Intelligente selectie van live videostromen afkomstig van mobiele toestellen op grote evenementen

Jarno Goossens

Promotoren: prof. dr. ir. Pieter Simoens, prof. dr. ir. Bart Dhoedt
Begeleiders: Steven Bohez, Lander Van Herzeele

Masterproef ingediend tot het behalen van de academische graad van
Master of Science in de industriële wetenschappen: informatica

Vakgroep Informatietechnologie
Voorzitter: prof. dr. ir. Daniël De Zutter
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2014-2015

UNIVERSITEIT
GENT

Voorwoord

De keuze voor deze masterproef was ingegeven door mijn interesse voor mobiele technologiën en beeldverwerking. Op mijn stage had ik reeds ervaring opgedaan met het ontwikkelen van mobiele applicaties en opvallend veel van deze hadden iets te maken met het streamen van beelden. Dankzij dit werk heb ik het ontwikkelproces eens uit de ogen van een backend developer kunnen aanschouwen. Dit was een zeer leerrijke ervaring.

Vooreerst wil ik mijn begeleiders, Steven Bohez en Lander Van Herzeele, bedanken voor hun uitstekende begeleiding en hun altijd constructieve feedback. Ook dank aan mijn promotoren prof. Pieter Simoens en prof. Bart Dhoedt voor de kans om dit onderzoek te mogen uitvoeren. Verder wens ik Merlijn Sebrechts te bedanken voor het telkens paraat staan om Linux gerelateerde zaken toe te lichten. Tot slot bedank ik nog mijn vriendin voor het begrip dat zij afgelopen maanden heeft opgebracht omdat dit werk meer aandacht kreeg dan haar.

Ook wil ik mijn klasgenoten bedanken voor de plezante momenten doorheen onze studies en de productieve samenwerking bij de vele projecten. Tot slot rest mij alleen mijn ouders te bedanken voor de steun doorheen mijn opleiding.

Jarno Goossens, juni 2015

Toelating tot bruikleen

"De auteur geeft de toelating deze masterproef voor consultatie beschikbaar te stellen en delen van de masterproef te kopiëren voor persoonlijk gebruik. Elk ander gebruik valt onder de bepalingen van het auteursrecht, in het bijzonder met betrekking tot de verplichting de bron uitdrukkelijk te vermelden bij het aanhalen van resultaten uit deze masterproef."

Jarno Goossens, juni 2015

Intelligente selectie van live videostromen afkomstig van mobiele toestellen op grote evenementen

door

Jarno Goossens

Scriptie ingediend tot het behalen van de academische graad van
Master of Science in de industriële wetenschappen: informatica

Academiejaar 2014–2015

Promotoren: Prof. Dr. Ir. P. Simoens, Dr. Ir. B. Dhoedt
Scriptiebegeleiders: Ir. S. Bohez, Ing. L. Van Herzeele
Faculteit Ingenieurswetenschappen en Architectuur
Universiteit Gent

Vakgroep Informatietechnologie
Voorzitter: Prof. Dr. Ir. D. De Zutter

Samenvatting

In deze masterproef gaan we na of het mogelijk is om op grote evenementen live videostromen te verzamelen afkomstig van mobiele toestellen. De methode die voorgesteld wordt gaat streams karakteriseren op een flexibele manier. Op basis hiervan is het mogelijk om het uploadproces in verschillende fases te laten verlopen, hierdoor wordt het netwerk minder belast en enkel potentieel interessante beelden doorlopen alle fases. Het is mogelijk om een overzicht van alle streams weer te geven op basis van relevantie, hierdoor wordt de intelligente selectie verwezenlijkt.

Trefwoorden

videostromen, mobiele toestellen, metadata, realtime, cloud-scaling

Intelligent selection of live video streams from mobile devices on large events

Jarno Goossens

Supervisor(s): prof. dr. ir. Pieter Simoens, prof. dr. ir. Bart Dhoedt, Steven Bohez, Lander Van Herzeele

Abstract—**Mobile devices with camera's and a data connection are more and more present in our lives. The need to stream live content from these devices is something that automatically arises with it. In this article we discuss the problem of gathering live streams on large events. We propose a framework based on filters as a solution to gather and query these streams. In this framework the available bandwidth on each network link will be an important factor in gathering process as well as the energy consumption on mobile devices. To handle a large amount of streams the framework is able to scale in the cloud. Eventually it is evaluated with a simulation for a large event with more than 200 simultaneous video streams.**

Keywords— **video streaming, mobile devices, selection, realtime, cloud-scaling**

I. Introduction

ON large events more and more people are shooting video's to share with friends, family or world wide. The organisers of the events like to gather these in realtime to show them in a live feed on the web or on big screens present at the event. Just streaming all video's will result in a huge stream of data that most networks aren't able to consume.

Another problem is making a selection out of all these streams. Only a few streams out of the possibly many will be candidates to show on a big screen or in a web feed. A possible solution is adding describing metadata making it able to query them.

It is very likely that streams aren't going to be distributed equally in time. On some occasions a lot of people will grab their device and start filming at the same moment. The needed infrastructure must be able to adapt accordingly.

II. Characteristics of streams

Streams can have lots of characteristics. For our purpose, displaying them live on a big screen or in a web feed, it is important that the characteristics can be calculated in realtime. In the next sections we're going to list some possibly interesting characteristics to query streams. They will be divided in two categories: structural and content characteristics.

A. Structural characteristics

This type of characteristics is relatively easy to be determined by a computer. For example the resolution and frame rate of a stream. Both values are most likely to be static and provide some information about the quality of a stream. It isn't possible to tell if it's a high quality stream but if they have a very low resolution or frame rate it isn't likely that they are going to be candidates. At the moment the standard for TV is a frame rate of 24 frames per second. According to [1] this will be upgraded to 48 or 60 FPS in the near future.

A characteristic that depends on the resolution and frame rate is the bitrate. This is the amount of data that is sent in a certain time period. The higher the bitrate of the stream the more likely it is that the quality will be better, but more bandwidth will be occupied.

Most of the smartphones these days are equipped with many sensors. They can provide useful information like the amount of movement a device is making or the light conditions of the environment. When a camera is moving quickly it is likely to assume that the stream of a more stable camera will produce a better quality. Similarly with the light conditions, camera's in a dark environment probably aren't producing useful streams. Another valuable sensor is the GPS sensor, certainly in combination with the magnetic field sensor. By using this data streams can be categorised by location and orientation. It doesn't say much about the quality of a stream but it can say a lot about the content, something that will be discussed in the next section.

Since the streams are live and shot from a mobile device, it is likely that they will be transmitted over a wireless network. Too much streams on a certain network link will produce unwanted delays and a reduction of quality. To prevent this the network characteristics of a stream can deliver valuable information, such as: the type of network connection of the mobile device and the amount of packet loss on a stream. The type of network connection will give an upper limit for the upload speed. In Figure 1 this illustrated for some typical network connections in combination with the recommended bitrate for a certain resolution. In [5] is researched what the influence is of packet loss in an MPEG-4 video stream. The quality drops heavily when more and more packets are lost, the reason being error propagation. They propose a solution by extending the RTP protocol to allow retransmission of the most important frames.

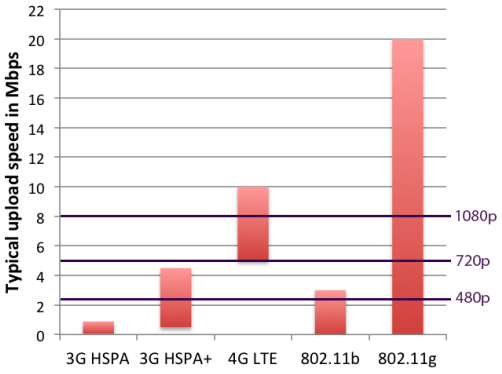


Fig. 1. Typical upload speeds of some types of network connections together with the recommended bitrate for a certain resolution by YouTube [2–4].

B. Content characteristics

These characteristics are mostly vague or subjective. For example creativity, in [6] they try to answer the question if a video is creative programmatically. Here fore they use several photographic, visual and audio features.

Another characteristic that is subjective is the subject of an image. In [7] *saliency moments* are used to determine the subject. They create a signature of the image based on several features like the place of colours. Based on this it's possible to determine a visual distance between two images. A similar approach is possible by using Google's, ImageNet or Flickr's annotated dataset to translate pictures into keywords.

III. Selecting based on features

Once the characteristics of a stream have been determined, we need to make a selection out them. This selection can be static, always selecting the same kind of streams, or dynamic, at one time we need streams from a certain location another time streams with one individual in the center are preferred.

If a static selection is necessary and a large dataset of annotated streams is available, then neural nets or naive Bayes classifiers can provide a solution. Another method is calculating a score based on weights for each characteristic and select the streams with the highest score. Optimal weights can be found with methods like simulated annealing.

The last solution, weights for each characteristic, can also be applied for a dynamic selection. Preconfigured sets of weight can be made to switch between them, also the weights can be configured at the moment and corrected by trial-and-error.

IV. Architecture of the framework

To provide a solution for the problem a modular framework is suggested. The deployment diagram is illustrated in Figure 2. The clients have only two requirements: a data connection to the back end and a camera. A native application must be developed for these devices that can analyse and stream the content.

The results of the analyzations will be stored in the central DB Server. This server will contain information about all the streams and servers and almost every module is connected to it.

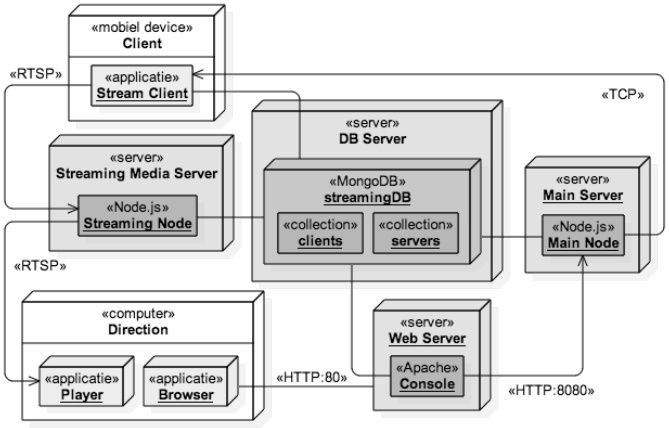


Fig. 2. Deployment diagram of the framework.

The Main Server will decide which clients can stream their content. He will make these decisions based on the information available in the database.

The Streaming Media Servers will receive content from the clients. They will execute extra analysations that require a lot of CPU power, memory or other information to compare with. Again the results are added to the database.

The last two modules are a Web Server and the computer(s) of the directors. The Web Server functions as a bridge between the database and the directors, it provides a graphical user interface where it is possible to set the filters, view a summary of all streams (textually) and view a graphical representation of the top-9 streams.

A. Upload process

The flow of data during the upload process is illustrated in Figure 3. This is divided into three phases:

1. The client will download the minimum requirements from the Main Server. When these are not met the process stops here. Until now only a few kilobytes are sent.
2. The content characteristics are added to the database. Depending on the settings this will happen about one time each second. At this moment a few kilobytes are sent every second and the client is performing some computations.
3. The Main Server discovers the client in the database and sends a command to start streaming. From now on the client will stream the content to a Streaming Media Server, typically a data stream with the size of several Mbps. The server will perform some extra analysations as mentioned earlier.

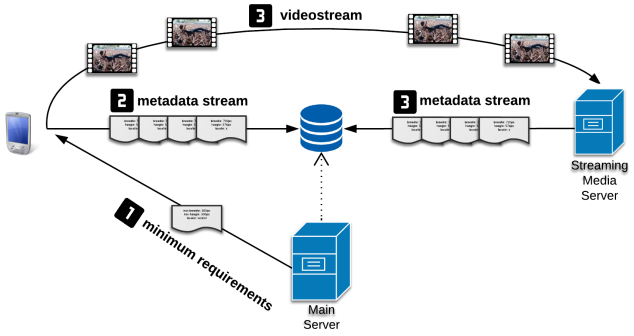


Fig. 3. The flow of data during the upload process.

V. Implementation

To analyse the content of the streams, OpenCV[1] is used. It provides lots of optimised algorithms for computer vision and is practically available on each platform.

Because the lack in structure in the data and the short lifetime it has MongoDB[2] is chosen as database management system.

The asynchronous nature and periodic analysing of the database are motivations to chose for Node.js[3] to implement the

[1] http://opencv.org
[2] https://www.mongodb.org
[3] https://nodejs.org

server logic. For the Streaming Media Servers a library is used for the implementation of codecs and protocols, it can be found at GitHub[4].

To prevent a network link from overloading the streams are monitored by the Streaming Media Servers. For each stream the packet loss rate in a time frame of five seconds (depending on the settings) is determined. When this rate exceeds more than 1% streams are stopped until the packet loss returns to a stabilised value.

The same thing happens with Streaming Media Servers. These servers will update there status to the database each second. This includes their CPU load, memory usage and the amount of incoming and outgoing traffic on each network interface. When the Main Server detects a server with a property above a certain threshold he will stop setting up new streams to this server and search for one that is not fully loaded. The thresholds are depending on many factors. For example the amount of streams that are started each period will have an intense effect. Also the type of streams and the hardware of the servers plays a crucial role. On the servers that were available with two Quad core Intel E5520 (2.2GHz) CPUs and 12GB RAM, thresholds of 80% for the CPU load and memory usage were acceptable for setting up two streams each second with a resolution of 1920x1080. These servers were able to process 30 to 40 streams of this resolution with this settings. With the same settings 100 to 120 streams of a low resolution (640x380) could be processed.

VI. Evaluation

To evaluate the framework a simulation of a large event is constructed. Over a time period of 250 minutes (about 4 hours) 3750 streams are started. To construct a realistic simulation the Pareto principle is applied, 80% of all streams are started in 20% of the time. The amount of started streams is also divided in several bursts with a period of 10-40 minutes between them. To be able to process a burst, the settings of the Main Server were adjusted: start maximum five streams per second, a threshold of 60% for the CPU load and 50% for memory usage.

To determine the influence of the bursts an other simulation was constructed were the streams are equally distributed in time. More specific the amount of streams that was started in a minute is Poisson distributed with $\lambda = 15$ (3750 streams $/250$ minutes $= 15$).

A. Results

The average amount of servers needed in the Pareto simulation was 2.014 servers, in the Poisson simulation this was 2.257 servers. In a cloud-based architecture the cost of both simulations won't differ that much. But during the bursts in the Pareto simulation more servers were needed, up to eight servers at one time (see Figure 4). With a private server setup the Pareto simulation would end up more expensive because of this.

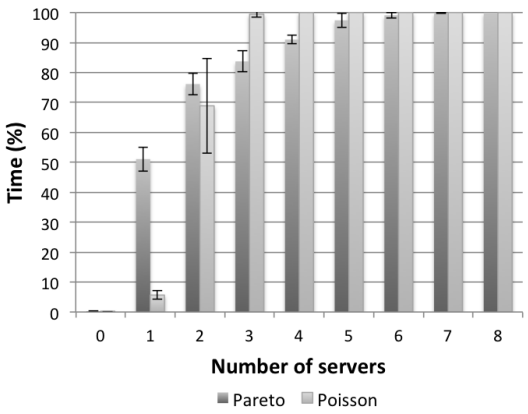


Fig. 4. The percentage of time a certain number of Streaming Media Servers was sufficient for the simulations.

VII. Conclusion

With the proposed architecture it is shown that making a intelligent selection out of realtime streams is possible. With the server setup that was available more than 200 simultaneous streams could be processed on the back end. The power of the framework is in the intelligent selection, the better this fits the needs of purpose, the better the results will be. The network won't be overloaded, the mobile devices will do less unnecessary calculations/uploads and the amount of servers will be reduced.

[4] https://github.com/iizukanao/node-rtsp-rtmp-server

References

[1] Tom Bert. *Getting ready for high frame rates in digital cinema.* Technical report, Barco, 2012.

[2] Prashant Panigrahi. *LTE vs HSPA+: Where is the future?* http://www.3glteinfo.com/lte-vs-hspa-where-is-the-future, 2012. [Online; retrieved 5-april-2015].

[3] *What is the actual real-life speed of wireless networks?* http://www.speedguide.net/faq/what-is-the-actual-real-life-speed-of-wireless-374, 2014. [Online; retrieved 5-april-2015].

[4] *Advanced encoding settings: recommended bitrates, codecs, resolutions and more.* https://support.google.com/youtube/answer/1722171, 2015. [Online; retrieved 3-april-2015].

[5] Nick Feamster and Hari Balakrishnan. *Packet loss recovery for streaming video.* In *12th International Packet Video Workshop*, pages 9–16. PA: Pittsburgh, 2002.

[6] Miriam Redi, Neil O'Hare, Rossano Schifanella, Michele Trevisiol, and Alejandro Jaimes. *6 seconds of sound and vision: Creativity in micro-videos.* In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*. IEEE Computer Society, 2014.

[7] Miriam Redi and Bernard Merialdo. *Saliency moments for image categorization.* In *Proceedings of the 1st ACM International Conference on Multimedia Retrieval*, page 39. ACM, 2011.

Inhoudsopgave

Lijst van tabellen

Lijst van figuren

Gebruikte afkortingen

HD	High Definition
fps	frames per second
GPS	Global Positioning System
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
RTP	Real-time Transport Protocol
RTMP	Real Time Messaging Protocol
RTSP	Real Time Streaming Protocol
FLV	Flash Video
HLS	HTTP Live Streaming
HTTP	Hypertext Transfer Protocol
kbps	kilobit per second
Mbps	Megabit per second
DB	Database
HSV	Hue, Saturation, Value
RGB	Red, Green, Blue
OpenCV	Open source Computer Vision
JSON	JavaScript Object Notation
AAC	Advanced Audio Coding
MPEG	Moving Picture Experts Group
PSNR	Peak Signal-to-Noise Ratio
CPU	Central Processing Unit
RAM	Random-Access Memory
BBC	British Broadcasting Corporation
ICoSOLE	Immersive Coverage of Spatially Outspread Live Events

Hoofdstuk 1

Inleiding

1.1 Probleemstelling

Camera's zijn overal rond ons aanwezig. Bijna elke smartphone is in het bezit van minstens één camera en de meeste mensen beschikken nog over tal van andere toestellen die kunnen filmen. Zo zijn er bijvoorbeeld tablets maar ook actiecamera's en spiegelreflexcamera's. Veel van deze toestellen worden meegenomen naar grote evenementen om beelden vast te leggen en deze later te delen. Grote evenementen spelen hier dan ook op in en motiveren bezoekers om deze beelden te delen, zodat ze eventueel enkele momenten later in een *videostream* verwerkt kunnen worden. Bijvoorbeeld om weer te geven op grote schermen aanwezig op de locatie van het evenement.

De volgende stap is dan logischerwijs het realtime delen van deze beelden. Zo heeft *Twitter* recent (op 26 maart 2015) *Periscope* gelanceerd [1], een applicatie om live video beelden uit te zenden vanaf een smartphone. Echter, wanneer een groot evenement deze beelden wil verwerken in een live videostream, wil men graag de beste beelden hierin verwerken. Als er duizenden videostreams beschikbaar zijn is dit onbegonnen werk om handmatig een selectie te maken. In deze masterproef wordt een manier gezocht om dit proces te automatiseren. Anders verwoord, er wordt op de volgende vraag een antwoord gezocht:

Hoe kan een groot aantal live videostreams realtime gekarakteri-

seerd worden?

Concrete voorbeelden waarbij het karakteriseren van een grote hoeveelheid live videostreams interessante resultaten kan opleveren:

- Op een festival wil de organisator beelden van bezoekers op het festivalterrein verwerken in een live feed die beschikbaar is op de website.
- Bij het veldrijden valt een deelnemer op een locatie waar geen vaste camera aan het filmen was en de regie wil deze valpartij semi-live op televisie tonen.
- Een televisieshow wil beelden van hun kijkers verwerken in de live show.
- De politie wil ongevallen detecteren op basis van beelden waardoor hulpdiensten sneller ter plaatse kunnen zijn.

Haperende videostreams met een lage resolutie leveren voor het voorbeeld van het festival en de televisieshow geen interessante beelden op. Deze hoeven we dus niet te tonen aan de regie die instaat voor het monteren van de live feed. Zo zijn er nog tal van andere classificatie's mogelijk die afhankelijk van het doel van de beelden nuttige informatie kunnen produceren.

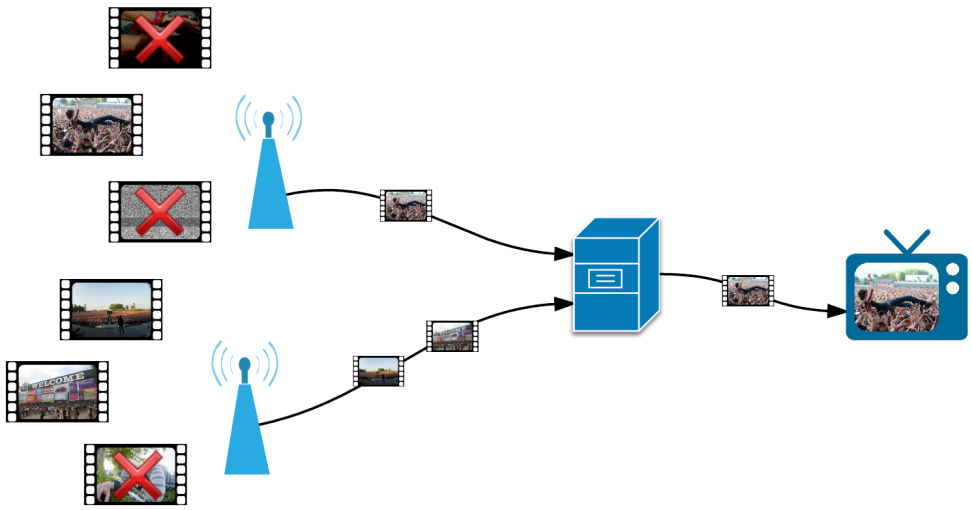
Welke kenmerken van een videostream kunnen interessant zijn om te doorzoeken of op te ordenen?

Bij het karakteriseren van videostreams zal soms onmiddellijk duidelijk worden dat we die beelden niet hoeven te tonen aan de regie, zoals bijvoorbeeld beelden van een smartphone die in een rugzak aan het filmen is. In principe is het dan ook niet nodig om die beelden door te sturen over het netwerk waardoor dit minder belast wordt en het mobiele toestel minder energie verbruikt. Hierbij moet een evenwicht gevonden worden tussen de analyses die gebeuren op het toestel en degene die op de backend worden uitgevoerd.

Welke analyses worden uitgevoerd op de backend en welke op het mobiele toestel?

Ondertussen is het reeds duidelijk dat er een raamwerk ontwikkeld moet worden dat bestaat uit twee delen. Het eerste deel is een applicatie op het mobiele toestel die beelden kan analyseren en streamen, het tweede deel is een backend die verdere analyses kan uitvoeren, een overzicht kan weergeven van de huidige live streams en toelaat queries uit te voeren zodat automatisch de meest geschikte streams getoond worden. Naarmate het aantal streams groeit, zal één server echter niet voldoende zijn om alle streams af te handelen. Het raamwerk moet dus schaalbaar zijn.

> *Hoe kan ervoor gezorgd worden dat het raamwerk schaalbaar is en duizenden streams kan verwerken?*



Figuur 1.1: Het basis probleem: welke streams zijn mogelijk interessant?

Er kunnen in theorie zoveel servers aangemaakt worden als nodig eens het raamwerk schaalbaar is, maar de netwerk infrastructuur kan ook een probleem vormen. Deze moet de grote datastroom van alle beelden kunnen verwerken. De videostreams zijn afkomstig van mobiele toestellen die een draadloze netwerk connectie hebben, dit kan een 3G of 4G verbinding zijn

maar ook bijvoorbeeld een Wi-Fi netwerk dat beschikbaar is op het evenement. Wanneer een bepaald *access point* te veel videostreams binnen krijgt zal dit overbelast worden en gaan er netwerk pakketten verloren, hierdoor zal de kwaliteit van de beelden dalen.

> *Hoe zwaar kan een access point belast worden zonder merkbaar verlies in kwaliteit?*

1.2 Overzicht

In Hoofdstuk 2 worden de mogelijkheden onderzocht om te bepalen of beelden al dan niet interessant zijn. Hiervoor worden kenmerken van beelden en streams geclassificeerd. De vraag hoe men op basis van die kenmerken een intelligente selectie van streams kan maken wordt op basis hiervan beantwoord.

In Hoofdstuk 3 wordt vervolgens de architectuur van het raamwerk besproken. Welke onderdelen heeft het raamwerk nodig om de nodige functionaliteit te realiseren en aan alle eisen te voldoen? Hoe kan er voor gezorgd worden dat het raamwerk schaalbaar blijft?

Hoofdstuk 4 werpt een blik op de beschikbare technologieën voor elk architectuurblok en de communicatie onderling. Vervolgens wordt besproken welke technologie gebruikt werd en waarom. Uiteindelijk wordt dit hoofdstuk afgesloten met een geïmplementeerde toepassing van het raamwerk, meer bepaald voor een festival. Het doel van de toepassing wordt uitgelegd om hierna verder te gaan naar de vereisten en een concrete implementatie.

Vervolgens wordt in Hoofdstuk 5 het raamwerk geëvalueerd aan de hand van een simulatie voor de uitgewerkte toepassing en worden de resultaten besproken.

In Hoofdstuk 6 worden conclusies getrokken uit de resultaten van deze masterproef en ook wordt er eventueel toekomstig werk besproken.

Hoofdstuk 2

Literatuurstudie

Om streams te kunnen selecteren is het belangrijk te weten welke metadata nodig zijn om te kunnen karakteriseren. Op basis van deze metadata kunnen dan beslissingen genomen worden zoals: Moet de stream verstuurd worden? Moet de stream getoond worden aan de regie? Is deze stream beter dan een andere? Belangrijk is dat deze metadata realtime geconstrueerd wordt vermits er met live streams wordt gewerkt.

Metadata kan ingedeeld worden in twee groepen: structurele en inhoudelijke kenmerken. De eerste groep kenmerken is relatief eenvoudig te bepalen door een computer. Voorbeelden zijn: de resolutie van het beeld, data van sensoren, richting waarin gefilmd wordt, geografische locatie en de kwaliteit van de verbinding. De inhoudelijke kenmerken zijn veel moeilijker te bepalen en vallen voornamelijk onder het vakgebied van computervisie. Voorbeelden zijn onder andere: uniciteit, relevantie, compositie, het onderwerp en andere semantische opvattingen. In deel 2.1 worden enkele structurele kenmerken besproken en in deel 2.2 worden enkele inhoudelijke kenmerken besproken.

Op basis van de verzamelde metadata van deze beelden moet een selectie gemaakt worden. Een eerste selectie beslist welke beelden er gestreamd worden voor een eventuele verdere analyse op een server. Een tweede selectie beslist welke beelden er uiteindelijk aan de regie getoond zullen worden. Hoe deze selectie het beste gebeurt is afhankelijk van de toepassing, in deel 2.3 wordt dit uitvoerig besproken.

2.1 Structurele kenmerken

Het belang van deze kenmerken is sterk afhankelijk van de toepassing. In dit deel worden enkele kenmerken overlopen en hun eigenschappen besproken. Eerst worden kenmerken die verband houden met de video codering besproken, vervolgens kenmerken die bepaald kunnen worden aan de hand van sensoren en tot slot kenmerken die iets meer kunnen vertellen over de netwerkverbinding.

2.1.1 Video codering

Bij het coderen van video wordt er gezocht naar een compromis tussen de compressiefactor en de kwaliteit van de beelden. Wanneer er geen compressie wordt toegepast zal dit resulteren in een gigantische datastroom en zal het netwerk snel overbelast raken. Hieronder volgen enkele eigenschappen die mee de kwaliteit en compressiefactor bij codering bepalen.

De eerste eigenschap is de resolutie die aangeeft aan hoeveel *pixels* een beeld bevat. Dit wordt beschreven door het aantal rijen en kolommen pixels in een beeld. In Tabel 2.1 worden de aangeraden resoluties voor YouTube video's opgesomd. Afhankelijk van het doel van de toepassing kan men een minimum resolutie vereisen en voorkeur geven aan beelden met een hogere resolutie.

Naam	**Resolutie (b x h)**	**Video Bitrate**	**Audio Bitrate**
2160p (Ultra HD of 4k)	3840 x 2160	35-45 Mbps	384 kbps
1440p (2k)	2560 x 1440	10 Mbps	384 kbps
1080p (Full HD)	1920 x 1080	8,000 kbps	384 kbps
720p (HD Ready)	1280 x 720	5,000 kbps	384 kbps
480p	854 x 480	2,500 kbps	128 kbps
360p	640 x 360	1,000 kbps	128 kbps

Tabel 2.1: Tabel met aangeraden resoluties en bitrates voor YouTube [2].

Een hogere resolutie is echter geen garantie dat de beelden van hogere kwa-

liteit zullen zijn. De kwaliteit hangt af van nog vele andere factoren zoals de gebruikte *codec* en de instellingen ervan, de kwaliteit van de beeldsensor, de scherpte van de beelden, enzovoorts.

De codec is het algoritme dat gebruikt wordt om ruwe gegevens van de beeldsensor om te zetten naar een structuur die minder ruimte in beslag neemt en/of makkelijker te verwerken is. Deze geeft aan welk compressie algoritme er gebruikt is en met welke instellingen. De meeste codecs maken gebruik van *lossy* compressie waarbij informatie verloren gaat. Het doel van een lossy codec is dus net zoveel mogelijk informatie weggooien zonder dat wij als mens dit gewaar worden. Het voordeel van dit soort compressie is dat streams veel minder bandbreedte nodig hebben dan streams waarbij gebruik gemaakt werd van *lossless* compressie (zonder verlies aan gegevens). In [3] wordt als vuistregel aangegeven dat een compressie met factor 60 nog steeds uitstekende kwaliteit oplevert bij moderne codecs maar dit is natuurlijk afhankelijk van het soort beelden. In sommige situaties, bijvoorbeeld wanneer de beelden vergroot moeten worden, is het soms nodig om een codec met lossless compressie te gebruiken. In Tabel 2.2 wordt een overzicht gegeven van veelgebruikte codec-bibliotheken in combinatie met hun gebruikte compressiemethode.

lossy	**lossy/lossless**	**lossless**
Xvid	x264	Huffyuv
libtheora	FFmpeg MPEG-4	Lagarith
VP6	VP9	

Tabel 2.2: Tabel van veelgebruikte codec-bibliotheken ingedeeld op basis van ondersteunde compressiemethodes

Ook is er nog de *frame rate*, het aantal beelden dat per tijdseenheid geproduceerd wordt en meestal uitgedrukt is in *frames per second* (FPS). Het menselijk oog kan 10 tot 12 beelden per seconde analyseren en bewaart elk beeld $^1/_{15}$ seconde [16]. Wanneer tijdens deze periode een ander beeld ontvangen wordt zullen de hersenen het effect van visuele continuïteit creëren.

Hierdoor lijken het vloeiende bewegingen, hoewel het in werkelijkheid verschillende statische beelden zijn. Voor video zal er dus minstens een frame rate van 20 á 25 fps nodig zijn indien we geen haperende beelden willen.

De huidige standaard voor televisie is momenteel 24 FPS, maar volgens [4] zal die in de nabije toekomst verhoogd worden naar 48 of 60 FPS. Er worden zelfs toestellen uitgebracht die een frame rate tot 120 FPS ondersteunen.

Opnieuw is het meestal zo dat hoe hoger de frame rate is, hoe beter de kwaliteit van de video ervaren wordt. Ook voegt dit extra mogelijkheden toe zoals het vertragen van beelden naar *slow motion* zonder merkbaar kwaliteitsverlies in de vorm van schokkerigheid in de video. Maar dit heeft als gevolg dat er meer gegevens in dezelfde tijdsperiode verzonden moeten worden. Afhankelijk van de toepassing zal naar een compromis moeten gezocht worden tussen een hoge frame rate en de hoeveelheid data die verzonden moet worden. Zie Tabel 2.3 voor enkele camera's en hun maximum frame rate voor het opnemen van video's afhankelijk van de resolutie.

Toestel	**Resolutie**	**Maximum Frame Rate**
Foscam FI8904W [1]	480p	15 FPS
	240p	30 FPS
iPhone 6 [2]	1080p	60 FPS
GoPro HERO4 Black [3]	2160p	30 FPS
	1080p	120 FPS
	720p	240 FPS

Tabel 2.3: Tabel met enkele camera's en hun maximum frame rate voor het opnemen van video's afhankelijk van de resolutie

Afhankelijk van onder andere de frame rate en de resolutie zal er dus meer of minder data verzonden moeten worden. Dit wordt aangeduid door de

[1] `http://www.foscam.nl/index.php/outdoor-ip-camera/foscam-fi8904w.html`
[2] `https://www.apple.com/iphone-6/specs`
[3] `http://shop.gopro.com/EMEA/cameras/hero4-black/CHDHX-401-master.html`

bitrate en deze wordt meestal uitgedrukt in het aantal bits per seconde. Hoe hoger de resolutie van de beelden, hoe meer data er verzonden moet worden binnen dezelfde tijdsperiode voor dezelfde kwaliteit. In Tabel 2.1 worden de aangeraden bitrates weergegeven voor video's op YouTube in functie van hun resolutie. Deze zijn opgesplitst in een bitrate voor beelden (video) en geluid (stereo audio). Hieruit kunnen we afleiden dat de bandbreedte nodig voor audio verwaarloosbaar is in vergelijking met die voor video.

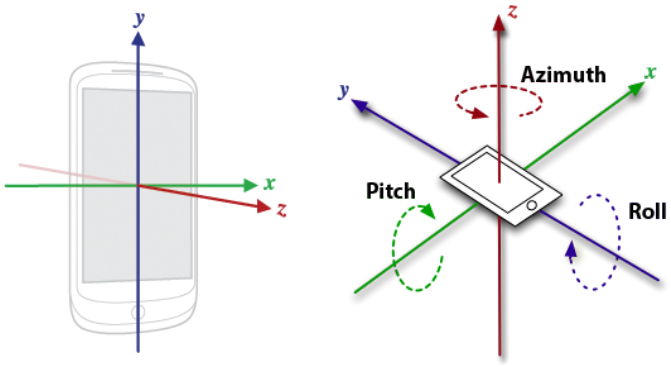
Een hogere bitrate wil zeggen dat er meer data verzonden wordt, wat zorgt voor een hogere netwerkbelasting. Anderzijds betekent een hogere bitrate, doorgaans betere kwaliteit. Opnieuw hangt de kwaliteit af van nog andere factoren en is een bepaling enkel op basis van bitrate niet voldoende.

2.1.2 Sensorinformatie

Veel mobiele toestellen zijn uitgerust met tal van sensoren die beweging, oriëntatie en verschillende omgevingsfactoren kunnen meten. Hieronder worden de interessantste sensoren en hun correlatie met de video besproken.

Data van bewegingssensoren kunnen een indicatie geven van hoeveel beweging er is in een video op een bepaald moment. Te veel beweging van de camera resulteert in bewegingsonscherpte en wordt meestal als storend ervaren. De accelerometer is een sensor die versnelling kan registreren en meten. Door deze data te analyseren is het mogelijk om te bepalen in welke mate het toestel op dat moment aan het bewegen is. De versnelling wordt uitgedrukt in $^{m}/_{s^2}$ langs de x-, y- en z-as. Naast de versnelling bij translaties van een toestel kan ook de rotatie versnelling gemeten worden door middel van de gyroscoop. Deze data wordt uitgedrukt in de vorm van hoeksnelheid, in radialen per seconde. Zie Figuur 2.1 voor een grafische voorstelling van het coördinatenstelsel gebruikt door de accelerometer en gyroscoop op het *Android* platform.

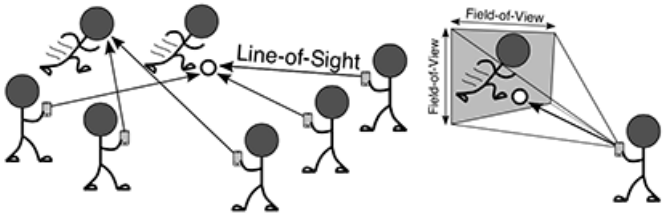
Het nadeel van deze sensoren is dat de gevoeligheid en accuraatheid verschilt van toestel tot toestel. Technieken zoals kalibratie, normalisatie en ruisonderdrukking door middel van filtering kunnen hiervoor compenseren.



Figuur 2.1: Data van bewegingssensoren: links het coördinatensysteem van de accelerometer, rechts de verschillende hoeksnelheden gemeten door de gyroscoop

Sensoren voor lichtsterkte, temperatuur, vochtigheid en luchtdruk vallen onder de categorie omgevingssensoren. Voor bepaalde toepassingen kan deze informatie nuttig zijn, maar voornamelijk de sensor voor lichtsterkte kan data aanleveren die belangrijk is voor het bepalen van de kwaliteit van beelden. Zonder de beelden te analyseren kan er bepaald worden in welke situatie iemand aan het filmen is. Zo kan bijvoorbeeld heel snel bepaald worden of een toestel in een zeer donkere omgeving beelden aan het opnemen is en hoogstwaarschijnlijk zich in een rugzak of broekzak bevind, maar in elk geval geen interessante beelden aan het produceren is.

Ook is er nog data van positiesensoren, met onder andere de GPS sensor. Aan de hand van die data is het mogelijk de absolute locatie te bepalen waar beelden gemaakt zijn. In combinatie met data van een kompas kan ook de richting van de camera bepaald worden. Opnieuw hebben deze sensoren slechts een beperkte nauwkeurigheid en moet dit in rekening gebracht worden. Wanneer men beschikt over een 3D model van de omgeving waarin alle beelden gemaakt worden kan deze data ook helpen bij het bepalen van de locatie en oriëntatie van de camera. In [10] wordt deze data gebruikt om de *Line-of-Sight* (zie Figuur 2.2) te bepalen en op basis daarvan de video's te groeperen.



Figuur 2.2: Groeperen van beelden op basis van Line-of-Sight, figuur uit [10]

2.1.3 Netwerkparameters

Op basis van het type verbinding kan bepaald worden hoeveel vertraging en storing we kunnen verwachten. Maar ook is het mogelijk om een onder- en bovengrens te bepalen voor de bandbreedte, dit in combinatie met de bitrate van de video bepaalt of het mogelijk is om de video realtime te streamen (zie Figuur 2.3).

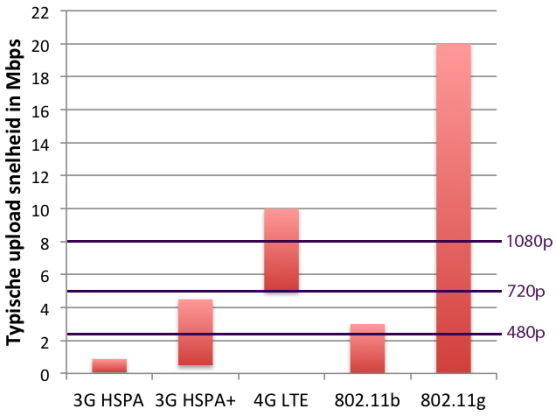
Het type verbinding kan ook nog een ander belang hebben, wanneer data verstuurd wordt over een mobiele verbinding gaan er meestal kosten met gepaard, iets wat een nadelig effect kan hebben op de gebruiker zijn tevredenheid.

Indien men niet over deze informatie beschikt is het nodig deze zelf te bepalen door middel van netwerk monitoring tools.

Om streams te verzenden over het netwerk kunnen verschillende protocols gebruikt worden. Zo is er bijvoorbeeld het *Real Time Messaging Protocol* (RTMP) dat gebruikt wordt door YouTube[4]. Dit is een TCP gebaseerd protocol en biedt dus ondersteuning voor het herverzenden van verloren pakketten. Een nadeel van dit protocol is dat het enkel kan gebruikt worden om beelden te versturen in het *Flash Video* (FLV) formaat, dit wordt momenteel niet standaard ondersteund op het Android- en iOS platform voor het opnemen van video beelden.

Een andere mogelijkheid is het *HTTP Live Streaming* (HLS) protocol, dit

[4]https://www.youtube.com



Figuur 2.3: Grafische voorstelling van de typische upload snelheid van enkele dataverbindingen samen met een paar aangeraden bitrates voor YouTube (zie Tabel 2.1) [17, 18]

is gebaseerd op HTTP en bijgevolg ook op TCP. Het grote voordeel is dat het bijna overal werkt, HTTP verkeer wordt zelden geblokkeerd door een firewall. Dit brengt meteen ook een nadeel met zich mee want dit zorgt voor een grotere overhead en dus een grotere belasting van het netwerk.

Tot slot is er ook nog het *Real-time Transport Protocol* (RTP) dat meestal gebruik maakt van UDP. Dit protocol definieert een pakketformaat om audio en video over een netwerk te verzenden onafhankelijk van de videocodering. Wanneer UDP gebruikt wordt is er ook nog eens weinig overhead en bijgevolg minder bandbreedte nodig, het is wel niet mogelijk om verloren pakketten opnieuw te verzenden maar vanwege de realtime vereiste is dit ook minder belangrijk. Bovendien bevat het protocol elementen om vertraging te compenseren en de oorspronkelijke volgorde van pakketten te herstellen.

Wanneer een server te veel data over een bepaalde verbinding krijgt of wanneer het netwerk overbelast wordt gaat er pakketverlies optreden. In [5] wordt onderzocht wat het effect is van pakketverlies op een MPEG-4 video stream. Zij ondervinden dat de kwaliteit hevig daalt naarmate er meer pakketten verloren gaan, met als reden dat fouten zich voortplanten bij het decoderen van de stream. Hun oplossing is een uitbreiding van het RTP protocol dat het

mogelijk maakt om de belangrijkste data opnieuw te verzenden met een aanvaardbare vertraging. Hierdoor is de stream niet meer real-time, er zal een kleine vertraging op zitten, maar men verkrijgt een aanzienlijke verbetering qua kwaliteit.

Een andere optie is voorkomen in plaats van genezen. Er kan pakketverlies gedetecteerd worden door een stream te monitoren en eens dit boven een aanvaardbaar niveau gaat gepast reageren. Bijvoorbeeld: detecteren op welk access point pakketverlies optreedt en het minder gaan belasten tot het pakketverlies terug aanvaardbaar is.

2.2 Inhoudelijke kenmerken

Deze kenmerken zijn meestal vaag of subjectief, het antwoord op de vraag of een video voldoet aan dit kenmerk is niet eenvoudig te beantwoorden met ja of nee maar zit ergens tussenin. Zo kan er bijvoorbeeld bepaald worden hoe mooi een videostream is. Uiteindelijk hangt het antwoord af van het doel van de verzameling streams, maar er bestaan wel algoritmes die dit kunnen benaderen op basis van een wetenschappelijke definitie van het kenmerk.

Het bepalen van een inhoudelijk kenmerk wordt meestal gedaan door verschillende inhoudelijke eigenschappen van een beeld te analyseren, deze informatie te combineren en hieruit tenslotte een conclusie te trekken.

Zo is er in [6] reeds onderzoek gevoerd om automatisch te bepalen of een *Vine* video als creatief wordt beschouwd of niet. Hier worden fotografische, visuele en audiologische kenmerken gebruikt voor de analyse. In Tabel 2.4 is een overzicht terug te vinden van de interessantste kenmerken samen met een beschrijving ervan.

Om een gevarieerde selectie te verkrijgen kan men het onderwerp van een beeld bepalen. Op basis van deze informatie kan er geclusterd worden om daarna het beste beeld te selecteren. Hier is het probleem: wanneer bevat een beeld hetzelfde onderwerp? Voor de ene persoon bevat een foto van

Eigenschap	Beschrijving
Fotografische kenmerken	
Regel van derden [8]	Compositieregel waarbij het beeld ingedeeld wordt in 9 gelijke vlakken (Figuur 2.4)
Bokeh [8]	De kwaliteit van onscherpte van de achtergrond in een beeld
Contrast [21]	Verhoudingen tussen lichte en donkere elementen in een beeld
Symmetrie [9]	Gelijkenis tussen de linker- en rechterhelft van het beeld
Uniciteit [9]	Gelijkenis met andere beelden
Visuele kenmerken	
Kleur [19]	Hoeveelheid rood, groen en blauw
GLCM [19]	*Gray-Level Co-Occurrence Matrix*
HSV [19]	Kleurtoon, verzadiging en helderheid
Audio kenmerken	
Geluidssterkte [20]	Hoeveelheid energie in het geluid
Ruwheid [20]	Een waarneming die geassociëerd is met amplitudemodulatie van zuivere tonen
Ritme [20]	Accenten die een zeker patroon of regelmaat vertonen

Tabel 2.4: Tabel met interessante kenmerken die gebruikt worden om de creativiteit van een Vine video te bepalen in [6]

een dennenbos en een loofbos hetzelfde onderwerp, een bos. Voor de andere persoon zijn dit twee verschillende onderwerpen, een bos met naaldbomen en een bos met loofbomen. Opnieuw bestaan er algoritmes die het antwoord op deze vraag kunnen benaderen.

Een mogelijkheid is gebruik maken van *saliency moments* [7], hiermee kunnen afbeeldingen op inhoud gecategoriseerd worden. Er wordt een handtekening gecreëerd op basis van eigenschappen van een afbeelding zoals de kleuren die deze bevat en de positie waar ze zich bevinden. Deze handtekening kan dan gebruikt worden om de visuele afstand met andere te berekenen en geeft een benadering voor de semantische afstand tussen deze twee afbeeldingen.

Tot slot is het ook mogelijk om gebruik te maken van reeds bestaande technieken zoals bijvoorbeeld het laten omzetten van een afbeelding naar kernwoorden door Google, ImageNet of Flickr. Deze beschikken over een enorme dataset van geannoteerde afbeeldingen waarmee ze de te categoriseren afbeelding kunnen vergelijken en een semantische afstand kunnen bepalen.

Figuur 2.4: Kwaliteit van beelden bepalen aan de hand van fotografische kenmerken, bijvoorbeeld de regel van derden.

2.3 Selectie op basis van kenmerken

Wanneer de beelden geanalyseerd zijn moet op basis van deze resultaten nog een selectie gemaakt worden. Deze kan statisch of dynamisch gebeuren. Een voorbeeld waar dynamische selectie van toepassing is: een festival wil op een bepaald tijdstip enkel beelden vanop een bepaalde locatie streamen, zoals tijdens een optreden beelden uit het gebied voor het podium en tussen de optredens door beelden die backstage gemaakt worden. Een voorbeeld van statische selectie: hulpverleners die ongevallen wil detecteren. Telkens wil men dezelfde soort beelden verkrijgen, die die mogelijk een ongeval in beeld brengen. Hieronder worden enkele methodes besproken die dit mogelijk maken, welke methode uiteindelijk het beste werkt is afhankelijk van de toepassing.

2.3.1 Aantal streams maximaliseren

Indien we de bitrate van een stream kunnen aanpassen door bijvoorbeeld de resolutie of de framerate te verlagen kunnen we het aantal streams maximaliseren. In [12] wordt het omgekeerde verwezenlijkt: in plaats van zoveel mogelijk streams te ontvangen worden zoveel mogelijk streams uitgezonden. Op basis van een populariteitsscore krijgt een bepaalde video een hoeveelheid bandbreedte toegekend. De populariteitsscore is een gewichtsfactor bepaald

in functie van het aantal kijkers. Hoe meer kijkers hoe meer bandbreedte een video krijgt, maar andere video's krijgen toch nog een gegarandeerde minimumbandbreedte eventueel met gereduceerde beeldkwaliteit. Indien dit principe omgedraaid wordt is het mogelijk om met één van voorgaande technieken een score toe te kennen aan een videostream en op basis hiervan de gereserveerde bandbreedte te berekenen.

2.3.2 Naïeve Bayes classificatie

De regel van Bayes is een regel uit de kansrekening die een relatie legt tussen de huidige kans voor een gebeurtenis en de kans uit het verleden. Op deze manier is het mogelijk om kennis uit het verleden te gebruiken voor beslissingen in de toekomst. Deze methode kan dus toegepast worden indien men telkens beelden wil met dezelfde kenmerken. In [22] werd aangetoond dat deze methode heel goed presteert voor het rangschikken van gegevens op de kans dat deze in een bepaalde categorie behoren.

Om gebruik te maken van deze methode is het nodig om een dataset te construeren die een lijst van streams bevat vergezeld van hun kenmerken en een attribuut dat aangeeft in welke categorie deze stream zit: niet streamen, streamen en analyseren of weergeven aan de regie. Hierna moet men voor elk kenmerk bepalen hoe de data verdeeld is en kan men aan de hand hiervan de kans bepalen of een kenmerk geselecteerd wordt. Vervolgens kan de globale kans dat een beeld geselecteerd wordt berekend worden door al deze kansen met elkaar te vermenigvuldigen.

Deze methode is aan te raden wanneer er veel gegevens uit het verleden beschikbaar zijn en enkele kenmerken volgens een bepaalde kansverdeling verdeeld zijn. Indien de classificatie moet gebeuren op basis van vele kenmerken zal deze methode veel statistische analyse met zich meebrengen en bijgevolg een tijdrovend werk zijn.

2.3.3 Selectie door middel van gewichten

Een andere mogelijkheid om de selectie te maken is door aan elk kenmerk een bepaald gewicht toe te kennen. Dit gewicht geeft het belang aan van dit kenmerk ten opzichte van een ander. Op basis hiervan kan een score berekend worden en is het mogelijk om streams te sorteren. Uit deze lijst kan er dan een aantal geselecteerd worden beginnend bij diegene met de beste score.

Indien er geweten is welk soort beelden gewenst is kan men opnieuw een dataset construeren zoals in 2.3.2. Hierna kan men voor deze set de optimale gewichten benaderen en in toekomst deze gebruiken voor de selectie.

De uitdaging hier zit hem in het benaderen van de optimale gewichten, hoe meer kenmerken we hebben hoe nauwkeuriger de selectie zal zijn maar hoe langer het duurt om de optimale gewichten te vinden. Elk kenmerk voegt namelijk een dimensie toe aan de zoekruimte. Gelukkig hoeft dit proces maar één keer te gebeuren voor een bepaald soort streams waardoor dit interessant kan zijn voor bepaalde toepassingen.

Simulated annealing is uitermate geschikt om de optimale gewichten te vinden. Het algoritme kan beperkt worden in tijdsduur en het vindt steeds een optimum, maar niet noodzakelijk het globale optimum. Het principe is als volgt:

1. Begin in een bepaalde toestand (bijvoorbeeld: alle gewichten op 1)
2. Vind een buur[5] die nog niet bezocht is
   (a) Gevonden → ga naar 3
   (b) Niet gevonden → dit zijn de benaderde optimale gewichten
3. Controleer resultaten met huidige gewichten
   (a) Betere resultaten → ga naar 2
   (b) Slechtere resultaten $\begin{cases} p \geq 0,5 & \text{ga naar 2} \\ p < 0,5 & \text{ga terug naar vorige toestand en dan naar 2} \end{cases}$

[5]Een buur is een toestand waarbij één van de gewichten een beetje veranderd is

De kans p bepaalt of we verder gaan met een buur die slechtere resultaten geeft of niet. Indien deze kans groot is ($p \geq 0,5$) zal er verder gezocht worden vanaf de buur met slechtere resultaten, indien deze kans klein is ($p < 0,5$) zal er naar een nieuwe buur gezocht worden met betere resultaten. Bij de start van het algoritme zullen kansen geproduceerd worden die groter zijn dan op het einde, hierdoor blijven we in het begin niet hangen bij lokale optima en gaan we op het einde naar een optimum toe.

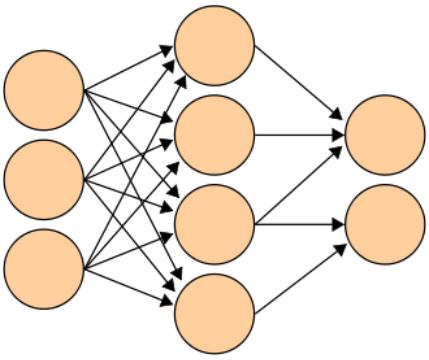
Wanneer op voorhand niet geweten is welk soort beelden gewenst is kan er een systeem ontworpen worden waarbij de gewichten handmatig worden ingesteld en door middel van *trial-and-error* geëvalueerd worden. Ook is het mogelijk om enkele voorgedefinieerde reeksen van gewichten te maken die dan ingeladen kunnen worden. Bijvoorbeeld een reeks waarbij enkel beelden met hoge kwaliteit geselecteerd worden of een reeks waar geopteerd wordt voor beelden die geografisch verspreid liggen.

2.3.4 Neurale netwerken

Deze netwerken zijn een vorm van informatieverwerkende eenheden die geïnspireerd zijn op biologische neurale netten. Deze zijn gericht om informatie te verwerken die vaag, gevarieerd en ingewikkeld is. Wat leren betreft is een neuraal netwerk ten zeerste geschikt om een computer taken te laten vervullen waarvoor men geen algoritme kent [23]. Met andere woorden: wanneer men op voorhand weet welk soort beelden gewenst is maar men ziet geen verband tussen de kenmerken, dan kunnen neurale netwerken misschien een oplossing bieden.

Een neuraal netwerk bestaat uit drie soorten neuronen: invoerneuronen, interneuronen en uitvoerneuronen (zie Figuur 2.5). De invoerneuronen worden in dit geval gekoppeld aan de data van de kenmerken. De interneuronen hebben inkomende verbindingen van verschillende invoerneuronen en/of andere interneuronen, ook hebben ze uitgaande verbindingen naar andere interneuronen en/of uitvoerneuronen. Deze verbindingen hebben elk hun eigen gewicht en afhankelijk daarvan zal een neuron al dan niet geactiveerd worden.

De uitvoerneuronen duiden aan in welke categorie een bepaalde stream zit of kennen er een score aan toe.



Figuur 2.5: Een eenvoudig voorbeeld van een neuraal netwerk met drie invoerneuronen (links), vier interneuronen (midden) en twee uitvoerneuronen (rechts)

Om correcte resultaten te bekomen met het neurale netwerk is het nodig de gewichten correct af te stemmen. Aangezien dit een complexe zaak is (om goede resultaten te bekomen zal men heel wat interneuronen nodig hebben) kunnen we deze gewichten vinden door middel van een gesuperviseerd leerproces. Hiervoor is opnieuw een dataset nodig zoals bij 2.3.2.

2.3.5 Vereenvoudigde selectie

Bovenstaande methodes vereisen telkens grondig onderzoek en/of een grote dataset. De focus van de masterproef ligt eerder bij het ontwikkelen van een flexibel raamwerk waarbij het makkelijk is om methodes zoals deze te gebruiken. Hierdoor is er voor de implementatie een systeem ontwikkeld waarbij aan streams een score wordt toegekend met als doel een selectie te maken waarbij de totaalscore maximaal is. De score wordt toegekend op basis van gewichten voor kenmerken, zoals in 2.3.3. Met het verschil dat gewichten handmatig worden ingesteld in plaats van op zoek te gaan naar de optimale gewichten aan de hand van een dataset.

Hoofdstuk 3

De architectuur van het raamwerk

3.1 Inleiding

In dit hoofdstuk wordt bekeken welke onderdelen nodig zijn om de uitdagingen die vermeld zijn in Hoofdstuk 1 te realiseren: realtime streamen en karakteriseren van video's, een schaalbare architectuur en verdelen van de analyses tussen backend en mobiel toestel.

Om dit verwezenlijken is een modulair raamwerk ontwikkeld waarbij elk element een specifiek doel heeft. Deze modulariteit laat toe om meer systeembronnen toe te kennen aan bepaalde elementen door het op- en neer te schalen in een cloud omgeving.

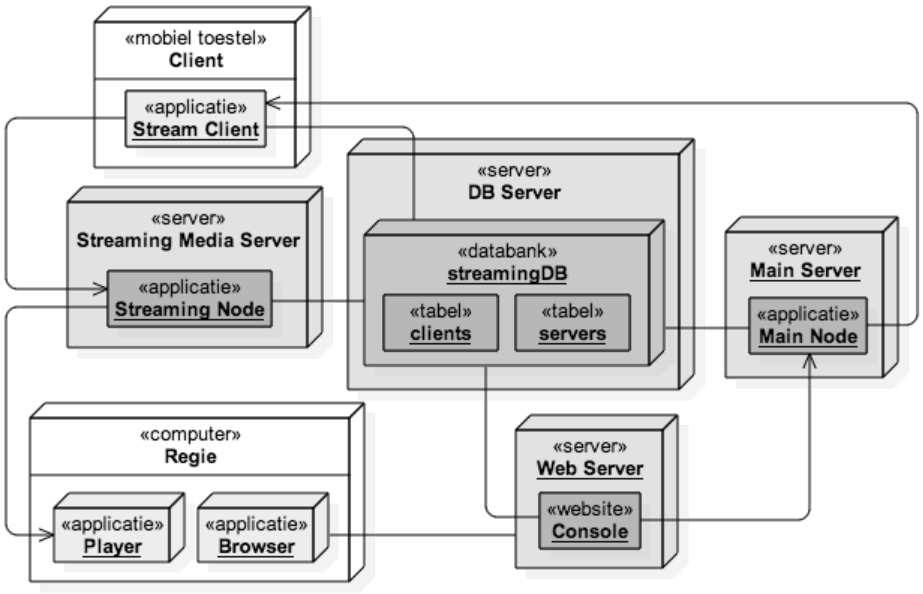
3.2 Onderdelen van het raamwerk

Op Figuur 3.1 is de architectuur van het raamwerk te zien. De component *Client* zijn de mobiele toestellen. Om het raamwerk flexibel te houden hebben die maar twee vereisten: er moet minstens één camera aanwezig zijn en het toestel moet over een dataverbinding beschikken die kan communiceren met de servers. Voorbeelden van zulke toestellen zijn: smartphones, tablets, actiecamera's, enzovoorts.

Deze clients hebben een applicatie nodig die het mogelijk maakt om beelden van de camera te analyseren en eventueel te streamen naar de volgende component in het raamwerk, de *Streaming Media Server*. Deze component buffert de binnenkomende streams en voert eventueel extra analyses uit. De resultaten hiervan maakt hij bekend in de databank.

De *DB Server* is de centrale component van het raamwerk, al de benodigde gegevens zijn hierop aanwezig. Op basis van deze gegevens maakt de *Main Server* beslissingen over welke streams gestart worden en welke Streaming Media Server deze moet verwerken.

Uiteindelijk is er dan nog de *Web Server* die de gegevens beschikbaar stelt aan de computer(s) van de regie. Deze toestellen moeten enkel beschikken over een browser om naar de webserver te surfen en een applicatie die streams kan afspelen.



Figuur 3.1: Deployment Diagram: de logische structuur van het raamwerk afgebeeld op de infrastructuur

3.2.1 DB Server

De databank is de centrale component van het raamwerk, alle elementen zijn hiermee rechtstreeks verbonden met als enige uitzondering de computer(s) van de regie. Ze bevat twee tabellen: clients en servers, beide met data van zeer korte levensduur.

De eerste tabel (clients) bevat metadata over de streams die momenteel beschikbaar zijn. Deze metadata wordt afhankelijk van de instellingen en gebruikte filters ongeveer één keer per seconde herberekend. Hiervan wordt een deel op de client berekend en een deel op de Streaming Media Server. Ook is deze verschillend van toestel tot toestel, bij het ene apparaat is bijvoorbeeld data van een accelerometer aanwezig en bij het andere niet. Een vaste structuur voor de tabel is dus afwezig.

De andere tabel (servers) bevat gegevens over de Streaming Media Servers. Hierin is onder andere per server de belasting van de processor terug te vinden, hoeveel procent van het geheugen in gebruik is en de hoeveelheid inkomende en uitgaande datatrafiek van alle netwerkverbindingen. Aan de hand van deze tabel is het mogelijk om snel de minst belaste server te vinden en het op- en neerschalen te implementeren. Ook deze data wordt afhankelijk van de instellingen ongeveer één keer per seconde herberekend.

Kortom indien er eens data verloren gaat is het dus zo erg nog niet, een ogenblik later wordt dit toch geüpdatet. Realtime beschikbaarheid daarentegen is wel belangrijk, gegevens van enkele ogenblikken geleden kunnen verkeerde informatie geven over de huidige toestand van een stream of server.

3.2.2 Streaming Media Server

Deze server(s) verwerken en analyseren de streams, met andere woorden deze moeten een grote hoeveelheid data realtime kunnen verwerken. Wanneer een client begint te streamen zal deze zijn content uploaden naar één van deze servers. Deze server gaat op zijn beurt die stream beginnen bufferen en onmiddellijk starten met de nodige analyses uit te voeren. De resultaten van

deze analyses worden dan telkens geüpdatet in de databank. Indien van op een regie computer een stream geselecteerd wordt kan deze vanuit de buffer verzonden worden naar die computer.

Het is zeer waarschijnlijk dat deze servers een knelpunt gaan vormen in de architectuur en een kandidaat zijn om op te schalen.

3.2.3 Main Server

De dirigerende server is de main server. Deze analyseert voortdurend de databank en onderneemt acties op basis van deze resultaten. Zo verstuurd hij commando's naar clients om te starten of stoppen met streamen naar een bepaalde Streaming Media Server maar ook bijvoorbeeld om te wisselen tussen servers om het neerschalen te realiseren. Welke server geselecteerd wordt om een nieuwe stream op te starten of een oude naar toe te verplaatsen zal afhangen van de gegevens in de server tabel.

3.2.4 Web Server

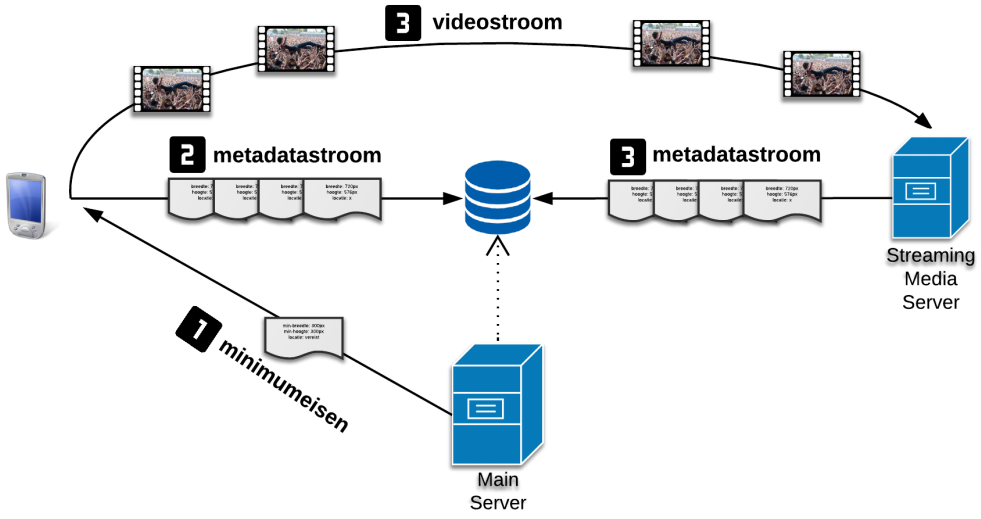
De laatste server is dan de web server, deze functioneert als spreekwoordelijke brug tussen de databank en de regie: een eerste deel visualiseert de gegevens in de databank, dit is een tekstueel overzicht van alle streams en hun metadata aanwezig in de tabel clients. Hierbij is het mogelijk om de data te sorteren op een bepaald veld en om alle details te bekijken.

Een tweede deel presenteert de top-9 streams in een raster waarbij je de enkel live beelden te zien krijgt zonder verdere informatie.

Tot slot is het mogelijk om in het derde deel de filter criteria aan te passen. Bijvoorbeeld bij een festival is het mogelijk om tijdens concerten een voorkeur te geven aan beelden die zich in een bepaald gebied bevinden, bijvoorbeeld voor het podium. Anderzijds tussen de concerten door is het interessant om streams te verkrijgen die geografisch verspreid liggen.

3.3 Uploadproces

In het ideale geval worden enkel de streams die nodig zijn geüpload. Omdat het niet mogelijk is op voorhand te weten welke content zal nodig zijn moet er aan de hand van kenmerken een selectie gemaakt worden. Er wordt geopteerd om zoveel mogelijk content naar de backend te zenden (zodat de kans groter wordt dat de gewenste streams erbij zijn), maar vroeg of laat zal er een limiet bereikt worden en moet er een selectie gemaakt worden. De beschikbare bandbreedte is een eerste grens die niet overschreden mag worden, wanneer er teveel data over een bepaalde netwerk link verstuurd wordt zal er congestie optreden en zal de kwaliteit van de streams dalen. Wanneer het netwerk geen probleem vormt is het ook mogelijk dat de servers een tekort aan rekenkracht of geheugen krijgen. Tot slot zijn er ook nog de beperkingen van mobiele toestellen: het batterijverbruik en de eventuele kosten verbonden aan het dataverbruik kunnen elementen zijn die niet verwaarloosd mogen worden.



Figuur 3.2: De dataflow bij het uploadproces binnen het raamwerk

Om zo weinig mogelijk berekeningen uit te voeren en het netwerk minimaal te belasten wordt het upload proces ingedeeld in drie fases (zie Figuur 3.2):

1. De client begint te filmen, maakt zich kenbaar in de databank[1] en vraagt aan de main server de minimumeisen. Deze bevatten de vereisten voor de metadata zoals een minimum of maximum waarde maar ook of bepaalde kenmerken aanwezig moeten zijn (bijvoorbeeld data van een accelerometer). Indien de vereisten niet voldaan zijn stopt het uploadproces hier, wordt het netwerk niet verder belast en verbruikt het mobiele toestel ook geen energie. Voorlopig is er eenmalig hoogstens enkele kilobytes verzonden.

2. Indien de vereisten wel voldaan zijn gaat de client de gevraagde metadata beginnen streamen naar de databank. Hierbij gaat hij afhankelijk van de instellingen ongeveer één keer per seconde de metadata updaten. Tot hiertoe is er dus een datastroom van hoogstens enkele kilobytes per seconde en voert de client afhankelijk van de gevraagde metadata lichte tot zware berekeningen uit.

3. De Main Server analyseert periodiek de databank (stippellijn op Figuur 3.2) en ontdekt zo nieuwe potentieel interessante streams. Wanneer zo een stream gevonden wordt stuurt deze een commando naar de client dat hij de beelden naar een bepaalde Streaming Media Server moet doorsturen. Vanaf nu worden de beelden gestreamd en ontstaat er een grote datastroom. Deze beelden zullen worden geanalyseerd door de server en op zijn beurt zal hij de resultaten periodiek (één keer per seconde) updaten naar de databank.

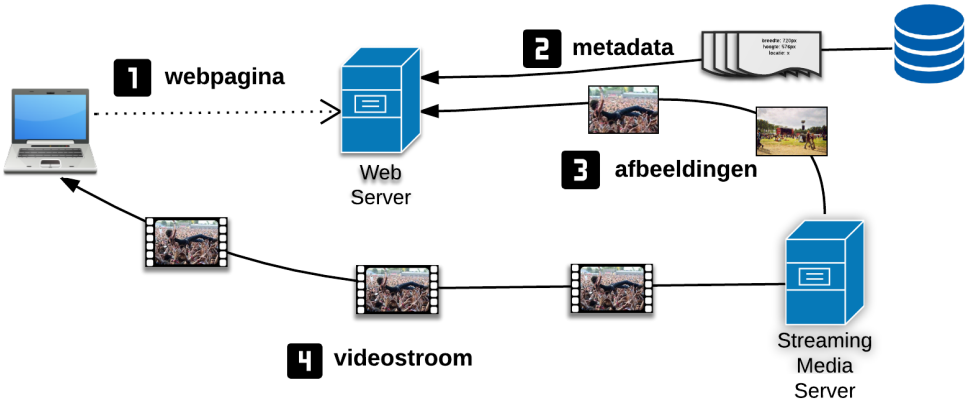
Doordat de Main Server de databank periodiek analyseert is het mogelijk om streams dynamisch te gaan starten en stoppen. Bijvoorbeeld: iemand begint te filmen en de stream wordt geselecteerd, echter na een tijdje stopt de gebruiker het toestel terug in zijn rugzak en vergeet de camera uit te zetten.

[1] Dit is nodig om het pollen van minimumeisen te vermijden. Als de minimumeisen worden geüpdatet kan de Main Server alle actieve clients uit de databank halen en deze verwittigen dat er een nieuwe versie beschikbaar is.

Wanneer dit kan afgeleid worden uit de kenmerken zal de Main Server dit detecteren en de stream stoppen wanneer nodig[2].

3.4 Bekijken van een stream

Het proces om een stream te bekijken wordt verduidelijkt in Figuur 3.3.



Figuur 3.3: De stappen die nodig zijn om een stream te bekijken

1. Om een overzicht van alle streams te verkrijgen wordt er vanuit een browser gesurft naar de webserver. Deze stelt enkele webpagina's ter beschikking met onder andere een tekstueel en grafisch overzicht van de streams die momenteel beschikbaar zijn. Deze pagina's worden periodiek ververst waardoor de gegevens steeds actueel blijven.

2. Op zijn beurt vraagt de webserver aan de databank de nodige gegevens op. Afhankelijk van de pagina is dit ofwel een lijst van alle clients die aan het streamen zijn (al dan niet gesorteerd op een kenmerk), ofwel de gegevens van de top-9 streams.

[2]Bijvoorbeeld als alle Streaming Media Servers volledig belast zijn en er betere video's beschikbaar zijn.

3. Indien de top-9 van streams werd opgevraagd worden de bijhorende frames ook gedownload. Enkel de frames vanop het huidig moment en nog niet de streams zelf, deze vergen wat tijd om op te starten. Omdat de top-9 variabel is kan het bijvoorbeeld voorkomen dat een stream wordt opgestart en daarna verwijderd voordat hij geladen is. Dit probleem komt niet voor bij afbeeldingen die in een fractie van dezelfde tijdsperiode ingeladen kunnen worden.

4. Lijken de beelden interessant? Dan is het mogelijk de stream op te starten in een mediaspeler. Hiervoor wordt rechtstreeks de bijhorende Streaming Media Server gecontacteerd (gegevens aanwezig in webpagina) en wordt de stream opgezet.

Hoofdstuk 4

Implementatie van het raamwerk

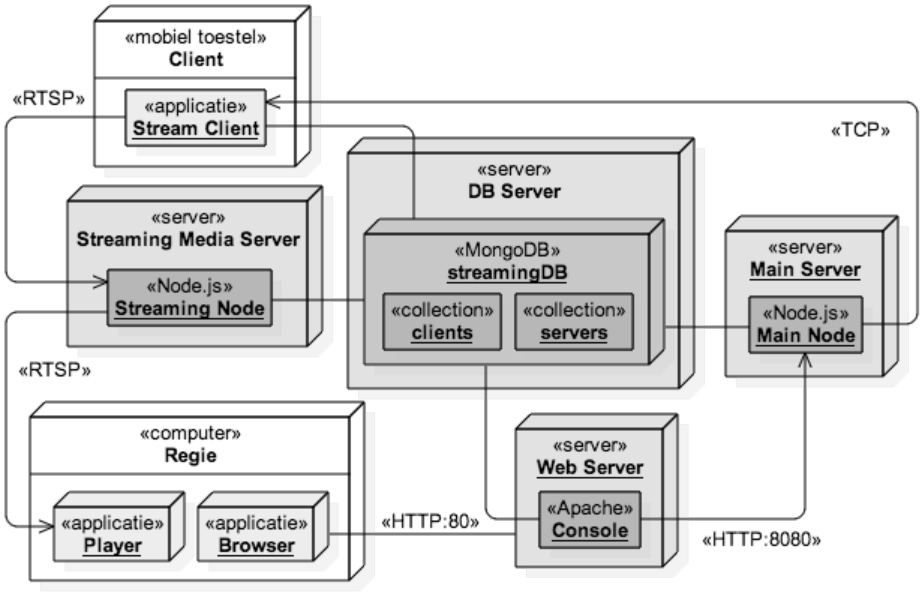
4.1 Inleiding

Dit hoofdstuk werpt een blik op de beschikbare technologieën voor elk architectuurblok om uiteindelijk een gemotiveerde keuze te maken. Ook de communicatie tussen de modules wordt verder onderzocht steunend op de literatuurstudie uit Hoofdstuk 2.

Uiteindelijk wordt er afgesloten met een uitgewerkte toepassing voor een festival. Eerst wordt het doel van de toepassing uitgelegd om zo verder te gaan naar de vereisten en een concrete implementatie. Deze implementatie wordt dan gebruikt om het raamwerk te evalueren in Hoofdstuk 5 en eventuele demonstraties te kunnen geven.

4.2 Implementatie onderdelen

In Figuur 4.1 wordt het deployment diagram uit het vorig hoofdstuk aangevuld met de implementatie details zoals de gebruikte platformen en protocols.



Figuur 4.1: Gedetailleerd deployment diagram met hierbij de gebruikte platformen en protocols

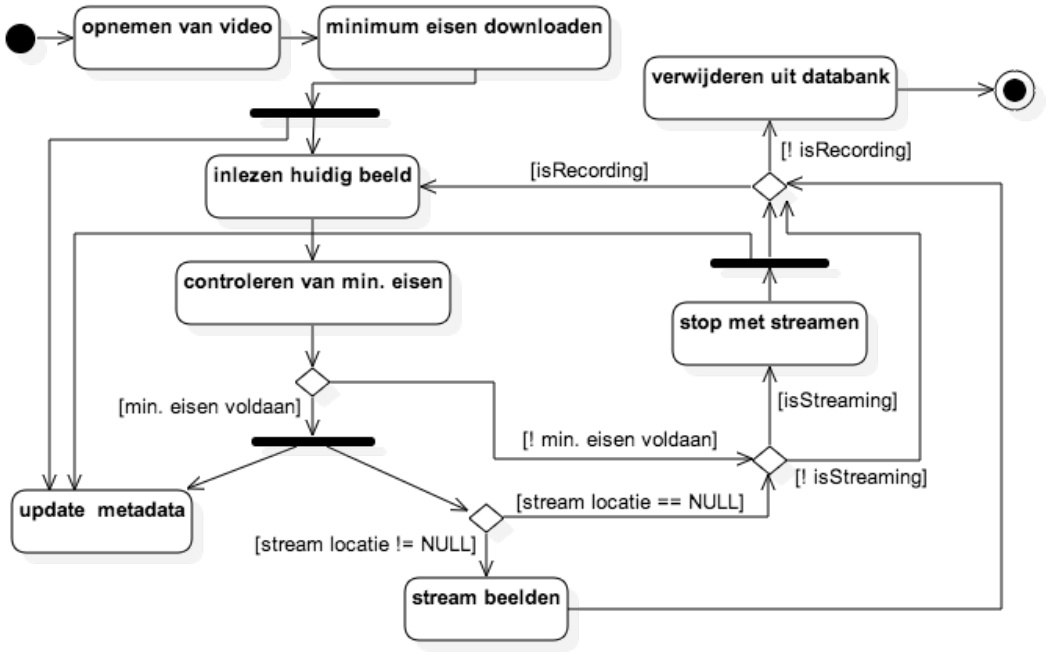
4.2.1 Client-side applicatie

De functionaliteit van de clients bestaat uit twee delen: de beelden analyseren en de beelden streamen. Voor het analyseren is er een bibliotheek nodig dat reeds efficiënte algoritmes bevat om beelden te verwerken. Bij voorkeur ondersteunt deze ook meerdere platformen zoals Android, iOS en Windows. Dit maakt het mogelijk om bij verschillende implementaties gebruik te maken van dezelfde functionaliteit. *OpenCV*[1] is hiervoor uitermate geschikt, het biedt ondersteuning voor een groot aantal platformen, is uitvoerig getest en

[1] `http://opencv.org`

geoptimaliseerd voor realtime beeldverwerking. Bovendien is de bibliotheek ook beschikbaar in C en C++ waardoor het bijna op elk toestel kan gebruikt worden. Tal van complexe analyses zijn te gebruiken met slechts één enkele functie-aanroep en zijn uitgebreid gedocumenteerd.

Voor het streamen van de beelden bestaan er tal van bibliotheken. Zo is er *libstreaming*[2] voor Android die het mogelijk maakt om beelden te streamen over RTP. Maar ook *RealTimeLibs*[3] een bibliotheek ontwikkeld voor zowel Android als iOS en met de mogelijkheid om beelden te streamen over RTMP en RTSP. Zo bestaan er nog wel een pak meer, maar *FFmpeg*[4] is vrijwel de enige die voor elk platform beschikbaar is. Bovendien ondersteund dit meer dan 30 protocols en heleboel codecs. Omwille van de flexibiliteit is er dus gekozen voor FFmpeg.



Figuur 4.2: Het proces bij een client vanaf het starten van een opname tot het einde in de vorm van een activiteitendiagram

[2] `https://github.com/fyhertz/libstreaming`
[3] `http://www.realtimelibs.com`
[4] `https://www.ffmpeg.org`

De functionaliteit van de applicatie bestaat uit vier delen: opdrachten ontvangen van de Main Server, de camera beelden analyseren, eventueel de resultaten hiervan updaten naar de databank en streamen van de beelden naar een Streaming Media Server. In Figuur 4.2 wordt het volledige proces vanaf het begin van een opname tot het einde geïllustreerd in de vorm van een activiteitendiagram.

De flow van de applicatie gaat als volgt: wanneer de gebruiker begint met het opnemen van een video wordt een lus gestart die periodiek een frame gaat analyseren. Hoeveel tijd er tussen elke analyse zit kan statisch of eventueel afhankelijk van de beschikbare rekenkracht ingesteld worden. Eens een frame ingelezen is worden de nodige analyses uitgevoerd die vermeld worden bij de minimumeisen, als de resultaten voldoen aan deze eisen wordt de metadata toegevoegd aan de databank. Zo blijft dit proces herhaald worden tot een commando ontvangen wordt van de Main Server om te starten met streamen. Wanneer dit gebeurt wordt de stream locatie ingesteld en worden vanaf de eerstvolgende iteratie de beelden gestreamd.

De commando's van de Main Server worden verzonden over TCP *sockets*. Hiervoor moet de client luisteren naar binnenkomende verbindingen op poort 8000. Deze opdrachten worden verzonden in JSON formaat volgens een vaste structuur, in code fragment 4.1 wordt als voorbeeld een mogelijk start commando gegeven.

```
{
    msg_type: "command",
    msg_body: {
        command: "start",
        server_info: {
            address: { host: "streamsrv1", port: 8000 }
        }
    }
}
```

Codefragment 4.1: Start commando van Main Server naar een client

4.2.2 Databank

De gegevens in de databank hebben een korte levensduur en wanneer deze opgevraagd worden is het belangrijk dat dit de realtime informatie is (zie deel 3.2.1). Bovendien hebben de gegevens geen vaste structuur, bij het ene toestel is bijvoorbeeld data van een accelerometer aanwezig en bij het andere niet. Door deze vereisten lijkt een niet relationele databank een goede keuze, bijvoorbeeld *MongoDB*[5]. Dit databank systeem is ontwikkeld om hoofdzakelijk flexibel, schaalbaar en performant te zijn. Het laatste punt is het belangrijkste. Query's kunnen uitgevoerd worden zonder er 100 procent zeker van te zijn of deze wel het volledig juiste resultaat terug geven of dat het toevoegen of updaten wel degelijk gelukt is. Dit is niet intuïtief, maar hierdoor wordt er minder overhead geproduceerd waardoor er efficiënter gewerkt kan worden. Het is bijvoorbeeld geen probleem wanneer het eens niet lukt om metadata te updaten want een seconde later wordt deze toch overschreven.

De flexibiliteit is ook een groot voordeel. Dankzij de document georiënteerde structuur van MongoDB vormt het geen probleem wanneer de gegevens geen vaste structuur vertonen. Het is mogelijk om één tabel van clients te hebben waarin alle clients aanwezig zijn: clients die niet voldoen aan de minimum vereisten, clients die beelden beschikbaar hebben en clients die aan het streamen zijn. Zie code fragment 4.2 ter illustratie.

```
{ _ID: <Id1>, client_IP: "10.12.53.2", brightness: 79, stream_loc: 10.1.9.1 }
{ _ID: <Id2>, client_IP: "172.18.130.71" }
{ _ID: <Id3>, client_IP: "87.66.169.200", bitrate: 573 }
```

Codefragment 4.2: Voorbeeld dataset uit de clients collectie

Tot slot is er nog de schaalbaarheid. In het geval dat de databank een bottleneck is kan de collectie opgesplitst worden, de ene helft van de clients in een databank en de andere helft in een andere. Hierdoor worden de gegevens verspreid over meerdere servers en dus ook het zware update verkeer van de clients. Bij het opvragen van alle gegevens door de Main Server of web interface zullen nog steeds alle servers gecontacteerd moeten worden maar

[5] `https://www.mongodb.org`

aangezien dit veel minder voorkomt (het aantal clients is in normale omstandigheden veel groter dan het aantal computers van de regie en de Main Server) lijkt dit een goede oplossing.

4.2.3 Streaming applicatie

Deze applicatie moet streams die afkomstig zijn van clients kunnen buffferen, analyseren en de resultaten hiervan updaten naar de databank. Voor de analyse is het handig dat er opnieuw gebruik gemaakt wordt van OpenCV omwille van dezelfde redenen als bij de clients (uitgebreid aanbod van algoritmes, uitvoerig getest en geoptimaliseerd voor realtime beeldverwerking). Er moet dus een mechanisme aanwezig zijn dat het mogelijk maakt om deze analyses te starten bij het opstarten van een stream en te stoppen indien nodig.

Uit de literatuurstudie (in deel 2.1.3) blijkt dat het Real-time Transport Protocol (RTP) uitermate geschikt is voor het realtime streamen van audio en video. Dus bij voorkeur maakt de applicatie hier gebruik van en is het bovendien geïmplementeerd op een platform dat hier goed met kan omgaan, zoals bijvoorbeeld Node.js. Een citaat van op hun website[6]:

> Node.js is een platform [...] voor het eenvoudig bouwen van snelle, schaalbare netwerkapplicaties. [...] ideaal voor data-intensieve realtime applicaties die uitgevoerd worden op gedistribueerde systemen.

Uiteindelijk is een reeds bestaande applicatie gekozen die verder uitgewerkt is. Nao Iizuka heeft in Node.js enkele streaming protocols in combinatie met enkele audio en video codecs geïmplementeerd [14]. De beschikbare protocols zijn RTSP en RTMP. Het eerste maakt achter de schermen gebruik van RTP voor de overdracht van audio- en videodata, dus deze voorwaarde is reeds voldaan. Bovendien is de code publiek beschikbaar waardoor het makkelijk is om extra informatie uit streams te halen zoals bijvoorbeeld pakketverlies.

[6] `https://nodejs.org`

Ook is het mogelijk om OpenCV rechtstreeks in Node.js aan te spreken maar vanwege de vele open en oude issues in [15] is er voor gekozen om dit te implementeren via een Python script.

De aangeboden codecs zijn op het moment van schrijven beperkt tot het H.264 formaat voor video en het AAC formaat voor audio, meer specifiek: AAC-LC, HE-AAC v1 of HE-AAC v2.

4.2.4 Dirigerende applicatie

De Main Server moet volgende taken voltooien:

- Minimum eisen beschikbaar stellen voor clients.
- Commando's versturen naar clients, zoals start en stop met streamen.
- De clients collectie analyseren met als doel nieuwe clients ontdekken en overbelaste netwerk links te ontlasten.
- Het schalen van Streaming Media Servers aan de hand van gegevens in de servers collectie.
- Mechanisme voorzien om de minimum eisen en filters aan te passen via de web interface en hierbij alle clients op de hoogte te brengen.

Deze applicatie moet dus voornamelijk communiceren met andere elementen in het raamwerk. Al deze taken gebeuren onafhankelijk[7] van elkaar dus is er nood aan een platform waarbij er makkelijk asynchrone methodes gemaakt kunnen worden. Het communiceren met de clients gebeurt via sockets en voor de communicatie met de databank zijn MongoDB drivers nodig. Deze eisen zijn dus niet zo strikt en praktisch elke programmeertaal of platform kan dit aan. Node.js is gekozen omwille van de eenvoud om asynchrone methodes te implementeren en de vlotte communicatie met MongoDB.

[7]Strikt gezien zullen de Streaming Media Servers niet gaan neerschalen wanneer er nieuwe clients worden toegevoegd omwille van stabiliteitsredenen. Dus dit is een kleine afhankelijkheid tussen taken van de Main Server.

Het starten en stoppen van clients gebeurt dynamisch. Zoals in Figuur 3.2 te zien is maakt een client zich onmiddellijk kenbaar in de databank wanneer hij begint te filmen. Door periodiek de databank door te nemen ontdekt de Main Server nieuwe clients. Indien er nog Streaming Media Servers zijn die niet volledig belast worden zal de Main Server een commando sturen naar de eventueel nieuwe client om te starten met streamen naar een bepaalde server. Wanneer alle Streaming Media Servers volledig belast zijn is het ook mogelijk dat een client gestopt wordt met een lagere score om plaats vrij te maken.

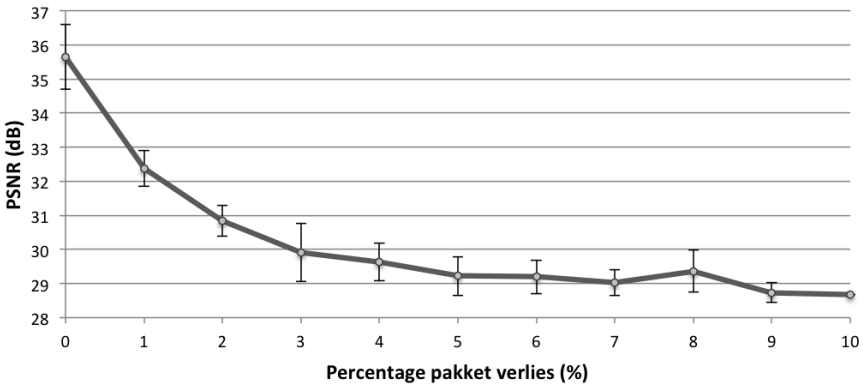
Als een bepaalde netwerklink overbelast is zal er pakketverlies optreden. De Streaming Media Servers zullen voor elke stream het aantal pakketten bijhouden die te laat zijn aangekomen of verloren gegaan in een bepaald tijdsinterval. Clients gaan hun netwerkadres bepalen en dit meesturen met de kenmerken van de beelden. Op deze manier is het mogelijk om voor elke netwerklink het totale pakket verlies te berekenen door middel van een aggregatie query. Indien het pakketverlies boven een bepaalde grens gaat worden de clients met de laagste score op die link gestopt tot het pakketverlies terug onder een aanvaardbaar niveau ligt.

Om te bepalen welke grens aanvaardbaar is werd volgende test uitgevoerd. Een client streamt een video naar een Streaming Media Server en deze gaat de gestreamde beelden bewaren. Tijdens het streamen wordt ervoor gezorgd dat er een bepaald percentage van de verstuurde pakketten verloren gaat.

Achteraf worden beide video's vergeleken en de gemiddelde *PSNR* (Peak Signal-to-Noise Ratio) bepaald van alle corresponderende frames. De PSNR wordt berekend volgens [25].

Deze test wordt vijf keer herhaald met dezelfde video voor verschillende niveau's van pakket verlies. Het totaal aantal verloren pakketten werd telkens bijgehouden om te kunnen verifiëren achteraf. In Figuur 4.3 worden de resultaten getoond voor een video met een hoge resolutie (1920 x 1080).

Hierbij valt op dat de kwaliteit onmiddellijk sterk begint te dalen. Visueel is vastgesteld dat één procent pakket verlies aanvaardbaar is maar dit is



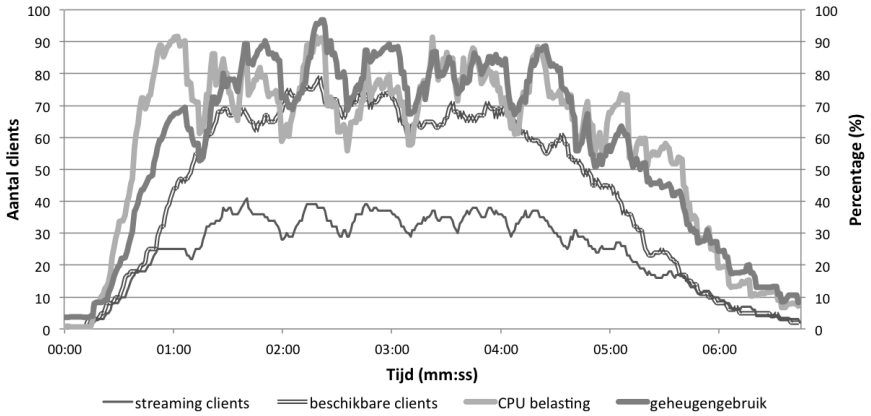
Figuur 4.3: De PSNR van de gestreamde video tegenover de oorspronkelijke video met een bepaald percentage pakketverlies.

natuurlijk afhankelijk van het doel van de gestreamde beelden.

Het schalen van Streaming Media Servers gebeurt aan de hand van gegevens in de servers collectie van de databank. Die collectie bevat onder andere de CPU belasting, het geheugengebruik en de hoeveelheid netwerkverkeer van alle Streaming Media Servers. Op basis van grenzen wordt bepaald wanneer een server al dan niet volledig belast is. Indien twee servers niet volledig belast zijn en er zijn geen nieuwe clients beschikbaar zal de Main Server beginnen met clients van de ene server naar de andere te verplaatsen. Met andere woorden de Streaming Media Servers zullen gaan neerschalen. Het opschalen gebeurt automatisch, indien een server volledig belast is wordt de volgende server gekozen. Een vereiste is dat er reeds een aantal servers beschikbaar zijn, opgestart en draaiende. Dit proces zou eventueel geautomatiseerd kunnen worden door servers dynamisch te laten opstarten in een cloud omgeving.

De grenzen die optimaal zijn voor het schalen zijn afhankelijk van vele factoren. Zo zal onder andere de snelheid waarmee clients worden opgezet van groot belang zijn. Indien er bijvoorbeeld één client per seconde wordt opgezet heeft de server tijd om te stabiliseren en zijn de gegevens over de server belasting accurater wanneer de volgende client wordt opgezet. Wanneer agres-

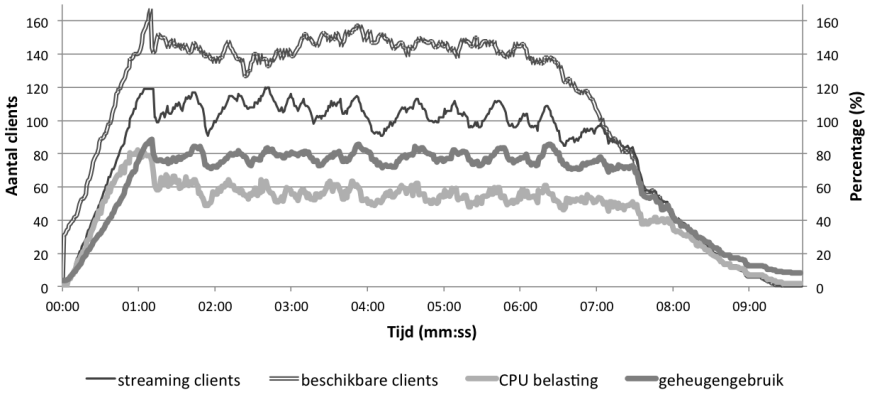
siever te werk wordt gegaan en bijvoorbeeld vijf clients per seconde worden opgezet zullen deze grenzen veel braver moeten zijn om ervoor te zorgen dat de server niet overbelast wordt.



Figuur 4.4: Stresstest: de CPU belasting en het geheugengebruik van één Streaming Media Server waarbij één video met **hoge resolutie** (1920x1080) vanop een groot aantal clients werd aangeboden

In Figuur 4.4 wordt de CPU belasting en het geheugengebruik weergegeven bij het streamen van een video met hoge resolutie (1920x1080). Dezelfde video wordt vanop een groot aantal clients aangeboden zodat er steeds meer beschikbaar zijn dan de server aankan (stresstest). Bij deze testopstelling werden twee clients per seconde opgezet en lag de limiet voor CPU belasting en geheugengebruik op 80%. Wanneer het experiment enkele keren herhaald werd kwam het geheugengebruik een paar keer in de gevarenzone boven de 95%, de grens van het geheugengebruik daalt dus best nog wat in deze situatie.

Een andere factor die een grote invloed heeft op de belasting van de Streaming Media Servers is de bitrate van de streams. Wanneer we het vorige experiment herhalen voor een video met een lagere resolutie (640x360) bekomen we de resultaten in Figuur 4.5.



Figuur 4.5: Stresstest: de CPU belasting en het geheugengebruik van één Streaming Media Server waarbij één video met **lage resolutie** (640x360) vanop een groot aantal clients werd aangeboden

Bij herhalingen van dit experiment is het geheugengebruik nooit meer dan 90% geweest en op de grafiek valt onmiddellijk op dat de gegevens over de belasting minder fluctueren. Hiermee komen we bij een laatste factor die nog een grote invloed heeft op deze grenzen en dat is de gebruikte hardware. Een server die meer dan honderd streams kan verwerken zal niet zo een grote hinder ondervinden wanneer er nog een stream van hetzelfde type wordt gestart.

Uit de figuren 4.4 en 4.5 kan tenslotte nog afgeleid worden dat een server van dit type, 2 x Quad core Intel E5520 (2.2GHz) CPU en 12GB RAM, ongeveer 30-40 hoge resolutie streams kan verwerken en ongeveer 100-120 lage resolutie streams met de vermelde instellingen.

4.2.5 Web interface

Als laatste is er de web interface die ervoor zorgt dat de regie de minimumeisen kan instellen en de databank visueel kan weergeven. In principe kan dit verwezenlijkt worden in de Main Server maar om modulair te kunnen werken is dit opgesplitst.

De webinterface bestaat uit drie delen:

- top-9 van streams grafisch voorgesteld,
- tekstueel overzicht van alle streams met hun kenmerken,
- een paneel om de minimum vereisten en filters in te stellen.

De top-9 van streams kan bepaald worden door aan de databank de negen clients met de hoogste score op te vragen. Aan de hand van deze informatie kunnen de bijhorende afbeeldingen gedownload worden van de Streaming Media Servers. Zie Figuur A.1 voor een screenshot.

Het tekstueel overzicht is gelijkaardig maar in plaats van de negen clients met de hoogste score op te vragen worden alle clients opgevraagd. De bijhorende afbeeldingen worden niet gedownload om het overzicht te bewaren. Zie Figuur A.2 voor een screenshot.

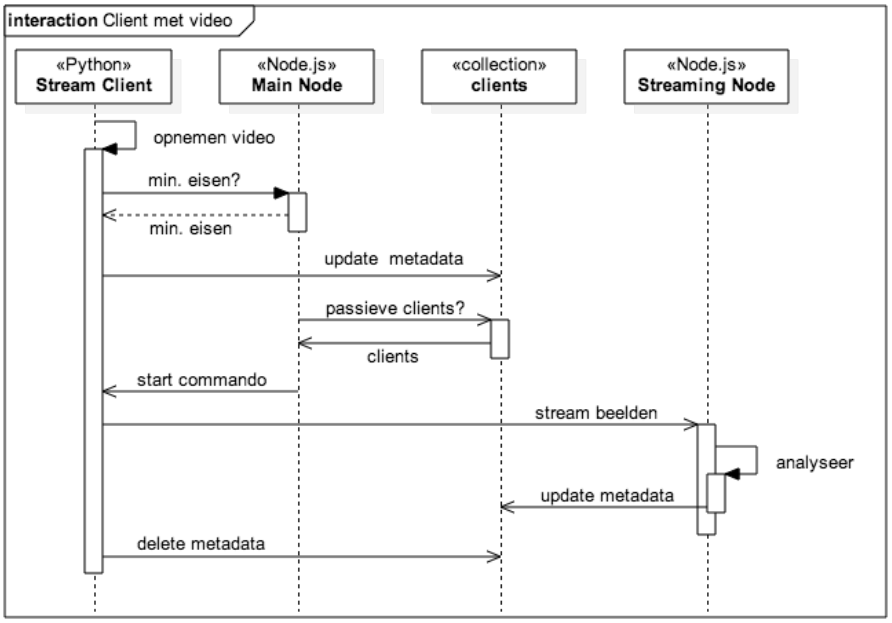
Het paneel met instellingen geeft een overzicht van de huidige instellingen van de Main Server en maakt het mogelijk om deze aan te passen in deel 4.4.2 wordt hier verder op in gegaan. Zie Figuur A.3 voor een screenshot.

Voor elke weergave is er een apart PHP bestand gemaakt maar dit kan in eender welke programmeertaal. Het enige dat deze (web)interface moet kunnen is communiceren met de databank en afbeeldingen downloaden over HTTP.

4.3 Interactie tussen de componenten

In Figuur 4.6 wordt verduidelijkt hoe alle componenten onderling interageren aan de hand van een vereenvoudigd sequentiediagram. Hierbij wordt er vanuit gegaan dat aan alle eisen voldaan is en dat er een Streaming Media Server beschikbaar is.

Eens de client begint met het opnemen van de video zal deze beginnen met het downloaden van de minimum vereisten. Wanneer hieraan voldaan is wordt de metadata verzonden naar de databank en vroeg of laat ontdekt door de Main Server. Als deze de nieuwe client ontdekt heeft wordt een start-commando



Figuur 4.6: De communicatie tussen de verschillende componenten van het raamwerk, weergegeven in een vereenvoudigd sequentie diagram.

verstuurd met hierin het adres van een beschikbare Streaming Media Server. Hierop gaat de client de beelden beginnen streamen naar de gewenste locatie. Op deze server worden dan nog eens de beelden geanalyseerd en extra metadata toegevoegd aan de databank. Als de opname gedaan is worden de beelden niet meer gestreamd en zal de server het proces automatisch stoppen. De client verwijdert tenslotte nog de metadata in de databank en de interactie is afgelopen.

4.4 Een uitgewerkte toepassing

In dit deel wordt een mogelijke toepassing uitgewerkt van het raamwerk. Het doel is een festival dat video content van bezoekers wil streamen naar een live videostroom op het web. De vereisten hiervoor zijn als volgt:

- Een gevarieerd aanbod van filters waardoor ongewenste beelden kunnen weggefilterd worden.
- Dynamisch aanpassen van de selectiecriteria.
- Optimaal gebruik van een aantal beschikbare Streaming Media Servers.

4.4.1 Filters

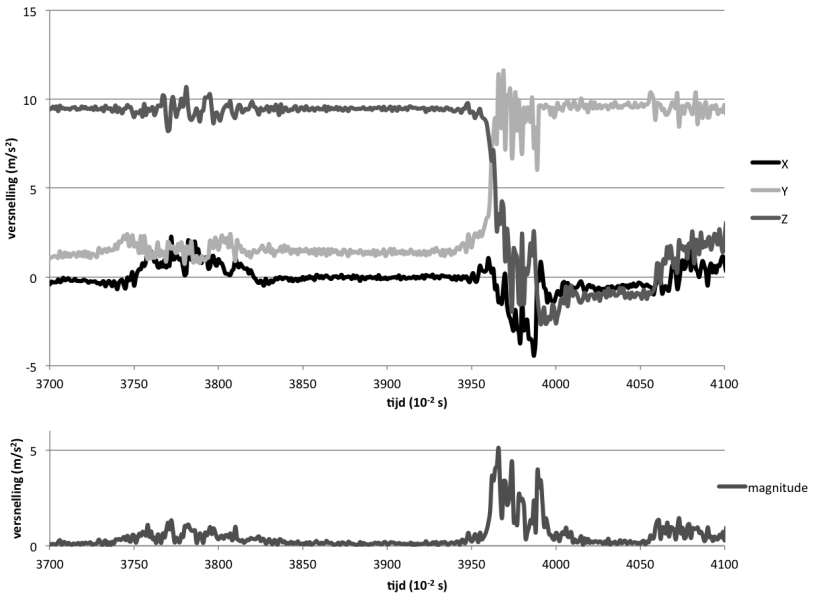
Voor het selecteren van content zijn er een aantal relevante filters geïmplementeerd. Één ervan werkt met data van de accelerometer, een sensor die beweging registreert. Met andere woorden het is mogelijk om te bepalen hoeveel een gebruiker aan het bewegen is met zijn toestel. De data komt binnen in van de vorm van versnelling langs een bepaalde as in een drie dimensionale ruimte. Om hieruit af te leiden hoeveel een toestel beweegt wordt de magnitude van het signaal berekend zoals in formule 4.1.

$$\text{magnitude} = \sqrt{x^2 + y^2 + z^2} \tag{4.1}$$

Hierdoor wordt de beweging van de camera uitgedrukt in een scalaire waarde. Wanneer de camera veel beweegt of schokt zal dit getal een grotere waarde hebben dan bij een toestel in rust.

Omdat de zwaartekracht steeds een invloed heeft op de accelerometer (g = $9.81 m/s^2$) zal het resultaat niet de werkelijke versnelling van het toestel weergeven. De oplossing hiervoor is de zwaartekracht uit het signaal te verwijderen. Maar omdat het toestel niet altijd in dezelfde positie wordt gehouden en bovendien niet steeds gealigneerd is met de zwaartekracht, kan het voorkomen dat de zwaartekracht wisselt van as of verdeeld is over meerdere

assen. Door een low-pass filter toe te passen kunnen we de invloed van de zwaartekracht isoleren. Trekken we hierna dit resultaat af van de oorspronkelijke data bekomen we de gegevens zonder invloed van de zwaartekracht (high-pass filter). Voor het resultaat van de filter en de berekening van de magnitude zie Figuur 4.7.



Figuur 4.7: Grafiek van accelerometer data. Bovenaan de ruwe gegevens van de sensor, onderaan de magnitude van het signaal na verwerking.

De lichtsensor meet de verlichtingssterkte van de omgeving, meestal in lux. Typisch liggen de waardes tussen 1 en 30 000 lux, hoe groter het getal, hoe helder de lichtsterkte van de omgeving. Bijvoorbeeld op een bewolkte dag is de lichtsterkte typisch 10 000 lux, op een heldere dag gaat dit naar 20 000 lux (indirect zonlicht). Hierdoor kan er bepaald worden of het toestel bijvoorbeeld in een rugzak of zeer donkere omgeving aan het filmen is. De kans dat er dan nuttige beelden worden geproduceerd zal gering zijn.

Nog een extra controle om donkere beelden er van tussen te halen kan op basis van de helderheid van de beelden. Om dit te berekenen wordt de gemiddelde waarde van elk kleur kanaal berekend over het gehele frame en

vervolgens het gemiddelde van alle kanalen (RGB). Omdat deze waardes nogal kunnen fluctueren, bijvoorbeeld wanneer iemand voor de camera loopt of als de camera zijn witbalans aan het aanpassen is, wordt het lopende gemiddelde van dit resultaat genomen over de laatste 50 frames.

Een andere vorm van beelden die niet bruikbaar zijn onscherpe beelden. Hiervoor wordt een eenvoudige methode gebruikt die weergeeft in welke mate beelden onscherp zijn. Eerst worden randen in het frame gedetecteerd door middel van het Canny randdetectie-algoritme [24]. In de gegenereerde afbeelding zijn pixels die niet zwart zijn een rand in de oorspronkelijke afbeelding. Het aantal niet zwarte pixels in een frame wordt dan genormaliseerd door het delen door de grootte van het frame (breedte $\times$ hoogte). Hoe hoger dit getal, hoe meer randen er gedetecteerd zijn of hoe scherper het beeld is.

Tot slot is er nog de hoogte en breedte van beelden beschikbaar. Zo is het mogelijk om een minimum resolutie te eisen, het is echter wel geen garantie dat beelden met een hoge resolutie van goede kwaliteit zijn. Ook wordt de gemiddelde bitrate bepaald maar dit is louter informatief om een idee te krijgen hoe groot de datastroom op een bepaalde netwerk link is.

Vanwege de beperkte rekenkracht die nodig is om vorige kenmerken te berekenen zijn deze telkens geïmplementeerd aan de client zijde. Maar deze kunnen eventueel beschikbaar gesteld worden door dezelfde logica toe te passen langs de server zijde. Ter illustratie is er een filter aan de server zijde uitgewerkt die bepaalt hoeveel variatie er tussen opeenvolgende frames is. Eerst worden de frames omgezet naar grijswaarden en vervolgens wordt het absoluut verschil berekend tussen de twee frames. Dit resultaat wordt vervolgens genormaliseerd door te delen door de resolutie van de frames.

4.4.2 Selectie

Via de web interface kunnen de filters ingesteld worden (zie Figuur 4.8). In gebied 1 bevindt zich een keuzelijst waarin alle beschikbare filters aanwezig zijn, hiermee is het mogelijk om een filter toe te voegen. Eens een filter is toegevoegd komt deze bij de lijst van filters te staan en is het mogelijk om

deze te configureren in gebied 2. Er zijn vier mogelijkheden om een filter in te stellen:

- **Blanco**: de gevraagde metadata wordt berekend en geüpload naar de databank, hierdoor is het mogelijk om enkel een minimumeis in te stellen (minimum of maximum waarde) of de data vooraf te inspecteren.
- **Lager is beter**: hoe lager de waarde van deze metadata is, hoe beter de score voor deze filter. Bijvoorbeeld voor beelden waarbij de camera weinig beweegt te selecteren.
- **Hoger is beter**: hoe hoger de waarde van deze metadata is, hoe beter de score voor deze filter. Bijvoorbeeld om heldere beelden te selecteren.
- **Dichter is beter**: hoe dichter de waarde van deze metadata bij een opgegeven waarde ligt, hoe beter de score voor deze filter. Bijvoorbeeld bij de resolutie van beelden: een te hoge resolutie zorgt voor een grote datastroom, een te lage resolutie voor slechte kwaliteit.

Figuur 4.8: Screenshot van web interface waar filters ingesteld kunnen worden

Elke manier heeft zijn eigen methode om een score te berekenen. Stel α gelijk aan de waarde van de metadata, dit kan elk positief reeël getal zijn. Dan wordt bij *lager is beter* de score berekend volgens formule 4.2.

$$\text{score} = \frac{\max - \alpha}{\max} \tag{4.2}$$

Er is nood aan een maximum waarde zodat de score kan geschaald worden naar een getal tussen 0 en 1. Wanneer de metadata boven de maximum waarde gaat voldoet deze niet aan de minimum eisen en wordt de videostroom niet geüpload.

Wanneer *hoger is beter* wordt gebruikt om de score te bereken zal formule 4.3 toegepast worden. Om hier de score te kunnen schalen naar een getal tussen 0 en 1 is er nood aan een minimum waarde (min) en een geschatte bovengrens (mid). De minimum waarde werkt gelijkaardig aan de maximum waarde uit vorige functie en zorgt ervoor dat er een minimumeis is voor deze metadata. De geschatte bovengrens is nodig voor te herschalen. Wanneer deze waarde overschreden wordt zal opnieuw een getal groter dan 1 worden geproduceerd en zal de score verbeteren.

$$\text{score} = \frac{\alpha - \text{min}}{\text{mid} - \text{min}} \tag{4.3}$$

Tot slot is er nog een laatste formule nodig om de afstandsfunctie *dichter is beter* te kunnen verwezenlijken. Hierbij moet de metadata in een bepaald interval zitten: $]\text{min}; \text{max}[$ waarbij geldt: $\text{min} < \text{mid} < \text{max}$. Hoe dichter de waarde bij de ideale waarde *mid* zit hoe dichter de score bij 1 zal aanleunen. Zie formule 4.4.

$$\text{score} = \frac{1 - |\text{mid} - \alpha|}{max(\text{mid} - \text{min}, \text{max} - \text{mid})} \tag{4.4}$$

Om verschillende filters te combineren is het ook nog mogelijk om een gewicht aan de score mee te geven. In formule 4.5 wordt de totaalscore voor een video berekend waarbij g_i het gewicht en s_i de score van filter i is.

$$\text{totaalscore} = \sum_{i=1}^{n} g_i \cdot s_i \tag{4.5}$$

Uiteindelijk zullen de negen streams met de hoogste totaal score getoond worden aan de regie op de top-9 pagina.

Hoofdstuk 5

Evaluatie van het raamwerk

5.1 Inleiding

In dit laatste hoofdstuk wordt de uitgewerkte toepassing gebruikt om het raamwerk te evalueren. Eerst wordt de testomgeving besproken waarop deze evaluaties hebben plaatsgevonden. Daarna wordt de dataset besproken die gebruikt is om de evaluaties uit te voeren, onder andere de beperkingen die deze met zich meebrengt. Vervolgens wordt de toepassing geëvalueerd met een simulatie die overeenstemt met wat men kan verwachten op een doorsnee festival.

5.2 Testomgeving

De infrastructuur die beschikbaar is om het raamwerk te evalueren is die van het technisch testcentrum van iMinds: iLab.t [1], meer specifiek de *Virtual Wall*. De omschrijving op de website luidt als volgt:

> De Virtual Wall is een algemene testomgeving voor geavanceerde netwerken, gedistribueerde software en dienstenevaluatie. Ze ondersteunt het onderzoek naar schaalbaarheid van toepassingen. [...]

[1] `http://ilabt.iminds.be`

De testomgeving bestaat uit een grote verzameling servers waarop het mogelijk is om de architectuur die besproken is in Hoofdstuk 3 volledig uit te werken. Bovendien kan er ook een netwerktopologie met een configureerbare bandbreedte en delay geconstrueerd worden die een realistische situatie kan nabootsen.

Het was niet haalbaar om voldoende mobiele toestellen te verzamelen en deze samen aan te sturen. Hierdoor worden deze gesimuleerd door de servers die beschikbaar zijn. Ter herhaling: deze toestellen hebben maar twee vereisten: een dataverbinding en een camera (zie deel 3.2). Een mobiele dataverbinding verbinding kan gesimuleerd worden door middel van netwerkverbindingen te creeëren met gelijkaardige eigenschappen. De camera en de eventuele sensoren die aanwezig zijn op een mobiel toestel ontbreken echter. Dit wordt echter opgevangen door gebruik te maken van een dataset waarin zowel videodata als metadata beschikbaar is.

Om de mobiele toestellen op één of meerdere servers te simuleren werd een Python script gebruikt dat de nodige functionaliteit verwezenlijkt. Dit script leest video bestanden in tesamen met de eventueel bijhorende metadata. Python is gekozen omdat hierin snel prototypes gemaakt kunnen worden van algoritmes die gebruik maken van OpenCV en de uitvoeringstijd is nagenoeg hetzelfde als die bij C of C++ [13].

Voor de Streaming Media Servers werden telkens servers gebruikt met twee Quad core Intel E5520 (2.2GHz) processors en 12GB RAM. Deze hardware zal een grote invloed hebben op de uitgevoerde testen zoals vermeld in 4.2.4.

5.3 Dataset

De dataset moet beschikken over zowel video- als metadata en voldoende groot en gevarieerd zijn. Zo een dataset construeren zou veel tijd in beslag nemen en hierdoor is er gezocht naar een bestaand alternatief. De dataset van het *ICoSOLE*[2] (Immersive Coverage of Spatially Outspread Live Events)

[2] `http://icosole.eu`

project beschikt over veel van de deze eigenschappen maar bevat ook enkele beperkingen die verder besproken worden.

De video's bevatten als onderwerp een deel van het BBC Philharmonic orkest en zijn ook opgenomen in hun oefenruimte te Salford, in de buurt van Manchester (Groot-Brittannië). Meteen brengt de locatie reeds een beperking met zich mee die op een doorsnee festival meestal niet van toepassing is. Zo zijn alle beelden indoor opgenomen en zullen de locatie gegevens van de beelden niet accuraat genoeg zijn om een selectie op uit te voeren. Vandaar dat er in het vorige hoofdstuk ook geen filter op basis van GPS data is besproken.

De beelden in de dataset zijn dan ook minder gevarieerd dan men doorgaans zou aantreffen op grotere evenementen. Bijna alle beelden bevatten dezelfde achtergrond (de muren van de oefenruimte) en het onderwerp is zo goed als altijd een muzikant. Bij een doorsnee festival verwacht je eerder beelden van mensen op het terrein, het podium, de bar, de camping, backstage, enzovoorts.

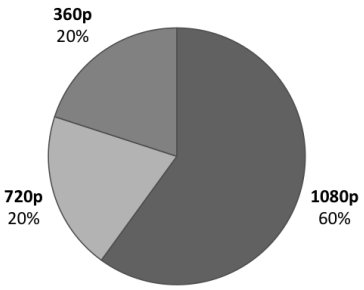
De gelijkenissen die wel van toepassing zijn met een festival, is bijvoorbeeld de variëteit aan toestellen. Zo zijn er originele beelden beschikbaar van onder andere GoPro's, iPhone's, iPad's, Android smartphones en Android tablets. Bovendien werd ook alle ruwe data van de sensoren geregistreerd en beschikbaar gemaakt in de vorm van een XML bestand.

5.4 Simulatie van een groot evenement

Om de schaalbaarheid van het raamwerk te kunnen evalueren werd een doorsnee festival gesimuleerd. De bedoeling is om het systeem te pushen, dus moeten zoveel mogelijk streams worden opgezet en bijgevolg worden de minimumvereisten uitgezet. Andere selectiecriteria hebben enkel invloed op welke video's getoond worden aan de regie en spelen dus geen rol bij het testen van de schaalbaarheid.

Eerst werd er manueel een representatieve selectie gemaakt uit de dataset

van beelden die men kan verwachten. Zo is ervoor gezorgd dat er een gevarieerd aanbod aan resoluties is (zie Figuur 5.1) en dat de video's verschillende lengtes hebben, gaande van enkele seconden tot maximum tien minuten. In totaal werden 30 verschillende video's geselecteerd, waarvan de helft beschikt over data van sensoren.

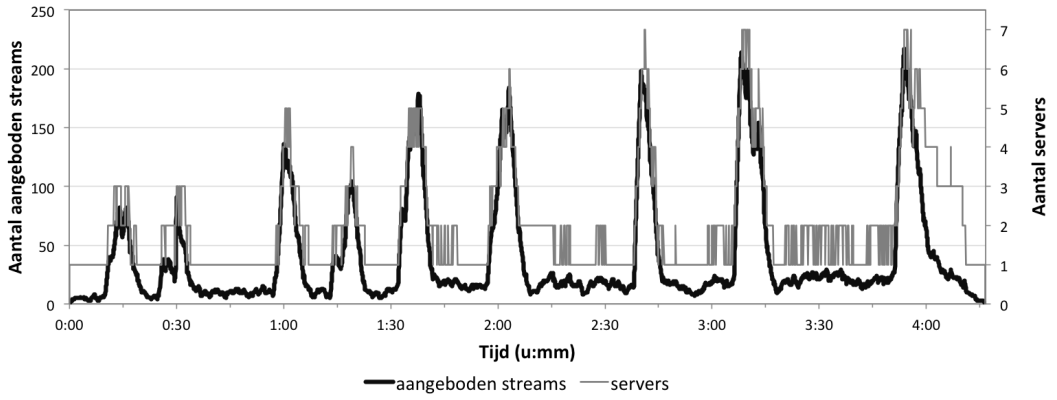


Figuur 5.1: Het aantal video's met een bepaalde resolutie dat aanwezig is in de manuele selectie uit de dataset, weergegeven in de vorm van een taartdiagram.

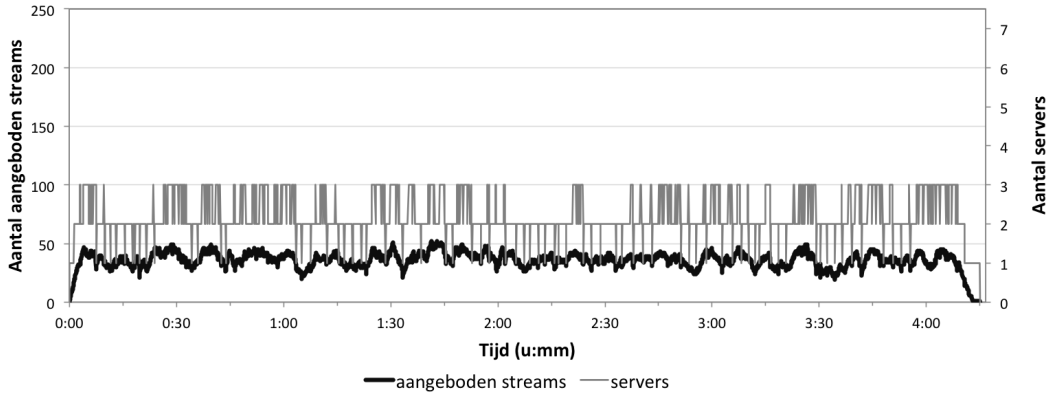
In totaal duurde elke simulatie ongeveer vier uur met een totaal van gemiddeld 3750 video's[3] die gestreamd werden. De video's werden telkens willekeurig gekozen uit de manuele preselectie. Om een realistische simulatie te verkrijgen werd een proces opgesteld waarbij het aantal gestarte streams geleidelijk aan toeneemt naar het einde toe. Met de gedachtegang dat naarmate het later wordt er steeds meer mensen aanwezig zijn op het festival. Ook zal er regelmatig een piek ontstaan, met tussenpauze's van 10 tot 40 minuten. Dit stelt bijvoorbeeld een grote hit van een groep voor of iets opmerkelijk waarbij veel aanwezigen naar hun mobiel toestel grijpen. Deze verdeling steunt op het *Paretoprincipe* waarbij een 80-20 verhouding van toepassing is, zo zal op 20% van de tijd 80% van het totale aantal clients aan het streamen zijn.

[3]Minimum: 3500, maximum 4000. Het aantal aangeboden streams en gesimuleerde clients was beperkt door het aantal beschikbare servers in de testopstelling.

In Figuur 5.2a zie je het resultaat van een simulatie, de ene curve stelt het aantal clients voor dat momenteel aan het streamen is en de andere het aantal Streaming Media Servers dat op ieder tijdstijd in gebruik is. Op de grafiek is te zien dat het aantal servers mee schaalt met het aantal streams en in staat is om alle streams te verwerken. Het schalen volgt vrij strikt het aantal clients omdat de Streaming Media Servers reeds opgestart zijn en klaar staan. Indien het opstarten van een server langer duurt kan er eventueel gekozen worden om langer te wachten met neerschalen zodat het aantal servers minder fluctueert.



(a) Pareto



(b) Poisson

Figuur 5.2: Het aantal aangeboden streams en benodigde Streaming Media Servers om deze te verwerken doorheen het verloop van de simulatie van (a) Pareto en (b) Poisson.

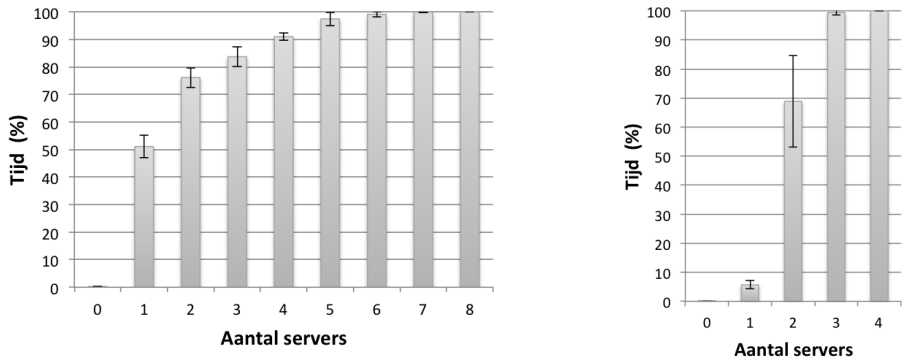
Vanwege de pieken in de verdeling kan het voorkomen dat er zich heel veel clients tegelijkertijd aanbieden en zullen de Streaming Media Servers snel moeten schalen. Hiervoor zijn de instellingen op de Main Server aangepast, om een piek te kunnen verwerken worden vijf clients per seconde opgestart, anders gezegd maximaal 300 clients per minuut. Zoals in deel 4.2.4 reeds vermeld is zullen de grenzen voor het schalen van Streaming Media Servers dus ook verzacht moeten worden. Met de grenzen ingesteld op een maximale CPU belasting van 60% en een maximaal geheugengebruik van 50% werden goede resultaten bekomen. De Streaming Media Servers bereikten soms de gevarenzone van 95% maar herstelden snel.

Om te kunnen bepalen wat de invloed van de pieken is op het aantal Streaming Media Servers werd ook een andere verdeling gebruikt, een verdeling waarbij de streams gelijkmatig verdeeld werden over de volledige periode. Bij het vorige experiment was het totale aantal streams dat werd opgezet gemiddeld 3750 of gemiddeld 15 streams per minuut (totale looptijd van het experiment was 250 minuten). Om de streams gelijkmatig te verdelen is gekozen om het aantal streams dat per minuut wordt opgestart te laten verlopen volgens een Poisson verdeling met $\lambda = 15$. Het resultaat van een simulatie volgens deze verdeling is te zien in Figuur 5.2b.

Wanneer het raamwerk gebruikt wordt in een cloud omgeving worden extra virtuele machines opgestart om extra Streaming Media Servers te voorzien. Typisch wordt een kost aangerekend voor de tijd dat een virtuele machine op staat. De totale kost is dan het gemiddeld aantal servers in gebruik tijdens de simulatie keer de totale tijdsduur van de simulatie.

Om te bepalen hoeveel verschil er op de kostprijs zit van beide simulaties werden de gemiddeldes berekend. De simulaties werden vijf maal herhaald voor beide toevalsprocessen om een statistisch significant resultaat te bekomen. Gemiddeld waren er bij de Pareto simulaties $2,014$ servers nodig en bij de Poisson simulaties lag dit iets hoger, namelijk: gemiddeld $2,257$ servers. In een cloud omgeving zou de kost dus gelijkaardig zijn, de meest waarschijnlijke verdeling (Pareto) zou in dit geval zelfs goedkoper zijn.

Bij een private server opstelling hangt de kost af van het maximaal aantal servers dat nodig is om alle streams te kunnen verwerken. In dit geval zou de Pareto simulatie een veel hogere kostprijs hebben aangezien deze dubbel zoveel servers nodig heeft dan de Poisson simulatie. In Figuur 5.3 wordt dit verduidelijkt door middel van een staafdiagram.



(a) Pareto **(b)** Poisson

Figuur 5.3: Het percentage van de tijd dat een bepaald aantal Streaming Media Servers voldoende is tijdens de simulaties.

Hoofdstuk 6

Conclusie

Op grote evenementen worden steeds meer en meer mobiele toestellen met camera's en een dataverbinding meegenomen. Organisatoren hebben dan ook steeds meer interesse in de beelden die worden opgenomen tijdens de evenementen met die toestellen. Vandaar was het doel van deze masterproef om een algemeen raamwerk te construeren waarbij het mogelijk is om op een intelligente manier een selectie te maken uit live videostromen afkomstig van mobiele toestellen.

De oplossing die is voorgesteld bestaat uit een modulair raamwerk waarbij centraal een databank staat die alle gegevens verzamelt over beschikbare streams. Op de mobiele toestellen is een applicatie aanwezig die de beelden kan analyseren die worden opgenomen. De resultaten van deze analyses worden dan in de databank geüpdatet. Een Main Server zoekt telkens in deze databank naar nieuwe potentieel interessante streams. Indien hij er zo één vindt stuurt hij een commando naar het mobiele toestel om te beginnen streamen naar een bepaalde Streaming Media Server. Deze servers bufferen de streams en voeren analyses uit waarvan zij ook de resultaten in de databank updaten. Op deze manier is het mogelijk om een intelligente selectie te maken uit de live videostromen. Om een groot aantal streams te kunnen verwerken is het mogelijk het raamwerk te laten schalen in een cloud omgeving.

Met de voorgestelde architectuur en implementatie is aangetoond dat het re-

altime verwerken en selecteren van een groot aantal videostromen mogelijk is. Voor het gegeven budget van acht Streaming Media Servers zijn we in staat om minstens 200 gelijktijdige streams te verwerken uit het totale aanbod.

De kracht van het raamwerk zit hem in de intelligente selectie. Hoe meer metadata beschikbaar is en hoe meer je weet welke waarden die metadata mag hebben, hoe beter je al kan filteren op het mobiel toestel zelf en dus hoe minder je het netwerk gaat belasten. Het is dan ook aangeraden om bij gebruik van het raamwerk grondig onderzoek te voeren naar welke filters werkelijk kunnen bijdragen in een selectie die het doel van de streams verwezenlijkt. Hierdoor zullen de mobiele toestellen minder energie verbruiken, het netwerk zal minder belast worden en het aantal benodigde servers zal sterk gereduceerd worden.

6.1 Toekomstig werk

Een mooie uitbreiding is het ontwikkelen van een mobiele applicatie die functioneert als client voor het raamwerk. Zo kan er bepaald worden hoe zwaar het berekenen van de kenmerken opweegt tegenover het streamen van de beelden en bijgevolg hoeveel energie een bepaalde verzameling filters verbruikt.

Een aanvulling die mogelijks zeer goede resultaten kan opleveren is het doorsturen van beelden met een variërende bitrate. Bijvoorbeeld door de resolutie te verkleinen of de framerate te verlagen. Zoals in deel 4.2.4 aangehaald is beïnvloed deze sterk het aantal streams dat een Streaming Media Server kan verwerken. Door de implementatie licht uit te breiden is het mogelijk om afhankelijk van de score een optimale bitrate te berekenen.

Bibliografie

[1] Kevin Weil. Introducing Periscope. `https://blog.twitter.com/2015/introducing-periscope`, 2015. [Online; geraadpleegd op 13-mei-2015].

[2] Advanced encoding settings: recommended bitrates, codecs, resolutions and more. `https://support.google.com/youtube/answer/1722171`, 2015. [Online; geraadpleegd op 3-april-2015].

[3] Jeremiah Golston. Comparing media codecs for video content. In *Embedded Systems Conference, San Francisco*, 2004.

[4] Tom Bert. Getting ready for high frame rates in digital cinema. Technical report, Barco, 2012.

[5] Nick Feamster and Hari Balakrishnan. Packet loss recovery for streaming video. In *12th International Packet Video Workshop*, pages 9–16. PA: Pittsburgh, 2002.

[6] Miriam Redi, Neil O'Hare, Rossano Schifanella, Michele Trevisiol, and Alejandro Jaimes. 6 seconds of sound and vision: Creativity in micro-videos. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*. IEEE Computer Society, 2014.

[7] Miriam Redi and Bernard Merialdo. Saliency moments for image categorization. In *Proceedings of the 1st ACM International Conference on Multimedia Retrieval*, page 39. ACM, 2011.

[8] Ritendra Datta, Dhiraj Joshi, Jia Li, and James Z Wang. Studying aesthetics in photographic images using a computational approach. In *Computer Vision–ECCV 2006*, pages 288–301. Springer, 2006.

[9] Miriam Redi and Bernard Merialdo. Where is the beauty?: retrieving appealing videoscenes by learning flickr-based graded judgments. In *Proceedings of the 20th ACM international conference on Multimedia*, pages 1363–1364. ACM, 2012.

[10] Puneet Jain, Justin Manweiler, Arup Acharya, and Kirk Beaty. Focus: clustering crowdsourced videos by line-of-sight. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, page 8. ACM, 2013.

[11] Mohammad Soleymani and Martha Larson. Crowdsourcing for affective annotation of video: Development of a viewer-reported boredom corpus. In *Proceedings of the ACM SIGIR 2010 workshop on crowdsourcing for search evaluation (CSE 2010)*, pages 4–8, 2010.

[12] Ali El Essaili, Eckehard Steinbach, Daniele Munaretto, Srisakul Thakolsri, and Wolfgang Kellerer. Qoe-driven resource optimization for user generated video content in next generation mobile networks. In *Image Processing (ICIP), 2011 18th IEEE International Conference on*, pages 913–916. IEEE, 2011.

[13] Brian Thorne and Raphaël Grasset. Python for prototyping computer vision applications. University of Canterbury. Human Interface Technology Laboratory, 2010.

[14] Nao Iizuka. RTSP, RTMP, and HTTP server implementation in Node.js. `https://github.com/iizukanao/node-rtsp-rtmp-server`, 2014. [Online; geraadpleegd op 27-maart-2015].

[15] Peter Braden. OpenCV bindings for Node.js. `https://github.com/peterbraden/node-opencv`, 2013. [Online; geraadpleegd op 29-mei-2015].

[16] P. Read and M.P. Meyer. *Restoration of Motion Picture Film*. Butterworth-Heinemann series in conservation and museology. Elsevier Science, 2000.

[17] Prashant Panigrahi. LTE vs HSPA+: Where is the future? `http://www.3glteinfo.com/lte-vs-hspa-where-is-the-future`, 2012. [Online; geraadpleegd op 5-april-2015].

[18] What is the actual real-life speed of wireless networks? `http://www.speedguide.net/faq/what-is-the-actual-real-life-speed-of-wireless-374`, 2014. [Online; geraadpleegd op 5-april-2015].

[19] Jana Machajdik and Allan Hanbury. Affective image classification using features inspired by psychology and art theory. In *Proceedings of the international conference on Multimedia*, pages 83–92. ACM, 2010.

[20] Cyril Laurier, Olivier Lartillot, Tuomas Eerola, and Petri Toiviainen. Exploring relationships between audio features and emotion in music. 2009.

[21] Mark Desnoyer and David Wettergreen. Aesthetic image classification for autonomous agents. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 3452–3455. IEEE, 2010.

[22] Harry Zhang and Jiang Su. Naive bayesian classifiers for ranking. In *Machine Learning: ECML 2004*, pages 501–512. Springer, 2004.

[23] Jan Cnops. Inleiding tot kunstmatige intelligentie, 2015.

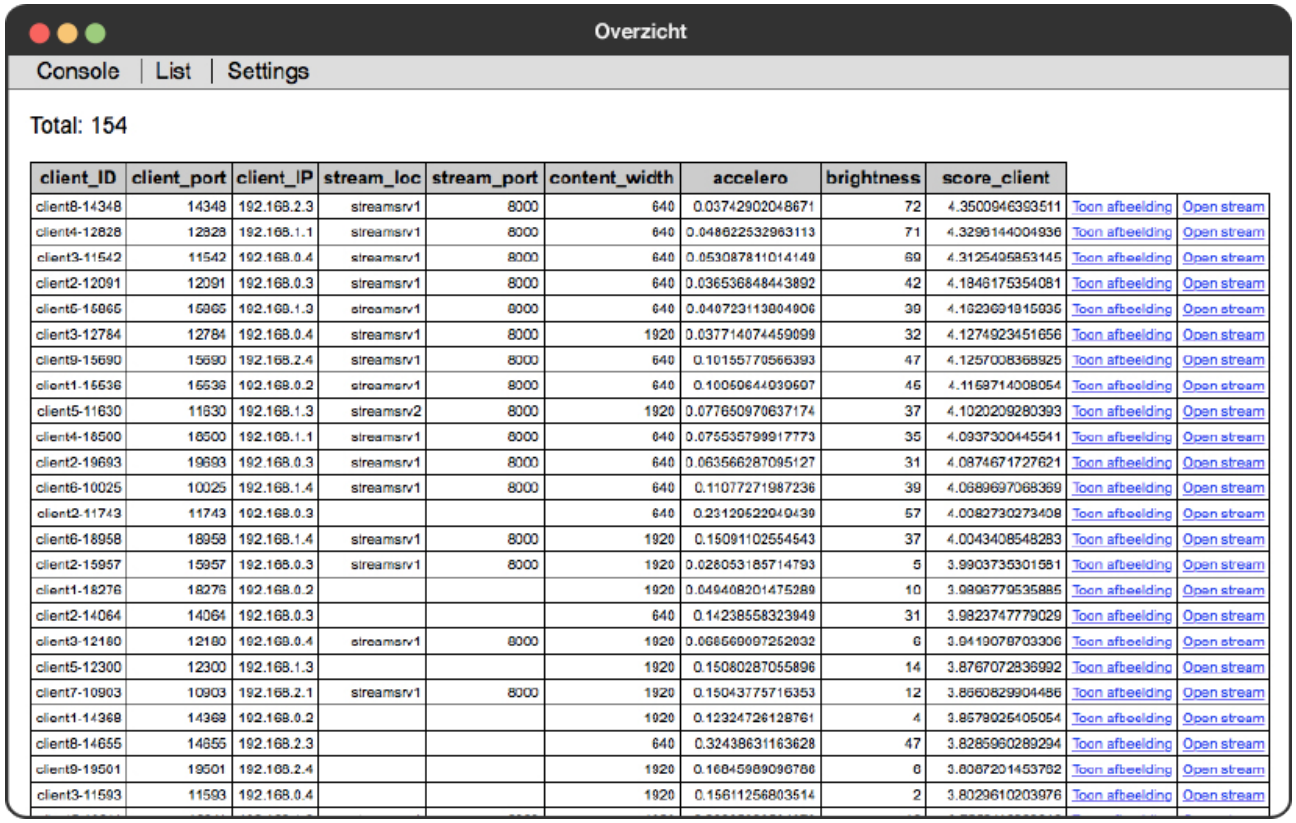
[24] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):679–698, 1986.

[25] Peak Signal-to-Noise Ratio as an Image Quality Metric. `http://www.ni.com/white-paper/13306/en`, 2013. [Online; geraadpleegd op 8-juni-2015].

Bijlage A

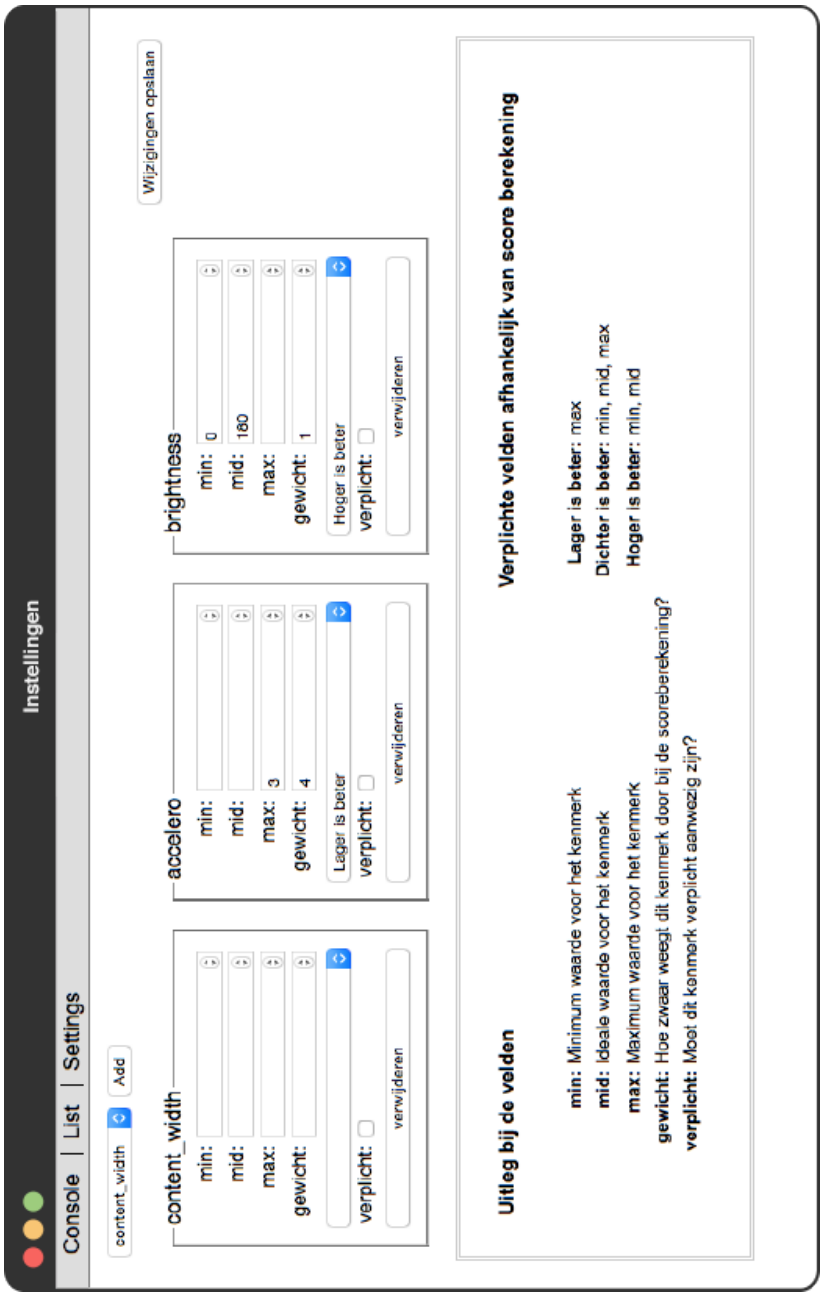
Screenshots van de webinterface

Figuur A.1: Pagina met de top-9 streams. Elke seconde worden de frames van de top-9 clients gedownload en weergegeven op deze pagina. Door op een afbeelding te klikken is het mogelijk om de stream te openen in een mediaspeler.



client_ID	client_port	client_IP	stream_loc	stream_port	content_width	accelero	brightness	score_client		
client8-14348	14348	192.168.2.3	streamsrv1	8000	640	0.03742902048671	72	4.3500946393511	Toon afbeelding	Open stream
client4-12828	12828	192.168.1.1	streamsrv1	8000	640	0.048622532963113	71	4.3296144004936	Toon afbeelding	Open stream
client3-11542	11542	192.168.0.4	streamsrv1	8000	640	0.053087811014149	69	4.3125495953145	Toon afbeelding	Open stream
client2-12091	12091	192.168.0.3	streamsrv1	8000	640	0.036536848443892	42	4.1846175354081	Toon afbeelding	Open stream
client5-15865	15865	192.168.1.3	streamsrv1	8000	640	0.040723113804906	39	4.1623691815936	Toon afbeelding	Open stream
client3-12784	12784	192.168.0.4	streamsrv1	8000	1920	0.037714074459099	32	4.1274923451656	Toon afbeelding	Open stream
client9-15690	15690	192.168.2.4	streamsrv1	8000	640	0.10155770566393	47	4.1257008368925	Toon afbeelding	Open stream
client1-15536	15536	192.168.0.2	streamsrv1	8000	640	0.10050644030607	46	4.1158714008054	Toon afbeelding	Open stream
client5-11630	11630	192.168.1.3	streamsrv2	8000	1920	0.07765097063717 4	37	4.1020209280393	Toon afbeelding	Open stream
client4-16500	16500	192.168.1.1	streamsrv1	8000	640	0.075535799917773	35	4.0937300445541	Toon afbeelding	Open stream
client2-19693	19693	192.168.0.3	streamsrv1	8000	640	0.063566287095127	31	4.0874671727621	Toon afbeelding	Open stream
client6-10025	10025	192.168.1.4	streamsrv1	8000	640	0.11077271987236	39	4.0689697068369	Toon afbeelding	Open stream
client2-11743	11743	192.168.0.3			640	0.23120522049439	57	4.0082730273408	Toon afbeelding	Open stream
client6-18958	18958	192.168.1.4	streamsrv1	8000	1920	0.15091102554543	37	4.0043408548283	Toon afbeelding	Open stream
client2-15957	15957	192.168.0.3	streamsrv1	8000	1920	0.028053185714793	5	3.9903735301581	Toon afbeelding	Open stream
client1-18276	18276	192.168.0.2			1920	0.049408201475289	10	3.9896779535885	Toon afbeelding	Open stream
client2-14064	14064	192.168.0.3			640	0.14238558323949	31	3.9823747779029	Toon afbeelding	Open stream
client3-12180	12180	192.168.0.4	streamsrv1	8000	1920	0.068569097262032	6	3.9419078703306	Toon afbeelding	Open stream
client5-12300	12300	192.168.1.3			1920	0.15080287055896	14	3.8767072936992	Toon afbeelding	Open stream
client7-10903	10903	192.168.2.1	streamsrv1	8000	1920	0.15043775716353	12	3.8660829904486	Toon afbeelding	Open stream
client1-14369	14369	192.168.0.2			1920	0.12324726128761	4	3.8579025405054	Toon afbeelding	Open stream
client8-14655	14655	192.168.2.3			640	0.32438631163628	47	3.8285960289294	Toon afbeelding	Open stream
client9-19501	19501	192.168.2.4			1920	0.16845989096786	6	3.8087201453762	Toon afbeelding	Open stream
client3-11593	11593	192.168.0.4			1920	0.15611256803514	2	3.8029610203976	Toon afbeelding	Open stream

Figuur A.2: Overzichtspagina van de web interface. Elke rij is een client die al dan niet aan het streamen is. Aan de rechterkant is het mogelijk om het laatste frame uit de stream als afbeelding weer te geven of om de stream te openen in een mediaspeler.



Figuur A.3: Pagina om de instellingen van de Main Server aan te passen. Zie deel 4.4.2 voor uitleg bij de verschillende velden.