RESEARCH REPORT

Wat is een geschikte manier om realtime video en audio te streamen in mijn webapplicatie?

Luka Spaninks

INHOUDSOPGAVE

1. INLEIDING

1.1 OMSCHRIJVING VAN DIT DOCUMENT

Dit document dient als onderzoeksverslag over wat een manier is om realtime video & audio te streamen in een webapplicatie.

1.2 LEESWIJZER

In dit document maak ik gebruik van de STARR-methode. Dit houdt in dat ik de volgorde: Situatie, Taak, Actie en Resultaat aanhoud. Ook maak ik gebruik van het DOT framework, dat betekent dat ik 3 strategieën gebruik om onderzoek te verrichten

2. SITUATIE

2.1 DE HUIDIGE SITUATIE

Ik ben dit semester gestart met het bouwen en ontwerpen van een webapplicatie genaamd: Meet a Stranger. Mijn applicatie heeft als voornaamste doel om willekeurige gebruikers met elkaar te verbinden via video- en textchat. Dit concept is heel vergelijkbaar met dat van Omegle. Omegle is namelijk ook een webapplicatie waarbij je kan video bellen met vreemden.

2.2 HET PROBLEEM VAN DE HUIDIGE SITUATIE

Het probleem is dat ik niet weet hoe ik realtime communicatie implementeer in mijn webapplicatie.

2.3 DE ONDERZOEKSVRAAG DIE HIER BIJ HOORT

Op basis van het beschreven probleem is mijn onderzoeksvraag: Wat is een geschikte manier om realtime video en audio te streamen in mijn webapplicatie?

Op basis van oriënterend onderzoek stel ik de hypothese dat WebRTC in combinatie met een javascript library een oplossing zou kunnen zijn.

3. HOOFD- EN DEELVRAGEN

In dit document zal ik mijn best doen om de volgende vragen te beantwoorden:

Wat is een geschikte manier om realtime video en audio te streamen in mijn webapplicatie?

1. Welke applicaties bestaan er met een realtime videofunctie?
2. Welke protocollen bestaan er voor het realtime streamen van video?
3. Wat zijn de voor- en nadelen van een aantal bestaande protocollen?
4. Wat is WebRTC?
5. Hoe verbind je twee of meer gebruikers met WebRTC?
6. Wat is een manier om WebRTC te implementeren in mijn WebApplicatie?

4. TAKEN

De volgende strategieën ga ik gebruiken om de hoofd- en deelvragen te beantwoorden:

Library: Available Product Analysis

Bij de library strategie wil ik gaan kijken naar wat al gemaakt is/bestaat. Dit kan heel handig zijn als je zelf iets wil ontwikkelen maar je weet nog niet hoe of wat een goede manier is. Ook hoef je vaak het wiel niet opnieuw uit te vinden en zijn er al bestaande technieken.

Field: Problem Analysis

Bij de problem analysis wil ik gaan uitzoeken waarom bepaalde protocollen/technieken wel of niet werken voor mijn use case. Ik wil graag dieper in gaan op de verschillende opties en weten of ze aan mijn criteria voldoen. De criteria voor een techniek/protocol zijn: nauwelijks of geen vertraging, gangbare video- en audiokwaliteit en het moet schaalbaar zijn.

Workshop: Prototyping

Bij de prototyping wil ik graag zelf een mvp bouwen. Ik ben benieuwd of ik een juiste oplossing voor mijn probleem kies en of dit ook aan de genoemde criteria voldoen. Ook wil ik graag weten of ik nog onverwachte problemen opmerk en of ik deze kan oplossen.

5. ACTIES

WELKE APPLICATIES BESTAAN ER MET EEN REALTIME VIDEOFUNCTIE?

Ik ben op het internet gaan kijken naar welke applicaties al bestaan? Hieronder noem ik een aantal producten die redelijk lijken op wat ik wil gaan maken.

Omegle:

Ik ben begonnen met een beetje meer onderzoek te doen naar het product waar ik mijn applicatie op baseer, namelijk: Omgele. Na een beetje gezocht te hebben op het internet kwam ik uit op deze GitHub pagina:

https://gist.github.com/nucular/e19264af8d7fc8a26ece

Op deze pagina staat een redelijk gedetailleerd onderzoek beschreven naar de source code van Omegle. In dit document komt ook WebRTC een aantal keer naar voren. Dit is interessant want in mijn oriënterend onderzoek heb ik dit ook een aantal keer voorbij zien komen.

Google Hangouts

Google Hangouts is een platform waar op gebruikers ook met elkaar kunnen videobellen. Het bestaat al sinds 2013 en gebruikt ook WebRTC voor de real time communicatie.

Discord

Discord is een communicatie app met een focus op gamers. Je kunt applicaties streamen, video- en audiobellen en tekst chatten. Ook Discord gebruikt WebRTC om het real time gedeelte van de applicatie te verzorgen.

https://blog.discord.com/how-discord-handles-two-and-half-million-concurrent-voice-users-using-webrtc-ce01c3187429

WELKE PROTOCOLLEN BESTAAN ER VOOR HET REALTIME STREAMEN VAN VIDEO?

RMTP

RMTP is een stream protocol (TCP) met een redelijk lage vertraging in video en audio. Vroeger was het alleen maar beschikbaar in combinatie met een flash player en een server. Nu is het ook voor het publiek beschikbaar. Het wordt nog steeds gebruikt maar is niet heel makkelijk te integreren met verschillende platforms zoals Android en IOS.

RTSP

RTSP is ook een streaming protocol die over het algemeen TCP gebruikt. Het wordt voornamelijk gebruikt in applicaties zoals YouTube, Spotify en Skype en in bijvoorbeeld beveiligingscamera's. RTSP regelt zelf niet het overdragen van de data maar functioneert als een soort afstandsbediening die de server instructies geeft met betrekking tot het streamen. RTSP gebruikt dus een onderliggend protocol om data over te dragen, dit onderliggend protocol is meestal RTP samen met RTCP.

HLS

HLS (HTTP Live Streaming) is een streaming protocol ontwikkeld door Apple in 2009. Het wordt ondersteund door heel veel mediaspelers en browsers. Het wordt voornamelijk gebruikt bij het streamen van beeldmateriaal naar HTML5- en Androidmediaspelers.

(S)RTP

Real-time Transport Protocol is een protocol die audio en video op een hele snelle manier van apparaat tot apparaat kan streamen. Het wordt veel gebruikt in applicaties waarbij je kan (video)bellen of media streamen. Er wordt meestal gebruik gemaakt van het UDP-protocol, maar het is ook mogelijk om gebruik te maken van TCP. WebRTC maakt ook gebruik van het SRTP-protocol (Secure RTP), net zoals het bovengenoemde RTSP-protocol.

WAT ZIJN DE VOOR- EN NADELEN VAN EEN AANTAL BESTAANDE PROTOCOLLEN

Ik ga deze deelvraag beantwoorden door onderzoek te doen naar wat nou de voor- en nadelen zijn van de bovengenoemde protocollen met als extra criteria dat het moet voldoen aan mijn use case.

Mijn criteria zijn:

- Er mag niet meer dan een halve seconde vertraging zijn
- Het moet schaalbaar zijn
- De kwaliteit moet gangbaar zijn
- Het moet makkelijk te integreren zijn met een webapplicatie

RMTP

Voordelen:

-Relatief lage stream vertraging (gemiddeld 5 seconden)

-Weinig buffering

Nadelen:

-Slechte support voor web

-Niet goed geoptimaliseerd voor het opschalen

Conclusie:

RMTP is geen geschikt protocol voor mijn use case. De relatief lage vertraging van 5 seconden is nog steeds te hoog voor het videobellen. Ook is het een gedateerd protocol en dus ook niet makkelijk te integreren met een html video-element. RMPT zou geschikter zijn bij het livestreamen.

RTSP

Voordelen:

- *Kan gebruik maken van zowel TCP als UDP*
- *Lage stream vertraging (gemiddeld 3 seconden)*
- *Flexibel*

Nadelen:

- *Slechte support voor web*

Conclusie:

RTSP is helaas ook niet geschikt voor mijn use case. De iets lagere vertraging dan die van RMTP is helaas ook niet goed genoeg. Ook de slechte integratie met web is een bottleneck.

HLS

Voordelen:
- *Adaptive bit rate (bij slechtere verbindingen een lagere kwaliteit)*
- *Wordt door veel platforms ondersteund (e.g. web, android & ios)*

Nadelen:
- *Hoge vertraging (20+ seconden)*
- *Kan alleen streams ontvangen (kan zelf niet streamen)*

Conclusie:

HLS is ook geen geschikt protocol voor mijn use case. Hoewel de brede ondersteuning en adaptive bit rate allebei aan mijn criteria voldoen wordt het onmogelijk om te werken met de vertraging van 20 seconden (of hoger). Daarnaast kun je zelf ook geen stream starten met het HLS protocol.

(S)RTP

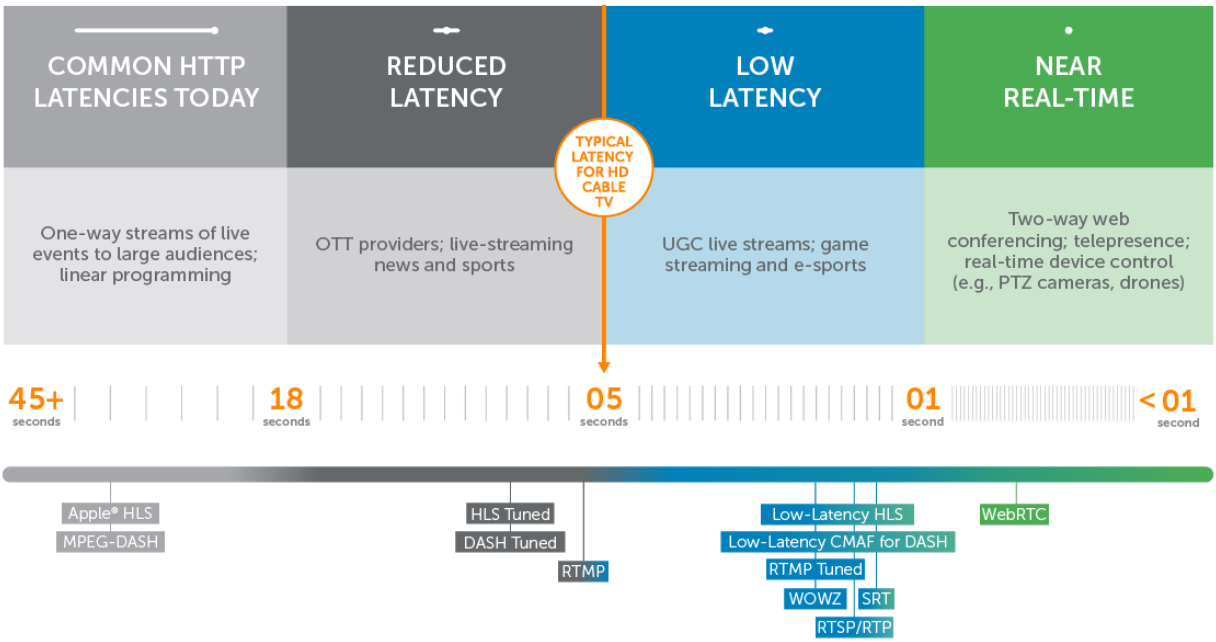
Voordelen:
- *Over het algemeen hele lage vertraging (onder de halve seconde)*
- *Wordt door web ondersteund in de vorm van WebRTC*

Nadelen:
- *Mogelijke kwaliteitsverlies door het gebruik van UDP*

Conclusie:

(S)RTP in de vorm van WebRTC is duidelijk de best optie uit het lijstje. Het heeft de laagste vertraging van alle protocollen en wordt ook nog eens ondersteund in de meeste moderne browsers. Het mogelijke kwaliteitsverlies is geen heel groot probleem als je kijkt naar wat de alternatieven zijn. Daarnaast hoeft er naast het opzetten van de verbinding tussen gebruikers geen server opgezet te worden voor het streamen. WebRTC geeft namelijk de mogelijkheid om media peer-to-peer te streamen.

STREAMING LATENCY AND INTERACTIVITY CONTINUUM



WAT IS WEBRTC?

Ik heb onderzoek gedaan naar bestaande webapplicaties met een videochat functie en bestaande protocollen onderzocht/vergeleken. Uit mijn onderzoek kan ik concluderen dat WebRTC eigenlijk de enige optie goede optie voor mijn project/use case. Ik wil WebRTC dus gaan gebruiken in mijn prototype en uiteindelijk in het eindproduct. Maar eerst wil ik de vraag beantwoorden: “Wat is WebRTC?”.

WebRTC is een technologie die real time communicatie mogelijk maakt in de meeste moderne internetbrowsers. Het is vooral bedoeld voor communicatie waarbij snelheid belangrijk is zoals bij (video)bellen of (grote) bestanden uitwisselen. Het is beschikbaar voor ontwikkelaars in de vorm van drie Javascript API’s.

Zoals ik al eerder vermelde maakt WebRTC gebruik van het SRTP-protocol, wat ervoor zorgt dat de verzonden data geëncrypt wordt. Ook bij het opzetten van een peer-to-peer verbinding (signaling) wordt er gebruik gemaakt van https.

JavaScript API’s:

- **MediaStream/getUserMedia.** Deze API zorgt ervoor dat een webpagina of applicatie toegang krijgt tot bijvoorbeeld de camera en microfoon van de gebruiker, en dat de audio- en videostreams worden gesynchroniseerd.
- **RTCPeerConnection**. Dit is het component binnen WebRTC dat de audio- en videocommunicatie tussen ‘peers’ (webbrowsers) afhandelt. Deze API ondersteunt functionaliteiten zoals signaalverwerking, codec handling, bandbreedtebeheer en security. De rol van RTCPeerConnection komt duidelijk naar voren in onderstaand architectuurdiagram.
- **RTCDataChannel**. Deze API maakt WebRTC geschikt voor het uitwisselen van data anders dan audio en video, en dat met een lage latency en een hoge doorvoer. De use cases voor deze API zijn onder andere gaming, remote-desktopapplicaties die direct vanuit de browser zijn te gebruiken, chat en de uitwisseling van bestanden. Om te komen tot een consistente werkwijze is de opzet van de RTCDataChannel API met opzet gelijk aan die van WebSockets. Het grote verschil tussen RTCDataChannel en WebSockets is dat RTCDataChannel direct tussen browsers werkt en daardoor sneller is dan WebSockets, zelfs als een relay-server nodig is om zaken als firewalls en Network Address Translation te omzeilen.

Bron: https://sping.nl/open-source-technology/webrtc/

HOE VERBIND JE TWEE OF MEER GEBRUIKERS MET WEBRTC?

Om een verbinding tussen twee of meer gebruikers te maken moet er “signaling” plaats vinden. Signaling is het proces waarbij ICE candidates en SDP (Session Description Protocol) aanvragen/antwoorden worden uitgewisseld. Dit betekent eigenlijk gewoon dat je een manier vindt om gebruikers met elkaar te verbinden doormiddel van bijvoorbeeld een server. Aangezien signaling best lastig kan worden en veel tijd kost als je het zelf maakt, kies ik ervoor om te kijken naar een bestaande API die het proces makkelijker maakt.

Na wat onderzoek te hebben gedaan naar de mogelijkheden heb ik besloten een YouTube tutorial van de gebruiker: “Web Dev Simplified” te volgen. De reden dat ik hiervoor kies is omdat de video nog niet zo oud is, kort van duur en de uitleg heel duidelijk is. De enige voorkennis die je hoeft te hebben is een beetje verstand van javascript en socket.io. In de video wordt er gebruik gemaakt van de javascript API PeerJS. Het idee is dat je clientside een peer object aanmaakt en deze vervolgens verstuurt via de websocket server naar de andere gebruiker. De andere gebruiker ontvangt dit object dan en heeft de mogelijkheid om de verbinding te accepteren. Als de verbindingen vanuit beide kanten zijn geaccepteerd kun je de stream starten. Deze stream wordt clientside opgehaald via de eerdergenoemde getUserMedia API en vervolgens verstuurd naar de andere gebruiker(s).

PeerJS is een manier om op een makkelijke manier een verbinding te starten tussen twee of meer gebruikers. Er zijn nog veel meer technieken/API’s beschikbaar die misschien zelfs nog beter zijn. De reden dat ik voor deze manier heb gekozen is omdat ik nog niet zoveel ervaring met WebRTC en ook niet heel veel tijd heb om alles zelf uit te zoeken.

WAT IS EEN MANIER OM WEBRTC TE IMPLEMENTEREN IN MIJN WEBAPPLICATIE?

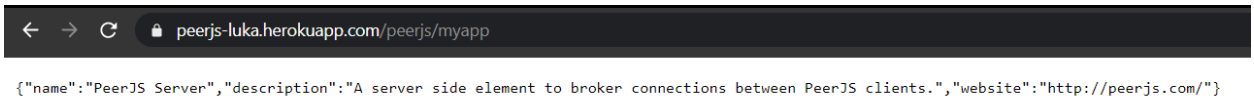
In dit gedeelte van mijn onderzoeksdocument ga ik beginnen met het prototypen. Ik wil graag met de React library een webpagina maken waarop 2 gebruikers met elkaar kunnen videobellen. Ik ga hierbij de eerdergenoemde socket.io en PeerJS library gebruiken om de verbinding tot stand te brengen.

Ik ben begonnen met het clonen van de repository van de gebruiker: “Web Dev Simplified”. Ik heb mijn best gedaan om zijn code zo goed mogelijk te snappen. Na anderhalve week begon ik toch eindelijk een beetje te begrijpen hoe socket.io in combinatie met PeerJS werkt. Ik heb toen gelijk een nieuw React project gemaakt en toen zelf geprobeerd dit te implementeren.

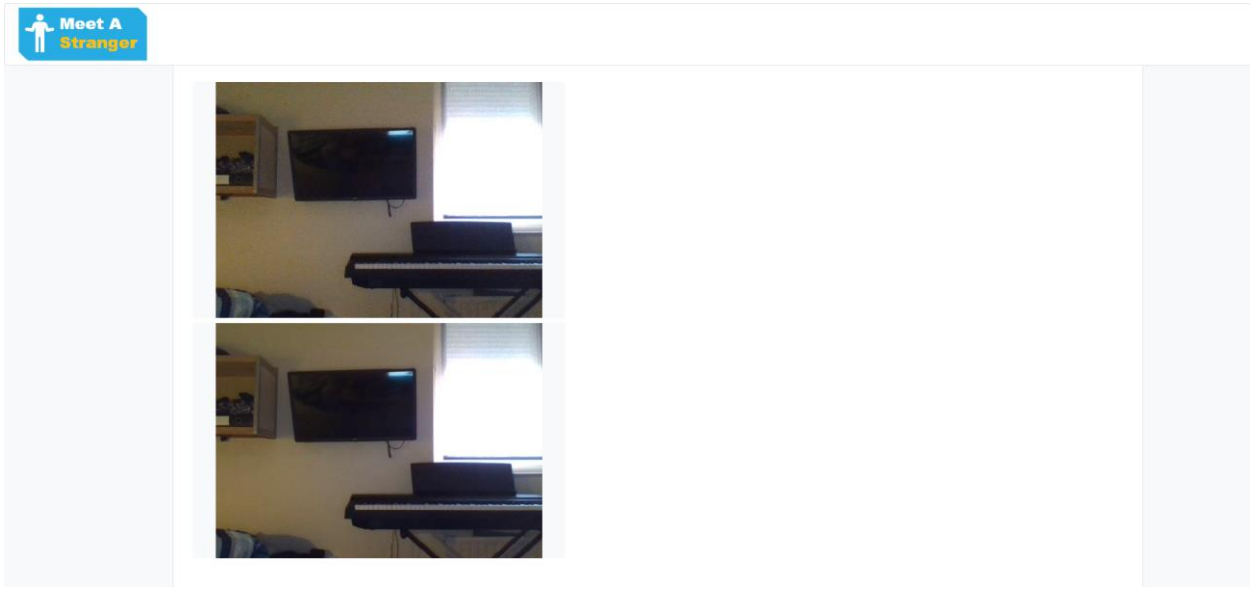
Op het begin wilde het niet werken omdat ik de volgorde van de events niet goed had geprogrammeerd. Toen ik hier eenmaal achter was wilde het eindelijk lukken. Maar er was een probleem, soms deed hij het wel en soms niet. Ik was heel erg verbaasd hierover en dacht dat er nog steeds een probleem was met de volgorde van de events. Ik heb mijn programma vaak aangepast en ben zelf een keer opnieuw begonnen. Maar tot mijn verbazing deed zich nog steeds hetzelfde probleem voor. Ik ben toen gaan kijken naar het peerobject en of deze wel verbinding maakte met de server (in de Cloud). Ik merkte dat deze het soms wel en soms niet deed. Ik vond dit raar en heb toen een peer on error event aangemaakt. Hier kwam uit dat de server dus vaak tegen zijn capaciteit aan zat en daarom geen peer-to-peer verbinding tot stand kon brengen.

Gelukkig is de PeerJSserver open source en kon ik de repository clonen. Ik probeerde de server lokaal te starten maar dit was helaas geen success. De server start wel op maar ik kon lokaal geen verbinding maken met mezelf. Ik heb dit opgelost door mijn eigen PeerJS server te hosten op Heroku.

Heroku PeerJS Server:



Prototype:



6. RESULTAAT

Ik ben redelijk blij met het resultaat. Ik vind dat ik de onderzoeksvraag goed heb weten te beantwoorden. Natuurlijk is er nog heel veel meer informatie beschikbaar over WebRTC, maar het onderzoek dat ik heb gedaan is in ieder geval heel nuttig voor mijn Use Case. Ik heb met succes een prototype weten te bouwen dat voldoet aan mijn criteria en ook nog heel veel geleerd van het proces.

7. REFLECTIE

Achteraf zou ik het onderzoek precies hetzelfde doen met vergelijkbare strategieën. Waar ik wel een beetje verbaasd over was dat eigenlijk dat in het begin van het onderzoek WebRTC al eigenlijk als beste naar voren kwam. Ik heb verder ook geen alternatief kunnen vinden. De rest van de protocollen/technieken waren allemaal niet geschikt. Dit maakte het begin van mijn onderzoek wel interessant maar niet heel erg nuttig uiteindelijk. Waar ik in het proces toch wel het meeste ben vastgelopen is bij prototyping. Ik kreeg het relatief snel aan de praat maar het heeft lang geduurd voordat ik erachter kwam dat het peer object soms wel/niet verbinding maakte met de PeerJS server. Dit heeft me gelukkig wel het een en ander geleerd over debuggen, dus zo erg is het achteraf niet. Ik ben over het algemeen tevreden met het resultaat en zou graag nog meer willen leren over WebRTC (specifiek het signaling gedeelte).

8. BRONNEN

- *https://www.wowza.com/blog/rtmp-streaming-real-time-messaging-protocol*
- *https://www.wowza.com/blog/rtsp-the-real-time-streaming-protocol-explained*
- *https://www.streamingvideoprovider.com/blog/streaming_video_protocols.html*
- *https://medium.com/av-transcode/hls-streaming-protocol-pros-and-cons-of-choosing-it-12ef080163b0*
- *https://www.wowza.com/blog/low-latency-hls-vs-webrtc*
- *https://www.3cx.nl/webrtc/wat-is-webrtc/*
- *https://sping.nl/open-source-technology/webrtc/*
- *https://webrtc.org/getting-started/peer-connections*
- *https://www.youtube.com/watch?v=DvlyzDZDEq4&t=1378s&ab_channel=WebDevSimplified*