Final Work Paper

JONATHAN DE WIT

Promotor: Rube Dejonckheere

Index

1 Project Beschrijving

Mijn final work is een Technical Excellence final work en heeft als doel om een videobewakingssysteem te maken aan de hand van microcontrollers.

Wat is een microcontroller?

Een microcontroller is een kleine, geïntegreerde computer die vaak wordt gebruikt om verschillende kleine elektronische apparaten besturen. Dit kan gaan van een sensor tot een motor.

De installatie en het beheren van het bewakingssysteem zal volledig plaatsvinden op een Android app. Het systeem is volledig modulair opgebouwd voor de gebruiker de mogelijkheid te geven om op elk moment een apparaat toe te voegen of te verwijden uit zijn systeem.

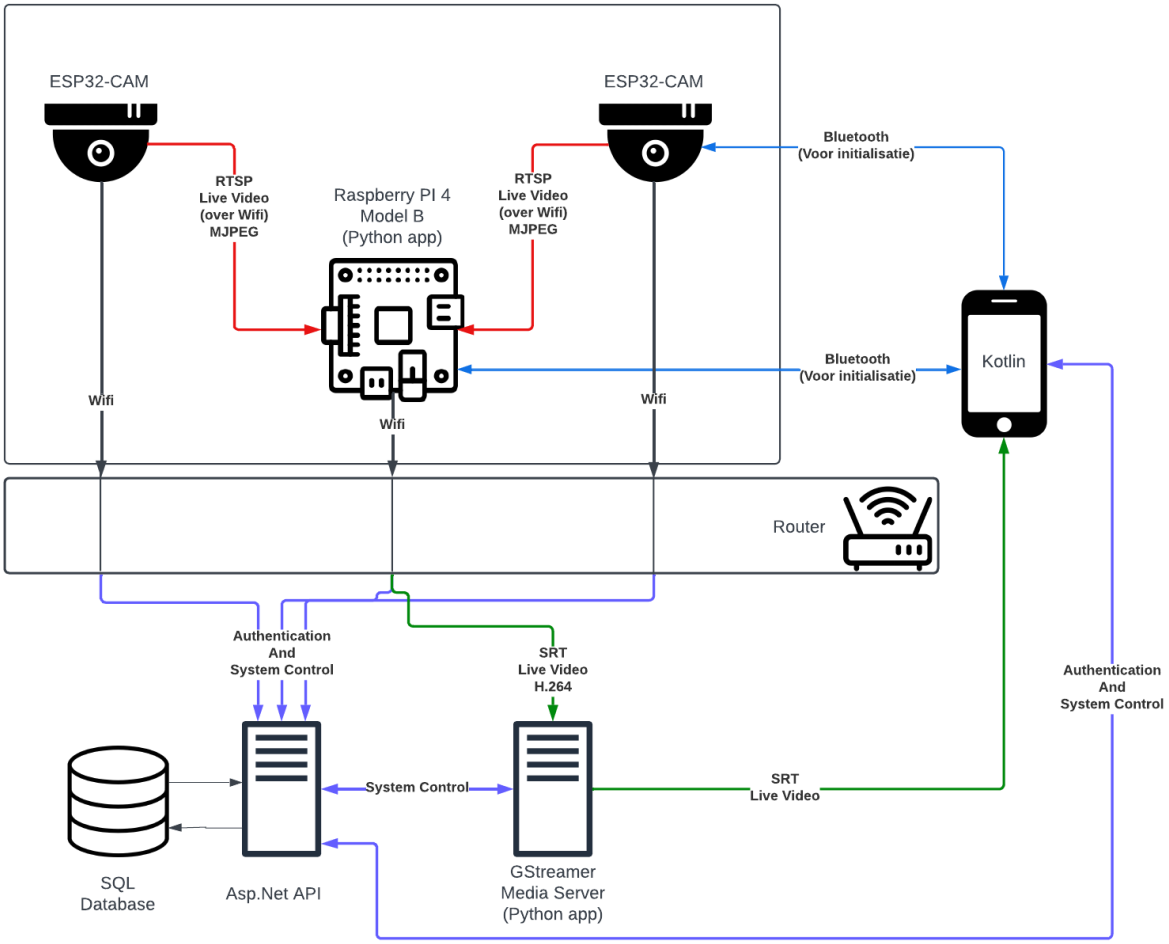
Om een nieuw apparaat toe te voegen kan de gebruiker zich aan het apparaat verbinden met zijn smartphone aan de hand van Bluetooth. Zodra deze verbinding tot stand is gebracht, moet de gebruiker het apparaat configureren. Dit houdt voornamelijk in om het apparaat te verbinden met de wifi en zijn product code in te voeren voor authenticatie. Zodra de configuratie is voltooid, wordt het apparaat automatisch toegevoegd aan het account van de gebruiker en kan hij zijn nieuw apparaat op afstand beheren.

Het is belangrijk om te vermelden dat het systeem werkt met een centrale hub die de videostreams van de verschillende camera’s zal ontvangen voor deze te verzenden naar de server.

Na het toevoegen van al zijn apparaten zal de gebruiker de mogelijkheid hebben om live naar de beelden van zijn beveiligingscamera’s te kijken.

Als uitbreiding zou ik een geautomatiseerd systeem kunnen toevoegen om het beveiligingssysteem te kunnen aan en uit te schakelen op basis van de locatie van de gebruiker. Dit zou werken op basis van Bluetooth zonder gebruik te maken van GPS.

1.1 Schets van het project



2 Gemaakte keuzes en tegengekomen problemen

2.1 Microcontroller: ESP32-CAM

Voor dit project had ik een microcontroller devloper board nodig met een WiFi/Bluetooth module en de mogelijkheid gaf om een camera aan te sluiten ofwel door middel van I/O-pinnen of een ander socket. Het was ook belangrijk dat het bord een grote community heeft omdat dit voor een groter aantal documentatie zorgt.

Wat is een microcontroller devloper bord?

Een microcontroller devloper bord is een printplaat die speciaal is gemaakt om de ontwikkeling van microcontroller-gebaseerde projecten te vergemakkelijken. Het heeft uiteraard een microcontroller, Maar het heeft ook input/output (I/O) pins om verschillende apparaten aan het bord aan te sluiten bv. sensoren, knoppen of kleine motors. Wat ook belangrijk is dat het een programmeerinterface heeft waarmee de programmeur eenvoudig zijn code naar de kaart kan uploaden

Wat zijn I/O-pinnen?

I/O-pinnen (Input/Output pins) zijn elektrische connecties van de microcontroller die hem toelaten om te communiceren met externe apparaten. Een I/O-pin kan geconfigureerd worden als input om data te verkrijgen of als output om data te verzenden.

Tijdens mijn research fase is het vlug duidelijk geworden dat het Arduino Framework het meest gedocumenteerd was. Zeker voor mensen (zoals ik) die nieuw zijn in het gebruiken van microcontrollers. Toen ik echter op zoek was naar specifieke documentatie voor het gebruik van camera's en live-video transmissie aan de hand van een microcontroller, vond ik de meeste resultaten voor de ESP32-cam microcontroller developer bord.

De ESP32-cam is gebaseerd op de ESP32 microcontroller, en wat dit specifiek board interessant maakt is dat hij een SCCB (Serial Camera Control Bus) connector heeft die gebruikt kan worden voor een OV2640 camera rechtstreeks te koppelen aan het board. Dit maakt het koppelen van een camera betrouwbaarder en minder complex dan het gebruik van I/O pins.

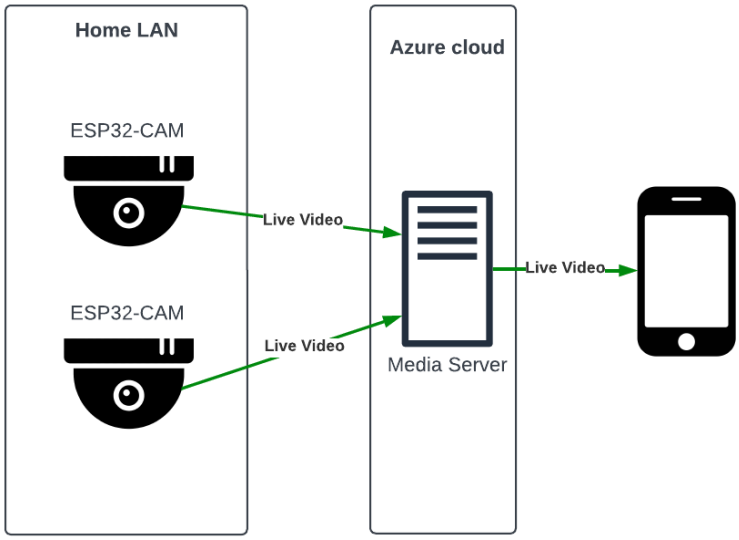
De ESP32 microcontroller is compatible met het Arduino Framework en heeft meer computerkracht en geheugen dan de meeste Arduino borden.

Wat wel vermeld moet worden is dat de ESP32-cam geen ingebakken programmeerinterface heeft waarmee ik mijn code direct kan uploaden op het bord. Hiervoor zal ik een TTL serial conversion module nodig hebben. Deze module kan gemakkelijk aan de I/O pins gekoppeld worden voor de code te kunnen uploaden.

2.2 Doorsturen van live video

Het verzenden van een live videostream vanuit mijn microcontroller naar mijn mobiele applicatie is een deel van het project die ik heb onderschat en waar ik de meeste problemen ben tegengekomen tijdens het project.

Wat ik origineel wou doen was om de livestream van de microcontrollers te versturen naar een server die op zijn beurt de stream moest versturen naar de smartphone app.



Ik ging ervan uit dat een video slechts een opeenvolging van afbeeldingen was en dat het voldoende was om de afbeeldingen één voor één te verzenden om hier een video van te krijgen.

Hierdoor kwam ik op het idee om een API te gebruiken waarbij ik de afbeeldingen één voor één ging versturen om ze ook één voor één te laten weergeven op de smartphone app.

2.2.1 Gebruiken van een API voor het live verzenden van video

De complexiteit van het live-streamen van video heb ik dik onderschat. Zelfs al is het waar dat een video slechts een opeenvolging van afbeelding is, zijn er veel meer factoren die erbij komen kijken als men live-video via het internet wil verzenden.

Een paar van deze factoren zijn de volgende:

- Real-time encoding en compression: Wanneer een camera de afbeeldingen maakt voor de live-video, mogen de afbeeldingen best niet rauw verzonden worden over het netwerk. Dit doet men best niet omdat de rouwe afbeeldingen veel te groot zijn om verzonden te worden over een netwerk die een beperkte bandbreedte heeft. Daarom moeten we een video codec gebruiken die de video in real-time kan coderen (encoding) en comprimeren (compression). Dit proces zal zorgen dat de video lichter is om te verzenden.

- Lage latency: Bij het live streamen van video is het de bedoeling dat de tijd tussen het verzenden van de videobeelden en het verkrijgen van de videobeelden zo laag mogelijk is. Als er voor security camera's bv. een latency zou zijn van een paar minuten zal men dan ook pas na een paar minuten kunnen zien wat er aan het gebeuren is. Een latency van een paar minuten is niet aanvaardbaar omdat men met live video de huidige toestand van zaken willen zien. Zelf al bestaat een latency van nul niet zal men toch proberen om deze zo laag mogelijk te houden.

- Synchronisatie: Het is ook belangrijk dat de video gesynchroniseerd blijft. Het is belangrijk dat de klank en het beeld van een video op elkaar afgestemd zijn of dat beelden die te laat zijn niet meer gebruikt moeten worden.

- Error correctie: Als een pakket van de video stream onderweg verloren gaat of als de inhoud van het pakket beschadigd raakt, is het best dat er een mechanismen is om de fout te herstellen. Error correctie kan bijvoorbeeld uitgevoerd worden aan de hand van Forward Error Correction (FEC) of Automatic Repeat Request (ARQ)

Deze factoren waar rekening mee gehouden moet worden maakt dat het gebruik van een API voor live-video te streamen niet optimaal is. Dit komt omdat een API hier simpel gezegd niet voor gemaakt is en dat ze dus de vereisten voor het live streamen van video niet implementeert. Dit zal resulteren in live-video met een extreem hoge latency met op zijn best een paar frames per minuut.

Aan het begin van het project was ik er niet bewust dat het gebruik van een API voor live video streamen geen haalbare oplossing was. Maar tijdens de development ben ik er snel van bewust gekomen toen ik zag dat het veel tijd kostte om mijn afbeeldingen vanuit mijn ESP32-CAM naar mijn API te sturen.

2.2.2 Gebruiken van socket connections

Nadat ik tot de conclusie was gekomen dat het niet haalbaar was om via een API live video te versturen, deelde ik mijn conclusies met meneer Dejonckheere. Hij had me aangeraden om te gaan kijken naar de kant van socket connections.

Wat is een socket connection?

Een socket connection is een manier waarop twee computerprogramma's via een netwerk met elkaar kunnen praten. Eén programma maakt een "socket" en bindt deze aan een specifiek poortnummer op de hostcomputer. Een ander programma kan dan verbinding maken met die socket door het IP-adres van de hostmachine en hetzelfde poortnummer op te geven. Dit creëert een tweerichtingscommunicatieverbinding tussen de programma's, waardoor ze gegevens heen en weer kunnen sturen.

Ik ben dan eerst gaan kijken welke protocollen het meest gedocumenteerd waren om een live-video te verzenden vanuit mijn ESP32-cam aan de hand van een socket connection.

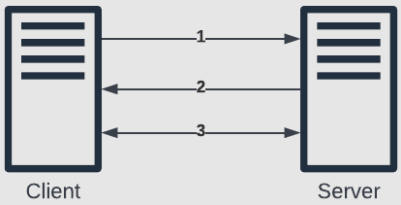
Tijdens mijn onderzoeksfase merkte ik op dat de documentatie die ik vond over het verzenden van live video aan de hand van een ESP32-cam board systematisch gebruik maakte van WebSocket.

Wat is een WebSocket?

WebSocket is een protocol die zal zorgen voor een full-duplex communicatiekanaal tussen een client en een server. Deze is geschikt voor het real-time verwisselen van kleine hoeveelheden data dat zoals chattoepassingen of het live streamen van gegevens.

Het tot stand brengen van een WebSocket-verbinding omvat de volgende stappen:

1. *De client verstuurt een http request met de vraag om te upgraden naar een WebSocket connectie.*
2. *De Server aanvaart de request en stuurt een HTTP successful antwoord terug dit aan de client te laten weten.*
3. *De client en de server brengen een websocketverbinding tot stand aan de hand van het WebSocket protocol.*



Wat is een full-duplex communicatiekanaal?

Een full-duplex communicatiekanaal zal ervoor zorgen dat men gelijktijdig in de 2 richtingen kunnen communiceren tussen de 2 partijen. Als men bijvoorbeeld een full-duplex kanaal hebben tussen een client en een server zal men gelijktijdig gegevens kunnen verzen vanuit de client en de server.

2.2.3 Waarom geen WebSocket

Het gebruik van WebSocket voor dit project is door 2 redenen niet wenselijk:

1. WebSocket is niet speciaal gemaakt voor live video te versturen en is hier dus ook niet voor geoptimaliseerd. Hierdoor kan men van WebSocket verwachten dat hij bijvoorbeeld een hogere latency, lagere reliability en hogere bandbreedte gebruik tegenover protocollen die speciaal gemaakt zijn voor live-video streaming.

2. In elke documentatie die ik vond werkte het ESP32-cam board als een WebSocket server, dit wil zegen dat mijn server als de client moet werken. Het probleem is dat mijn server op de cloud zit met een public IP-adres, en dus niet rechtstreeks met mijn ESP32-cam board kan spreken die op een lokaal netwerk zit met een private IP-adres. Hierdoor kan mijn server het initiële rekwest voor het aanmaken van een WebSocket connectie niet versturen.

Door deze redenen kon ik WebSocket niet gebruiken om mijn videostream te verzenden.

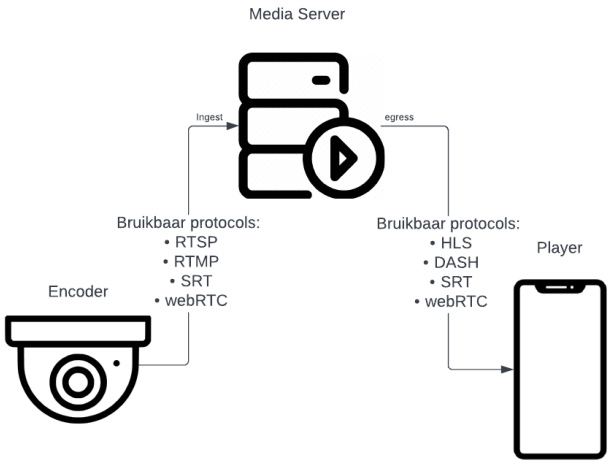
2.2.4 Gebruiken van videostreaming protocollen

Toen ik eenmaal begreep dat WebSocket ook niet zou werken ben ik specifiek gaan kijken naar de verschillende protocollen die gespecialiseerd zouden zijn in het live streamen van video.

Video streaming protocollen werken met behulp van socket connections en zijn speciaal gemaakt voor het live verzenden van video en houd dus rekening met alle behoeftes die hiervoor nodig is.

Door meer te leren over de verschillende specifieke protocollen, leerde ik dat er twee grote groepen protocollen waren voor het live verzenden van video. Een eerste groep protocollen die gebruikt wordt tussen de het apparaat die de video produceert (encoder) en de mediaserver. En een 2de groep protocollen voor de communicatie tussen de mediaserver en de het apparaat die de video wil consumeren (player).

In de eerste groep van protocollen kan men protocollen terugvinden zoals: RTSP en RTMP. En in de 2de groep vind men protocollen zoals: HLS en DASH. Er bestaan ook protocollen die gebruikt kunnen worden voor beide use-cases zoals: SRT en webRTC.



2.2.5 Research naar het beste video-streaming protocol

Ik ben eerst begonnen met het protocol te vinden die ik zou willen gebruiken voor de communicatie tussen de encoder en de mediaserver. Wat belangrijk was bij de keuze van het protocol, was dat de initiële aanvraag voor de socketverbinding tussen de encoder en de mediaserver door de encoder kon uitgevoerd worden.

De meest gebruikte protocollen voor deze use-case zijn de RTMP (Real-Time Messaging Protocol) en SRT (Secure Reliable Transport).

2.2.5.1 RTMP (Real-Time Messaging Protocol)

Het RTMP-protocol werd gepubliceerd in 1990 en is eigendom van Adobe. Dit protocol is één van de meest gebruikte protocollen voor live-video te streamen vanuit een lokaal netwerk met een private IP-adres naar een mediaserver. RTMP wordt bijvoorbeeld gebruikt door Youtube en Twitch.

RTMP ondersteunt een groot aantal codecsen voor compressie, heeft een lage latency en word door de meeste systemen ondersteund.

Het belangrijkste nadelen van RTMP is dat hij geen ondersteuning biedt voor build-in encription en hij heeft geen error correctie.

2.2.5.2 SRT (Secure Reliable Transport)

SRT is een videostreaming protocol gemaakt door Havision in 2013 en is nu open source sinds 2017.

SRT biedt dezelfde voordelen dan RTMP: ondersteuning voor een groot aantal codecsen voor compressie en een lage latency dan RTMP. Daarnaast biedt SRT ondersteuning voor build-in AES 128/256 bit encription en error correction.

Het nadeel van SRT is dat hij nog redelijk nieuw is en hierdoor minder ondersteuning heeft dan RTMP.

2.2.6 Gekozen video-streaming protocollen

Nadat ik meer had geleerd over de verschillende videostreaming protocollen, kwam ik tot de conclusie dat SRT ideaal zou zijn voor dit project. Dit komt voornamelijk omdat hij open source is en build-in ecriptie ondersteund voor veilig de live-video te kunnen uitzenden.

Het grootste probleem van SRT is dat hij minder verspreid is tegenover RTMP waardoor er voor dit protocol minder documentatie en ondersteuning is.

Ik ben toen gaan kijken hoe ik het SRT-protocol kon gaan gebruiken op mijn microcontroller. Na lang zoeken heb ik geen library, tutorial of andere vormen van documentatie teruggevonden die me zou kunnen helpen om het SRT-protocol te implementeren op mijn ESP32-cam bord.

Hierdoor ben ik gaan kijken als er misschien wel iets van documentatie zou zijn voor het RTMP-protocol. Helaas kwam ik hier op hetzelfde resultaat tegen als voor het SRT-protocol. Ik kon geen library, tutorial of andere vormen van Documentatie terugvinden.

Doordat ik het SRT of RTMP-protocol niet kon implementeren, wist ik niet hoe ik ging zorgen dat mijn ESP32-cam bord live zijn beelden ging doorsturen naar mijn mediaserver.

Ik heb me dan herinnerd dat ik tijdens mijn onderzoek een implementatie van het RTSP-protocol voor het Arduino-framework was tegengekomen, en dacht dat het toch mogelijk was om het te gebruiken op voorwaarde dat ik de structuur van het systeem licht wijzigt.

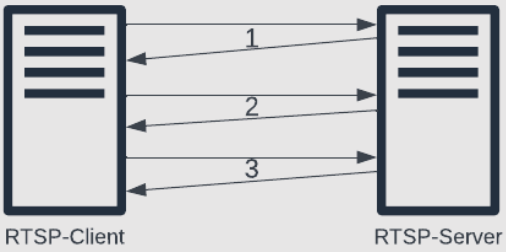
Wat is het RTSP-protocol?

RTSP (Real-Time Streaming Protocol) is een wijdverspreid open standaard protocol gemaakt voor audio en video te verzenden met een lage latency over een IP-netwerk. Dit protocol kan zowel live video en audio streamen alsook statische vooraf opgenomen content.

RTSP is een wijdverspreid protocol en ondersteund een groot aantal codecen. Het grootste nadeel aan RTSP is zijn gebrek aan securityfeatures zoals encryption.

Een RTSP-connectie wordt als volgt aangemaakt:

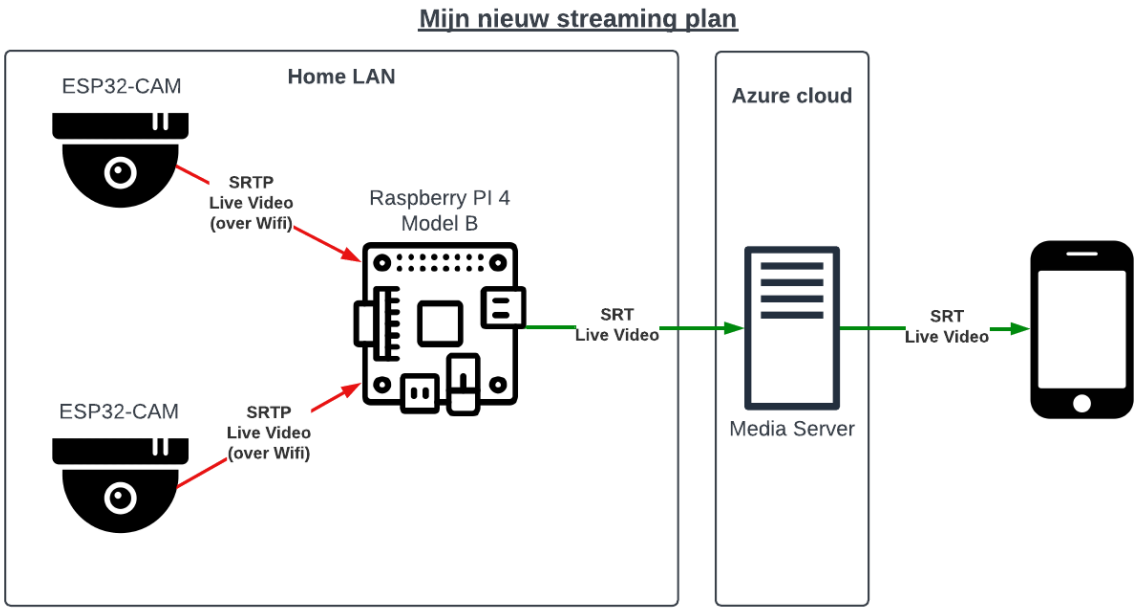
1. *De RTSP-client begint met het verzenden van een Describe-pakket hier zal de RTSP-server op antwoorden met een SDP (Session Description Protocol). De SDP zal informatie bevat zoals: het media type, codecs, en network details.*
2. *De RTSP-client verzendt een Setup-pakket voor een socket connectie aan te maken. Als deze ook aanvaard wordt door de RTSP-sever word deze opgezet.*
3. *De RTSP-client verzendt een Play-pakket voor aan te tonen aan de RTSP-server dat hij klaar om data te ontvangen. De RTSP-server begint de content dan te versturen naar de client.*



Net zoals WebSocket wordt de connectie tussen de server en de client gemaakt aan de hand van een initieel pakt die moet komen van de partij die de content opvraagt (client). Zoals eerder vermeld is dit niet mogelijk omdat mijn microcontroler slechts een private IP-adres heeft en hierdoor van buiten het netwerk niet direct aanspreekbaar is.

Het is wel mogelijk om de live-video stream op te vragen binnen het lokaal netwerk van de microcontroler. Dit bracht me tot het idee om de live-video stream op te vangen op een ander device op het lokaal netwerk die de RTSP live-video stream zou kunnen converten naar een SRT live video stream.

De systeemarchitectuur zal er dus nu als volgt uitzien:



2.2.7 Converter van het RTSP-videostream naar SRT-videostream

Voor de conversie te maken van het RTSP-protocol naar het SRT-protocol heb ik een Raspbarry PI 4 Model B (4Gb ram) te gebruiken. Deze Raspberry PI is een open source single-board computer waarop men een volledig OS kunnen laten draaien.

Ik heb gekozen om Raspbian OS te gebruiken. Dit heb ik gedaan omdat dit OS geoptimaliseerd is om te werken met Raspberry PI's en dus zorgt voor een grootere stabiliteit.

Raspbian OS is gebaseerd op Debian. Tijdens dit project heb ik gewerkt met Raspian OS onder Debian bullseye die tijdens het project de laatste Debian versie was.

Voor de conversie te maken van RTSP naar SRT heb ik de open source tool FFMpeg gebruikt. Ik heb een custom installatie moeten maken van deze tool omdat hij default niet kan werken met het SRT-protocol. Ik verwijs u naar het document "FFmpegCustomInstallGuid.pdf" waar ik deze installatie in detail heb beschreven als u hier meer over wilt weten.

Vanwege compatibiliteitsproblemen tussen het videoformaat van de microcontroller en de mediaserver/media player in Android, moest ik ook het compressieformaat en het pixelformaat van de videostream wijzigen.

De microcontroller gebruikt het MJPEG-formaat als compressieformaat. Deze is bij het gebruik van het SRT-protocol niet compatible met de mediaserver.

Hierdoor verander ik het compressieformaat van MJPEG naar H.264 die een veel grotere ondersteuning heeft. H.262 zal ook een efficiëntere compressie toepassen dan MJPEG waardoor de live video minder bandbreedte gaat gebruiken.

Hiernaast heb ik ook het pixelformaat moeten veranderen. Dit heb ik moeten doen omdat het origineel formaat voor problemen zorgde bij het weergeven van de video op een fysiek Android device (in een Android VM werkte het wel prima zonder pixel formaat verandering). Ik heb het pixelformaat veranderd naar YUV420p.

Dit is het commando dat ik heb gebruikt voor de conversie:

fmpeg -i rtsp://192.168.1.28:8554/mjpeg/1 -pix_fmt yuv420p -c:v libx264 -r 30 -b:v 1000k -f mpegts srt://192.168.1.32:40005

Voor meer detail over het commando verwijs ik opnieuw door naar het "FFmpegCustomInstallGuid" document.

2.2.8 Gekozen mediaserver

Omdat het SRT-protocol relatief nieuw is, is het nog niet weid verspreid. Hierdoor was het niet gemakelijk om een media server te vinden die het SRT-protocol ondersteunt. De meeste mediaservers die ik gevonden heb waren betalend en waren niet open source.

Ik heb uiteindelijk een mediaserver gevonden met de naam 'Nimble streamer die een gratis versie van het softwarepakket heeft die standaard het SRT-protocol ondersteunt. De versie van Nimble streamer die gratis is voldoet aan alle basisbehoeftes die ik nodig heb voor het project maar heeft één groot nadeel. Nimble streamer is niet open source.

Dit is niet ideaal omdat ik persoonlijk wenste dat alles zo veel mogelijk open source was, maar omdat de deadline van het project steeds meer naderde heb ik besloten om hier toch verder mee te gaan voor om hier geen tijd meer te verliezen.

Vervolgens heb ik Nimble streamer gebruikt voor de demo te maken voor de tusentijdse evaluatie. deze demo omvatte het verzenden van een videostream vanuit mijn microcontroller naar mijn mobiele applicatie. Deze demo was nog hardcored en zou moeten geautomatiseerd voor de eindpresentatie.

Tijdens de tussentijdse evaluatie kreeg de opmerking dat Nimble streamer proprietary software is en dat het beter zou zijn om een open source oplossing te gebruiken.

2.2.8.1 GStreamer

Door de feedback van de tussentijdse evaluatie ben ik opnieuw op zoek gegaan naar een open source solution die ik als media server zou kunnen gebruiken. Dit bracht me tot GStreamer. GStreamer is een open-source multimedia-framework dat een op pijplijnen gebaseerde architectuur biedt voor het verwerken en streamen van audio- en video.

GStreamer is bijna ideaal om als mediaserver te gebruiken. Het enigste probleem is dat hij het SRT-protocol niet ondersteund. Dit heb ik kunnen oplossen door het SRT plugin te installeren van Justin Kim die het werken met SRT wel mogelijk maakt.

Dit zorgt wel dat het gebruik van GStreamer minder betrouwbaar is dan Nimble streamer aangezien ik nu afhankelijk ben van een plug-in die is geschreven door een derde partij. Maar aan de andere kant is mijn project nu voledig open source.

Hierdoor heb ik gekozen voor GStreamer als media server.

2.3Automatiseren van het video-streaming

Het is de bedoeling dat het videobewakingssysteem opgestart word wanneer de gebruiker deze op zijn app opstart. Voor dat de Microcontrollers, Raspberry Pi en de Media Server op te starten of te stoppen met het doorsturen van de live video stream heb ik gekozen om het observer patern te gebruiken.

De 3 apparaten die de video stream afhandelen zullen regelmatig aan de hand van de API checken als de status van het systeem is veranderd. Als de status veranderd zal elk divice de nodige procesen uitvoeren om de video stream op/af te zetten.

Voor meer info over de individuele code van elk divice verwijs ik naar de documentatie op hun GitHub repo:

Microcontroller code:

https://github.com/JonathanDeWit/FinalWorkESP32CamLiveCamera.git

Raspberry Pi code:

https://github.com/JonathanDeWit/FinalWorkRaspberryPiConvertor.git

Media Server:

https://github.com/JonathanDeWit/FinalWorkSrtServer.git

2.4API: ASP.Net

Ik koos voor ASP.Net voor het maken van mijn API, vooral omdat ik me wou verdiepen en meer vertrouwd wilde maken met het ASP.Net framework.

Ik ga mijn API en mijn mediaserver in Azure implementeren omdat ik als student recht heb op 100 euro credit voor hun verschillende clouddiensten. Azure maakt het ook gemakelijk om ASP.Net applicaties te deployen op hun cloud solution aan de hand van een built-in ondersteuning voor ASP.Net deploying in Visual Studio.

De hoofdtaken van mijn API zijn de controle van het beveiligingssysteem en de authenticatie voor de verschillende applicaties binnen mijn project. Ik koos voor ASP.Net-identiteit als het framework voor authenticatie, omdat het een goede integratie bied voor ASP.Net en omdat hij goed gedocumenteerd is.

Voor meer detail in verband met mijn ASP.Net API verwijs ik door naar mijn API documentatie:

https://github.com/JonathanDeWit/FinalWorkApi

3 Literatuurlijst

Sebastian, L. (2014, 10 augustus). TECKQUICKIE – VIDÉO COMPRESSION AS FAST AS POSSIBLE (Vidéo). Youtube. https://www.youtube.com/watch?v=qbGQBT2Vwvc

Sebastian, L. (2014, 3 april). TECKQUICKIE – WHAT IS A CODEC AS FAST AS POSSIBLE (Vidéo). Youtube. https://www.youtube.com/watch?v=GhWki9a7s18

M5Stack (S.D.). ESP32 Camera Module Development Board. M5Stack. https://m5stack.hackster.io/products/esp32-camera-module-development-board

Random Nerd Tutorials (2016, 28 december). Getting started with the ESP32 Development. Random Nerd Tutorials. Geraadpleegd op 27 oktober 2022 op https://randomnerdtutorials.com/getting-started-with-esp32/

Random Nerd Tutorials (2020, 6 februari). How to program 1/upload code to ESP32-CAM al-thinker (Arduino IDE). Random Nerd Tutorials. Geraadpleegld op 27 oktober 2022 op https://randomnerdtutorials.com/program-upload-code-esp32-cam/

Romero, A. (2018, 27 maart). What is video encoding? Codecs and compression techniques. https://blog.video.ibm.com/streaming-video-tips/what-is-video-encoding-codecs-compression-techniques/

Wowza Media Systems. (2022, 9 september). What is low latency and who needs it ? (update). https://www.wowza.com/blog/what-is-low-latency-and-who-needs-it

Ruether, T. (2023, 3 january). SRT: The secure reliable transport protocol explained (update). https://www.wowza.com/blog/srt-the-secure-reliable-transport-protocol-explained#history

Microsoft. (2023, 2 januari). *QuickStart: Deploy an ASP.net web app*. Microsoft. https://learn.microsoft.com/en-us/azure/app-service/quickstart-dotnetcore?tabs=net60&pivots=development-environment-vs

WebSocket. (2023, 13 januari). In *Wikipedia*. https://en.wikipedia.org/wiki/WebSocket

Real-Time Messaging Protocol. (2022, 3 november). In *Wikipedia*. https://en.wikipedia.org/wiki/Real-Time_Messaging_Protocol

IBM. (2021, 8 februari). *What is a TCP/IP Socket Connection?.* IBM. https://www.ibm.com/docs/en/zvse/6.2?topic=SSB27H_6.2.0/fa2ti_what_is_socket_connection.htm

PenzeyMoog, L. (2019, 24 mei). *Understanding socket connections in computer networking.* Medium. https://medium.com/swlh/understanding-socket-connections-in-computer-networking-bac304812b5c

Ruether, T. (2022, 27 oktober). Streaming Protocols: Everything you need to know (update). https://www.wowza.com/blog/streaming-protocols

Dasic, A. (2021, 8 januari). *RTMP vs. RTSP: Everything publishers should know*. Brid.TV.https://www.brid.tv/rtmp-vs-rtsp/#:~:text=Both%20RTMP%20and%20RTSP%20streaming%20protocols%20have%20their,other%20hand%2C%20RTSP%20is%20suitable%20for%20localized%20streams

Wowza (2021, 23 july). WOWZA – LIVE STREAMING PROTOCOL COMPARISON (Vidéo). Youtube. https://www.youtube.com/watch?v=jYYhwbF3tvM

Secure Reliable Transport. (2022, 10 februari).In *Wikipedia*. https://en.wikipedia.org/wiki/Secure_Reliable_Transport

Mela, M. (2006). Utilizing DSP for IP telephony applications in mobile terminals (Master's thesis, University of Helsinki) https://www.researchgate.net/publication/27516280_Utilizing_DSP_for_IP_Telephony_Applications_in_Mobile_Terminals

Ruether, T. (2022, 2 december). RTSP: The real-time streaming protocol explained (update).https://www.wowza.com/blog/rtsp-the-real-time-streaming-protocol-explained

UA Server RTSP Communication. (2021, 21 januari). In *W3C*. https://www.w3.org/2008/WebVideo/Fragments/wiki/UA_Server_RTSP_Communication#DESCRIBE

Nikols, L. (S. D.) SRT: Everything you need to know about the secure reliable transport protocol.https://www.haivision.com/blog/all/srt-everything-you-need-to-know-about-the-secure-reliable-transport-protocol/

GStreamer. (2023, 23 février). Nouvelles - GStreamer 1.22.1 version de correction de bogue stable. GStreamer. https://gstreamer.freedesktop.org/

GStreamer. (s. d.). Documentation SRT. GStreamer. https://gstreamer.freedesktop.org/documentation/srt/index.html?gi-language=c