

# 2019 samenvatting - Lars Lemmens

Met dank aan de [Github van Martijn](#) en Lars Lemmens

## Hoofdstuk 2

### 2.1 Principes van netwerkanvullingen

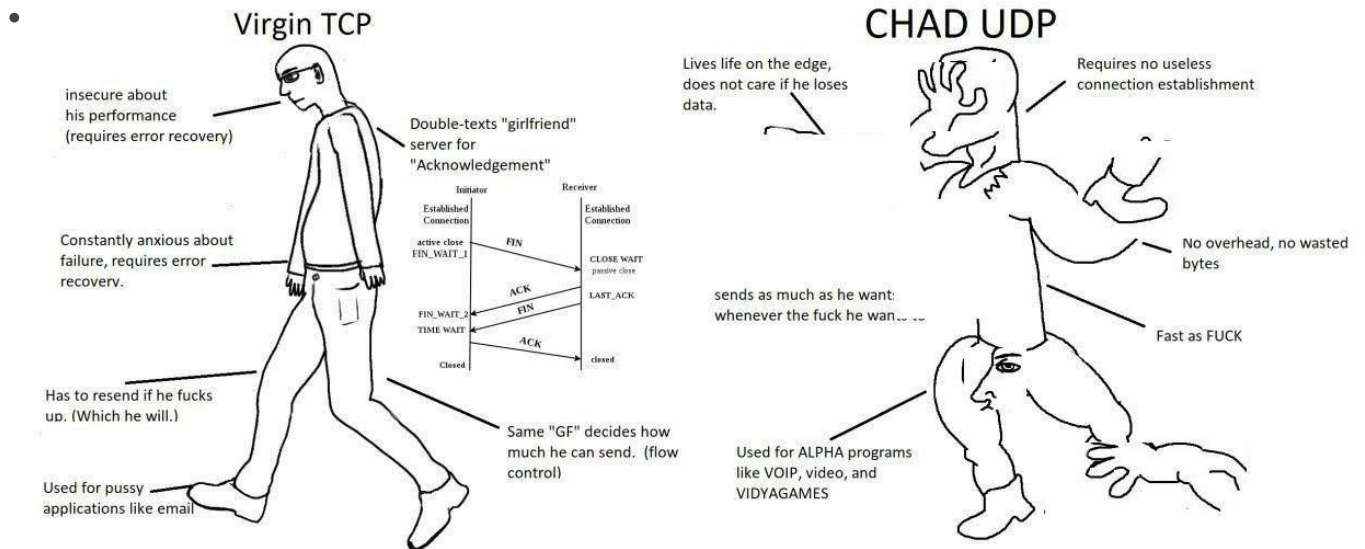
#### 2.1.1 Structuren van netwerkanvullingen

#### **Client-server structuur**

- Host altijd-aan (server) & handelt verzoeken van andere hosts (clients) af
- Meestal datacenters → anders overbelasting

#### **P2P-structuur**

- Infrastructuurservern die altijd aan zijn → minimaal of afwezig
- Maakt gebruik van rechtstreekse communicatie → periodiek met elkaar verbonden hosts → peers
- Zelfschaalbaarheid & goedkoper



## 2.1.2 Communicerende processen

- Processen in 2 verschillende hosts → communiceren met elkaar door berichten uit te wisselen → verzendend proces creëert & verzendt berichten over netwerk → ontvangend proces ontvangt berichten en verzendt eventueel antwoordbericht

## Client – en serverprocessen

- Voor elk paar communicerende processen → 1 van de 2 processen **client** & andere **server**
- Definitie client- en serverprocessen:
  - In context van een communicatiesessie tussen 2 processen noemen we het proces dat de communicatie initieert de **client**. Het wachtende proces noemen we de server
- In P2P kan een proces ophalen en beschikbaar stellen

## Interface tussen proces en computernetwerk

- Proces verzendt berichten naar en ontvangt berichten van het netwerk via een netwerkinterface → **socket AKA API ( Application Programming Interface)**
- Socket = soort deur → proces bericht naar ander proces sturen → bericht door deur (socket) naar buiten → verzendende proces neemt aan → andere kant deur → infrastructuur aanwezig is waarmee bericht → bij deur van huis van bestemming zal bezorgd worden → zodra bericht arriveert bij ontvangende host → passeert het de toegangsdeur van ontvangende proces → waarna ontvangende proces bericht verwerkt

# Adresseren van processen

- Om ontvangende proces te kennen moet het 2 gegevens bevatten:
  - Het adres van de host
  - Unieke aanduiding van ontvangende proces in ontvangende host

## 2.1.3 Transportdiensten voor applicaties

### | Data Integriteit:

- Transportlaagprotocol → mogelijk een betrouwbare gegevensoverdracht tussen processen aan applicatie leveren → verzendend proces gegevens in socket duwen & is zeker van geen fouten
- Als er geen betrouwbare gegevensoverdracht is → mogelijkheid dat deel gegevens niet aankomen bij ontvangend proces → \*\*verliestolerante apps (\*\*Skype,...) | **Doorvoer**
- Sommige apps (bijv. multimedia) vereisen een minimale verwerkingscapaciteit om "effectief" te zijn = **bandbreedtegevoelige apps** → stellen eisen aan doorvoercapaciteit
- Elastische apps gebruiken de doorvoercapaciteit die op het moment beschikbaar is

### Timing:

- Sommige apps (bijv. internettelefonie, interactieve games) hebben een korte vertraging nodig om 'effectief' te zijn **Security**
- Encryptie, data integriteit,...

## Diensten van TCP & UDP

68747470733a2f2f65787465726e616c2d636f6e74656e742e6475636b6475636b676f2e636f6d2f69

## Beveiligen van TCP

TCP gebruikt SSL/TLS → TCP met SSL/TLS → doet niet alleen wat oorspronkelijke TCP doet → levert ook beveiligingsdiensten voor communicerende processen → SSL niet een 3e transportprotocol voor internet is dat op zelfde niveau werkt als UDP & TCP → uitbreiding van TCP → uitbreidingen geïmplementeerd in applicatielaag

# Diensten die niet geleverd worden door internettransportprotocollen

- Huidig internet presteert goed voor tijdgevoelige apps → geen garanties voor timing of doorvoercapaciteit
- Internettelefonieapps kan enig verlies verwerken maar wel minimale snelheid eisen om te kunnen werken → Internettelefonieapps meestal UDP → grotendeel van firewalls blokkeren UDP → ontworpen voor TCP → als back-up communicatie via UDP niet lukt



## 2.1.5 Protocollen voor applicatielaag

In applicatielaag volgende aspecten gedefinieerd:

- Soorten berichten die uitgewisseld worden, request & antwoordberichten
- Syntaxis van verschillende soorten berichten, velden in bericht & manier waarop velden van elkaar gescheiden worden
- Semantiek van velden → betekenis informatie in velden
- Regels voor bepalen wanneer en hoe proces berichten verzendt & beantwoordt

## 2.1.6 Netwerkapplicaties die in dit boek beschreven worden

## 2.2 Web & HTTP

### 2.2.1 Meer over HTTP

- Webpagina bestaat uit aantal objecten → object is een bestand
- HTTP definieert hoe webclient een webserver verzoekt om een webpagina op te zoeken & hoe servers webpagina's versturen naar een client
- HTTP → TCP als transportprotocol
- HTTP-client initieert TCP-verbinding met server → verbinding tot stand → processen van browser & server TCP via socketinterfaces → client verzendt HTTP-verzoekberichten door socket → server ontvangt op zelfde manier HTTP-verzoekberichten via socket & verzendt HTTP-antwoordberichten door socket
- HTTP is staatloos → Server houdt geen informatie bij over eerdere clientaanvragen

## 2.2.2 Non-persistente en persistente verbindingen

### Non persistent

Basis HTML-file met volgende url: <http://www.someschool.edu/someDepartment/home.index>

1. HTTP-client starts TCP-verbinding met server → client & server gebruiken socket
2. HTTP-client verzendt HTTP-verzoekbericht naar HTTP
3. HTTP-serverproces ontvangt verzoekbericht
4. HTTP-serverproces TCP opdracht → verbreek verbinding als bericht ontvangen
5. HTTP-client ontvangt antwoordbericht
6. Herhaal stap 1-4 voor elk object

RTT → tijd die packet nodig heeft → client naar server & omgekeerd

Klikt op een hyperlink → 3-way handshakeproces nodig:

1. Client verzendt TCP-segment naar server
2. Server bevestigt & antwoord met TCP-segment
3. Client verzendt 2e bevestiging naar server  
→ 1 RTT

- Na 1&2 → verzendt client HTTP-verzoekbericht + 3e deel van 3way-handshake
- Totale responstijd = 2 RTT's + transmissietijd HTML-bestand server

### Persistent

- Bij non persistent → elk opgevraagd object een nieuwe verbinding tot stand gebracht & gehouden worden
- Bij non persistent → elk object 2 RTT's vertraging → 1 voor TCP-verbinding starten & 1 om TCP-verbinding object op te vragen & ontvangen
- HTTP 1.1 persistent → server laat TCP-verbinding intact na respons

## 2.2.3 Indeling HTTP-berichten

### HTTP-verzoekbericht

- Eerste regel → requestregels
- Volgende regels → headerregels
- Host → host waar opgeslagen object is
- Connection: close → geen persistente verbinding nodig
- User Agent: → type browser
- POST → stuurt velden dat gebruiker heeft ingevuld door
- GET → 2 velden ( x & y ) → structuur URL → [www.eensite.nl/zoeklets?x&y](http://www.eensite.nl/zoeklets?x&y)
- HEAD → server reageert HTTP-bericht zonder opgevraagde object te verzenden

### HTTP-antwoordbericht

- Statusregel, 6 headerregels & entity body
- Connection: close → client weet TCP-verbinding weg na bericht verstuurd
- Server: → geeft aan gegenereerd door Apache-webserver
- Last-Modified: cachen objecten → lokaal & server
- Content-length: → grootte object

68747470733a2f2f65787465726e616c2d636f6e74656e742e6475636b6475636b676f2e636f6d2f69

## 2.2.4 Interactie gebruikers & servers: cookies

1. Cookieheaderregel in HTTP-antwoordbericht
2. Cookieheaderregel in HTTP-verzoekbericht
3. Cookiebestand opgeslagen op host gebruiker & browser van gebruiker beheerd
4. Back-end database op website

## 2.2.5 Webcaching

Netwerkentiteit die HTTP-verzoeken afhandelt namens oorspronkelijke webserver waar verzoek oorspronkelijk naartoe is gestuurd

Browser object <http://www.someschool.edu/campus.gif>

1. Browser start TCP-verbinding met webcache & verzendt HTTP-verzoek voor object naar webcache
2. Webcache checks → exemplaar opgevraagde object aanwezig? → if yes → webcache verzendt object in HTTP-antwoordbericht naar browser client
3. Opggevraagde object niet op webcache → TCP-verbinding met oorspronkelijke server → webcache verzendt HTTP-verzoek voor object via TCP-verbinding → wanneer server ontvangen → verzendt object in HTTP-antwoordbericht naar webcache
4. Opslaan kopie lokaal & stuurt een exemplaar naar browser

2 redenen webcaching

- Responstijd clientverzoek verkorten
- Webcaches belasting van link van instituut verlagen

**CDN** (Content Distribution Networks) → veel geografische verspreide cachegeheugens in internet → groot deel dataverkeer lokaal

## 2.2.6 The conditional GET

- Verzoekbericht methode GET
- Verzoekbericht bevat If-Modified-Since

1. Client maakt get request
2. Server reageert met header
3. Client checkt de Last-Modified header
4. Is Last nieuwer is dan cache → haal pagina opnieuw op & zet opnieuw in cache
5. anders → laad van cache

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
```

```
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

# HTTP/2

**Goal:** vertraging verlagen in multi-object HTTP requests **HTTP1.1**\*\* :\*\* meerdere, pipelined GETs over 1 TCP connectie

- Server antwoord in-order ( **FCFS : first-come-first-served scheduling** ) naar GET requests
- Met FCFS, mogelijkheid dat kleine objects moeten wachten voor transmissie (Head-of-line (HOL) blocking) achter grote objects
- Herstel van loss (retransmissie van verloren TCP segmenten) stalls object transmissie

**HTTP/2**\*\* :\*\* flexibiliteit verhogen server in versturen van objects naar client **Goal:** vertraging verlagen in multi-object HTTP requests

- Methoden, status codes, meeste header velden onveranderd tov HTTP 1.1
- verzendvolgorde van aangevraagde objecten op basis van door de klant opgegeven objectprioriteit (Niet noodzakelijk FCFS)
- Push unrequested objects naar cleint
- Verdeel objects in frames, plan frames om HOL-blokkering te verminderen

68747470733a2f2f65787465726e616c2d636f6e74656e742e6475636b6475636b676f2e636f6d2f69

## HTTP/2 to HTTP/3

**Goal:** vertraging verlagen in multi-object HTTP requests

HTTP/2 over single TCP connectie wil zeggen:

- Herstel packet loss vertraagt nog steeds alle objecttransmissies
- In HTTP 1.1 browsers → stimulans om meerdere TCP-verbindingen te openen → vertraging verminderen → algehele doorvoer te verhogen
- Geen security over vanilla TCP-verbinding
- HTTP/3 → adds security → per object fout- & congestie-controle ( meer pipelining) over UDP



## 2.3 E-mail op het internet

### 2.3.1 SMTP

Werking:

1. Alice → UA opdracht verstuur bericht
2. Alice's UA stuurt bericht mailserver → in berichtenwachtrij
3. Clientzijde van SMTP opent TCP-verbinding met Bob's e-mailserver
4. SMTP-client verzendt het bericht van Alice via de TCP-verbinding
5. Bob's mailserver plaatst het bericht in Bob's mailbox
6. Bob roept zijn user agent aan om bericht te lezen

68747470733a2f2f65787465726e616c2d636f6e74656e742e6475636b6475636b676f2e636f6d2f69

### 2.3.2 Vergelijking met HTTP

#### SMTP

- Transfers files van 1 mail server naar een andere mail server
- **PUSH** protocol: verzendende e-mailserver duwt bestand naar e-mailserver van ontvanger
- Elk bericht moet in 7-bit ASCII zijn
- Plaatst alle bericht objecten in 1 bericht

#### HTTP

- Transfers files van web server naar web client
- **PULL** protocol: iemand zet informatie op een webserver & gebruiker gebruikt HTTP om informatie van server ophalen wanneer het hem uitkomt
- Restrictie niet nodig
- Encapsuleert elk object in zijn eigen HTTP response message

### 2.3.3 Formats e-mailberichten

## 2.3.4 Mail accessprotocollen

SMTP: levering/opslag van e-mailberichten op de server van de ontvanger

Mail Access Protocol: Ophalen server

IMAP: **Internet Mail Access Protocol** [RFC 3501]: berichten opgeslagen op de server, IMAP biedt ophalen, verwijderen, mappen met opgeslagen berichten op de server

HTTP: Gmail, Hotmail, Yahoo! Mail, etc. biedt webgebaseerde interface bovenop SMTP (om te verzenden), IMAP (of POP) om e-mailberichten op te halen

### POP3

UA → TCP-verbinding naar mailserver

1. Autorisatiefase: UA → stuurt username & password → identiteit gebruiker vaststellen
2. Transactiefase: UA haalt berichten op
3. Updatefase: client opdracht quit → POP3-sessie closed

### IMAP

- Koppelt elk bericht aan een map
- Biedt opdrachten → UA afzonderlijke componenten van bericht

## Webmail

Ontvanger wil mailbox bekijken → bericht van mailserver naar browser van gebruiker verzonden → behulp van HTTP-protocol

## 2.4 DNS

### 2.4.1 Diensten van DNS

- Een gedistribueerde database → geïmplementeerd in hiërarchie van DNS-servers

- Applicatielaagprotocol waarmee hosts & DNS-servers kunnen communiceren om vertaalslag ( omzetten IP → "te onthouden" adres & omgekeerd)

DNS verzorgt aantal andere diensten naast vertalen hostnamen in IP-adressen:

- **Host-aliasing:** 1 of meerdere aliases wanneer hostname te lastig is → canonieke hostnaam → aliashostnamen gebruikt → gemakkelijker te onthouden dan canonieke hostnamen
- **Mailserver-aliasing:** Hostnaam van een mailserver (bv. Yahoo) → moeilijker dan → eenvoudige Yahoo.com → DNS kan door mailapplicatie gebruikt worden → achterhalen canonieke hostnaam v/e bepaalde host + IP-adres van host → MX record maakt mogelijk dat mailserver & webserver van bedrijf dezelfde hostnamen hebben → mailserver & webserver van bedrijf kunnen dus bv. enterprise.com heten.

(Een MX-record (Mail eXchange-record) is een gegevenstype in het Domain Name System (DNS). Het specificeert de mail server die e-mailverkeer voor het betreffende domein afhandelt. Een domein kan meerdere MX-records hebben met een verschillende prioriteit waardoor het mogelijk is om bijvoorbeeld een back-up mailserver aan te geven als de computer met de hogere prioriteit niet bereikbaar blijkt. De naam die in het MX-record wordt gevonden kan via DNS op zijn beurt in een ip-adres worden vertaald. Src: Wikipedia)

- **Loadbalancing:** Belasting van gerepliceerde servers te verminderen → gerepliceerde webservers een serie IP-adressen gekoppeld aan canonieke hostname → DNS-database bevat serie IP-adressen

## 2.4.2 Overzicht van de werking van het DNS

Gecentraliseerd ontwerp levert volgende problemen op:

- **Single point of failure:** als DNS crashed hele internetonbruikbaar
- **Netwerkbelasting** : 1 DNS-server → alle DNS-verzoekberichten verwerken ( alle HTTP-verzoekberichten & e-mails van honderden miljoenen hosts)
- **Afstand tot gecentraliseerde database** : 1 enkele DNS-server kan niet 'in de buurt' van alle verzoekende clients staan → centrale DNS bijvoorbeeld in Antwerpen → verzoeken uit Nieuw-Zeeland → kan vertraging veroorzaken
- **Onderhoud** : die DNS-server informatie over alle internethosts moeten beheren → database wordt dan heel groot & telkens bijgewerkt moeten worden wanneer nieuwe host wordt aangemeld.

# Een gedistribueerde, hiërarchische database

- **Root-DNS-servers:** meer dan 400 root-nameservers verspreid over de wereld → door 13 verschillende organisaties gemanaged → leveren IP adressen van TLD-servers
- **Topleveldomein (TLD) -servers:** voor elk TLD (com,org,net,edu,...) & landTLD(nl,fr,ca,...) → is er een TLD → TLD leveren IP-adressen voor authoritative DNS-servers
- **Authoritative DNS-servers:** eigen DNS-server(s) van de organisatie, die gezaghebbende hostnaam biedt aan IP-toewijzingen voor de benoemde hosts van de organisatie + kan Dutch translation. kakjkworden onderhouden door organisatie of dienstverlener
- Lokale DNS-server behoort niet tot hiërarchie maar wel belangrijk voor DNS-structuur → elke ISP zoals een standaard ISP of ISP van instituut → lokale DNS-server → wanneer host verbinding tot stand wil brengen met ISP → ISP geeft aan die host IP-adressen van 1 of meer van diens lokale DNS-server → lokale DNS-server niet ver weg van host → bij residentiële ISP DNS-server → gescheiden enkele routers van host van client → host verzoekbericht aan DNS verzendt → doorgestuurd lokale DNS-server → fungeert als proxy → verzoek verder gestuurd in DNS-serverhiërarchie
- Recursief DNS-verzoek = host stuurt verzoek aan lokale DNS-server, die op zijn beurt een verzoek doet aan de root-server, die op zijn beurt een verzoek doet aan de TLD-server ... (zie p127)
- Iteratief DNS-verzoek = host stuurt verzoek aan lokale DNS-server, die op zijn beurt een verzoek doet aan de root-server, de root-server stuurt antwoord naar lokal DNS-server die op haar beurt een verzoek doet aan de TLD-server... (zie p126)

## DNS-caching

DNS-server → in verzoekberichtenketen → wanneer DNS-antwoord ontvangt → verwijzing in lokale cachegeheugen plaatsen

## 2.4.3 DNS-records en -berichten

DNS-servers bevatten bronrecords → (Naam, Waarde, Type, TTL)

1. **If Type = A** then naam bevat hostnaam en Waarde bevat IP-adres van hostnaam
2. **If Type = NS** then naam bevat domeinnaam en Waarde bevat hostnaam van authoritative DNS-server die de hostnaam en IP combinaties voor de hosts in dat domein weet
3. **If Type = CNAME** then waarde is canonieke hostname voor aliashostname

4. **IF Type = MX** then waarde is canonieke naam van mailserver met alias hostname name

Een authoritative DNS-server bevat een A-record

Een niet-authoritative DNS-server bevat een NS-record voor het domein waarin de DNS-server zich bevindt en ook een A-record met het IP-adres van de DNS-server die in het veld Waarde van het NS-record staat

## DNS-berichten

- Eerste 12 bytes voor de header:
  1. Eerste veld is een uniek 16 bit getal waarmee het verzoek geïdentificeerd kan worden. Dat getal wordt gekopieerd in het antwoordbericht, zodat de client het antwoord en het verzoek kan koppelen
  2. Vlaggen veld bevat een aantal vlaggen. Een 1-bit verzoek/antwoord-vlag (verzoek = 1 en antwoord = 0), 1-bit authoritative-vlag wordt in een antwoordbericht gezet als de DNS-server de authoritative DNS-server voor de hostnaam is, 1-bit recursienoodzaak-vlag wordt gebruikt als de client vraagt om recursief te werken als het gevraagde record niet op die DNS-server staat en een 1-bit recursiemogelijkheids-vlag die in het antwoordbericht staat na een verzoekbericht met een recursienoodzaak-vlag
- Vraagveld bevat informatie over het verzonden verzoek:
  1. Bevat een naamveld met de naam waarvoor het IP-adres wordt gezocht
  2. Bevat een typeveld met het type verzoek (Type A, Type NS...)
- In een antwoordbericht staat in het antwoordveld de bronrecords van het oorspronkelijke DNS-verzoek
- Het autorisatieveld bevat gegevens van andere authoritative DNS-servers
- Het aanvullingsveld bevat aanvullende bronrecords (Het antwoordveld in een antwoord op een MX-verzoek bevat een bronrecord met de canonieke hostnaam van de mailserver en het aanvullingsveld bevat het een Type A record met het IP-adres van de canonieke hostnaam van de mailserver)

## DNS-Security

### DDOS-attack:

- root servers belasten met verkeer
- Tot op heden niet succesvol

- Verkeersfiltering
- Lokale DNS-servers cache-IP's van TLD-servers waardoor rootserver bypassed

#### **TLD servers bombarderen:**

- Kan gevaarlijker zijn maar moeilijker → TLD servers niet zo gemakkelijk bypassed

#### **Aanvallen omleiden**

- Man in the middle → DNS queries intercepten
- DNS poisoning → valse afhankelijkheden verzenden naar dns-server, die cachten

#### **DNS voor DDoS exploiteren**

- Queries versturen met vervalste bronadres → target IP
- Vereist versterking

#### **Laatste 2 vormen DNSSEC**

## 2.5 Peer-to-peer bestandsdistributie

### P2P file distributie: BitTorrent

Peer neemt deel aan torrent → meldt bij tracker → peer informeert tracker met regelmaat of nog aanwezig in torrent → nieuwe peer → tracker random # peers → verzendt IP-adressen van # peers → naar nieuwe peer → proberen TCP-verbinding met peers op lijst → bepaald tijdstip → elke peer → # chunks bestand → verschillende peers → verschillende verzamelingen chunks hebben → na een bepaalde tijd → gebruiker vraagt elke peer om lijst met chunks die ze hebben → gebruiker vraagt de "missing chunks" van de peerst → zeldzaamste eerst of **Rarest first**

### Tit-for-tat principe:

Gebruiker stuurt chunks naar die vier peers die momenteel haar chunks in het hoogste tempo verzenden → andere peers gestikt door gebruiker (ontvangen geen chunks van gebruiker) → herbeoordeeld top 4 elke 10 seconden

Elke 30 seconden → selecteert willekeurig een andere peers, begint met verzenden van chunks → optimistisch unchoked deze peer → nieuwe gekozen peer kan lid worden van top 4

# 2.6 Videostreaming en content distribution networks

## Internetvideo

- Video → sequentie van afbeeldingen → afgebeeld constante snelheid Bijvoorbeeld: 24 afbeeldingen / seconde
- Digital image → array van pixels Elke pixel gerepresenteerd door bits
- Coding: Gebruikt redundantie binnen en tussen afbeeldingen om # bits te verminderen die worden gebruikt om afbeeldingen te gebruiken Spatial → binnen afbeelding Temporal → van 1 afbeelding naar volgende
- CBR → constant bit rate → video encoding rate is fixed
- VBR → variable bit rate → wijzigingen in videocoderingssnelheid als hoeveelheid spatial, temporal codering wijzigt

### 2.6.1.1 Streaming stored video

#### **Simpel Scenario:**

#### **Hoofddoelen:**

- Server to client bandbreedte varieert over tijd met veranderende netwerkcongestieniveaus → ( in huis, in acces network, network core, video server)
- pakketverlies en vertraging als gevolg van congestie zullen de play-out vertragen of resulteren in een slechte videokwaliteit

### 2.6.1.2 Streaming stored video : challenges

Continuous playout constraint → zodra play-out van client begint → afspelen overeenkomen met oorspronkelijke timing → **maar** network delays variabel (jitter) → heeft buffer aan clientzijde nodig om aan play-out vereisten te voldoen

Andere challenges

Client interactiviteit → pause, voortspoelen, terugspelen, verder in video gaan → video packets loss mogelijk → opnieuw verzonden

# HTTP-streaming en DASH

**D**ynamic **A**daptive **S**treaming over **H**TT**P**

- Server
  - Deelt video bestand in meerdere chunks
  - Elke chunk stored, gecodeerd met verschillende snelheden
  - **Manifestbestand** : biedt URL's voor verschillende chunks
- Client
  - Meet periodiek server-naar-client bandbreedte
  - Adviesmanifest, vraagt 1 chunk tegelijk
    - Kiest voor max coderingssnelheid duurzaam gezien huidige bandbreedte
    - Kan verschillende coding rates kiezen op verschillende punten in tijd → afhankelijk vrije bandbreedte

STREAMING VIDEO = CODERING + DASH + PLAYOUT BUFFERING

## Content Distribution Networks (CDNs)

Challenge: Hoe content streamen naar 100 tot 1000'en gebruikers tegelijk

- **Optie 1: 1 grote server**
  - Single point of failuire
  - Punt van netwerkcongestie
  - Lange weg naar clients die verweg zijn
  - meerdere kopieën van video verzonden via uitgaande link
- Deze oplossing **schaalt niet**
- Optie 2: meerdere kopieën van video's opslaan/weergeven op meerdere geografisch gedistribueerde sites → **(CDN)**
  - Enter deep → push CDN servers diep in veel access networks → dichtbij users



- Bijvoorbeeld: Akamai → 240k in meer dan 120 landen
- Bring home: kleinere nummers (10's) of grotere clusters in POP's dichtbij access netwerken
- CDN → slaagt kopieën van content op aan CDN nodes
- Subscriber vraagt content van CDN → gericht nabijgelegen kopie & haalt inhoud op
- kan een andere kopie kiezen als het netwerkpad overbelast is

# Hoofdstuk 8: Security in computer networks

## 8.1 Wat is netwerkbeveiliging

1. **Vertrouwelijkheid** : Alleen zender & beoogde ontvanger inhoud van verzonden bericht begrijpen
2. **Berichtintegriteit** : de afzender en ontvanger zeker zijn dat inhoud van communicatie niet wordt gewijzigd
3. **Authenticatie op eindpunt**: zender & ontvanger identiteit andere partij vaststellen zeker te zijn dat ander is wie hij beweert
4. **Operationele beveiliging**: bijna alle organisaties hebben netwerken aangesloten op het openbare internet. Deze netwerken kunnen daarom mogelijk worden aangetast.

## 8.2 Principes van cryptografie

**Verzender ( X )** verstuurt bericht naar **ontvanger (Y)**

1. X gebruikt sleutel  $K^{**}$   $A^{**} \rightarrow$  invoer versleutelalgoritme
  2. Versleutelalgoritme gebruikt sleutel  $\rightarrow$  onversleutelde bericht  $m \rightarrow$  versleutelde tekst  $\rightarrow K^{**} A^{****}(m)^{**}$
  3. KA onderling afspreken
  4. Y ook sleutel  $K^{**}$   $B^{****} \rightarrow$  invoer onversleutelalgoritme  $\rightarrow$  versleutelde bericht  $X \rightarrow$  plaintext
  5. Y ontvangen versleutelde bericht  $KA(m) \rightarrow$  ontsleutelen  $\rightarrow$  berekenen van  $Kb(Ka(m)) = m$
- Symmetrische sleutelsystemen  $\rightarrow$  sleutels X & Y identiek en geheim
  - Openbare sleutelsystemen  $\rightarrow$  2 verschillende sleutels  $\rightarrow$  1 v/d 2 zowel bij X als Y bekend

# 8.2.1 Cryptografie met symmetrische sleutels

1. **First** → Caesar cipher → elke letter in platte tekst → letter → k-letters in alfabet te vervangen
2. **Daarna** → monoalfabetisch cijfer → lettervervanging maar moet uniek zijn

**Bruteforce-benadering** → uitproberen alle  $10^{26}$  → teveel werk

- **Aanval met alleen versleutelde tekst** → indringer beschikt alleen over onderschepte versleutelde tekst → niet over informatie van inhoud → onversleutelde bericht
- **Aanval met bekende onversleutelde tekst → Indringer (Z)** → kent enkele combinaties van onversleutelde & versleutelde tekst
- **Aanval met specifieke onversleutelde tekst** → Z kiest onversleuteld bericht → corresponderende versleutelde tekst krijgen → als Z → X een bepaalde zin kan laten verzenden → versleutelmethode verbroken

**Polyalfabetische codering** → verschillende monoalfabetische ciphers gebruikt → afwisselend ingezet → bepaalde positie in onversleutelde bericht te versleutelen

## Block ciphers (DES = data encryption standard, 3DES, AES= advanced encryption standard)

**2 categorieën** van symmetrische versleuteltechnieken

- Stream cipher
- Block cipher

### Blockciphers

1. versleutelen bericht → verwerkt blokken k bits
2. **IF**  $k = 64$  → bericht opgesplitst in 64 blokken → elk blok onafhankelijk versleuteld
3. Codering 1 op 1 toewijzing om k-bit blok cleartext toe te wijzen aan k-bit blok Ciphertext

### Hoeveel mogelijke verwijzingen?

- $k = 3$  dan zijn er 23 mogelijke ingangen. Deze ingangen kunnen in 8 worden gepermuteerd! = 40 320.

Zeer moeilijk uit te voeren. Voor  $k = 64$  moeten Alice en Bob een tabel onderhouden met  $2^{64}$  invoerwaarden → onmogelijk → blokcoderingen meestal functies die willekeurig gepermuteerde tabellen simuleren.

## Cipher-block chaining (CBC)

Blockcipher → twee of meer blokken identiek zijn → aanvaller mogelijk **cleartext raden** en misschien het hele bericht decoderen. → solution → willekeur in ciphertext

### Werkwijze:

1. voor bericht versleutelt → genereert Afzender **een willekeurige k-bit string, initialisatievector (IV) =  $c(0)$**  genoemd → afzender stuurt IV naar ontvanger in leesbare vorm
2. Eerste blok berekent de afzender  **$m(1) + c(0)$  → exclusieve OR van eerste blok onversleutelde tekst & IV. → verzender verwerkt met BC → bijhorende blok als versleutelde tekst  $c(1) = Ks(m(1) + c(0))$**  → verzender versleutelde blok ( $c(1)$ ) naar ontvanger
3. Voor het i-blok genereert de afzender  **$c(i) = Ks(m(i) + c(i-1))$**

## 8.2.2 Cryptografie met openbare sleutel

- 2 partijen delen gedeeld geheim → Symmetrische sleutel voor encryptie & decryptie

## Diffie-Hellman key exchange

1. Alice haalt Bob's publieke sleutel
2. Alice versleutelt bericht ( $m$ ) aan Bob → door public key van Bob en bekend encryptie-algoritme  $K+B(m)$ .
3. Bob ontvangt → gecodeerde bericht van Alice → gebruikt private key & bekend decoderingsalgoritme → gecodeerde bericht decoderen
4. Bob berekent  $K-B(K+B(m))$ .
5. Berekenen van  $K-B(K+B(m))$  resulteert in  $m$

**Note :** each party generates a public/private key pair and distributes the public key. After obtaining an authentic copy of each other's public keys, Alice and Bob can compute a shared secret offline. The shared secret can be used, for instance, as the key for a symmetric cipher.

# RSA

## Maken van publieke en private RSA keys:

1. Kies 2 grote priemgetallen  $p$  &  $q \rightarrow$  hoe groter de waarden hoe moeilijker RSA-algoritme te kraken
  2. Bereken  $n = p * q$
  3. Bereken  $z = (p - 1) * (q-1)$
  4. Kies nummer  $e$ , dat kleiner is dan  $n$  & geen factoren (buiten 1 ) gemeenschappelijk heeft met  $z$
  5. Zoek een getal  $d$ , zodanig dat  $ed - 1$  precies deelbaar is door  $z$ . Wij kiezen  $d$  zodanig dat  $e*d \bmod z = 1$
  6. Openbare sleutel (K+B) is het paar van de getallen  $(n, e)$
  7. The private key(K-B) is the pair of the numbers  $(n, d)$
- Encrypteren  $\rightarrow C = me \bmod n$
  - Decrypteren  $\rightarrow m = cd \bmod n$ . Which requires the use of the private key  $(n, d)$

**NOTE:** Diffie-Hellman niet zo veelzijdig als RSA omdat het niet gebruikt kan worden om berichten met willekeurige lengte te coderen  $\rightarrow$  toegepast om symmetrische sessiesleutel tot stand te brengen  $\rightarrow$  daarna coderen berichten

# Berichtintegriteit en digitale handtekeningen

## 8.3.1 Cryptografische hashfuncties

- Rekentechnisch onmogelijk  $\rightarrow$  2 verschillende berichten  $x$  en  $y$  te vinden  $\rightarrow$  waarbij

$$H(x) = H(y)$$

- Onmogelijk voor indringer  $\rightarrow$  bericht vervangen door ander bericht dat beveiligd is met hashfunctie
- **Om veiligheidsredenen** hebben we een krachtigere hash-functie nodig dan een checksum  $\rightarrow$  het **MD5-hash-algoritme**. Het berekent een 128-bits hash in een proces in **vier stappen** dat bestaat uit:

- een opvulstap
- een toevoegstap
- een initialisatie van een accumulator
- een laatste loopingstap waarin de blokken van 16 woorden van het bericht in vier rondes worden verwerkt
- Het tweede belangrijke hash-algoritme is het **Secure Hash Algorithm (SHA1)**. Het produceert een 160-bits berichtcijfer. De langste output maakt SHA1 veiliger.

## 8.3.2 Berichtauthentificatiecode

berichtintegriteit uit te voeren → Alice en Bob → naast gebruik van cryptografische hashfuncties, → gedeeld geheim nodig → niets meer dan een reeks bits = verificatiesleutel. → door gedeelde geheim kan berichtintegriteit als volgt worden uitgevoerd:

- Alice maakt bericht  $m$ , samenvoegt  $s$  met  $m$  om  $m + s$  te maken en berekent de hash  $H(m+s)$ . → berichtverificatiecode (MAC = **Message Authentication Code**)
- Alice voegt vervolgens de MAC toe aan het bericht  $m$ , maakt een uitgebreid bericht  $(m, H(m+s))$  en stuurt het naar Bob
- Bob ontvangt het bericht  $(m, h)$  en weet  $s$ , berekent de MAC  $H(m+s)$ . Als  $H(m+s) = h$ , dan is alles in orde met de boodschap

## 8.3.3 Digitale handtekeningen

### Certificering van openbare sleutels

Certificate authority → echtheid identiteiten authenticceert & certificaten uitgeeft

1. Een certification authority controleert of een entiteit is wie het zegt dat het is.
2. certificeringsinstantie identiteit van entiteit verifieert, maakt certificeringsinstantie een certificaat → openbare sleutel van de entiteit aan de identiteit bindt → certificaat bevat openbare sleutel + wereldwijd unieke identificerende informatie over eigenaar van openbare sleutel. Het certificaat digitaal ondertekend door certification authority.

## Authenticatie op het eindpunt

Eindpuntverificatie → proces waarbij de ene entiteit zijn identiteit aan een andere entiteit bewijst via een computernetwerk.

## 8.4.1 Authenticatieprotocol ap1.0

Trudy (indringer) stuurt bericht naar Bob → zegt "ik ben Alice" → Bob weet niet of het werkelijk Alice is  
8.4.2 Authenticatieprotocol ap2.0 Alice bekend netwerkadres waaruit ze altijd communiceert → Bob probeert Alice te verifiëren → bronadres IP-datagram met verificatiebericht overeenkomt met het bekende adres van Alice.

niet moeilijk om IP-datagram te maken, zet elk IP-bronadres dat we willen in het IP-datagram.

## 8.4.3 Authenticatieprotocol ap3.0

Het wachtwoord is een gedeeld geheim tussen de authenticator en de persoon die wordt geverifieerd.

- Alice stuurt haar geheime wachtwoord naar Bob.
- De beveiligingsfout hier is dat als Trudy Alice's communicatie afluistert, ze Alice's wachtwoord kan leren.

## 8.4.4 Authenticatieprotocol ap3.1

Door het wachtwoord te versleutelen → voorkomen Trudy Alice's wachtwoord leert → aannemen dat Alice en Bob een symmetrische geheime sleutel delen,  $K_A - B$ , dan kan Alice het wachtwoord versleutelen en haar identificatiebericht en het gecodeerde wachtwoord naar Bob sturen. Bob decodeert vervolgens het wachtwoord en verifieert Alice.

**De fout: the playback attack :** Trudy hoeft alleen maar de communicatie van Alice af te luisteren, de gecodeerde versie van het wachtwoord op te nemen en de gecodeerde versie van het wachtwoord af te spelen naar Bob om te doen alsof ze Alice is.

## 8.4.5 Authenticatieprotocol ap4.0

**Een nonce** is een getal dat een protocol maar één keer in een leven zal gebruiken. Dat wil zeggen dat zodra een protocol een nonce gebruikt, het het nummer nooit meer zal gebruiken. Onze ap4.0 gebruikt een nonce in als volgt:

1. Alice verzendt bericht 'ik ben Alice' aan Bob
2. Bob kiest een nonce en verzendt die naar Alice

3. Alice versleutelt de nonce met de symmetrische sleutel van Alice en Bob,  $K_{A-B}$ , en stuurt de gecodeerde nonce  $K_{A-B}(R)$  terug naar Bob.
4. Bob ontsleutelt het ontvangen bericht. Als de gedecodeerde nonce gelijk is aan de nonce die hij Alice stuurt, dan is Alice geauthenticeerd

## 8.5 E-mail beveiligen

### 8.5.1 Ontwerp van veilige e-mail

#### Vertrouwelijkheid (Systeem 1)

- Bericht versleutelen → symmetrische sleuteltechnologie (DES) → ontvanger bericht ontsleutelen
- Alternatief → cryptografie openbare sleutel (RSA)
- Om dit efficiëntieprobleem op te lossen, maken we **gebruik van een sessie key**:
  1. Alice selecteert een willekeurige symmetrische sessiesleutel ( $K_s$ )
  2. Versleutelt haar bericht  $m$  met de symmetrische sleutel
  3. Versleutelt de symmetrische sleutel met Bob's openbare sleutel  $K_B$
  4. Voegt het gecodeerde bericht en de gecodeerde symmetrische sleutel samen om een pakket te vormen
  5. stuurt het pakket naar Bob's e-mailadres. **(zie p 586 onderaan voor schema)**
- Bob ontvangt pakket:
  1. Gebruikt geheime sleutel  $K_B$  → verkrijgen symmetrische sleutel  $K_s$
  2. Symmetrische sleutel  $K_s$  → ontsleutelen bericht

#### Sessie key = inefficiënt

##### 1. Berichtintegriteit

- Digitale handtekeningen:
  1. Alice past hashfunctie  $H$  toe
  2. Versleutelt resultaat met hashfunctie → met geheime sleutel  $K_A$  → digitale handtekening
  3. Oorspronkelijke bericht samen met handtekening → 1 pakket
  4. Verzend pakket → e-mail Bob
- Bob ontvangt het pakket
  1. Openbare sleutel Alice  $K_A$  → op ondertekende hash
  2. Vergelijkt resultaat bewerking met eigen hash  $H$  van bericht

#### The 2 combined:

- Alice maakt eerst een voorlopig pakket, dat bestaat uit haar originele bericht samen met een digitaal ondertekende hash van het bericht
- Vervolgens behandelt ze dit voorlopige pakket als een bericht op zich en stuurt dit nieuwe bericht via **de stappen van de afzender van (a)**, waardoor een nieuw pakket wordt gemaakt dat naar Bob wordt verzonden (**zie schema p 586\*\*** )\*\*

## 8.5.2 PGP

PGP-software gebruikt:

- MD5 of SHA → berekenen hash
- CAST, Triple-DES of IDEA → versleutelen symmetrische sleutel
- RSA versleuteling openbare sleutel

Als PGP installed:

1. Openbaar sleutelpaar voor gebruiker
  2. Openbare sleutel → website gebruiker of openbare sleutelservers
- Persoonlijke sleutel beschermd → wachtwoord

PGP → mogelijkheid bericht digitaal ondertekenen

## 8.6 TCP-verbindingen beveiligen

Noodzaak SSL:

- Geen versleuteling → indringer Bobs bestelling onderscheppen → creditcardgegevens achterhalen
- Integriteit gegevens niet gecontroleerd → bestelling veranderen
- Server niet geauthenticeerd → Valse website maken → gegevens stelen

SSL lost problemen op door volgende bovenop TCP uit te voeren:

1. Vertrouwelijkheid
2. Gegevensintegriteit
3. Serverauthenticatie
4. Clientauthenticatie

SSL → eenvoudige API vergelijkbaar API van TCP



# 8.6.1 Het hele verhaal, maar vereenvoudigd

## Fase 1: Handshake

1. Bob TCP-verbinding met Alice maken
2. Verzekeren dat Alice echt Alice is
3. Alice geheime mastersleutel zenden → Alice & Bob gebruiken → symmetrische sleutel genereren → nodig voor SSL

## Fase 2: Verkrijgen van een sleutel

Alice & Bob moeten MS gebruiken om vier sleutels te genereren:

- EB: **session encryption key** for data sent from Bob to Alice
- MB: **session MAC key** for data sent from Bob to Alice
- EA: session encryption key for data sent from Alice to Bob
- MA: session MAC key for data sent from Alice to Bob

## Fase 3: Gegevensoverdracht

Geen goed idee → integriteit alle gegevens tijdens hele sessie dat Bob gestuurd heeft controleren

1. SSL splitst gegevensstream → records
2. SSL voegt berichtauthenticatiecode toe aan elk record → versleutelt combinatie
3. Maken van berichtauthenticatiecode → Bob hashfunctie toe → combinatie recordgegevens & sleutel Mb
4. Versleutelen → Bob gebruikt sessievercijfersleutel Eb

**Man in the middle attack ( MITM )** : kansegmenten in de TCP-stream vervangen, invoegen en verwijderen tussen Alice en Bob.

Aannemen dat elk TCP-segment exact 1 record verpakt is → kijken hoe Alice segmenten verwerkt

1. TCP in host Alice → denkt alles is OK → 2 records aan SSL-sublaag
2. SSL in host Alice → 2 records ontsleutelen
3. SSL in host Alice → berichtauthenticatiecode in elk record gebruiken → integriteit van gegevens in 2 records
4. SSL → ontsleutelde bytestream van 2 records doorgeven → applicatielaag → door Alice ontvangen bytestream → gevolg verwisseling records → niet in juiste volgorde

**Oplossing: gebruik volgnummers.**

1. Bob onderhoudt een reeksnummerteller, die begint bij nul en wordt verhoogd voor elke SSL-record die hij verzendt.
2. Wanneer hij de MAC berekent, neemt hij het volgnummer op in de MAC-berekening.

Zo is  $MAC = \text{hash van gegevens} + \text{MAC-toets} + \text{huidig volgnummer}$ .

Alice kan Bob's volgnummers opsporen, zodat ze de gegevensintegriteit kan verifiëren.

## SSL record

Bestaat uit → typeveld, versieveld, lengteveld, gegevensveld & berichtauthenticatiecode

# 8.6.2 Het hele verhaal, maar wat minder vereenvoudigd

## SSL handshake

Alice & Bob begin SSL-sessie zelf afspraken maken over cryptografische algoritmen → aka handshakeprocedurefase → + Alice & Bob zenden elkaar nonces toe → gebruikt bij maken van sessiesleutels (EB,MB,EA,MA)

Handshakeprocedure bij SSL:

1. Client verzendt lijst → versleutelalgoritmen die hij ondersteunt & zelfgekozen nonce
2. Server kiest uit ontvangen lijst algoritme voor → symmetrische sleutel, openbare sleutel & berichtauthenticatiecode → server verstuurt bericht met voorkeuren, certificaat & zelfgekozen nonce
3. Client authenticceert certificaat + berekent openbare sleutel server + genereert geheime pre-mastersleutel (PMS) → versleutelt PMS met openbare sleutel server → verzendt versleutelde PMS naar server
4. Afsproken functie bepalen sleutel → berekenen client & server onafhankelijk → geheime mastersleutel met PMS & nonces → geheime mastersleutel in stukken gehakt → 2 coderingssleutels & 2 berichtauthenticatiecodes genereren → gekozen symmetrische codering werkt met cipher-block-chaining → 2 initialisatievectoren gemaakt → geheime mastersleutel → daarna alle berichten versleuteld & geauthenticceerd
5. Client verzendt berichtauthenticatiecode → alle handshakeprocedureberichten
6. Server verzendt berichtauthenticatiecode → alle handshakeprocedureberichten

Alleen nonces → niet mogelijk "replay attack" te voorkomen

# Verbinding verbreken

Iemand geeft in het typeveld aan of de record dient om de SSL-sessie te beëindigen. Door zo'n veld op te nemen, zou Alice weten dat als ze een TCP FIN zou ontvangen voordat ze een SSL-sluitingsrecord zou ontvangen, ze weet dat er iets grappigs aan de hand is.

## 8.7 Beveiliging op netwerklaag: IPsec & VPN

### 8.7.1 IPsec & VPN

Met VPN → interne dataverkeer van de instelling verzonden via het publiekelijk toegankelijke internet in plaats van via een fysiek gescheiden netwerk. → dataverkeer eerst versleuteld

**CHECK PAGINA 597 IN HANDBOEK VOOR AFBEELDING**

### 8.7.2 Authentication header- protocol en het encapsulation security payload-protocol

Protocolsuite IPsec → Authentication header (**AH-protocol**) & encapsulation security payload (**ESP-protocol**)

**AH-protocol** → bronauthenticatie & gegevensintegriteit maar geen vertrouwelijkheid **ESP-protocol** → bronauthenticatie & gegevensintegriteit & vertrouwelijkheid Vertrouwelijkheid essentieel bij VPN & andere IP-sec applicaties

### 8.7.3 Beveiligingsassociaties

IPsec-datagrammen → verzonden tussen 2 netwerkentiteiten → voor bronentiteit IPsec-datagrammen verstuurt → 2 entiteiten → logische verbinding tot stand = **beveiligingsassociatie**

→ logische simplexverbinding → unidirectioneel van bron naar bestemmingsentiteit → beide entiteiten beveiligde datagrammen naar elkaar willen verzenden → noodzakelijk om 2 beveiligingsassociaties tot stand te brengen → voor elke richting 1

## 8.7.4 Het IPsec-datagram

2 verschillende packetvormen → **tunnelmodus & transportmodus** **PAGINA 589 HANDBOEK**

- R1 voegt achter aan oorspronkelijke IPv4-datagram een ESP-trailerveld toe
- R1 versleutelt resultaat met behulp van het algoritme & de sleutel die voor de beveiligingsassociatie zijn overeengekomen
- R1 voeg aan voorkant van versleutelde geheel → veld toe → ESP-header → Resulterende pakket → enclilada
- R1 voegt berichtauthenticatiecode toe aan achterkant van versleutelde geheel → alles te samen → inhoud van payloadveld
- R1 creëert een nieuwe IP-header met alle klassieke IPv4-headervelden → voegt voor payloadveld toe

## 8.7.5 sleutelbeheer in IPsec (IKE)

IKE kent twee fase

Fase1:

- Vaststellen bidirectionele IKE SA

*opmerking: IKE SA anders dan IPsec SA ook bekend als ISAKMP security association*

Fase 2:

- ISAKMP wordt gebruikt om veilig te onderhandelen over IPsec-paar SA's

fase 1 heeft twee modi: agressieve modus en hoofdmodus

- Agressieve modus gebruikt minder berichten
- Hoofdmodus biedt identiteitsbescherming en is flexibeler

IKE-berichtenuitwisseling voor algoritmen, geheime sleutels, SPI-nummers

- AH- of ESP-protocol (of beide) AH biedt integriteit, bronverificatie
- ESP-protocol (met AH) biedt bovendien versleuteling

- IPsec-peers kunnen twee eindsystemen zijn, twee routers/firewalls, of een router/firewall en een eindstelsysteem

## 8.8 Securing wireless LANs

### 8.8.1 Wired equivalent privacy

LEER VANUIT SLIDES  $\rightarrow$  DUIDELIJKER

## 8.9 Operationele beveiliging: firewalls & intrusion- detectionssystemen

### 8.9.1 Firewalls

3 doelen:

- Alle dataverkeer van buiten naar binnen & omgekeerd passeert firewall
- Alleen geauthenticeerd dataverkeer  $\rightarrow$  gedefinieerd in lokale beveiligingspolicy's mag passeren
- Firewall kan niet zelf benaderd worden

## Traditionele packetfilters

Filterbeslissingen meestal genomen op basis van:

- IP-bronadressen of IP-bestemmingsadressen
- Typeveld in IP-datagram
- TCP- of UDP-bronpoorten en -bestemmingspoorten
- TCP-vlagbits: SYN, ACK,...
- ICMP-berichttype
- Verschillende regels voor datagrammen die netwerk binnenkomen en verlaten
- Verschillende regels voor verschillende routerinterfaces

Organisatie kan filteren op:

- TCP-SYN-segmenten → geen inkomende TCP-verbindingen toestaan behalve voor publieke webserver → alle TCP-SYN-segmenten blokkeren behalve → TCP-SYN-segmenten bestemmingspoort 80 & bestemmings-IP overeenkomt met webserver
- TCP-ack-bit → interne clients verbinden met externe servers → externe clients niet verbinden met interne servers
  - 2e mogelijkheid → DNS-packets netwerk kunnen binnenkomen & verlaten → blokkeert al het dataverkeer → behalve webdataverkeer binnen organisatie & DNS-dataverkeer

Filterpolicy kan gebaseerd zijn op combinatie van adressen & poortnummers

## Stateful packetfilters

Bewaken alle bestaande TCP-verbindingen → firewall kan nieuwe verbinding detecteren → door 3-wayhandshake (SYN, SYNACK & ACK) → + eind verbinding detecteren → FIN-packet → firewall kan ook veronderstellen → verbinding niet meer nodig is → geen activiteit

Packet bereikt firewall

1. Firewall controleert lijst met **toegangsbeheer** ( traditionele packetfilters )
2. Verbindingstabel controleren voor packet in netwerk van organisatie kan komen
3. Controleert verbindingstabel → geen deel van lopende TCP-verbinding → weigert
4. IF webserver stuurt packet terug → firewall controleert tabel → overeenkomstige verbinding → packet passeren

## Application gateway

Firewalls moeten packetfilters combineren met applicatiegateways → die kijken verder dan headers van IP, TCP & UDP → beslissingen op basis van applicatiegegevens **Applicatiegateway** → applicatiespecifieke server die door alle applicatiegegevens gepasseerd moet worden → verschillende applicatiegateways kunnen op dezelfde host uitgevoerd worden → elke gateway afzonderlijke server met eigen processen

**Stel:**

Firewall → geselecteerde groep interne gebruikers → Telnet-verbindingen met externe netwerken → tegelijk voorkomen → externe clients → Telnet-verbinding maken met interne host

### Stel nu:

Interne gebruiker wil verbinding tot stand brengen met buitenwereld

1. Gebruiker Telnet-sessie starten met applicatiegateway → op gateway draait applicatie → wacht voor inkomende Telnet-sessies tot stand komen
2. Applicatie vraagt username & password
3. IF informatie = correct → applicatiegateway checkt IF gebruiker = gerechtigd is → als dat het geval is
4. Gateway vraagt gebruiker → hostname externe host ingeven
5. Gateway Telnet-sessie → tussen gateway & externe host
6. Gateway verzendt alle gegevens afkomstig van externe host naar gebruiker & omgekeerd

## 8.9.2 Intrusion-detectionsystems

- Intrusiondetectionsysteem (**IDS**) → waarschuwt wanneer mogelijk schadelijk dataverkeer detecteerd
- Intrusion-preventionsysteem (**IPS**) → Apparaat dat schadelijk dataverkeer blokkeert

### IDS + IPS = IDS

Organisatie → meerdere IDS's sensoren implementeren → meestal samenwerkend → sturen verdachte verkeersactiviteit → centrale IDS-processor → verzamelt info → alarmen verzendt naar netwerkbeheerder wanneer nodig

### Pagina 619 afbeelding 3.6

Organisatie → 2 delen opgesplitst

1. Streng beveiligd deel → afgeschermd door → packetfilter & applicatiegateway → bewaakt door IDS sensoren
2. Minder streng beveiligd deel → **gedemilitariseerde zone (DMZ)** → alleen beveiligd door packetfilter maar ook bewaakt door sensoren IDS *(is een netwerksegment dat zich tussen het interne en externe netwerk bevindt. Het externe netwerk is meestal het Internet. Een DMZ is feitelijk een andere naam voor een extranet, een gedeelte van het netwerk dat voor de buitenwereld volledig toegankelijk is. Op het netwerksegment van de DMZ zijn meestal servers aangesloten die diensten verlenen die vanuit het interne en externe netwerk aangevraagd kunnen worden)*

Sensoren voor IDS → verderop in systeem → elke sensor deel van dataverkeer → gemakkelijker taak uitvoeren

## IDS systemen in 2 categorieën

### 1. Systemen die werken met handtekeningen

- database met aanvalskennmerken (handtekeningen) → elke handtekening → verzameling regels → gebruikt verdachte activiteit → kan lijst met kenmerken 1 packet → vergelijkt packets met handtekening in database → overeenkomst in database → waarschuwing

### Beperkingen:

1. dit soort IDS → alleen als voorkennis over aanval is → gebruikt nauwkeurige handtekening te vervaardigen → blind als er nieuwe aanvallen zijn
2. packet → zelfs als er bekende handtekening is → niets te maken met een aanval → false-positive warning
3. IDS kan overvoerd geraken → elk packet vergeleken moet worden → uitgebreide verzameling handtekeningen → kan zover komen dat IDS schadelijke packets niet detecteert
4. **Op anomalie gebaseerde systemen**
  - Creëert profiel → normale verloop → gecontroleerde dataverkeer → zoekt packetstreams statistisch ongebruikelijk zijn
  - Niet afhankelijk van voorkennis bestaande aanvallen
  - Wel moeilijk → efficiënt onderscheid maken → normaal dataverkeer & statistisch ongebruikelijk verkeer

## Snort

Maakt gebruik van generieke sniffingsinterface, libpcap

Enorme groep gebruikers & beveiligingsexperts → houden handtekeningdatabase actueel

## 9 Multimedianeetwerken

### 9.1 Multimedianeetwerkapplicaties



## 9.1.1 Eigenschappen van video

- **Hoge bit rate** : 100kbps voor low-quality video conferencing → 3 Mbps streaming high-definition movies. → bitsnelheidvereisten van video belangrijk ontwerpen netwerkvideotoepassing
- Een niet-gecomprimeerde, digitaal gecodeerde afbeelding → reeks pixels → elke pixel gecodeerd in aantal bits om luminantie en kleur weer te geven.
- 2 Types redundantie in video → exploited door video compressie
  1. **Tijdelijke redundantie** → werkt met verschillen tussen opeenvolgende afbeeldingen
  2. **Spatial redundancy** → redundantie binnen een bepaalde afbeelding

## 9.1.2 Eigenschappen van audio

- Digitale audio → lagere bandbreedte nodig dan video → Pulse Code Modulation (PCM) → coderen spraak → 8000 samples/seconde & 8 bits per sample → bitsnelheid 64 kbps
- Compressietechniek voor stereomuziek → MP3 → bitsnelheid 128 kbps → geluidskwaliteit slechter
- Advanced Audio Coding (AAC) → meerdere versies vooropgenomen audio stream kan gemaakt worden → elke klein verschil bit rate

## 9.1.3 Soorten

## multimedianeetwerkapplicaties

## Streamen van opgeslagen audio / video

Streamen van opgeslagen video 3 belangrijke onderscheidende kenmerken

- Streamen → client begint video → binnen enkele seconden na ontvangst weer te geven → client video weergeeft → bepaalde plaats in opname & tegelijk latere delen van die opname ontvangt van server = **streamen**
- Interactiviteit → media vooraf opgenomen → gebruiker weergave onderbreken, doorspoelen,... → voor aanvaardbare gebruikskwaliteit → tijd tussen moment gebruiker actie start & uitvoering actie → enkele seconden
- Continue weergave → zelfde snelheid weergegeven als origineel → gegevens op tijd van server ontvangen → voor moment client moet weergeven → anders haperingen

- Gemiddelde doorvoercapaciteit → netwerk streamapplicatie → gemiddelde doorvoercapaciteit bieden → ten minste even groot als bitsnelheid video

# VOIP

Timing is belangrijk → spraak- en videoapplicaties → vertragsingsgevoelig → meeste multimedianeetwerkapplicaties → bestand tegen een zekere mate van gegevensverlies → resulteert in korte onderbrekingen van audio of video

## Streamen van live audio & video

Meestal via CDN's → zelfde snelheid weergegeven als origineel → gegevens op tijd van server ontvangen → voor moment client moet weergeven → anders haperingen → omdat evenement = live → vertraging probleem zijn → timingereisen minder streng dan voor spraakgebrekken

## 9.2 Streamen van opgeslagen video

2 belangrijke voordelen bufferen door client

1. Fluctuaties in vertraging tussen server en client opvangen
2. Banbreedte tussen server & client → daalt onder sneheid → waarmee videocontent wordt weergegeven → blijven kijken zolang buffer van clientcomponent niet leegraakt

### 9.2.1 Streamen met UDP

Kleine buffer op clientcomponent van applicatie gebruikt → net groot genoeg voor minder dan seconde video → server die video aan UDP-verbinding vertrouwt → stukjes video verpakken in transportpackets speciaal ontworpen voor transporteren audio & video → Realtime Transport Protocol ( RTP )

Server & client → onderhouden verbinding voor videostream → ook afzonderlijke besturingsverbinding die door client wordt gebruikt geven opdrachten

Systeem 3 belangrijke nadelen:

1. Streamen met constante snelheid → voor continue weergave → problemen opleveren als gevolg van onvoorspelbare & wisselende beschikbare bandbreedte
2. Streamen met UDP → server nodig om media te besturen → interactieve verzoeken tussen client en server afhandelen & toestand client bewaren → voor elke sessie
3. Veel firewalls geconfigureerd om UDP verkeer te blokkeren

## 9.2.2 Streamen met HTTP

Video in HTTP server → gewoon bestand met specifieke URL → gebruiker wil video zien

1. Client start TCP-verbinding met server
2. Verzendt HTTP-GET-bericht
3. Server verzendt video bestand in HTTP-antwoordbericht
4. Clientcomponent applicatie verzamelt bytes in buffer
5. Zodra # bytes in buffer > bepaalde drempelwaarde
6. Client begint met weergave + videoframes periodiek uit buffer opgehaald & gecomprimeerd

Packets → vertraagd als gevolg opnieuw verzenden packets

Gebruik van HTTP over TCP → firewalls en NAT's gemakkelijker gepasseerd kunnen worden → van het UDP-dataverkeer tegen houden → HTTP-dataverkeer door te laten → streamen HTTP geen mediabesturingsserver nodig (RTSP-server) → kosten lager → meeste videostreamapplicaties werken met HTTP over TCP als streamprotocol

## Prefetchen van video

Client probeert video downloaden → snelheid hoger dan weergavesnelheid → voorraad krijgen van videoframes → toekomst worden weergegeven

## Buffers van clientcomponent van de applicatie & TCP-buffers

Volledige client applicatie buffer → legt indirect limiet op rate → video verstuurd van server naar client wanneer streamen over HTTP

# Analyse van clientcomponent van applicatie en TCP-buffers

If beschikbare rate  $<$  video rate  $\rightarrow$  continue weergave afgewisseld worden  $\rightarrow$  periodes beeld stilstaat  $\rightarrow$  wanneer beschikbare rate in netwerk  $>$  video rate  $\rightarrow$  na initiele buffering vertraging  $\rightarrow$  continuous playout tot einde video

## Vroegtijdige beëindiging van weergave & verplaatsen van weergavetijdstip

HTTP-byterange-headerveld in HTTP-get-verzoekbericht  $\rightarrow$  bevat informatie  $\rightarrow$  bereik in bytes van gewenste video  $\rightarrow$  client wil ontvangen  $\rightarrow$  If gebruik springt naar ander tijdstip in video  $\rightarrow$  client verzendt nieuw HTTP-verzoekbericht  $\rightarrow$  in byterangeheaderveld van bericht  $\rightarrow$  clientapplicatie specificeert vanaf welke byte in bestand  $\rightarrow$  gegevens wil ontvangen  $\rightarrow$  server nieuw HTTP-verzoek ontvangt  $\rightarrow$  alle eerdere verzoeken weg & bytes verzenden

## 9.2.3 Dynamic Adaptive Streaming over HTTP (DASH)

Gebruikt meerdere versies  $\rightarrow$  zelfde video  $\rightarrow$  elk een bepaalde snelheid waarmee gestreamd wordt  $\rightarrow$  gecombineerd  $\rightarrow$  CDN's vaak gebruikt distribueren opgeslagen & live video

## 9.3 Voice-over-IP (VoIP)

### 9.3.1 Beperkingen van best-effortdienst van IP

IP  $\rightarrow$  zo snel mogelijk van bron naar bestemming  $\rightarrow$  geen beloften  $\rightarrow$  vertraging of packet loss  $\rightarrow$  belangrijke consequenties  $\rightarrow$  ontwerp realtime spraakapplicaties  $\rightarrow$  gevoelig voor packetvertraging, jitter & verlies

1. Verzender genereert bytes met snelheid → 8000 bytes/seconde → iedere 20 ms verzamelt bytes in stuk → **chunk**
2. Chunk + speciale header → verpakt in UDP-segment → elke 20 ms UDP-segment verzonden

Als packet ontvanger bereikt → constante end-to-endvertraging → packets 20 ms na elke spraakactie van andere partij arriveren → ideale situatie → ontvanger elke chunk direct bij aankomst weergeven → sommige packets kwijt + niet zelfde end-to-endvertraging

Ontvanger moet

1. Bepalen wanneer chunk moet weergegeven worden
2. Bepalen wat er moet gebeuren als chunk ontbreekt

## Packetverlies

Verlies zou voorkomen → packets over TCP versturen → mechanismen opnieuw verzenden packets → niet geschikt voor VoIP → vergroten end-to-endvertraging

Packetloss tussen 1% en 20% is acceptabel → afhankelijk hoe spraakgegevens gecodeerd & verzonden zijn → manier hoe ontvanger verlies maskeert → Forward Error Correction (FEC) → hulpvol zijn packetverlies maskeren

## End-to-endvertraging

Totaal van

- Transmission delay
- Processing delay
- Queing delays routers
- Propagation delays in verbindingen
- Processing delays hosts

Ontvanger bij VoIP-applicatie → packets negeren die vertraging groter dan bepaalde drempelwaarde

## Packetjitter

Variatie in queuing delays → packet in netwerkrouers ondervindt = cruciaal → deze vertragingen variëren → dus ook verstreken tijd tussen moment → verzenden packet & ontvangen packet → fenomeen = **jitter** → compenseren aan de hand van **volnummers** , **tijdstempels** & **weergavevertraging**

## 9.3.2 Jitter voor audio compenseren bij ontvanger

Gedaan door 2 mechanismes combineren

1. Elke chunk vooral laten gaan door tijdstempel → verzender voegt aan elk nieuw gegenereerd packet → informatie over tijdstip → packet werd gegenereerd
2. Weergave chunks → bij ontvanger vertragen → weergave ontvangen chunks → vertraagd worden → zodat grootste deel packets ontvangen is voor weergeven

### 1) Onveranderlijke weergavevertraging

Ontvanger kan weergavevertragingen variëren

Chunk tijdstempel bij afzender op tijdstip  $t$  → ontvanger speelt chunk ( $q$ ) af op tijdstip  $t + q$  → assuming chunk tegen die tijd gearriveerd → packets na hun geplande speeltijd arriveren → weggegooid

### 2) Variabele weergavevertraging

Door initiële weergavevertraging groot te maken → meestg packets op tijd aankomen → verloren packets = klein → weergavevertraging variëren aan begin elke spraaksessie → stiltes voorafgaand aan spraaksessie → verkort of verlengd → niet hoorbaar wanneer verleningen / verkortingen stiltes niet te groot zijn

- Schatting van de pakketvertraging ( $d_i$ ) = schatting gemiddelde netwerkvertraging moment  $i$ -de packet arriveert bij ontvanger
- 

Schatting gemiddelde afwijking  
van vertraging ( $v_i$ ) =  
geschatte gemiddelde afwijking

vertraging ten opzichte van de  
 geschatte gemiddelde  
 vertraging  $\rightarrow$  di gemiddelde  
 werkelijke netwerkvertragingen  
 $\$rarr; r_1 - t_1, \dots, r_i - t_i$

- Schattingen  $d_i$  &  $v_i$  berekend elk ontvangen pakket  $\rightarrow$  als packet  $i \rightarrow$  1e packet in spraaksessie  $\rightarrow$  weergavemoment  $p_i$  berekend  $\$rarr; p_i = t_i + d_i + K_{vi}$
- Term  $K_{vi} \rightarrow$  weergavemoment zover in toekomst verschoven  $\rightarrow$  klein gedeelte packets te laat arriveren in spraaksessie
- Stel dat  $q_i = t_j + q_i \rightarrow$  tijd tussen moment 1e packet in spraaksessie genegeerd & moment packet weergegeven  $\rightarrow$  als packet  $j$  ook hoor bij spraaksessie  $\rightarrow$  weergegeven op tijdstip  $\rightarrow p_j = t_j + q_i$

## 9.3.3 Compenseren van packetverlies

### Forward Error Correction ( FEC )

#### Mechanisme 1:

Verzendt redundant gecodeerde 'chunk' na elke  $n$  chunks  $\rightarrow$  geconstrueerd door exclusieve OR-bewerking uit te voeren op  $n$  oorspronkelijke chunk.

Als willekeurig packet van groep  $n+1$  kwijtgeraakt  $\rightarrow$  ontvanger verloren packet reconstrueren  $\rightarrow$  meer dan 2 niet mogelijk  $\rightarrow$  verhoogt overdrachtssnelheid & weergavevertraging

#### Mechanisme 2:

Versturen van lagere resolutie audio stream  $\rightarrow$  verzender genereert  $n$ -de packet door  $n$ -de chunk van nominale stream achter  $(n-1)$  de chunk van redundante stream te plaatsen  $\rightarrow$  wanneer

meerdere niet opeenvolgende packets kwijtraken → ontvanger verlies compenseren door chunk met lage bitsnelheid → volgende packet "meelifit" geven → lagere geluidskwaliteit

## Interleaving

Verzender verstuurt eenheden audiogegevens in andere volgorde → oorspronkelijk aangrenzende eenheden in verzonden stream gescheiden zijn door afstand → effect packetverlies verkleinen → if packetloss → meeste van elke originele chunk → geen redundantie overhead → verhoogt playout delay → voordeel → benodigde bandbreedte voor een stream niet vergroot

## Maskeren van fouten

Herhalen van packets → ontvanger vervangt kwijtgeraakte packets door kopieën

Interpolatie → kwijtgeraakte packet berekend op basis van voorgaande & volgende packet in stream

## 9.3.4 Casus: VoIP met Skype

- Skype werkt met hiërarchisch overlaynetwerk
- Peer kan "superpeer" of gewone peer zijn
- Werkt met index → koppelt IP-adressen aan gebruikers
- Index gedistribueerd over superpeers
- Alice belt Bob → Skype client zoekt distribueerde index → Bob's IP adres bepalen

Meeste thuisnetwerken → NAT netwerken → NAT voorkomt host buiten thuisnetwerk verbinding met host binnen thuisnetwerk → beide Skype-bellers NAT → probleem

Super peers lossen probleem op

1. Alice logt in → superpeer buiten netwerk toegewezen → Alice & superpeer besturingsberichten uitwisselen → idem voor Bob
2. Alice belt Bob → informeert Alice superpeer → superpeer Bob informeert → brengt Bob op hoogte → inkomende oproep Alice
3. Bob accepteert → 2 superpeers kiezen 3e superpeer zonder NAT → gegevens Alice en Bob uitwisselen aan elkaar koppelen

Conference calls → elke gebruiker verzendt audiostream naar deelnemer die gesprek start → deelnemer combineert audiostreams tot 1 enkele stroom → verzendt kopie van elk gecombineerde stream → elk van andere N-1 deelnemers → video elke deelnemer → gestuurd in servercluster



# 9.4 Protocollen voor realtime spraakapplicaties

## 9.4.1 RTP

Gebruikt PCM,MP3,... te transporteren

### Basisprincipes van RTP

RTP boven op UDP

1. Verzendende component verpakt chunk media-gegevens in RTP-packet
2. Verpakt packet in UDP-segment → naar IP
3. Ontvangende haalt RTP-packet uit UDP-segment
4. Chunk → mediaplayer → decodeert & weergeven
5. Verzendende omponent voegt voor elke chunk audiogegevens → RTP-header toe
  - Type van audiocodering
  - Volgnummer
  - Tijdstempel
6. RTP packet → in UDP socket interface
7. Ontvanger → applicatie ontvangt RTP van socket interface
8. Applicatie filtert audiogegevens & headervelden van RTP packet → gegevens decoderen & afspelen

RTP geen mechanismen → tijdige bezorging van gegevens of kwaliteit diensten → geen garantie of packets aankomen of juiste volgorde

RTP → elke bron → eigen onafhankelijke RTP-packetstream → routers geen onderscheid tussen IP-datagrammen met RTP-packets & zonder RTP-packets

### Headervelden van RTP-packet

- Volgnummerveld (16 bits) → verhoogt met 1 voor elk verstuurd RTP-packet → gebruikt door ontvanger om packet loss & volgorde van packets herstellen

- Tijdstempelveld (32 bits) → ontvanger tijdstempels → packetjitter compenseren & packets met zelfde snelheid weergeven als waarin verzonden → als applicaties 160 gecodeerde samples maakt → tijdstempelveld verhoogt met 160 voor elk RTP-packet wanneer bron actief → tijdstempelveld klok loopt met constante snelheid → zelfs if bron = inactief
- Bronidentificatieveld / ( **Synchronization Source Identifier** of **SSRC** (32 bits) → identificeerd bron van RTP stream → elke RTP sessie heeft unieke bronidentificatie

## 9.4.2 SIP

- Levert mechanismen → gesprekken over IP-netwerk → beller kan diegene die gebeld wordt waarschuwen → gesprek wil → beide partijen afspraken maken over codering van media → beide deelnemers ook gesprek beëindigen
- Mechanismen → beller huidige IP-adres bepalen van gebelde → gebruikers geen vast IP → dynamisch toegewezen → verschillende IP-apparaten
- Mechanismen → beheren gesprekken → bv. Toevoegen nieuwe mediastreams, doorschakelen van gesprekken, ...

## Gesprek tot stand brengen met bekend IP-adres

1. Alice stuurt Bob → SIP INVITE bericht → over UDP naar poort 5060 voor SIP → geeft poort aan
2. INVITE-bericht identificatie voor Bob + indicatie huidig IP van Alice & preferred codering
3. Bob antwoord met 200 OK bericht → poort weer, IP & preferred codering
4. Na ontvangen Bob antwoord → SIP ontvangst bericht → erna praten

SIP kan over TCP of UDP verstuurd worden → default port 5060 → SIP berichten verstuurd en ontvangen in sockets → andere dan voor media data

SIP berichten → ASCII leesbaar → lijken op HTTP-ber

## SIP-berichten

Kijk pagina 663. Voor voorbeeld

## Vertalen van namen en het traceren van een gebruiker

Alice kent alleen e-mailadres Bob → dit adres ook voor SIP-gesprekken

## Hoe kent proxyserver het huidige IP-adres van bob@domain.com

Elke SIP-gebruiker → registrar gekoppeld

1. Gebruiker start SIP-applicatie
2. Applicatie verzendt SIP-registerbericht naar registrar → informatie huidig IP gebruiker
3. Registrar Bob → bijhouden huidig IP-adres Bob → ander SIP-apparaat → nieuw registerbericht met nieuw IP

Gebruik langere tijd → register blijft registerberichten sturen → registrar overeenkomsten met DNS-name-server → vertaalt vaste hostnamen naar vaste IP → SIP-registrar vaste menselijke identificatiegegevens → dynamische IP-adressen → SIP-registrars & proxy's op zelfde host

## Hoe kan SIP proxy huidige IP-adres van Bob achterhalen

1. Alice verstuurt INVITE-bericht → SIP-proxy Bob
2. Proxy stuurt bericht naar SIP-apparaat van Bob
3. Bob ontvangt Alice INVITE-bericht → kan antwoord sturen naar Alice

Jim@umass.edu (217.123.56.89) wil VoIP starten met

Keith@upenn.edu (197.87.54.21)

1. Jim verzendt INVITE-bericht naar SIP-proxy van umass
2. Proxy → DNS-lookup voor SIP-registrar upenn.edu → verzendt bericht naar registrarserver
3. Keith.upenn.edu → niet bekend bij registrar upenn → registrar omleidingsantwoordbericht → melding → umass keith.nyu.edu moet proberen
4. Proxy umass → INVITE-bericht → SIP-registrar van NYU
5. Registrar NYU → kent IP van keith@upenn.edu → stuurt INVITE-bericht door naar host 197.87.54.21 → SIP-client van Keith uitvoert.
6. Verzendt SIP-antwoordbericht → registrars/proxy's → terug naar SIP-client → 217.123.56.89

7. Idem 6
8. Idem 6
9. Media → rechtstreeks tussen 2 clients verzonden → ook ACK

## 9.5 Netwerkondersteuning voor multimedia

- **Beste maken van de best-effortdienst** → toenames van vraag voorspeld → ISP's extra bandbreedte & switches in → acceptabele mate vertraging & packetloss te garanderen
- **Diensten in verschillende categorieën aanbieden** → 1 type van verkeer → kan hogere prioriteit krijgen dan een ander type
- **Per verbinding andere QoS-garanties geven** → per verbinding QoS garantie → elke instantie van applicatie → bandbreedte reserveren → garanties end-to-endperformance → **harde garantie** → applicatie zeker de gevraagde QoS zal ontvangen → **Zachte garantie** → grote waarschijnlijkheid de gevraagde QoS zal ontvangen

### 9.5.1 Best-effortnetwerken dimensioneren

Eerste manier kwaliteit multimedia-applicaties te verbeteren →

#### **Meer geld uitgeven**

Bij multimedia in netwerken → voorkomen tekort aan resources → als linkcapaciteit → groot genoeg → packets door huidige internet getransporteerd → zonder queuing delays of kans op vermissing

Vraag is → hoeveel capaciteit nodig? → kosten leveren benodigde bandbreedte → reële zakelijke optie is voor ISP's → hoe groot capaciteit netwerklinks → bepaalde topologie → bepaalde end-to-endperformance te leveren = **netwerkdimensioneringsprobleem**

**Volgende problemen moeten opgelost worden om performance van applicatielaag tussen 2 eindpunten in netwerk te kunnen voorspellen**

1. Modellen van het benodigde dataverkeer tussen eindpunten in het netwerk
  - Mogelijk modellen gedefinieerd worden op gespreksniveau
2. Goed gedefinieerde performance-eisen

- Performance eis stellen dat gevoelig is voor vertraging → kans end-to-endvertraging van packets groter is dan max te tolereren vertraging
3. Modellen om end-to-endperformance bij een bepaald netwerkbelastingsmodel te voorspellen en technieken om zo gering mogelijke kosten zodanig bandbreedte toe te wijzen dat aan alle eisen van de gebruikers wordt voldaan

## 9.5.2 Verschillende soorten diensten verlenen

Eenvoudigste uitbreiding → dataverkeer voor unitaire & egalitaire best-effortdienst → huidige internet opsplitsen in categorieën → bij dienstverlening aan verschillende categorieën → verschillende niveaus

### Een paar scenarios

Principe 1: **Packet markering** → markeren van packets → router packets die horen bij verschillende dataverkeercategorieën van elkaar onderscheiden → originele doel (ToS) veld in IPv4

Principe 2: **Isolatie dataverkeer** → zekere mate van isolatie tussen categorieën implementeren → ene klasse niet nadelig beïnvloed kan worden → als iets mis is met andere categorie

2 aanpakken mogelijk:

1. **Dataverkeerpolicy** → als dataverkeercategorie moet voldoen aan bepaalde criteria → controlemechanisme → zorgen policy nageleefd → als applicatie niet aan criteria houdt → mechanisme handelend optreden → dataverkeer netwerk binnenkomt voldoet aan criteria
2. **Packetschedulingmechanisme op datalinklaag** → expliciet een constante hoeveelheid van linkbandbreedte te laten reserveren → elke categorie

Principe 3: Belangrijk categorieën van elkaar scheiden → wenselijk resources → efficiënt mogelijk te benutten → manier packets in wachtrij voor verzending over link worden geselecteerd = **link-schedulingmethode**

## Leaky bucket

Policing = belangrijk QoS-mechanisme

3 policycriteria:

1. **Gemiddelde snelheid:** Netwerk kan gemiddelde snelheid van packets van stream langere tijd beperken → cruciale factor → tijdsduur waarover de gemiddelde snelheid zal worden geregeld → bron begrensd 100 packets per seconde → sneller afgeremd dan bron 6000 packets/min → zelfs beide bronnen → zelfde gemiddelde snelheid
2. **Maximale snelheid:** beperking van gemiddelde snelheid → begrenst hoeveelheid dataverkeer → in netwerk gezonden kan worden → relatief lange periode → beperking van maximale snelheid → begrenst # packets verzonden in korte periode
3. **Burstgrootte** → Netwerk kan ook max # packets → begrenzen → dat gedurende extreem korte periode via netwerk wordt verzonden

Buckets bestaan uit → bucket dat max  $b$  tokens bevat

1. Nieuwe tokens → in bucket → snelheid van  $r$  tokens per seconde genereerd
2. IF bucket  $\leq b$  tokens → token direct in bucket
3. Else → token genegeerd → bucket blijft gevuld met  $b$  tokens

**Stel packet voor het verzonden wordt → token uit bucket halen**

Omdat maximaal  $b$  tokens in bucket zitten → maximale burstgrootte voor een met leaky bucket begrensde stream gelijk aan  $b$  packets

Tokens genereerd met snelheid  $r$  → maximale aantal packets → netwerk kan binnenkomen in willekeurige periode met lengte  $t$  →  $rt + b$  → snelheid  $r$  waarmee tokens genereerd worden → maat om gemiddelde snelheid → packets netwerk kunnen binnenkomen op lange termijn → begrenzen

## 9.5.3 Diffserv

Diffserv → differentiatie in dienstverlening → mogelijkheid verschillende categorieën dataverkeer → internet schaalbare manier

2 functionele elementen

1. **Functies aan de edge:** → classificeren en conditioneren van dataverkeer → ingaande edge netwerk ( bij Diffserv-host die dataverkeer genereert of eerste Diffserv-router waarlangs dataverkeer passeert) → arriverende packets gemarkeerd
2. **Functies in de core** → doorverzenden → wanneer Diffserv gemarkeerd packet → arriveert Diffserv router → doorverzonden naar volgende hop → volgens 'per-hop' voorschrift → geldt voor betreffende categorie packets → 'per-hop' voorschrift uitsluitend gebaseerd op markering van packet ( Diffserv-model)

Packets die bij edgerouter aankomen → geclassificeerd & gemarkeerd (AFBEELDING p.679)

1. Classificeerder selecteert packets → basis 1 of meer waarden in headervelden
2. Verzendt packet naar betreffende markeerfunctie

Sommige gevallen host afgesproken → packetverzendsnelheid → sturen → voldoet aan bepaald **dataverkeerprofiel** → kan limiet op maximale overdrachtsnelheid bevatten of maximale # packets verzonden in korte tijd

Zolang gebruiker packets verzendt → voldoen aan overeengekomen waarden in dataverkeerprofiel → packets voorkeursbehandeling → wanneer verzender niet houdt aan dataverkeerprofiel, packets buiten overeenkomst vallen → andere markering of vertraagd worden of zelfs genegeerd worden aan edge netwerk

**Meetfunctie** → ingaande stream vergelijken met overeengekomen dataverkeerprofiel → bepalen packet binnen dat profiel past → eigenlijke beslissing over packets → netwerkbeheerder

In per-hop gedrag belangrijke overwegingen

- 'per-hop' voorschrift → zorgen verschillende categorieën dataverkeer → verschillende diensten
- Per-hop voorschrift verschilt → behandeling tussen categorieën definieert → geen expliciete informatie hoe voorschrift uitvoeren
- Verschillen in performance moeten dus zichtbaar & meetbaar zijn

## 2 'per-hop' voorschriften gedefinieerd

1. **Expedited forwarding** → 'per-hop' voorschrift → vertreksnelheid van categorie → bij router = > dan geconfigureerde snelheid
2. **Assured forwarding** → 'per-hop' voorschrift → verkeer in 4 categorieën → elke AF-categorie gegarandeerd → bepaalde minimale hoeveelheid bandbreedte & buffers

**End-to-end-Diffserv-dienst** leveren → alle ISP's tussene eind systemen → service leveren & samenwerken & afspraken maken → klant → gedifferentieerde end-to-end dienst te bieden

# 9.5.4 Per verbinding Qos-garanties geven: resources reserveren & streams toelaten

Principe 4: Nood aan nieuwe netwerk mechanics en protocollen → duidelijk wanneer stream gegearandeerde service moet ontvangen zodra gestart is

1. **Recources reserveren** → enige manier garanderen → stream beschikken over benodigde recources → recources gereserveerd → stream naar behoefte benutten → gedurende gereserveerde tijd ongeacht behoeften andere streams
2. **Streams toelaten** → recources gereserveerd → netwerk mechanisme hebben → streams verzoeken kunnen richten → recources niet oneindig → verzoek stream afgewezen → als gevraagde recources niet beschikbaar zijn
3. **Signaleren set-up streams** → toelatingsproces → stream die QoS nodig heeft → voldoende recources reserveren → elke netwerkrouter tussen bron & bestemming → zeker zijn end-to-end-QoS-eisen hebben → elke router → lokale recources voor nieuwe stream nodig zijn berekenen → bekijken hoeveel beschikbare recources in gebruik zijn → opgestarte streams → of bepalen beschikbare per hop → toereikend QoS-eisen → honoreren Signaliseringsprotocol nodig activiteiten coördineren = **call-setupprotocol** → **RSVP-protocol** → voorgesteld voor dit doel binnen internetarchitectuur (PAGINA 683 AFBEELDING)

# Active vs Passive FTP

## Active FTP

1. Client contacteert server op command poort
2. Server stuurt ACK terug naar client's commandpoort
3. Server initieert connectie op lokale datapoort → naar datapoort client eerder aanduidde
4. Client stuurt ACK terug

## Passive FTP

1. Client contacteert server op command poort
2. Server antwoord met poort 2024 → server verteld welke poort luisterd voor data connectie
3. Client initieert data connectie van zijn datapoort → gespecificeerde data poort
4. Server stuurt ACK terug naar client's datapoort

Active → SYN door client Passive → SYN door client

---

Revision #5

Created 17 June 2021 13:57:47 by Jasper G.

Updated 30 December 2021 19:13:38 by Jasper G.