Samenvatting Infrastructure

Noeël Moeskops

September 17, 2017

Contents

1 Computer Networks and the Internet

TODO

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)
```