voor de netwerk laag. Deze word weer bepaald door de **maximum transmission unit (MTU)** dit is de maximale frame grote op de link laag. Dit is allemaal plus alle headers van de verschillende lagen en de data.

3.5.2 TCP segment Structure

tpc heeft net als udp **source en destination port numbers** in zijn header staan. Hij heeft ook een **checksum**.

- **sequence number field en acknowledgment numberfield** (beide 32bit) word gebruikt voor het verfiëren van het pakketje. Elk pakketje dat verstuurt word met TCP heeft een **ack** en **seq** nummer daarmaa kan de ontvanger controleren of hij alle pakketjes wel ontvangen heeft. Als een host een pakketje stuurt met seq: 1 van 8 bytes, stuurt de ontvanger een pakketje terug met ack: 10 ($1 + 8 = 9$). Daarma wil hij zeggen dat hij alles tot 9 heeft ontvangen en klaar is voor pakketje 10.
- **receive window** (16bit) Dit veld word gebruikt voor flow-control. Hierin staat hoeveel bytes de ontvanger kan accepteren.
- **header length field** (4bit) In deze header staat hoegroot de header is.
- **options field** In dit veld onderhandelen de twee eind-systemen over de MSS.
- **flag field** Dit zijn allemaal mini flags van 6bit
  - **ACK** de ACK waar we het eerder overhadden.
  - **SYN FIN** worden gebruikt voor het opzetten en afbreken van de connectie
  - **PSH** als deze aan staat betekent het dat de ontvanger het bericht meteen moet pushen naar de applicatie laag
  - **URG** geef aan dat de data "urgent" is. Dit wijst naar een **urgent data pointer field** (16bit), maar word nauwlijks gebruikt.

De **sequance van een segment** is belangrijk om de goed volgorde aan te houden. Pakketjes hoeven niet altijd op dezelfde volgorde te worden ontvangen.

3.5.3 Round-Trip Time Estimation and Timeout

word niet behandeld (pag 269)

3.5.4 Reliable Data Transfer

word niet behandeld (pag 272)

3.5.5 Flow Control

TCP bied **flow-control service**. Dit is een speed-matching systeem zodat de rate van data wat je verzend even hoog is als de bottleneck van je netwerk. Dit kan op meerde methodes. Een van die methodes noem je **congestion control** als je de snelheid op een "file" in het netwerk.

receive window word gebruikt door de verzender om te kijken hoeveel vrije buffer er nog is om dingen te verzenden.

4 The Network Layer: Data Plane