Samenvatting Webdevelopment

D.E.R.P.

2 juni 2019

Inhoudsopgave

Inleiding

Deze samenvatting is gemaakt voor De Examen Repo Plaats (DERP), een github organisatie die te vinden is op `http://github.ugent.be/derp`, die dergelijke samenvattingen bevat. Het doel van DERP is om een plaats te hebben voor de LaTeX broncode van samenvattingen, zodat deze altijd up to date kunnen gehouden worden. Deze specifieke samenvatting is te vinden in de releases van `http://github.ugent.be/derp/itech`.

Als je een fout tegenkomt, of bepaalde stukken wilt uitbreiden, kan dat altijd door de repo te `forken` en een `pull request` aan te maken met jouw aanpassingen. Na een review kan jouw contributie deel uitmaken van het DERP project en zo toekomstige studentjes uit de nood helpen.

Contributors

Deze samenvatting is FOSS met dank aan:

- Arne Gevaert (2017)
- Mats Myncke (2016)
- Eloïse Piret (2016)
- Lorin Werthen-Brabants (2016)

Hoofdstuk 1

Architectuur van het Web

1.1 Overzicht

Internet Globaal communicatie netwerk dat toestellen verbindt. Technologieën: TCP, IP, DNS, ...

Web Informatieruimte bovenop het Internet. Technologieën: HTTP, HTML, URL, ...

1.1.1 Web voor mensen

Technisch perspectief

Een combinatie van factoren zorgt voor de enorme schaalbaarheid van het Web:

- **Client-server architectuur** zorgt ervoor dat niemand afhankelijk is van elkaar (geen application state)
- **Decentralisatie** wordt bereikt door links in slechts één richting aan te maken (*Individual links are allowed to break so the entire Web does not*)
- **Systeem-onafhankelijkheid** zorgt voor nooit eerder geziene compatibiliteit (zie Lynx)

Sociale impact

Bedrijven groeiden samen met het Web. Zo begon Pizza Hut online bestellingen aan te nemen in 1994. Wikipedia begon in 2001, en biedt gratis, soms onbetrouwbare, kennis aan.

Het Web is **democratisch**. Iedereen kan informatie lezen en schrijven.

Sociale media creëert vaak een **"filter bubble"**; je ziet meer dingen die je al gezien hebt, en ziet vaker dingen waar je al voor staat.

1.1.2 Web voor machines

Het Web is **gedecentraliseerd**. Dit heeft als gevolg dat niemand een volledig overzicht heeft van het Web. **Crawlers** van zoekmachines (zoals Google, Bing, Baidu, ...) doorzoeken het Web en maken een index van de inhoud.

De ranking van een pagina op een zoekmachine is van groot belang, dit heeft nieuwe jobs gecreëerd: **Search Engine Optimization (SEO)**.

Ook de zoekopdrachten zelf vormen een waardevolle databron: ze laten toe om bepaalde **trends te spotten.**

Het **Semantisch Web** is een laag bovenop het bestaande web. Het heeft als doel data met elkaar te **linken**, en zo het opzoeken van data over verschillende sites heen te vergemakkelijken.

1.2 Core Web Standaarden

Het web bestaat uit drie verschillende — maar verbonden — uitvindingen. Deze zijn **URL**, **HTML** en **HTTP**:

- **URL** definieert op unieke wijze een resource
- **HTTP** laat ons toe om een voorstelling van een resource te verkrijgen via een URL
- **HTML** can een resource afbeelden, en linken naar andere resources via hun URL

1.2.1 URL

Een **Web URL** definieert en situeert een resource op unieke wijze, eender waar in het universum.

Een typische **HTTP URL** ziet er als volgt uit.

`http://<host>/<path>?<search>#<fragment>`

1.2.2 HTTP

HTTP normaliseert hoe clients een request sturen voor een voorstelling van een resource via zijn **URL**. HTTP normaliseert ook hoe servers antwoorden met een response die een voorstelling kan bevatten.
Na het **resolven** van een server's IP adres, kan de client een HTTP request sturen. Deze start met een **request line**, die de **methode**, **request URL pad** en **HTTP versie** bevat. Ook kan de request een aantal **header fields** bevatten. Deze kunnen zijn: Host, Accept, User-Agent, ...
De response kan ook een body hebben (als de methode het toestaat).

De vijf methoden die HTTP toelaat zijn:

- **GET** draagt een voorstelling over
- **HEAD** draagt enkel de status en headers over
- **POST** voert een resource-specifieke operatie uit
- **PUT** vervangt voorstellingen
- **DELETE** verwijdert voorstellingen

Een HTTP methode is **veilig (safe)** als ze alleen-lezen is. **GET** en **HEAD** zijn zulke methoden.
Een HTTP methode is **idempotent** als herhalingen het resultaat niet veranderen. Alle veilige methoden zijn idempotente methoden, alsook **PUT** en **DELETE**.
De client stuurt de **hostname** door, zodat een server meerdere websites kan hosten. Hierdoor is er *geen* one-to-one mapping tussen IP adressen en domeinen. Wanneer een server een request krijgt, genereert die een response. Deze bevat een status line, header fields en een body.

HTTP heeft 5 categorieën van status codes die een indicator zijn van hoe de request behandeld werd:

- **100–199** info — de client mag door gaan
- **200–299** succes — de request werd aanvaard
- **300–399** redirection — verdere actie is nodig
- **400–499** client error — de request is ongeldig
- **500–599** server error — de server kan de request niet verwerken

1.2.3 HTML

HTML is een markup taal die de structuur van documenten vastlegt. HTML verdeelt een document in elementen, aangeduid door opening en closing tags. Merk op dat HTML gebruikt dient te worden voor het structureren van de pagina (*markup*) en **niet** voor het opmaken ervan (*makeup*). *Makeup-only*-elementen (zoals `<font>`) werden dan ook verwijderd uit HTML5.

1.3 Web Architectuur

1.3.1 Clients

Het web ondersteunt vele clients. Interactieve browsers (Firefox, Chrome, Opera, ...), applicaties (Web Apps, mobiele applicaties), crawlers, embedded toestellen en sensoren (Web of Things). Alle clients moeten dus een aantal core technologieën ondersteunen, zoals **TCP/IP**, **DNS**, **HTTP** en een of meer voorstellingsformaten, zoals **HTML** of **JSON**. Browsers bieden een interactieve omgeving aan om websites te bekijken. Ze renderen HTML-elementen als interactieve controls, hebben styles, media en scripts.

Een script op een webpagina kan HTTP requests maken, waarop de server doorgaans antwoordt met **JSON** of **XML**. Zo kunnen desktop- en mobiele applicaties soortgelijke HTTP requests maken en gebruiken.

Crawlers extraheren, verwerken en indexeren text. Ze analyseren ook gestructureerde annotaties (Google's Rich Snippets). Gebruik makende van links ontdekken ze andere paginas.

1.3.2 Servers

Het **HTTP** protocol geeft *geen* betekenis aan URL-paden en query strings.

Er zijn verschillende soorten servers en implementaties:

File servers

Een statische **file server** beeldt HTTP URLs af op interne bestand URLs.

Applicatieserver

Een **applicatieserver** gebruikt server-side code om pagina's on demand te genereren. De request wordt verwerkt door een **applicatieframework**. Vervolgens reageert de server op bepaalde URLs of patronen, waardoor er responses gegenereerd worden gebruik makende van templates. **Ruby on Rails** is een voorbeeld van een applicatieframework.

Reverse proxy

Elke machine kan maar *één* poort 80 open hebben. Een **reverse proxy** laat toe om vele servers te draaien. Om dit te realiseren, laat men de verschillende servers op interne poorten draaien. Vervolgens wordt een reverse proxy (bv. NGINX) gestart op poort 80, die binnenkomende requests doorstuurt naar de interne poort van de gepaste server.

1.3.3 Intermediaries

Er zijn vele **intermediaries** tussen een client en een server. **HTTP** zorgt ervoor dat intermediaries transparant zijn, gebruik makende van verschillende headers. Verschillende **GET** requests kunnen **gecached** worden, omdat GET veilig is. Indien echter een onveilige request (zoals POST) gebruikt wordt, mag een volgende GET request niet uit de cache gelezen worden.

Intermediaries hebben verschillende rollen:

- **caching** om prestaties en beschikbaarheid te verbeteren
- **veiligheid** om authorizatie en authenticatie af te handelen
- **routing** om door te verwijzen naar de juiste server
- **load balancing** om de last over servers te verdelen
- **anonymizing** om identificatie of logging te omzeilen

1.4 Ondersteunende technologieën

1.4.1 HTTP/2

Het Web wordt tegenwoordig op een heel andere manier gebruikt dan wanneer HTTP voor het eerst ontworpen werd. **HTTP/2** is een update van het HTTP-protocol die een aantal belangrijk bottlenecks probeert weg te werken. Het is een binair protocol dat 10 verschillende types van frames definieert. Het gebruik van een binair protocol zorgt ervoor dat deze frames makkelijker opgebouwd kunnen worden. Daarnaast wordt multiplexing gebruikt om meerdere objecten over dezelfde verbinding te versturen en zo de latency-bottleneck weg te werken.

1.4.2 Client-side technologieën

Cookies

Cookies laten toe om informatie te behouden over meerdere HTTP requests. HTTP is per design een stateless protocol, om schaalbaarheid te verhogen. Een cookie is een **key/value paar** gegenereerd door de server en verstuurd door de client bij opeenvolgende requests. Hiervoor moet de server een `Set-Cookie` header toevoegen aan responses.

CSS

Cascading Style Sheets leggen op declaratieve wijze vast hoe elementen er uit moeten zien. Een CSS document bestaat uit regels, die gegroepeerd kunnen worden in scopes. Een regel start met een **selector** die aangeeft welke elementen de regel moeten volgen. De body van de regel bestaat uit key/value pairs van een eigenschapsnaam en zijn gewenste instelling.

1.4.3 Veiligheid

HTTPS vormt een veilige uitbreiding van het HTTP-protocol, door HTTP over TLS te gebruiken (in plaats van over TCP). De voorganger van TLS is SSL, de beiden worden veel verward.

Code injection is het uitvoeren van (kwaadaardige) code op de server. Dit wordt veroorzaakt doordat input van de client niet deftig wordt gecontroleerd aan de serverzijde. SQL injection is een bekend voorbeeld.

Cross-site scripting is het uitvoeren van (kwaadaardige) code op de client. Een typisch voorbeeld hiervan is het injecteren van Javascript in een request (zie de *self-retweeting tweet*). Dit wordt opnieuw veroorzaakt door onnauwkeurige controle van input van de client aan de serverzijde.

Cross-origin Access is het stelen van informatie van andere websites met behulp van HTTP requests in een browser script. Browsers vemijden deze aanval door scripts enkel resources van dezelfde oorsprong te laten ophalen. Servers kunnen deze restricties omzeilen aan de hand van **Cross-Origin Resource Sharing (CORS)**. Hiervoor moet de server een `Access-Control-Allow-Origin` header toevoegen aan responses, en clients een `Origin` header. Als deze headers overeenkomen, kunnen scripts wel resources ophalen.

Hoofdstuk 2

Hypermedia & Web APIs

Het Web is veranderd. Internetverbinding is snel genoeg voor videostreaming, en er zijn meer gebruikers langer online om sociale media aan te drijven.

Affordance Een "affordance" is een eigenschap van een object dat zijn gebruik helpt en uitlegt. Een bepaalde deurklink "affords" het openen van een deur. Het hint naar de openingsrichting en helpt bij het openen zelf, maar is op zich niet nodig om de deur te openen. Affordances zijn cruciaal op het Web.

Een **Web API** is een **server-side HTTP interface** die data en functionaliteit blootstelt aan clients. Als deze goed ontworpen is, voorziet een Web API de affordances voor mensen en/of machines die er mee willen werken. Echter is de wereld van Web APIs enorm heterogeen, en ligt de verantwoordelijkheid van affordances te maken in de handen van de (al dan niet competente) programmeur.

2.1 REST

2.1.1 Beperkingen

De **REST** architecturale stijl bevat een aantal beperkingen voor gedistribueerde hypermedia systemen. Deze bestaat uit vijf verplichte beperkingen:

1. client-server beperkingen
2. staatloosheidsbeperkingen
3. cache beperkingen
4. gelaagde systeem beperkingen
5. uniforme interface beperkingen

Client-server beperking

REST architecturen erven de client-server beperkingen over. De server luistert naar requests, de clients sturen requests door via een connector. Dit zorgt voor een *"separation of concerns"*. We scheiden de user interface van de data, we versimpelen de server, we verbeteren de draagbaarheid over verschillende platforms en zorgt voor onafhankelijke ontwikkeling van de client en server.

Staatloosheidsbeperkingen

Elke client request moet alle informatie bevatten om verstaanbaar te zijn voor de server. De server houdt *geen* staat bij van de client. Staatloze requests kunnen onafhankelijk van elkaar geïnterpreteerd worden. Deze staatloosheid is enkel geldig voor **applicatie staat**, servers behouden nog steeds **resource staat**.

Staatloosheid verbetert **visibiliteit** (intermediaries verstaan elke request zonder het bestaan van andere te weten), **betrouwbaarheid** (herstel van gedeeltelijk falen is gemakkelijker aangezien er geen geschiedenis van requests nodig is), en **schaalbaarheid** (geen data moet gehouden worden tussen requests. Elk request kan behandeld worden door een verschillende, onafhankelijke server).

Staatloosheid heeft meer **bandbreedte** nodig (informatie moet soms herhaald worden in verschillende requests), en er is minder **server-side controle** (clients zijn verantwoordelijk voor het consistent overgaan van een staat naar een ander. Ze zullen niet in se alles correct doen).

Cache beperkingen

Een server respons geeft expliciet aan of een request cacheable is. Caching verbetert netwerk **efficiëntie**, **schaalbaarheid** en **waargenomen performantie**.

Gelaagde systeem beperkingen

Een gelaagd systeem laat flexibele hiërarchische lagen toe. **Intermediaries** kunnen op transparante wijze toegevoegd worden om complexiteit en details te encapsuleren (proxies, caches, load balancers, firewals, ...).

Lagen verhogen de **flexibiliteit**, maar kunnen **overhead** en **latency** introduceren.

Uniforme interface beperkingen

REST introduceert vier uniforme interface beperkingen:

1. identificatie van resources
2. resource manipulatie door voorstellingen
3. zelfbeschrijvende berichten
4. hypermedia als de aandrijver van applicatiestaat

Identificatie van resources Alles dat benoemd kan worden is een **resource**. Een identifier wijst naar hoogstens 1 resource. Een resource kan meerdere **identifiers** hebben. Één identifier voor meerdere resources creëert een **collision**.

De waarde van een resource kan veranderen over tijd.

Resource manipulatie door voorstellingen Afhankelijk van de noden van de client, kan een resource op een andere manier voorgesteld worden. Dit kan een andere taal zijn, andere representatie (HTML, JSON), ...

Zelfbeschrijvende berichten Elk bericht in een REST interactie moet zelfbeschrijvend zijn. Op deze manier kunnen intermediaries berichten interpreteren. Ook zijn reponse en request bodies gevormd in expliciete, afgesproken **media** types met goed gedefinieerde syntax en semantiek, waarbij al de semantiek in het bericht zit.

Hypermedia controle Interactie is gedreven door **hypermedia controls** in de responses. Ze bevatten hyperlinks en forms die aangeven wat de mogelijke stappen zijn, en uitleggen hoe ze die stappen moeten nemen. In plaats van een voorgedefinieerd script uit te voeren, volgt de client de server.

2.1.2 REST toegepast op het Web

REST APIs zijn vaak HTTP APIs, maar andere platforms kunnen deze stijl ook implementeren. Vele "REST" APIs zijn gewoon HTTP APIs. De architectuur laat REST toe, maar kan het niet afdwingen.

Web resources zijn conceptuele relaties, uniek geïndentifieerd door HTTP URLs. Een HTTP URL wijst hoogstens naar één resource. Als het een informatie resource is, laat HTTP het toe om een voorstelling er van te krijgen.

Clients kunnen verschillende voorstellingen krijgen via **HTTP Content negotiation**. De client geeft zijn voorkeuren aan met de `Accept` header, de server adverteert zijn keuze met `Content-Type`.

Goed gedefinieerde HTTP methodes, headers en media types zorgen voor **zelfbeschrijving**. HTTP heeft een gelimiteerd aantal methoden, een uitbreidbare verzameling headers, en de `Content-Type` header beschrijft het media type.

HTML (en andere hypermedia) kunnen hypermedia controls bevatten. In HTML kunnen we daarvoor de `<a>` tag gebruiken. Hierdoor moeten gebruikers bijna nooit de adresbalk gebruiken.

2.2 Duurzaamheid van informatie

Vele websites volgen de REST principes. Meestal is er maar één voorstelling per resource. Om een REST API te hebben, volstaat het om meerdere voorstellingen per resource ter beschikking te stellen. Zo wordt een website een API (in theorie).

TODO

De REST architectuur mikt naar duurzaamheid, aangezien resources een URL behouden ook al veranderen technologieën. Maar zelfs waar de duurzaamheid belangrijk is, wordt REST niet altijd toegepast.

Europeana is een webportaal voor metadata van Europees cultureel erfgoed. Ze verzamelt metadata van meet dan 2.000+ instellingen. Europeana publiceert hun inhoud op een website en als een afzonderlijke **Web API**. Deze Web API is echter niet RESTful en daardoor moeilijk te gebruiken. Deze heeft gevolgen voor de duurzaamheid. Een URL kan niet gebookmarked worden, niet gedeeld worden, ...

2.3 Web API technologieën

Web APIs begonnen als **Web Services**. Het verschil tussen de twee is niet goed gedefinieerd. Meestal worden oudere XML-gebaseerde interfaces gelabeled als "Web Services", en interfaces dat gebruik maken van HTTP en JSON worden "Web APIs" genoemd.

2.3.1 Simple Object Access Protocol (SOAP)

SOAP verstuurt **XML berichten** bovenop HTTP. Een SOAP bericht bestaat uit drie hoofdelementen:

envelope root met (optionele) header en body

header modulair extensiemechanisme

body details van een methode invocatie

SOAP biedt **taalonafhankelijk programmeren** over een netwerk. SOAP werkt zelfs zonder het Web via e-mail.

2.3.2 Web API beschrijvingen

Web API beschrijvingen bevatten machine-interpreteerbare details over een API. Er bestaan drie voornamelijke gebruiken voor beschrijvingen:

in-band resource en hypermedia control details

interface structurele eigenschappen

functional effect en/of doel kenschetsing

Web Services Description Language (WSDL)

Een WSDL beschrijving is een XML document dat de methodes en parameters van een diesnt oplijst. Dit is enkel structuur, geen semantiek. Een WDSL beschrijving en SOAP wrapper kunnen automatisch gegenereerd worden van programmacode. Code om te communiceren met de beschreven SOAP service kan gegenereerd worden van de WDSL.

OWL-S (Semantic Markup for Web Services)

OWL-S verrijkt WSDL met functionele aspecten. Een OWL-S beschrijving beschrijft de semantiek van een SOAP bewerking gebruik makende van RDF. Een OWL-S beschrijving bestaat uit drie delen:

profile high-level functionaliteit voor ontdekking

grounding verbinding naar SOAP parameters

model diepgaande functionele relaties en beperkingen

OWL-S is heel krachtig voor RPC, maar is ongebruikt.

Web Application Description Language (WADL)

WADL is de WSDL van niet SOAP gebaseerde Web APIs. WADL beschrijvingen detailleren structurele eigenschappen. Hier is er ondersteuning voor XML en JSON. Een WADL beschrijving is een XML document dat HTTP methodes en parameters beschrijft, alsook de voorstellingsstructuur. WADL kan ook zorgen voor geautomatiseerde codeontwikkeling.

Verder

De meest recente Web API beschrijvingsformaten hebben een focus op ontwikkelaarsondersteuning (OpenAPI, API Blueprint, ...). Structurele beschrijvingen bevorderen de ontwikkeling, maar veranderen Web APIs niet op fundamenteel niveau. Ze moedigen **tight coupling** aan, hebben weinig te doen met het Web en verhinderen een duurzaam Web API ecosysteem.

Hoofdstuk 3

Semantiek & technologieën voor het Semantisch Web

3.1 Metadata

We organiseren informatie om ze efficiënter terug te vinden. Met andere woorden: het effectief vinden stoelt op een of andere manier van organisatie/ordening die toegepast wordt op informatie.

Definitie **Metadata** is **data over data**. Het verschaft informatie voor een beter begrip van data, concepten en entiteiten uit de reële wereld. De oplossing voor **information overload** is meer metadata.

Vandaag bestaat metadata uit gestandaardiseerde en gestructureerde informatie die automatische processen kunnen sturen, met als doel de gevonden data te **begrijpen**.

Het ultiem doel van metadata is dat het een medium wordt waarin zowel mensen als software bij het verwerken van data en content tot dezelfde conclusies kunnen komen.

Metadata management De levenscyclus van metadata is langer dan die van data. Metadata wordt vaak gecreëerd nog voor de data gecreëerd of gecapteerd wordt. Ze dient dan ook vaak langer bewaard te worden dan de data zelf.

3.1.1 Levenscyclus van metadata

1. Creatie
2. Onderhoud
3. Updaten
4. Opslaan
5. Publiceren
6. Omgaan met verwijderen van data

Creatie Metadata wordt manueel ingevoerd, of automatisch gegenereerd (of een combinatie van beide).

Onderhoud De data kan statisch zijn, waardoor de metadata manueel geüpdatet moet worden. De data kan ook real-time sensordata zijn, waardoor de metadata vaker automatisch geüpdatet kan worden.

Updaten Metadata wordt gebruikt op globale schaal, met veranderende context. Nieuwe toepassingen worden gemaakt, thesauri evolueren, technologie en standaarden veranderen, ...

Opslaan Dit kan ingebed zijn in het bestand (Office-documenten, MP3, JPEG), of gescheiden (bv. databank).

Publiceren

Omgaan met verwijderen van data Metadata blijft om verwijdering aan te geven, eventueel met verwijzing naar een gearchiveerde kopie.

3.1.2 Metadatakwaliteit

Er zijn een aantal dimensies van metadatakwaliteit:

accuracy worden de eigenschappen van de databron correct weerspiegeld?

availability zijn de metadata toegankelijk?

completeness zijn alle eigenschappen/aspecten van de databron aanwezig?

conformance in welke mate worden standaarden gebruikt?

consistency bevatten de data tegenstrijdigheden?

credibility & provenance is de databron te vertrouwen?

processability is de data machine-leesbaar?

timeliness data op tijd beschikbaar en nog steeds geldig?

relevance welke data is nodig voor een gegeven toepassing/taak?

3.1.3 Taxonomieën

TODO

3.1.4 Metadata tags

Een tag is een annotatie in vrije tekst. Ze kunnen keywords, termen, commentaren, ... bevatten. Vaak zijn ze ook informeel en persoonlijk.

Origineel werden taxonomieën of woordenlijsten gebruikt om content te beschrijven. Deze zijn uiteindelijk geëvolueerd in **folksonomieën**.

Folksonomie

Folksonomie is een porte-manteauwoord bestaande uit folk + taxonomie. Dit is een systeem waarbij er annotatie is in vrije tekst. Een folksonomie heeft geen voorgedefinieerde structuur of hiërarchie. Gebruikers voegen zelf metadata toe. Alle termen zijn evenwaardig.

Problemen van tags

- Er kunnen zich **culturele verschillen** voordoen. Voor sommigen is Genghis Kahn een held, voor anderen een crimineel.
- Gebruikersgemeenschappen kunnen verschillende betekenis geven aan tags (Movie, Film, Cinema).
- Er kunnen zich **taalproblemen** voor doen.
- Er kan **ambiguïteit** zijn, bijvoorbeeld bij apple.
- **Spellingsfouten**
- Semantiek van tags kunnen veel verschillen. Zo zijn er **feitelijke tags**, **subjectieve tags**, **persoonlijke tags** en andere.

3.1.5 Metadatastandaard

Door gebruik te maken van **metadatastandaarden** wordt gedeelde informatie voorgesteld in een algemene structuur. Standaardsoftware kan dan gebruikt worden om de informatie te verwerken.

Er zijn verschillende standaarden die de **structuur** van metadata vastleggen voor verschillende domeinen, communities, formaten, doelen, ...

XML **XML Schema** gebruikt XML om de structuur van een document te beschrijven. Textuele specificatie bepaalt dan de semantiek van de elementen.

Problemen met metadatastandaarden

Als er verschillende standaarden in één systeem zich bevinden zijn er mappingen nodig. Ook kunnen verschillende waarden voor een bepaalde property bij een metadatastandaard voor problemen zorgen. Zo betekenen "William Shakespeare" en "Shakespeare, William" inhoudelijk hetzelfde, maar een machine zou dit niet kunnen interpreteren.

Het is ook moeilijk voor een machine om de **semantiek** van metadata te achterhalen. Momenteel is dit gewoon pure tekst wat niet verstaanbaar is voor machines.

Ook kunnen verschillende standaarden hetzelfde beschrijven maar op een totaal verschillende manier.

3.1.6 Samenvattend

Er zijn twee belangrijke aspecten van metadata: **syntax** en **semantiek**. Het huidige Web is bijna uitsluitend syntactisch opgebouwd.

Syntax

Syntax bepaalt de representatie van metadata.

Semantiek

Semantiek bepaalt de **eigenlijke betekenis** van metadata-elementen

3.2 Semantiek toevoegen aan data

Metadata dat op verschillende manieren is gedefinieerd kunnen hetzelfde betekenen voor een persoon, maar een machine zou dit niet automatisch kunnen interpreteren.

Momenteel is **XML** de standaard voor data uitwisseling. XML heeft echter een aantal nadelen. Ze drukken geen semantiek uit, enkel de structuur van een document. Ook is er weinig **semantische interoperabiliteit**. Dezelfde tags kunnen een verschillende betekenis hebben, en tags met dezelfde betekenis een verschillende structuur.

Semantic Web technologieën (RDF, RDF Schema, OWL, ...) kunnen dit probleem oplossen. Hierbij maken we gebruik van **ontologieën**.

> **Ontologie**
>
> Een formele voorstelling van een verzameling concepten binnenin een **domein** and de relaties tussen deze concepten. Het wordt gebruikt om te redeneren over eigenschappen van dat domein, en kan gebruikt worden om het domein te definiëren.

Een **ontologie** bevat volgende concepten: instances, klassen, attributen, relaties, functies en beperkingen. De ontologie wordt voorgesteld als een graaf met behulp van **tripels**, waarbij **URIs** gebruikt worden.

3.2.1 Wat is er nodig?

De beschikbare informatie moet **uitwisselbaar** en **begrijpbaar** zijn voor machines op een **ondubbelzinnige manier**. Het moet gemakkelijk zijn om deze informatie te combineren. Machines moeten kunnen redeneren over informatie.

Er is dus een nood aan een **Web van data (Semantisch Web)**, in plaats van een Web van documenten **Syntactisch Web**.

Het Semantisch Web moet geïntegreerd kunnen worden in het huidige web, en mag dus geen tweede web creëren.

3.2.2 Structuur van het Semantisch Web

Informatie wordt voorgesteld door middel van **gerichte grafen**. In de praktijk wordt dit geserialiseerd in **RDF** (bv. RDF/XML, N3/Turtle).

De graafrepresentatie is onafhankelijk van bestaande dataformaten. Een verandering aan het schema van een lokale database stuurt het geheel aan informatie niet in de war. Nieuwe informatie en nieuwe connecties kunnen op triviale manier aangebracht worden.

3.2.3 Problemen met Syntactisch Web

Markup bestaat uit renderen van informatie en hyperlinks naar gerelateerde content. Semantische content is begrijpbaar voor mensen maar niet voor computers.

De meeste content op het Web vandaag is gemaakt om door mensen geïnterpreteerd te worden, niet om door machines gemanipuleerd te worden.

Het Sematisch Web brengt structuur niet de content zelf, maar wel in de betekenis van de content op Web pagina's.

3.2.4 Semantisch Web vs. kennisrepresentatie

Alles draait rond **betekenisvolle representatie van data**. Dit bestond reeds lang voor het World Wide Web. Traditionele technieken van kennisrepresentatie gaan steeds uit van een gecentraliseerd systeem waarin de betekenis van alles éénduidig is vastgelegd. Een nadeel hierbij is dat schaalbare groei heel moeilijk wordt.

3.3 Open Linked Data

Het Semantisch Web is meer dan data op het Web plaatsen. Het legt de focus op **links** creëren. Er werd een vijf-sterrensysteem gemaakt die data-aanbieders kunnen 'verdienen'.

1 ster: toegankelijk via het Web, maar met een open licentie ('Open Data')

2 sterren: machine-leesbaar/gestructureerde data (bv. Excel-bestand i.p.v. een gescand document)

3 sterren: gebruik van niet-propriëtaire formaten (bv. CSV)

4 sterren: gebruik W3C standaarden om data te beschrijven

5 sterren: includeerd links tussen eigen data en andere databronnen

3.4 Toepassingen

Mashups

Mashups maken gebruik van linked data sets. In essentie houdt het de **integratie** en/of **visualisatie van verschillende data sets** in.

Zo kan er bijvoorbeeld **gouvernmentele data** opengesteld worden, die dan machine-leesbaar is.

Smart Queries

Google analyseert de query en past Semantic Web queries toe met behulp van de zoekstring.

schema.org

schema.org is een gezamenlijk initiatief van Google, Bing en "Yahoo!", wat een schema is dat toelaat te beschrijven wat op webpagina's staat. De bedoeling is om de betekenis (semantiek) van de inhoud uit te drukken, zodat het automatisch verwerkt kan worden door machines, gebruik makende van HTML-microdata.

3.5 Wat is het Semantisch Web *niet*

Het is geen magische artificiële intelligentie. Het biedt echter wel de mogelijkheid voor machines om goed-omschreven problemen op te lossen door goed-omschreven operaties uit te voeren op goed-omschreven data. Dit vergt een extra inspanning van ons allemaal.

3.6 Uitdagingen

- **Generieke user interfaces** voor data-exploratie ontwikkelen, waarbij het een adaptieve interface is voor elke vorm van data, met visualisaties en data-analyse.
- **Policies** voor informatie en data ontwikkelen, met betrekking tot identiteit, privacy, transparantie, ...
- **Robuustheid van het Web** verbeteren. Netwerk-gerelateerde problemen moeten opgelost worden (bv. Slashdot-effect), web-gebaseerde fouten kunnen voorkomen (bv. 404), phishing, spam, trolling, ... Deze fouten duiken vaak pas op bij schaalvergroting.
- **Multimediale informatie** wordt moeilijker geanalyzeerd. Veel informatie zit hier in 'verborgen'.
- De **hoeveelheid aan data** kan een probleem vormen.

3.7 Linked Data

Tim Berners-Lee stelde vier principes voor om **Linked Data** te publiceren:

1. Gebruik URIs als namen voor dingen
2. Gebruik HTTP URIs zodat mensen deze namen kunnen opzoeken
3. Wanneer iemand een URI opzoekt, voorzie nuttige informatie gebruik makende van de standaarden.
4. Geef links naar andere dingen mee, zodat mensen meer kunnen ontdekken.

Deze principes hebben een gelijkenis met de **REST uniforme interface beperkingen**.

Informatie en niet-informatie resources zouden **uniek identificeerbaar** moeten zijn. Gebruik maken van **HTTP URIs** verzekert dat eender wie de resource kan opvragen.

Het **derefereren** van een URI moet leiden tot zinvolle informatie van die resource, waarbij we met **zinvol** betekenen dat de data ter beschikking wordt gesteld gebruik makende van standaardtechnologieën. Ook moet de informatie context kunnen geven aan de resource.

Door links te maken naar andere resources, maken we een Web van Data.

3.7.1 Werking

TODO MET FOTOTJES ENZO.

3.7.2 Linked Data op het Web

Er is een immens aantal Linked Data beschikbaar op het Web voor hergebruik. Op structureel niveau zijn er honderden **vocabularies** die de bouwstenen kunnen vormen om jouw data te modelleren.

Op inhoudelijk niveau zijn er duizenden datasets die identifiers en data van individuen leveren.

Aangezien er geen enkele Linked Data set compleet is, maken we de **open-world** aanname. Geen enkele bron heeft de volledige waarheid. Andere bronnen kunnen meer data hebben over een bepaald onderwerp.

3.7.3 Vocabularies

Dublin Core

De Dublin Core termen zijn een verzameling van 15 vaak voorkomende metadata eigenschappen. Elke eigenschap is generisch en dus toepasbaar in vele gevallen (title, description, date, creator). Veel applicaties maken hier gebruik van.

Schema.org

Schema.org werd gemaakt en onderhouden door de grote **zoekmachines**. Het voorziet voornamelijk data voor opzoekingen. De concepten zijn relatief losjes gedefinieerd, zodat het gebruik flexibel is voor ontwikkelaars.

Open Graph

Open Graph is gelijkaardig aan Schema.org en wordt voornamelijk gebruikt voor **Facebook integratie**. Zo laat facebook je toe om te "liken" en te "commenten" buiten het facebook.com domein. Als deze dingen beschreven zijn met Open Graph kan facebook het object preciezer linken.

Facebook raadt de **RDFa** standaard aan.

3.8 Semantisch Web

3.8.1 RDF

Het **Resource Description Framework (RDF)** is een model voor data-uitwisseling op het Web. RDF is een gestandaardiseerde manier om Linked Data voor te stellen. Het RDF model definieert **RDF datasets**.

Een RDF dataset heeft een **default graph** en ≥ 0 **named graphs**. Een RDF graaf is een verzameling van **RDF tripels**. Een RDF tripel bestaat uit een **subject**, **predicate** en **object**.

RDF heeft verschillende concrete syntaxen (tripel-gebaseerd, JSON-gebaseerd, XML-gebaseerd).

Samenstelling

Tripels bestaan uit RDF-termen als volgt:

named node een resource, geïdentificeerd door een IRI (subjects, predicates en objects)

blank node een ongenoemde resource (subjects, objects)

literal een waarde, met een datatype (IRI) of taal (objects)

Een RDF dataset heeft één **default graph** en nul of meer **named graphs**. De default graph is een RDF graph die leeg kan zijn. Een **named graph** is een RDF graph geïdentificeerd door een IRI. Een tripel behoort tot een of meerdere grafen. Een tripel met graaf wordt soms een **quad** genoemd.

Niet alle syntaxen ondersteunen named graphs.

Er zijn **onafhankelijke** syntaxen voor RDF die **triple-gebaseerd** zijn (N-Triples, Turtle, N-Quads, TriG), **JSON-gebaseerd** (JSON-LD) en **XML-gebaseerd** (RDF/XML).

Ook zijn er **geëmbedde syntaxen** voor **HTML/XML** (RDFa).

TODO: Syntaxen overlopen

3.8.2 RDFS

RDF Schema is een RDF vocabulary om RDF vocabularies te modelleren. RDFS definieert **klassen**, **eigenschappen** en **datatypes** die gebruikt worden om vocabularies te definiëren.

Gebruikers in de RDF wereld verwijzen vaak naar vocabularies als **ontologieën**. Strikt genomen is een vocabulary een verzameling woorden. Een **ontologie** is een verzameling concepten en hun relaties.

TODO: RDF termen beschrijven

Kennis van RDFS helpt het verstaan van meeste vocabularies (RDF, RDFS, FOAF).

3.8.3 OWL

De **Web Ontology Language (OWL)** voorziet concepten voor gedetailleerde ontologieën. OWL breidt RDFS met geavanceerde concepten uit. RDFS en OWL worden zij aan zij gebruikt.

TODO: OWL in-depth bespreken

3.9 Queryen en beredeneren

3.9.1 SPARQL

Het **SPARQL Protocol And Query Language** laat queryen en updaten van RDF datasets toe. Het SPARQL protocol is een WEB API definitie voor het queryen in de SPARQL taal over HTTP.

Er zijn momenteel vier red-only query vormen:

SELECT vind waarden dat aan bepaalde condities voldoet

CONSTRUCT creëer tripels dat voldoen aan condities

ASK vraag of data bestaat

DESCRIBE toon informatie over een resource

De voornaamste bouwsteen van een SPARQL query is een **Basic Graph Pattern (BGP)**. Een BGP is een verzameling **tripel patronen**. Een tripelpatroon is een triple waarbij elk component een **variabele** kan zijn. Deze starten met een vraagteken.

TODO: In detail gaan met SPARQL

Het doel van het SPARQL protocol is het versturen van queries en het verkrijgen van resultaten. De server is dan doorgaans een **RDF databank (triplestore)** met een **SPARQL query engine**.

De client stuurt een query gebruik makend van een URI template, en de server stuurt een response in een gestandardizeerd formaat (XML, JSON, CSV, ...).

3.9.2 RDFS, OWL en N3 inferencing

TODO

Notation3

Notation3 (N3) is een regel-gebaseerde taal gedefinieerd als een superset van **Turtle**.

TODO

Hoofdstuk 4

Linked Data Publishing

4.0.1 Linked Data Life Cycle

Er zijn vele **Linked Data life cycles**. De simpele cycle bestaat uit vijf stappen.

Generation

Meeste voorstellingen op het web zijn gegenereerd door **templates**. Data zit in een **back-end databank**. De **front-end** web applicatie vertaalt databank entries met behulp van templates naar voorstellingen.

Templating is vaak niet voldoende om **5-star Linked Data** te maken. Linked Data kan gegenereerd worden door middel van een **mapping** proces. Een **mapping processor** neemt bronnen van data en een mapping file als input. Deze **mapping file** beschrijft hoe input data geconverteerd moet worden naar het RDF model.

R2RML is een RDF vocabulary voor het beschrijven van een mapping van relationele data naar RDF. Een tripel map heeft toegang tot tabellen en queries door logische tabellen. Deze **logische tabellen** zijn gemapped met **term maps**. Term maps genereren individuele RDF termen.

TODO: vis R2RML uit

Validation

Ontologieën laten veel preciezere validaties toe. Meer types kunnen gedefinieerd worden, en hun definities kunnen hergebruikt worden. **Type checking** kan meer dingen in acht nemen (domeinen, ranges).

Publishing

Er zijn ruwweg drie manieren om Linked Data op het Web te publiceren:

data dump geef een **archiefbestand** met de gehele dataset

SPARQL endpoint stel een tripelstore query interface ter beschikking

Linked Data Documents zoek triples op per resource/onderwerp

Data Dump Dumps moeten geheel gedownloadet worden vooraleer ze kunnen bevraagd worden. Ze geven de client volledige flexibiliteit in hoe de data verwerkt wordt. Om deze data echter up-to-date te houden, vergt het een extra inspanning.

SPARQL endpoint Een SPARQL endpoint laat clients toe om arbitraire (read-only) queries te evalueren op een server. Dit geeft de clients directe toegang aan data waarin ze geïnteresseerd zijn. Deze data is altijd up-to-date, maar de per-request cost voor SPARQL endpoints is veel hoger dan andere HTTP servers.

Linked Data documents Linked Data documents geven per onderwerp toegang aan een dataset. Ze volgen de Linked Data principes.

Querying

Net zoals op het "menselijke" web, gaat queryen verder dan browsen. De mogelijkheden van query evaluatie hangt af van hoe de data beschikbaar gemaakt werd. Als de data beschikbaar is in RDF, kunnen we deze bevragen aan de hand van de gebruikte ontologieën.

Enhancing

Wanneer gebruikers databronnen opvragen, kunnen ze fouten of ontbrekende informatie opmerken. Open data zou open moeten zijn voor verbeteringen. Jammer genoeg is dit zeldzaam voor Linked Data.

4.0.2 Semantic Web Challenges

De huidige generatie van **agents** voert enkel voorgeprogrammeerde handelingen uit. Het doel van het semantisch web is dat we dit toelaten met onbekende services en date. Machines zouden services en date moeten kunnen ontdekken en gebruiken zonder voorkennis.

Meeste publieke SPARQL endpoints hebben minder dan 95% beschikbaarheid. De gemiddelde SPARQL endpoint is **1.5 dagen elke maand** offline.

4.0.3 Linked Data Fragments

Trade-offs voor het Semantisch Web

Elk type van **Linked Data Fragment** is gedefinieerd door drie karakteristieken.

Data dump:

data alle dataset tripels

metadata aantal tripels, bestandsgrootte

controls geen

SPARQL query resultaat:

data triples die voldoen aan de query

metadata geen

controls geen

Linked Data document:

data tripels over het onderwerp

metadata creator, maintainer, ...

controls links naar andere Linked Data documenten

Triple Pattern Fragment We ontwerpen een nieuwe mix met lage kost en hoge beschikbaarheid. Een **Triple Pattern Fragement** heeft een lage kost en laat clients toe te queryen.

data matches van een tripelpatroon (gepagineerd)

metadata totaal aantal matches

controls toegang tot alle andere Triple Pattern Fragments van dezelfde dataset

Triple Pattern Fragments zijn lightweight, omdat ze geen triplestore nodig hebben. Een interface kan gemaakt zijn met verschillende back-ends. Het **Header-Dictionary-Triples (HDT)** formaat slaat tripels op in een gecomprimeerd bestand.

TODO: in detail Triple Pattern Fragments bespreken

Hoofdstuk 5

Video in het Web

5.1 Inleiding

Er is een explosieve groei van de hoeveelheid multimediale data die door gebruikers online geplaatst wordt. In 2007 waren er ca. 6 miljard foto's die gehost werden door Flickr, met ca. 6000 uploads per minuut.

Bij YouTube was de hoeveelheid video die per minuut geüploadet werd in 2011 60 uur.
4 miljard uur video wordt per maand bekeken op YouTube.
In 1 jaar tijd verdrievoudigde de mobiele trafiek van YouTube
YouTube 'Content ID' scant per dag 100 jaar video.

Wegens de **kwaliteitsverhoging** van multimediale content, is er een nood aan efficiënte adaptatie en aflevering van multimedia ontstaan.

Schattingen door o.a. Cisco voorspellen dat 86% van alle IP-trafiek op het Internet afkomstig zal zijn van video tegen eind 2016 (**exaflood**).

5.1.1 Afbeeldingen

Kunnen op het Web voorgesteld worden met de `<img>` tag. Dit is geïntegreerd in de **Webtechnologie stack**. Browsers hebben ingebouwde decoders voor afbeelding. Deze kunnen dan ook gestyled worden met **CSS** net als tekst-elementen. Eigenschappen van die afbeeldingen kunnen opgevraagd worden via JavaScript APIs.

5.1.2 Tijdsgebaseerde media

Met tijdsgebaseerde media bedoelen we **audio** en **video**, maar ook **presentaties**. We zien dat deze media vaak een extensie bovenop het Web is, eerder dan een integraal deel. Vaak zijn deze ondersteund via **plug-ins** (Flash). Tot voor kort hadden browsers geen audio- of videodecoders. HTML5 lost dit probleem op.

Een bijkomende complexiteit is de **tijdsdimensie**. Annotaties van multimedia wordt veel zinvoller als ze gekoppeld kunnen worden aan een tijdstip binnen de mediabron.

Een van de 24 'activities' van het W3C — naast Semantic Web, Web Services, HTML, Security, ... — is video een **"First class citizen"** te maken van het web.

5.2 Mediafragmenten

Er zijn **temporele**, **spatiale** en **track** media fragmenten. We kunnen deze beschrijven aan de hand van een metadataformaat. Een probleem dat zich hierbij vormt is echter dat het een **indirecte stap** vereist. De **fragmentidentificator** zit namelijk vervat in een apart document.

Clients moeten zich ook bewust zijn van de metadata. In het geval van RDF zullen RDF-annotaties gaan over een fragment van een document dat verwijst naar eigenlijke multimedia content.

Beschrijving van mediafragmenten Gebruikmakend van een **URI-mechanisme**:

```
MPEG-21:  hnp://foo.com/file.mp4#mp(/ Bme('npt','10','30'))
TemporalURI:  hnp://foo.com/file.ogg?t=10/30
SVG:  hnp://foo.com/file.svg#svgView(14,15,146,147)
```

URI-gebaseerde aanpakken zijn vaak **te beperkt** of **te complex**. Een goede aanpak is **onafhankelijk** van het mediaformaat, bevat drie **fragmentassen (temporeel, spatiaal, track)**, bevat een **relatie met de context**, heeft een **lage complexiteit**, **minimale impact** op de bestaande Web-infrastructuur en bevat fragmenten door middel van media-extractie.

5.2.1 Media Fragments 1.0

URI's

Volgende fragment identifiers kunnen gebruikt worden om media fragmenten aan te duiden:

- Temporele fragmenten
  - *#t=20,50*
  - *#t=npt:20,50*
  - *#t=20*
- Spatiale fragmenten
  - *#xywh=30,20,10,10*
  - *#xywh=pixel:30,20,10,10*
- Track fragmenten
  - *#track=video1*
  - *#track=spanish%20audio*
  - *#track=video1;audio1*

Tip: test dit allemaal eens uit op http://ninsuna.elis.ugent.be/MediaFragmentsPlayer

Fragment URI	Query URI
#t=20,30	?t=20,30
notie van context	geen notie van context
adaptaties in byte ranges	geen beperkingen
client-side processing	key-value pairs worden naar de server gestuurd
cacheable	niet cacheable

5.2.2 Verwerking van HTTP Media Fragment URIs

Het **URI-fragment** is niet beschikbaar aan de server-zijde. Deze kan op verschillende manieren verwerkt worden aan de client-zijde.

Het mediafragment kan geïnterpreteerd worden door de user agent.

- Gehele video aanvragen ,springen naar het begin van het fragment en stoppen op het einde.

\+ Lokaal de tijd van het fragmen omzetten naar bytes en dit aanvragen.

Het mediafragment wordt gecommuniceerd via HTTP-headers.

- mapping van tijd naar bytes aan server-side
- mapping server-side met setup info
- mapping server-side - proxy chacheable (redirect)

5.3 HTTP adaptive streaming

HTTP adaptive streaming is de gulden middenweg tussen HTTP en Real time streaming.

	HTTP	RTS	HTTP adaptive
latency		laag	laag
bandbreedte		OK	OK
server-side monitoring		ja	nee (stateless)
firewall issues	nee		nee
speciale server vereist	nee	mogelijk	nee
trick modes	nee	ja	nee
cacheability	ja	nee	ja
intrinsieke adaptiviteit	nee	nee	ja

Principes Media resource wordt opgedeeld in meerdere chunks(van typsich zo'n 2 à 10 seconden). Deze worden apart doorgestuurd, samen met een manifest. Dit manifest bevat alle metadata. Hetgeen vereist is bestaat uit volgende punten:

- media resource opsplitsen
- chunks onderbrengen in containerformaat
- genereren manifest
- server moet de chunks afleveren (statisch)
- client moet manifest en containerformaat begrijpen
- client moet de chunks kunnen agregeren
- client moet kunnen schakelen tussen de verschillende versies

Flow De client bepaalt de media delivery. Deze leest het manifest en download de verschillende segmenten. HTTP Streaming komt dus overeen met een sequentie van HTTP downloads.

HTTP adaptive streaming: De client kan van versie veranderen (bvb kwaliteit aanpassen). Dit kan a.d.h.v. het manifest.

5.4 HTTP adaptive streaming: bestaande technologiën

5.4.1 HTTP Live Streaming

Manifest in M3U8 formaat, dit is een uitbreiding op M3U-playlists. Nu zijn er twee opties voor de versies. Elke playlist bevat verschillende versies ofwel bestaat er een playlist voor elke versie. Vereist is dat de server een HTTP 1.x webserver is, en de client een van volgende: iPhone, iPad, QuickTime X, Android 3.0

5.4.2 Smooth Streaming

Smooth streaming maakt gebruik van gefragmenteerde MP4 bestanden. Het manifest is een xml gebaseerd bestand in propriëtair formaat. De server moet een IIS met media extension zijn, de client is telkens silverlight.

5.4.3 HTTP Dynamic Streaming

Ook deze technologie maakt gebruik van gefragmenteerde MP4-bestanden. Werkt enkel met Flash media server en het manifest is ook een xml-gebaseerd bestand in propriëtair formaat. In het manifest worden verder enkel de versies omschreven, de chunks zelf worden beschreven in het containerformaat. De segmenten worden geadresseerd met byte rangesuit het Flash index bestand.

5.4.4 MPEG DASH

DASH = Dynamic AdapBve Streaming over HTTP Enkel ondersteuning voor MP4 en MPEG-2 TS. Het manifest formaat is gebaseerd op Media PresentaBon DescripBon van 3GPP. Dit is de huidige specificatie.

Hoofdstuk 6

Contentnetwerken

6.1 Data- en contentswitching

De huidige internetarchitectuur is op end-to-end communicatie gebaseerd. De transport en applicatielaag wordt enkel in de client en de server aangesproken en in het netwerk zelf wordt maximum tot laag 3 (routing) gewerkt. Dit zorgt wel voor grote operationele uitdagingen.

6.1.1 Layer 4 switching

Layer 4 switching betekent dat het switchen op laag 4 gebeurd, dit is de transportlaag met TCP of UDP. De switch voert "**content-blind routing**" ook wel "**immediate binding**" uit:

- Switch doet zich voor als eindserver (end-host) bij aankomst van het eerste TCP SYN pakket.
- De switch gebruikt informatie uit de transportlaag om trafiek gericht te switchen/routeren (+ binding table).
- Er is efficiënte routering maar niet voor het dispatchen van content (er is geen kennis van de HTTP content).
- In de transportlaag wordt de poort informatie gebruikt om bepaalde applicaties te identificeren.

De **Web Switch** kan dus op basis van de inkomende aanvraag iets redirecten naar de SMTP server, Web Server of FTP server.

6.1.2 Layer 7 switching

Er bestaat ook layer 7, de applicatielaag, switching. De switch voert "**content aware switching**" ook wel "**delayed binding**" uit:

- De switch doet zich eveneens voor als eindserver maar zet een complete TCP connectie op met de client.
- Client pakketten worden door de switch ontleed tot in de applicatielaag en dan naar de server gestuurd.
- De routering is minder snel aangezien deze tot laag 7 loopt.
- Er is goede kennis voor dispatchen omdat men weet wat er wordt aangevraagd.

De apparatuur die dit doet wordt **contentswitching** apparatuur genoemd. De meeste switches kunnen zowel layer 4 als layer 7 switching aan en worden daarom ook wel **contentswitches** of **layer 4–7 switches** genoemd.

6.2 Contentnetwerken

Deze netwerken hebben de mogelijkheid om gebruikers toegang te verlenen tot bepaalde objecten op een locatie-onafhankelijke manier. URL's zijn daarvoor niet goed geschikt en de oplossing bestaat eruit om objecten op verschillende locaties te publiceren door replicatie van data. De uitdaging bestaat er in om te beslissen wie wanneer naar waar wordt verwezen.

Deze technologie wil een oplossing bieden aan opstopping van IP-lijnen, overbelasting van webservers, vertragingen door lange communicatiewegen, flash crowds, ...

Verkeer kan op 4 mogelijk plaatsen opgestopt raken:

1. **Internet acceslijn van de gebruiker.**
   Congestion op de accesslijn van de gebruiker kan niet worden aangepakt, enkel de gebruiker zelf heeft hierover de controle

2. **Het netwerk van een ISP (backbone).**
   Cruciale parameters voor de kwaliteit van het ISP netwerk zijn:

   - **Vertragingen**: delay, latency, round-trip times tussen de verschillende netwerkelementen.
   - **Jitter**: de variatie in vertragingen (vooral voor VoIP) belangrijk.
   - **Pakketverlies**: uitgedrukt in procent.

   Een goed ISP netwerk heeft een gelaagde structuur met eigen functionaliteit binnen de lagen en meestal zijn er 3 lagen:

   (a) **Edge laag**: alle klanten worden hierop aangesloten en er is dus veel routering van beperkte capaciteit.

   (b) **Metro laag**: de eerste laag van de backbone en er zijn minder connecties met lokale trafiek (bv. binnen België) met middelmatige capaciteit.

   (c) **Transit laag**: dit is de tweede laag van de backbone en verzorgt de capaciteit naar de rest van het netwerk/internet. Er is bijna geen routering meer en pure switching op zeer hoge capaciteiten

3. **Peering punten (interconnecties tussen ISP's).**
   Dit zijn punten waarop de ISP met andere communiceert. Dit kan zowel direct gebeuren als via een Internet Exchange zoals bv. BNIX in België.

4. **Internet acceslijn en belasting van de webserver.**
   Een server heeft uiteraard ook een toegangslijn die verzadigd kan raken. Er moet vooral rekening gehouden worden met hoeveel dat er wordt verstuurd van de server naar het internet:

   - Hoeveel **simultane connecties** naar een webserver men wil toelaten.
   - Wat is de gemiddelde **grootte van de webobjecten**, wat is de **structuur** van de websites, het **soort content**.
   - Moeten er nog andere protocollen dan HTTP geprioriteerd worden.
   - Wat is de **belasting** van de webserver.

Contentnetwerken kunnen de content zo dicht mogelijk bij de gebruiker plaatsen **[2]**, content verspreiden over verschillende ISP's **[3]** en de belasting van de webserver verminderen door content te spreiden over meerdere servers of acceslijnen**[4]**.

Het **doel** van contentnetwerken is het beheren van de replicatie van content door middel van 2 taken:

1. **Distributie**: dit garandeert het kopiëren en synchroniseren van objecten van een originele server naar diverse replica servers.

2. **Redirection**: dit zorgt ervoor dat gebruikers het object downloaden van de dichtstbijzijnde server. Dit wil zeggen minimale latency en/of hops.

De **constructie** van een contentnetwerk bestaat uit een 'Master Server' met meerdere 'Replica Servers'. Nu zijn er verschillende **classificaties** van contentnetwerken:

- Lokale/Globale distributie applicatieservers.
- DNS-gebaseerde/Applicatieserver gebaseerde routeringsmechanismen.
- ...
- Op basis van wie de eigenaar is van het contentnetwerk en het beheert (ISP's, Contentleveranciers, operatoren zoals Akamai)

Netwerkoperatoren of dus ISP's maken gebruik van caching proxies om bandbreedte en dus kosten in de backbone te besparen. Proxies kunnen recursief worden gebruikt, dit wil zeggen dat ze ouderproxies hebben. Op deze manier wordt een boomstructuur van proxies gebruikt. Voor aanvraag van niet populaire objecten zal er dus zeker een vertraging zijn en dit werkt niet ideaal als de originele servers niet dicht zitten bij de proxies uit de boomstructuur.

Er zijn drie modellen voor webcaching:

1. **Explicit caching**: de gebruiker bepaald het gebruik van proxies in de browser en dit moet door de ISP ondersteund worden.
2. **Forced explicit caching**: sommige ISP's forceren proxie gebruik. HTTP trafiek wordt dan niet doorgelaten als het geen ISP proxyserver passeert.
3. **Transparent caching**: trafiek naar TCP poort 80 kan door de ISP worden afgeleid naar een proxiserver zonder expliciete configuratie in de browser van de gebruiker

Proxies moeten dus door gebruikers worden ingesteld in hun browsers. Het Web Cache Communication Protocol (WCCP) is een protocol ontworpen om problemen met deze configuraties te vermijden door gebruik te maken van interception of redirection proxies. Er wordt gebruik gemaakt van transparent caching en de netwerkelementen die met WCCP geconfigureerd zijn zullen HTTP trafiek afleiden naar proxy servers zonder dat de gebruikers dit merken. Er is volledige transparantie voor de gebruikers en WCCP bevat ook een fail-safe mechanisme wat betekent dat als de web cache niet bereikbaar is, het verkeer er ook niet naartoe zal worden afgeleid.

Contentleveranciers zijn organisaties die grote hoeveelheden content publiceren op internet. Voorbeelden zijn CNN, Microsoft, Adobe, ... en men wil deze content zo wijd mogelijk beschikbaar stellen bij de gebruikers, controle houden over de content en dit zonder grote investeringen in netwerken en bandbreedte. Er zijn nu twee mogelijkheden:

1. Clustergebaseerd websysteem of webcluster
2. Gedistribueerde websystemen

6.2.1 Clustergebaseerd websysteem

De contentservers maskeren hun echte IP-adressen ("Real IP" of "RIP") voor de client ("ClientIP" of "CIP"). De client ziet 1 "Virtual IP" of "VIP" adres dat gerelateerd is aan een toestel voor de contentservers. Dit toestel is een webswitch (layer 4 of layer 7). In lokale server topologieën komen Webclusters het vaakst voor
Er zijn twee client/server dataflow mogelijkheden:

1. **One-way architectuur**: de contentserver antwoordt direct naar de client.
2. **Two-way architectuur**: de contentserver antwoordt aan de contentswitch, die vervolgens aan de client antwoordt. Bij Layer 7 switching zal men gegarandeerd two-way gebruiken.

Volgende twee aspecten zijn belangrijk bij load balancing van een webcluster:

1. Routeringsmechanismen

bepaalt hoe de request bij de contentserver(s) terecht komt.

Oplossingen:

- **Op basis van Layer 4 switches (two-way architecturen)**:

  Packet double-rewriting: Dit is gebaseerd op NAT en zowel voor ingaande als uitgaande pakketten moet de TCP en IP header worden herschreven en de checksums worden herberekend. Voor inkomende pakketen herschrijft de webswitch het VIP naar het RIP van de content server. Voor uitgaande pakketten herschrijft de webswitch het bronadres van RIP naar VIP.

- **Op basis van Layer 4 switches (one-way architecturen)**:

  Packet single-rewriting: De webswitch vervangt het VIP door RIP (contentserver) en herberekent de IP en TCP header checksum. De contentserver antwoordt rechtstreeks aan client maar gebruikt als bron IP-adres het VIP.

  Packet tunneling (ook IP tunneling of IP encapsulation): Dit is een techniek om een IP datagram te encapsuleren in een ander IP datagram. De webswitch tunnelt het inkomende pakket naar de contentserver door het te encapsuleren in een nieuw IP datagram met bron IP het VIP en destinatie IP het RIP. De contentserver stript het IP datagram eraf en zit het origineel pakket dat bestemd was aan het VIP adres. De contentserver antwoordt rechtstreeks aan de client met als bron IP het VIP.

  Packet Forwarding: Alle content servers hebben een RIP x met x het nummer van de content server, ook hebben ze een VIP. Packet forwarding gebeurt nu doordat de webswitch in het inkomende pakket het MAC adres vervangt door dat van de contentserver. Er gebeurt dus op laag 2 een MAC adres translation. Als de contentserver het pakket ontvangt ziet het er uit alsof het voor hemzelf is aangezien hij een gedeeld VIP-adres heeft. De contentserver kan rechtstreeks antwoorden aan de client zonder modificatie van het IP datagram.

- **Op basis van Layer 7 switches (two-way architecturen)**:

  TCP gateway: de webswitch bevat een applicatielaag proxyserver die alle inkomende verbindingen accepteert. De proxyserver heeft een persistente TCP connectie met alle contentservers. De proxiserver stuurt de client request door naar de contentserver en deze antwoord met data die de proxy doorstuurt naar de client.

  TCP splicing:

  TODO: IMG

  Stel dat de webswitch twee connecties heeft die het moet splicen, één naar **A** en één naar **B**. Het concept bestaat er in dat de volgende data(byte) die de Webswitch verwacht van A moet verstuurd worden op de connectie naar B met het juiste sequence nummer (namelijk dat dat het verwacht van de webswitch).

  We hebben:

  - **splice_irs** (splice Initial Receive Sequence number):
    Dit is het sequence number van de volgende byte die de **webswitch** verwacht te ontvangen van **A**.
  - **splice_iss** (splice Initial Send Sequence number):
    Het sequence number dat **B** als volgende verwacht te ontvangen van de **webswitch**.
  - <**splice_irs, splice_iss**>:
    Dit is een mapping tussen de sequence numbers spaces van de spliced connecties van **A naar B**.

Om ervoor te zorgen dat het volgende segment er voor B uitziet als een segment uit zijn originele connectie met de webswitch moeten we in de **IP header** het bron/doel ip aanpassen naar dat van de uitgaande connectie en de checksum updaten.
In de **TCP header** moeten we de bron/doel poort aanpassen naar die van de uitgaande connectie. en we moeten het sequence number van de inkomende space op de uitgaande space mappen:

TODO CODE

2. Dispatching algoritmes

Bepaalt bij welke content server de request terecht komt. Deze algoritmes voor load balancing kunnen statisch of dynamisch zijn.

De **statische** zijn de snelste oplossing maar houden geen rekening met de actuele status van de webservers waardoor de **dynamische** beter presteren. Wel zorgen de dynamische voor extra overhead omdat de status van de servers moet worden gecapteerd.

6.2.2 Gedistribueerde websystemen

De contentservers tonen hun RIP aan de clients en de sturing gebeurt niet vanuit een contentswitch. Men gebruikt deze systemen vooral bij een geografisch verspreid gebruik van de data.

Ook hier hebben we verschillende soorten:

- **DNS gebaseerde routeringsmechanismen**: dit is origineel ontworpen voor gedistribueerde websystemen, maar wordt heel veel gebruikt in geografisch gedistribueerde system en zoals Akamai.
- **Webserver gebaseerde routeringsmechanismen**: routering en eventuele verwijzingen gebeuren op de contentserver.

  Men kan hier gebruik maken van:

  - **Triangulation**: dit gebeurt in volgende stappen:
    * De client stuurt pakketten naar de eerste gecontacteerde content server die ze doorstuurt naar de tweede content server.
    * De routering is op IP tunneling gebaseerd wat wil zeggen dat het origineel datagram wordt geëncapsuleert in een nieuw IP datagram.
    * De tweede content server herkent dat het IP-datagram werd geherrouteerd en antwoordt rechtstreeks naar de client na aanpassing van het pakket waarbij zijn IP-adres vervangen wordt door dat van de eerste content server.
    * Alle volgende pakketten blijven het traject "client, eerste content server, tweede content server" volgen.
  - **HTTP redirection**: er kan een 301 'Moved permanently' of 302 'Moved temporarily' worden verstuurd. Dit laat controle toe op een heel fijn niveau maar introduceert extra latency.
  - **URL rewriting**: In dit geval zal de eerste gecontacteerde contentserver dynamisch de linken van de ingebedde objecten herschrijven. Dit kan vaak in combinatie met intelligente DNS-gebaseerde routering en dit wordt veel door CDN's gebruikt. Er is wel een grote load en latency door dit mechanisme.

6.3 Akamai

Akamai is de grootste CDN provider op het internet en wordt algemeen beschouwd als de marktleider. Het is in 1995 in MIT ontstaan omdat Tim Berners-Lee het probleem van network congestion voorzag en zijn collega's uitdaagde een goede techniek te bedenken. 85% van de globale internet gebruikers bevinden zich binnen 1 hop van een Akamai server. Het is voor veel contentleveranciers niet mogelijk om een wereldwijd servernetwerk uit te bouwen. CDN's bouwen wel dit netwerk en vragen dan een vergoeding om hiervan gebruik te maken. Zij kunnen de kosten aan door een gedeelde infrastructuur die door meerdere

gebruikers wordt betaald.

Oorpsronkelijk was het de missie van CDN operatoren om content voor hun klanten (content providers) op een geografisch verspreide manier te leveren. Tegenwoordig bieden CDN operatoren ook andere diensten aan:

- Ze beschikken over zeer hoog technologische algoritmes en protocollen (bv. DNS tools)
- Ze proberen meer taken op zich te nemen zoals scripting taken

Meestal gebruiken CDN's DNS om hun URL's te transformeren in locatie-onafhankelijke verwijzingen. Akamai's mapping proces gaat in volgende stappen. Stel dat een url (bv. www.mysite.net) in een webpagina verwijst naar **a7.g.akamai.net**:

1. De client DNS resolver krijgt als DNS de Akamai Top Level DNS server terug.
2. De client DNS vraagt aan deze server de DNS server voor g.akamai.net terug [TTL typisch 1 uur].
3. De client DNS vraagt aan akamai waar a7.g.akamai.net zit en krijgt een IP adres van een edge server terug [TTL typisch enkele seconden tot een minuut].
4. De client maakt verbinding met de edge server.
5. De client haalt eventueel een deel van de content bij een andere Akamai server.

Een pagina bestaat de dag van vandaag uit veel dynamische content wat proxy server caches typisch niet kunnen. Daarom zal er een **Edge Side Include** of **ESI** zijn welke een eenvoudige markup taal is, XML gebaseerd. Daarin staat beschreven welke delen van de site cacheable zijn en welke niet. De componenten worden beheerd als onafhankelijke objecte nin de cache van de edge server. Akamai zal dan enkel de statische fragmenten serveren en de dynamische ophalen bij de originele webserver ophalen. ESI wil zeggen dat de processing op de edge server gebeurt.

Andere diensten zijn bijvoorbeelde EdgeComputing for Java waarbij J2EE applicaties op de Akamai servers kunnen worden uitgevoerd, een Virtual Waiting Room om shopping carts en zo op te slaan, geografische landherkenning, . . .