UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga

Uporaba tehnologije WebRTC za oddaljeni nadzor

(Using WebRTC technology for remote control)

Ime in priimek: Matej Zemljak
Študijski program: Računalništvo in informatika
Mentor: doc. dr. Peter Rogelj

Koper, julij 2017

Ključna dokumentacijska informacija

Ime in PRIIMEK: Matej ZEMLJAK

Naslov zaključne naloge: Uporaba tehnologije WebRTC za oddaljeni nadzor

Kraj: Koper

Leto: 2017

Število listov: 51 Število slik: 14

Število prilog: 1 Število strani prilog: 1 Število referenc: 22

Mentor: doc. dr. Peter Rogelj

Ključne besede: WebRTC, Kurento medijski strežnik, oddaljen nadzor, stereo kamera, JavaScript, Node.js, navidezna resničnost

Izvleček:
V zaključni nalogi smo opisali in zgradili komunikacijski del sistema z uporabo tehnologije WebRTC. WebRTC je tehnologija prihodnosti. V svetu računalništva se je pojavila pred petimi leti in od takrat žanje velike uspehe. Razlog hitre popularnosti protokola je predvsem v njegovi preprostosti in načinu povezovanje med uporabniki. WebRTC primarno uporablja P2P (peer-to-peer) povezovanje, kar omogoča hitro odzivnost zaradi načina vzpostavitve povezave. S P2P povezovanjem je lahko vsak računalnik odjemalec ali strežnik. Omogoča povezovanje med napravami brez dostopa po osrednjem strežniku. Skupini razvijalcev WebRTC protokola je bilo glavno načelo preprostost, kar WebRTC tudi je. Z njegovo izgradnjo lahko sedaj uporabniki komunicirajo neposredno preko brskalnikov. Integracija avdio in video vsebine neposredno v spletni brskalnik je bila včasih zelo težko izvedljiva. Z WebRTC protokolom pa se je to spremenilo. WebRTC je najboljša rešitev kadar želimo ustvariti aplikacijo kjer nočemo, da bi uporabnik moral namestiti programsko opremo. Vse kar uporabnik potrebuje je spletni brskalnik, ki pa je v večini sodobnih računalnikih že nameščen. WebRTC protokol je idealen za realizacijo sistema za oddaljen nadzor. Uporabnik v brskalnik vpiše IP naslov oddaljene nadzorne naprave in že ima popolni nadzor nad njo. Naš cilj je bil ustvari sistem za oddaljen nadzor na preprost način z možnostmi nadgradnje, razširljivosti in uporabnosti.

Key words documentation

Name and SURNAME: Matej ZEMLJAK

Title of final project paper: Using WebRTC technology for remote control

Place: Koper

Year: 2017

Number of pages: 51 Number of figures: 14

Number of appendices: 1 Number of appendix pages: 1 Number of references: 22

Mentor: Assist. Prof. Peter Rogelj, PhD

Keywords: WebRTC, Kurento media server, remote control, stereo camera, JavaScript, Node.js, virtual reality

Abstract: In the final thesis, we described and built the communicational part of the system with the use of WebRTC technology. WebRTC is the technology of the future. In the world of computers, it appeared five years ago and has had great success since then. The reason for the protocol's rapid popularity is particularly in its simplicity and the type of connection between users. WebRTC primarily uses P2P (peer-to-peer) connection, which enables fast response due to the way it establishes the connection. Through P2P connection, each computer can be the receiver or server. It allows connection between devices without access to the main server. The main principle of the group of WebRTC protocol group of developers was simplicity and this is what WebRTC undoubtedly is. With its implementation, users can now directly communicate through browsers. In the past, the integration of audio and video content directly into the web browsers was very hard to accomplish. This has changed with WebRTC protocol.

WebRTC is the best solution, when we want to create an application, where we do not want the user to have to install additional software. The only thing the user needs is a web browser, which is mostly already installed in the case of modern computers. The WebRTC protocol is ideal for the realization of a system for remote control. The user types the IP-address of the remote controll device into the browser and he already has full controll over it. Our goal was to create a simple remote controll system with possibilities for upgrades, extension and usability.

Zahvala

Zahvaljujem se mentorju, dr.Petru Roglju, za strokovno pomoč pri izdelavi zaključne naloge.
Zahvaljujem se tudi družini, ki me je skozi celoten študij podpirala, in prijateljem za pomoč in podporo med študijem.

Kazalo vsebine

Kazalo slik

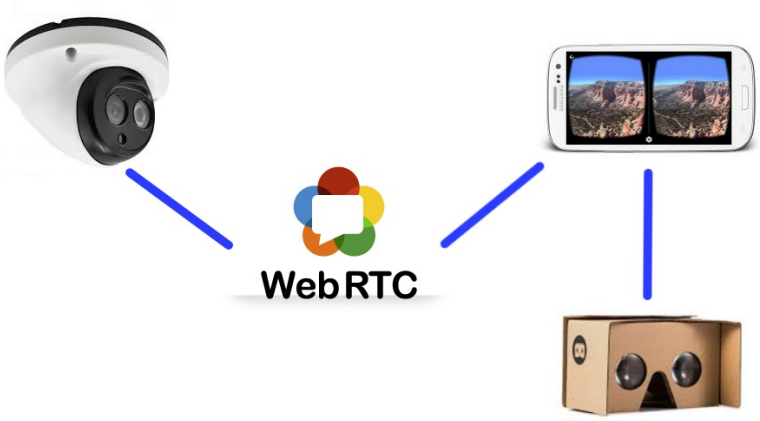
Kazalo prilog

Seznam kratic

tj.	To je
npr.	Na primer
itd.	In tako dalje
API	Application programming interface - aplikacijski programski vmesnik
framework	Ogrodje
LTE	Long Term Evolution - standard na področju mobilnih telekomunikacij
EDGE	Enhanced Data rates for Global Evolution - izboljšane vrednosti hitrosti prenosa za globalni napredek
IP	Internet Protocol adress - protokol internetnega naslova
NAT	Network address translation - prevajanje omrežnih naslovov
ICE	Interactive Connectivity Establishment - vzpostavitev medsebojne povezave
STUN	Session Traversal Utilities for NAT - sejni pripomoček za NAT
TURN	Traversal Using Relays around NAT - sejni pripomoček z uporabo relejev pri NAT
HTML	HyperText Markup Language - jezik za označevanje nadbesedila
IoT	Internet of Things - Internet stvari
SDP	Session Description Protocol - protokol opisa seje
NPM	Privzeti upravitelj paketov za JavaScript izvajalno okolje Node.js
TCP	Transmission Control Protocol - protokol za nadzor prenosa
XML	Extensible Markup Language - razširljivi označevalni jezik
TLS	Transport Layer Security - kriptografski protokol
SIP	Session Initiation Protocol - protokol inicializacije seje
SCTP	Stream Control Transmission Protocol - protokol nadzora prenosnega toka

1 Uvod

WebRTC je zastonjski in odprtokodni projekt, ki nam zagotavlja komunikacijo v realnem času (RTC - Real-Time Communications) med internetnimi brskalniki preko preprostega programskega vmesnika (API). Komponente WebRTC protokola so bile optimizirane, da bi kar se da najbolše upravljale svoje delo. WebRTC je zaživel po iniciativah korporacij kot so Google, Mozilla, Opera in drugih. Njihov prvotni cilj je bil ustvariti protokole, ki bi povezovali kakovostne RTC aplikacije znotraj brskalnikov, mobilnih naprav in IoT napravah. [7] Ideja zaključne naloge je vzpostaviti povezavo s



Slika 1: Delovanje sistema.

pomočjo protokola WebRTC na način kot delujejo sistemi za oddaljeni nadzor. Sistem za oddaljeni nadzor deluje tako, da imamo nekje postavljeno kamero, ki je povezana z računalnikom. Računalnik nam predvaja sliko iz kamere in tako lahko opazujemo dogajanje v živo iz območja, kjer je kamera postavljena. Sistem bo poleg samega prenosa žive slike, omogočal prenos podatkov. Podatki se bojo pošiljali iz smeri odjemalca proti kameri in tako bo moč kamero tudi usmerjati. Naloga bo vsebovala razvoj aplikacije po korakih do končnega cilja. Sliko kamere bi v posbnih primerih želeli tudi predhodno obdelati. Takšen primer bi bi v primeru uporabe stereo para kamer in prikaza na prikazovalniku navidezne resničnosti, kjer je v slike potrebno vnesti geometrijsko popačenje.

Cilj zaključne naloge je, da preko razvoja aplikacije pridobimo znanje o protokolu WebRTC. Ker pa protokol WebRTC bazira svoje API-je na programskem jeziku JavaScript, bomo poglobili svoje znanje tudi na področju programskih jezikov.
Zaključna naloga vsebuje 4 poglavja, vključno z uvodom. V vsakem od naslednjih poglavij je predstavljen postopek realizacije aplikacije. V zaključku pa so navedene ideje za nadgradnjo sistema ter kratek povzetek zaključne naloge.

2 Predstavitev WebRTC

Za delovanje našega sistema potrebujemo že definirane protokole in programske jezike s katerimi bomo te protokole med seboj povezali in ustvariti eno celoto. Sistem bo baziran na odprtokodnem ogrodju WebRTC, ki bo uporabljen na dva načina. Prvi namen komunikacije pri oddaljenem nadzoru je prenos slike med kamero in predvajalnikom pri katerem je mogoča obdelava slike. Drugi namen je posredovanje krmilnih podatkov v smeri od opazovalca do napreve oddaljenega nadzora, s katero lahko vplivamo na smer pogleda kamere. Zaključna naloga potrebuje obdelavo slike v realnem času, s katero prilagodi sliko potrebam odjemalca. V mislih imamo predvsem možnost implementacije združevanja več slik in radialnega popačenja, ki je potrebno za prikaz virtualne resničnosti (VR), kot je na primer Google cardboard. Obdelava slike bo realizirana s pomočjo Kurento medijskega strežnika, ki ponuja različne možnosti procesiranja slike. V nadaljevanju bom uporabljene protokole in programske jezike tudi opisal.

2.1 Programsko ogrodje WebRTC

WebRTC je sveža stvar v svetu računalništva. V letu 2017 je dopolnila peto leto obstoja, od kar je uradno izšla za množično uporabo. Interaktivni avdio in video je sedaj videti kot običajni dodatek znotraj internetnih aplikacij, tako na spletu kot v izvornih aplikacijah. Mnogi izdelki in storitve nam sedaj omogočajo funkcionalnosti, ki neopazno prečkajo mejo med spletom in izvornimi aplikacijami. Vse to omogočajo osnovni standardni programski vmesniki (API), kateri so bili razviti znotraj odprte skupnosti in v veliki meri vgrajeni v Google WebRTC projekt.
Obstoj skupnega niza sprejetih standardov, je poleg Google-a spodbudila tudi druge ponudnike spletnih brskalnikov, kot so Firefox, Opera in nedavno tudi Microsoft Edge, k integraciji WebRTC standarda. To je privedlo do več kot dveh milijard brskalnikov, ki omogočajo WebRTC. Razširjenost WebRTC ogrodja je privedla do ene milijarde avdio-video minut na teden zgolj v brskalniku Chrome. Po podatkovnem kanalu se pošlje preko 500 TB podatkov na teden.
WebRTC je implementiran v najbolj popularne aplikacije kot so Facebook, Messenger, Hangouts, Snapchat in mnogo drugih, ki skupaj štejejo več kot milijardo prenosov za Android in iOS naprave.

Danes je WebRTC robusten komplet protokolov, kodekov in programskih vmesnikov (API), ki se je izkazal kot dobra alternativa že obstoječim rešitvam istega problema. V naslednjih nekaj letih se pričakuje še več uporabnih funkcij, ki se bojo dograjevali na že obstoječem in hitro rastočem uspehu protokola WebRTC. [6]
WebRTC je postal priljubljen zaradi:

- Neprestanega nadgrajevanja avdio in video kvalitete
  Kodeki so že predefinirani, vendar je na tem področju še dovolj prostora za napredek.

- Svežega pogleda na povezovanje
  Današnja komunikacija poteka v različnih vrstah omrežij, od LTE omrežja pa vse do Ethernet omrežja. WebRTC se sooča z raznolikim naborom pogojev omrežja in zato se mora ustrezno prilagoditi. S tem ko se WebRTC prilagaja novim pogojem omrežij, prihaja tudi do poenostavitve medijskih protokolov. Protokoli so osnova za vsa WebRTC povezovanja in to posledično privede do lažje integracije WebRTC v lastne aplikacije.

- Universal RTC
  Pred petimi leti, je večina komunikacije poteka preko aplikacij za osebne računalnike. To se je sedaj vse spremenilo, in WebRTC sledi novemu trendu. V prihodnosti lahko pričakujemo WebRTC v drugih oblikah komunikacije vključno z komunikacijo naprav v navidezni resničnosti.

Cilj WebRTC protokola je omogočiti multimedijske klice z uporabo spletnih brskalnikov, brez potrebe po vtičnikih, kot je naprimer Adobe Flash. To stori tako, da podpira povezave neposredno v HTML5. Na ta način se HTML koda, ki teče v brskalniku lahko neposredno sklicuje na osnovno WebRTC infrastrukturo.
Vsaka rešitev z WebRTC potrebuje klicatelja in prejemnika klica. Veliko imajo skupnega, vendar obstaja nekaj zelo pomembnih razlik. [1] Skupni potek, ki ga sledijo tako klicatelj kot prejemnik klica je sledeč:

Vzpostavitev povezave s pomočjo signalizacijskeg strežnika

To je mogoče doseči na več različnih načinov, vendar je najlažji način za oba uporabnika, da obiščeta isto spletno stran in se povežeta preko skupnega signalizacijskega strežnika. Uporabniki si izmenjujejo neke vrste imena ali žetone, ki omogočajo edinstveno identifikacijo seje. Ta skupni žeton je lahko številka sobe ali identifikacijska števila pogovora.
Najpogostejši način povezave dveh uporabnikov s signalizacijskim strežnikom je z uporabo WebSocket API-ja.

Začetek signalizacije med dvema uporabnikoma

Uporabnika si medseboj izmenjata žetone, da se prepoznata. Sledi izmenjava signalizacijskih sporočil preko WebSocket povezave. WebRTC nima določenega signalizacjiskega protokola, zato je na razvijalcih, da izberejo najprimernejšo rešitev.

Izmenjava informaij o svojih omrežjih in njihova možnost povezove

Ta korak pogosto označujemo kot "iskanje kandidatov". Njen namen je omogočiti spletnim brskalnikom izmenjavo o omrežju za pošiljanje neposrednih medijskih podatkov. Večina uporabnikom uporablja zasebne IP naslove, zato je potreba po metodi prevajanja omrežnega naslova, ki nam ga omogoči NAT.
Če želi poiskati javni IP naslov, WebRTC izkoristi STUN in/ali TURN strežnike. Ti strežniki zagotavljajo odjemalca s naslovom IP, s katerim je mogoče vzpostaviti medijsko povezavo.
WebRTC imenuje proces uporabe STUN in TURN strežnikov Interactive Connectivity Establishment (ICE) ogrodje. ICE najprej poizkuša vzpostaviti povezavo z uporabo STUN strežnikov, če to ni mogoče uporabi TURN strežnike.

Pogajanje o medijski seji

Ko uporabniki vedo, kako komunicirati eden z drugim, nastopi faza dogovora o vrsti in obliki medijev. Ta proces se izvede s pomočjo JavaScript Session Establishment Protocol (JSEP). JSEP uporabi Session Description Protocol (SDP) za prepoznavanje kodekov, ločljivosti, bitne hitrosti, velikosti okvirja, itd., ki jih podpirata uporabnika.

Začetek RTCPeerConnection

Potem, ko je signalizacijaska povezava vzpostavljena in uporabniki so se dogovorili o svojih medijskih sposobnostih, lahko začne pretok medijskih vsebin. To se izvede s pomočjo WebRTC konstruktorja, RTCPeerConnection.

RTCPeerConnection API

Z RTCPeerConnection API-jem se zgodi dokončna vzpostavitev P2P povezave med dvema brsklanikoma. Upravlja ICE opravilnik, medijske tokove, dostop do lokalnega mikrofona in spletne kamere in procesira JSEP ponudbe in odgovore.
Spletni brskalnik bo vstvaril RTCPeerConnection podobno naslednjemu:

```
var myPeerConnection = RTCPeerConnection(configuration)
```

Spremenljivka “configuration” vsebuje ključ iceServers, ki vsebuje polje informacij STUN in TURN strežnikov.
myPeerConnection je objket, ki ga obe strani povezave uporabljata, vendar je njegovo delovanje odvisno glede na to na kateri strani se uporablja.

Za klicoče uporabnike

- Registracija ”onicecandidate”upravitelja:
  ”onicecandidate”upravitelj pošlje ICE kandidate klicatelju preko signalizacijskega kanala.

- Registacija upravitelja medijskega toka:
  Upravitelj medijskega toka prikazuje video tok od klicanega uporabnika.

- Registracija upravitelja sporočil:
  Upravitelj sporočil se uporablja za obdelavo sporočil, prejetih od klicanega uporabnika.

- Dostop do lokalne kamere in mikrofona:
  Fukncija getUserMedia() zajema lokalni medijski tok podatkov, ki se potem prikaže na lokalni spletni strani. Ta tok podatkov se nato doda k objektu myPeerConnection z metodo addStream().

- Pogajanje medijev:
  V tem koraku, spletni brskalnik opravlja JSEP ponudbe/odgovore z ustvarjanjem ponudb z uporabo metode myPeerConnection, createOffer(). Povratni klic se shrani v objekt RTCSessionDescription. RTCSessionDescription se doda k myPeerConnection z uporabo metode setLocalDescription(). Nato se RTCSessionDescription pošlje oddaljenemu uporabniku preko signalizacijskega kanala. Končni rezultat je, da bo SDP kličočega nastavljen tako za klicatelja kot kličočega.

Za klicane uporabnike

Koraki za klicane stranke potekajo podobno kot tiste za klicatelja z izjemo, da klicani se odzove z odgovorom na ponudbo klicatelja

- Registracija onicecandidate upravitelja:
  ”onicecandidate”upravitelj pošlje ICE kandidate klicanega preko signalizacijskega kanala.

- Registracija upravitelja medijskega toka:
  Upravitelj medijskega toka prikazuje video tok od klicočega uporabnika.

- Registracija upravitelja sporočil:
  Upravitelj sporočil se uporablja za obdelavo sporočil, prejetih od klicatelja uporabnika.

- Dostop do lokalne kamere in mikrofona:
  Fukncija getUserMedia() zajema lokalni medijski tok podatkov, ki se potem prikaže na lokalni spletni strani. Ta tok podatkov se nato doda k objektu myPeerConnection z metodo addStream().

- Pogajanje medijev:
  Tukaj se pokažejo razlike med klicateljem in klicanim. Ko preko signalizacijskega kanala prispe sporočilo z "new session description", bo klicani to zaznal kot novo povezavo in bomo klicali funkcijo myPeerConnection.setRemoteDescription() in nastavi opis seje.

Nato se aktivira funkcija myPeerConnection.createAnswer(), ki vrne opis seje klicanega.

Ko sta obe strani vzpostavili signalizacijsko povezavo in izmenjali medijske opise, lahko medijski podatki med uporabniki stečejo. Sejo nato lahko upravljamo (sprostimo, itd.) preko signalizacijskega kanala.

2.1.1 Signaliziranje

WebRTC uporablja RTCPeerConnection za vzpostavitev komunikacije prenosa podatkov med brskalniki(uporabniki). Signalne metode in protokoli niso določeni s strani WebRTC, zato potrebujemo tudi mehanizem za pošiljanje nadzornih sporočil. [15] Proces se imenuje signalizacija, ta pa nima definiranih signalnih protokolov, razlogi za to so:

- Različne aplikacije lahko zahtevajo različne protokole. Razvijalci WebRTC so želeli, da je sistem dostopen vsem razvijalcem.

- Izvaja se v spletnem brskalniku, predefinirana podpora za signaliziranje bi zahtevala, da se ob vsakem ponovnem zagonu spletne strani, ponastavi signalizacijski kanal.

WebRTC sam po sebi ne skrbi za signalizacijo. Signalizacija je potreba za nastavitev klicev, zato WebRTC potrebuje rešitev z dodanim signalizacijskim strežnikom.
Edina zahteva, ki jo morajo razvijalci upoštevati pri signalizaciji je uporaba SDP-ja. Za pogajajanja pri multimedijskih zmogljivosti med uporabniki uporabljamo SDP

protokol, ki lahko samostojno opravi zahtevano delo. Signalizacijski protokol mora delovati tako, da uporabnik pošlje SDP paket in SDP paket dobi nazaj. Spremembna protokolov med potjo, ne vpliva na delovanje WebRTC-ja.

Vzpostavitev seje

Mehanizem za vzpostavitev seje med dvema uporabnikoma (klicatelj, klicoči) je opisana v speficikaciji JSEP. Koraki za vzpostavitev seje so naslednji:

- Ponudba klicatelja vsebuje njegov lokalni SDP.
- Klicatelj doda svojo ponudbo v RTCPeerConnection objekt.
- Klicatelj pošlje novo ustvarjen projekt preko signalizacijskega strežnika z uporabo WebSocket protokola. WebSocket protokol omogoča polno dvojno povezavo (full-duplex) komunikacijskega kanala. Standardiziran je z WebSocket protokolom, ki je način za pošiljanje signalizacijskih informacij med spletnim brskalnikom in signalizacijskim streženikom.
- Klicoči sprejme ponudbo klicatelja z uporabo WebSocket-a.
- Klicoči ustvari odgovor, ki vsebuje njegov lokalni SDP.
- Klicoči doda svoj odgovor, skupaj s ponudbo klicatelja, v njegov novo ustvarjen RTCPeerConnection objekt.
- Klicoči odgovori na signalizacijski strežnik z uporabo WebSocket-a.
- Klicatelj sprejme ponudbo klicočega s pomočjo WebSocket-a.

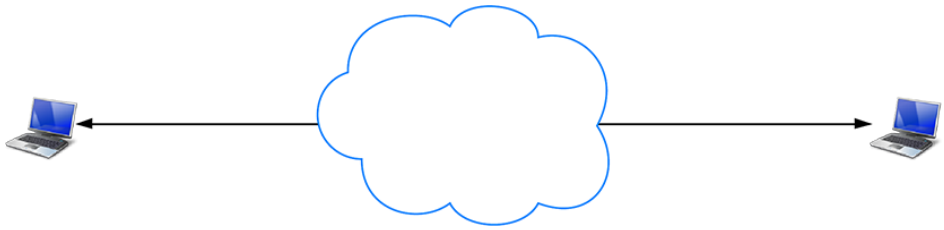
Za lažje razumevanje postopka povezave med dvema uporabnikoma, sem izpustil uporabo ICE, STUN in TURN, kateri bodo predstavljeni v nadaljevanju.

Varnost

Šifriranje je obvezno za vse komponente protokola WebRTC. Ker signalni mehanizmi niso določeni s strani WebRTC, je na uporabniku(razvijalcu), da poskrbi za varno signalizacijo. Nezavarovana signalizacija lahko privede do zaustavitve prenosa, preusmerjanja povezave in celo do snemanja seje.
Najpomembnejši dejavnik pri zagotavljanju varne signalizaije je uporaba varnih protokolov HTTPS in TLS, ki zagotavljajo da sporočila med signalizacijo ni mogoče dešifrirati. [4]

2.1.2 Vzpostavljanje medijskih povezav

Po signalizaciji pride na vrsto soočanje z NAT in požarnimi zidovi. Za signalizacijo metapodatkov WebRTC aplikacija uporablja vmesni signalizacijski strežnik. Za vzpostavitev medijske povezave, WebRTC poskuša vzpostaviti neposredno povezavo peer-to-peer kot lahko vidimo na sliki 2. Slika 2 prikazuje povezavo dveh računalnikov v oblak, ki predstavlja internetno omrežje. V enostavnejšem svetu bi vsaka končna točka imela enoličen naslov, ki bi si ga lahko izmenjavali z drugimi uporabniki, da bi zagotovili neposredno komunikacijo ob predpostavki, da se vnaprej medsebojno poznata svoja naslova(vsaj enega). [4] V realnem internetu, temelječemu na IPv4 protokolu,



Slika 2: Peer-to-peer povezava [4].

večina naprav vsebuje eno ali več plasti NAT. Na sliki **??** je prikaz povezave med dvema računalnikoma, kadar se vmes postavi NAT sistem. Nekateri uporabniki imajo protivirusno programsko opremo, ki blokira določena vrata in protokole. Požarni zid in NAT se dejansko lahko izvajata na isti napravi. Primer takšnega delovanja je domači wifi usmerjevalnik. Na sliki **??** lahko opazimo NAT sistem, ki ni vezan na signalizacijsko pot, ampak ima vsak sistem znotraj svojega omrežja NAT pregrado. Uporabnik znotraj svojega omrežja mora tako posredovali zunanji IP naslov, s katerim bo lahko P2P povezava vzpostavljena.
WebRTC aplikacije uporabljajo ICE(Interactive Connectivity Establishment) ogrodje, ki je opisan v nadaljevanju. Ogrodje nam omogoča premagovati kompleksnost povezovanja v realnem svetu. Za učinkovito realizacijo mora aplikacija posredovati URL ICE strežnika RTCPeerConnection objektu, kot bo v nadaljevanju opisano.
ICE najde najboljšo pot za povezovanje. Vzporedno preverja vse možnosti in izbere najbolj učinkovito med njimi. ICE sprva poskuša vzpostaviti povezavo z naslovom gostitelja pridobljenega iz operacijskega sistema naprave in mrežne kartice. Če to ne uspe (če je naprava za NAT sistemom) ICE pridobi zunanji naslov s pomočjo STUN (Session Traversal Utilities for NAT) strežnika, in če tudi to ne uspe, je ves promet preusmerjen preko TURN (Traversal Using Relays around NAT) relejskega strežnika. [4] Z drugimi besedami:

- STUN strežnik se uporablja za pridobitev zunanjega omrežnega naslova.

- TURN strežnik se uporabi kot posrednik za preusmeritev prometa, če neposredna povezava ne uspe.

Oba bosta podrobneje opisana v naslednjih poglavjih.

2.1.3 ICE

Interactive Connectivity Establishment (ICE) je tehnika v računalniškem omrežju, ki poišče najučinkovitejšo pot povezave med dvema računalnikoma v P2P omrežju. Najpogosteje jo uporabljamo pri interaktivnih vsebinah, kot so VoIP(Voice over Internet Protocol), P2P, video in tekstovna komunikacija. S tem, si želimo izogniti komunikaciji prek centralnega strežnika, kateri bi komunikacijo upočasnil in podražil.

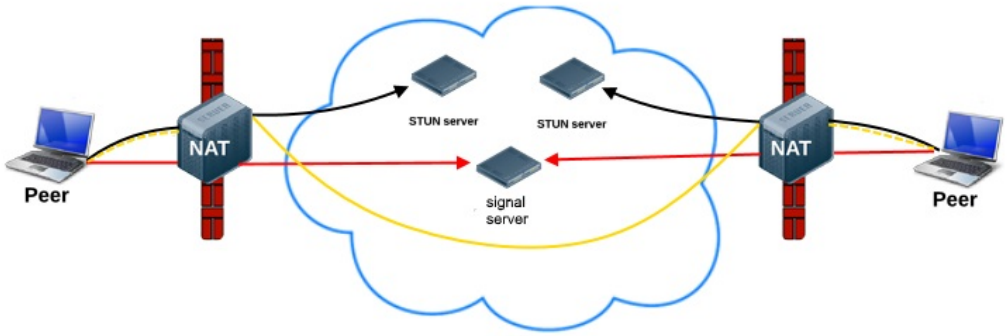
2.1.4 STUN in TURN

WebRTC je osnovan da deluje po principu peer-to-peer, tako da se lahko uporabniki povežejo med seboj po najhitrejši možni poti. WebRTC je namenjen uporabi v realenm času in se zato spopada z raznimi preprekami. Prepreke na katere je potrebno biti pazljiv so NAT vrata, požarni zidovi ter potreba po drugačnem načinu povezave, če neposredna povezava zataji. WebRTC uporablja API protokol s katerim preko STUN strežnika pridobi zunanji IP naslov računalnika. V primeru, da peer-to-peer komunikacija zataji, pa uporabi TURN strežnik za vzpostavitev povezave.
Vsak TURN strežnik podpira tudi STUN način delovanja. Razlika je zgolj v temu, da ima TURN strežnik dodano relejsko funkcijo. Kar pomeni, da se podatki posredujejo preko javnega strežnika. ICE tehnika, znotraj STUN in TURN strežnika, omogoča delovanje preko NAT naprav med povezujočimi uporabniki. STUN in TURN strežniki dobijo URL, ki je definiran v objektu RTCPeerConnection. WebRTC aplikacije vsebujejo iceServers konfiguracijo, ki je sestavi del RTCPeerConnection objekta.
RTCPeerConnection uporablja ICE ogrodje za optimalno izbiro poti med uporabniki, po potrebi z uporabo STUN ali TURN strežnikov. [15]

- STUN
  V primeru uporabe NAT so v lokalnem omrežju uporabljeni lokalni IP naslovi, ki v zunanjem omrežju niso veljavni/dosegljivi. IP naslove, ki so definirani za NAT napravo ni mogoče neposredno povezati. Zahteva po povezavi mora vedno prihajati s strani lokalnega omrežja (izza NAT). Omejitev je ta, da napave za NAT-om vzpostavi povezavo, s tem določi preslikavo naslovov na NAT-u, to isto preslikavo pa potem lahko koristimo pri prenosu. WebRTC reši problem tako, da uporabi STUN strežnik. Potek pridobitve IP naslovov in pretok medijskih

podatkov, lahko vidimo na sliki 3.

STUN strežniki so prisotni na internetu in imajo eno preprosto nalogo: preveriti IP naslove in vrata dohodnih zahtev (v aplikaciji, ki teče za NAT infrastrukturo) in priskrbeti pravilen uporabnikov IP naslov kot odgovor. Z drugimi besedami, aplikacija uporablja STUN strežnik za odkrivanje uporabnikovega IP naslova in vrat iz javne prespektive. S tem procesom pridobimo podatke, ki so potrebni pri posredovanju drugi napravi. Nato s pomočjo mehanizma za signalizacijo lahko vzpostavimo neposredno povezavo [15]. STUN strežniki zelo dobro opravijo svoje delo in večina WebRTC aplikacij se uspešno poveže z njihovo pomočjo [4].

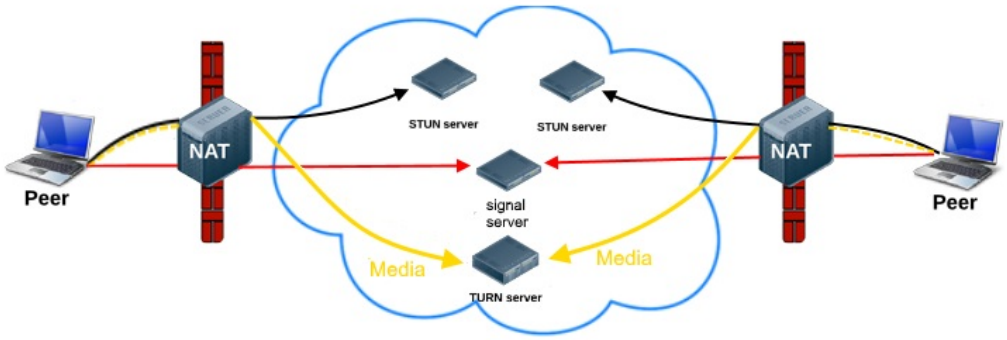


Slika 3: Uporaba STUN strežnikov za pridobitev javnih IP naslovov z pripadajočimi vrati.

- TURN

  Traversal Using Relays around NAT (TURN) je protokol, ki pomaga pri prečkanju NAT in požarnih zidov za multimedijske aplikacije. Uporablja se lahko z Transmission Control Protocol (TCP) in User Datagram Protocol (UDP). Koristno je pri omrežjih, zakritih za simetričnimi NAT napravami. Gre za NAT, ki ne omogoča, da se paketi iz zunanjega dela omrežja vračajo iz drugega naslova, kot je tisi, kamor so posredovani, zaradi načina dodelovanja transportnih naslovov(vrat). TURN omogoči povezavo uporabnika, ki je za NAT sistemom, s samo enim uporabnikom (primer telefonska povezava) [21].

RTCPeerConnection poizkuša vzpostaviti neposredno komunikacijo med uporabniki s pomočjo UDP paketov. Včasih takšen pristop ne uspe, ker požarni zidovi preprečijo pretok UDP paketov, zato se RTCPeerConnection zateče k TCP paketom. Vendar tudi TCP včasih ne uspe vzpostaviti povezave, takrat se uporabi TURN strežnik. Vsak od treh pristopov nam omogoča posredovanje podatkov in posledično vzpostavitev povezave med vozlišči. Treba je opozoriti, da se TURN strežnik uporablja za prenašanje avdio/video/podatkov med uporabniki, ne pa zgolj signalnih podatkov. Delovanje si lahko predstavljamo s pomočjo slike 4. Med dvema vozliščema se vzpostavi ena povezava s TURN strežnikom preko katerega poteka pretok medijskih podatkov. Potrebno je poudariti, da pretok medijskih podatkov poteka preko TURN strežnika. S pomočjo TURN strežnika



Slika 4: Povezovanje prek TURN strežnikov.

se izognemo dvema problemoma pri napravah, ki so izza NAT sistema. Pogledati moramo s stališča, kdaj se nek paket uspe prebiti preko NAT sistema. Prvič ko podatki prihajajo iz notranjega omrežja in NAT lahko definira novo preslikavo naslovov. Drugič ko podatki prihajajo od zunaj, na naslov, ki je bil predhodno že definiran ob prenosu podatkov od noter navzven. TURN strežnik imajo preprosto nalogo - obdržati delujočo povezavo. Njihova slabost je, da potrebujejo veliko količino pasovne širine, saj preko njih tečejo vsi podatki komunikacije, ne le signalizacija [4].

3 Programski jeziki in ogrodja

3.1 JavaScript

JavaScript je objektni skriptni programski jezik, ki ga je razvil Netscape, da bi spletnim programerjem pomagal pri ustvarjanju interaktivnih spletnih strani.
Jezik je bil razvit neodvisno od Jave, vendar si z njo deli številne lastnosti in strukture. JavaScript lahko sodeluje s HTML-kodo in s tem poživi stran z dinamičnim izvajanjem. JavaScript podpirajo velika programska podjetja in kot odprt jezik ga lahko uporablja vsakdo, ne da bi pri tem potreboval licenco. Podpirajo ga vsi novejši spletni brskalniki. [18]
JavaScript je skozi svoj obstoj zamenjala veliko imen, od Mocha do LiveScript. Ime pod katerim ga poznamo danes je dobil pod okriljem Sun, predvsem zaradi markentiških razlog, saj je takrat Java bila zelo popularna. Protokol WebRTC je osnovan na programskem jeziku JavaScript.
Skozi svoj obstoj se je JavaScript prilagajala novim trendom v svetu računalniška. Razvoj nove platforme kot je Node.js, ki bo predstavljen v nadaljevanju. Node.js nam omogoča uporabo JavaScript na strežniški strani in API-jev HTML5 za nadzor uporabniških medijev, odpiranja spletnih vtičev za neprestano komunikacijo, pridobivanje geografske lokacije in še veliko tega. JavaScript pripomore k hitrejšemu in boljšemu izvajanju Express ogrodja, ki je potreben za delovanje Node.js platforme.

Pomembni WebRTC JavaScript API-ji

- MediaStream: predstavlja sinhronizirane tokove (Ang. streams) medijev. Podatke pridobljene iz kamere in mikrofona sinhonizira, da se avdio in video podatki časovno ujemajo.
- MediaRecorder: preprost mehanizem, ki nam omogoča snemanje medijskih tokov iz uporabnikovih vhodnih naprav. Vhodnih podatkov ni potrebno ročno kodirati.
- RTCPeerConnection: vmesnik, ki nam omogoča vzpostavitev povezave med lokalnim računalnikom in oddaljenim odjemalcem. Nudi nam metode, ki nam omogočajo povezavo z oddaljenim odjemalcem, vzdrževanjem in nadzorovanjem povezave ter prekinitvijo povezave.

- RTCDataChannel: vmesnik, ki nam ustvari podatkovni kanal. Uporabljamo ga lahko za peer-to-peer dvosmerni prenos poljubnih podatkov [4].

3.2 Express

Express je minimalistično in prilagodljivo spletno ogrodje, ali angleško framework, za delo z Node.js. Zagotavlja učinkovit nabor funkcij za spletne in mobilne aplikacije. Z nešteto uporabnimi metodi za delo z HTTP protokolom in vmesno programsko opremo. Zagotavlja hiter in enostaven programski vmesnik (API). Vse kar Express ponudi so temeljne funkcije za spletne aplikacije, ne da bi kakorkoli vplival na delovanje Node.js platforme. [16]

3.3 WebSocket

WebSocket je računalniški komunikacijski protokol, ki zagotavlja polno dvosmerno povezavo komunikacijskega kanala, preko ene TCP povezave.
WebSocket je zasnovan tako, da se lahko izvaja v spletnih brskalnikih in spletnih strežnikih. Protokol WebSocket je neodvisni protokol, ki temelji na TCP protokolu. Omogoča večjo interakcijo med brskalnikom in morebitnim spletnim strežnikom ter olajša prenos podatkov v realnem času iz in na strežnik. S standardiziranim načinom komunikacije strežnika s spletnim brskalnikom, kjer strežnik ne čaka na sporočila s strani brskalnika, omogoča da se sporočila pošiljajo v obe smeri in pri čemer povezava ostaja odprta. [19]

3.4 JSON

JSON - JavaScript Object Notation je odprt standardni format, ki uporablja človeku berljiv tekst za prenos podatkovnih objektov oblike atribut-vrednost. To je najbolj pogosta oblika podatkov, ki se uporablja za asihrono komunikacijo brskalnik-strežnik in v veliki meri nadomešča XML format. Spodaj je prikazan kratek primer JSON sporočila, ki se uporabalja pri kurento medijskem strežniku. V sporočilu imamo podane informacije o imenu in verziji projekta ter o dodatnih paketih, ki jih projekt potrebuje za svoje delovanje [22].

```
{
  "name": "kurento",
  "version": "6.6.1-dev",
  "private": true,
  "scripts": {
```

```
    "postinstall": "cd static && bower install"
  },
  "dependencies": {
    "express": "~4.12.4",
    "minimist": "^1.1.1",
    "ws": "~1.0.1",
    "kurento-client": "Kurento/kurento-client-js"
  },
  "devDependencies": {
    "bower": "^1.4.1"
  }
}
```

3.5 Node.js

Node.js je ogrodje na strani strežnika, ki temelji na "Google Chrome JavaScript Engine", za enostavno gradnjo hitre in razširljive mrežne aplikacije. Node.js uporablja način komuniciranja tako, da vsak korak komunikacije tretira kot dogodek. Zaradi njegovega načina delovanja ne zaustavlja I/O dogodke in s tem posledično omogoča lahek in učinkovit sistem, ki je kot nalašč za podatkovno intenzivne aplikacije v realnem času. S pomočjo Node.js komuniciramo med dvema uporabnikoma in vzpostavimo komunikacijo. [17]
Node.js je odprtokodna platforma, ki jo lahko uporabljamo v več različnih računalniških okoljih. Njegove lastnosti lahko vključimo v razvoj mrežnih aplikacij in za aplikacije na strežniški strani. Node.js je napisan v programskem jeziku JavaScript, torej se ga lahko izvaja na večini operacijskih sistemov, kot so Microsoft Windows, OS X in Linux. [17]
Node.js je zelo priljubljen med uporabniki, saj nam ponuja naslednje značilnosti:

- Asinhronost in dogodki
  Vsi API za Node.js knjižnice so asinhrone, kar pomeni, da ni nobenega blokiranja prometa. Zaradi takšnega načina delovanja, Node.js nikoli ne čaka povratnih podatkov od API-jev. Strežnik se preprosto premakne na naslednji API in preko mehanizma za obveščanje dogodkov čaka na odgovor prejšnjega API-ja. [17]

- Hitrost
  Zaradi uporabe Google Chrome JavaScript je Node.js knjižnica zelo hitra pri izvajanju kode.

- Eno nitni način delovanja, vendar stopnjevano prilagodljiv
  Node.js uporablja model eno nitnega načina delovanja z uporabo dogodka v zanki.

Mehanizem dogodka pomaga strežniku, da se odzove brez blokiranja in s tem posledično naredi strežnik zelo prilagodljiv v primerjavi s standardnim strežnikom, ki ustvarijo omejeno število niti za obravnavanje prošenj.

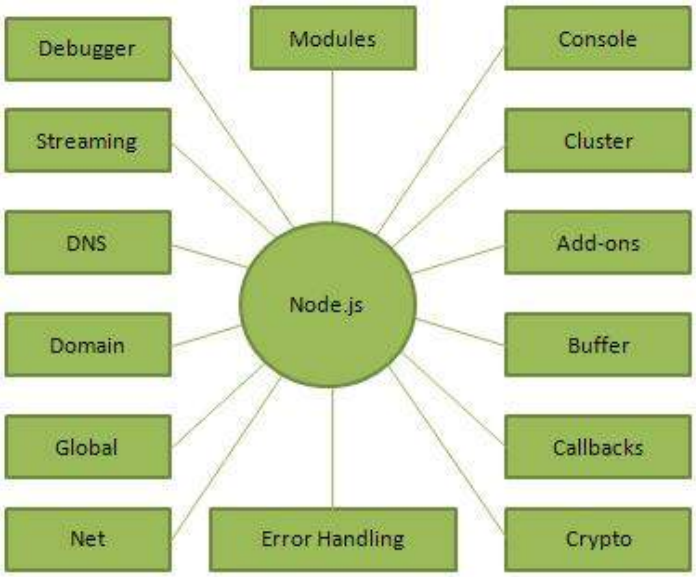
- Brez vmesnega shranjevanja
  Node.js aplikacije nikoli ne uporabljajo vmesnega shranjevanja. Aplikacije preprosto vračajo podatke v koščkih.

Kdo uporablja Node.js

Node.js je zelo razširjen v svetu računalništva. Zaradi zgoraj naštetih sposobnostih je Node.js implementiran v sistemih, ki jih upobljajo korporacije kot so eBay, Microsoft, PayPal, Uber, Yahoo! in veliko drugih.

Koncept

Njegov koncept je preprosta uporaba in hitrost izvajanja ukazov. Na spodnji sliki lahko vidimo vse možne razširitve, ki jih lahko izkoristimo. Na sliki lahko vidimo sposobno-



Slika 5: Pomembni moduli Node.js [17].

sti, ki nam jih Node.js ponuja. Dodatno lahko dodamo podprte module in razširimo njegovo delovanje. Optimalno uporablja medpolnilnik in poslednično odlično razvršča API klice. Ima zelo dober razhroščevalnik, s katerim lahko najdemo napake v našem sistemu. Njegovo delovanje sloni na "callbacks"klicih, kar pomeni, da pričakuje odgovor API-jev. Zelo dobro se spopada s pretočnimi vsebinami v realnem času. Node.js

se uporablja na strežniški strani, zato zelo dobro uporablja protokol DNS in protokole, ki so potrebni za delovanje v spletu. [17]

Kje uporabljati Node.js?

V navezah z naslednjimi področji se Node.js izkaže kot dober partner.

- I/O aplikacijah
- Pretočnih vsebinah
- Podatkovno intenzivnih aplikacijah v realnem času
- Aplikacijah, ki so grajene z JSON API-jem
- Aplikacije, ki za svoje delovanje uporabljajo zgolj eno spletno stran

Kje ne uporabljati Node.js?

Node.js ni priporočljivo uporabljati pri aplikacijah, ki so zelo požrešne za CPU. Node.js lahko uporabljamo kot samostojno okolje za strežniške zahteve, vendar je obdelava slike za Node.js prezahtevna. Zaradi tega bom za oddaljeni nadzor uporabil Kurento medjiski strežnik, ker je v tem primeru lahko potrebna obdelava podatkov (slik) v realnem času, kar je procesorsko zahtevno. [17]

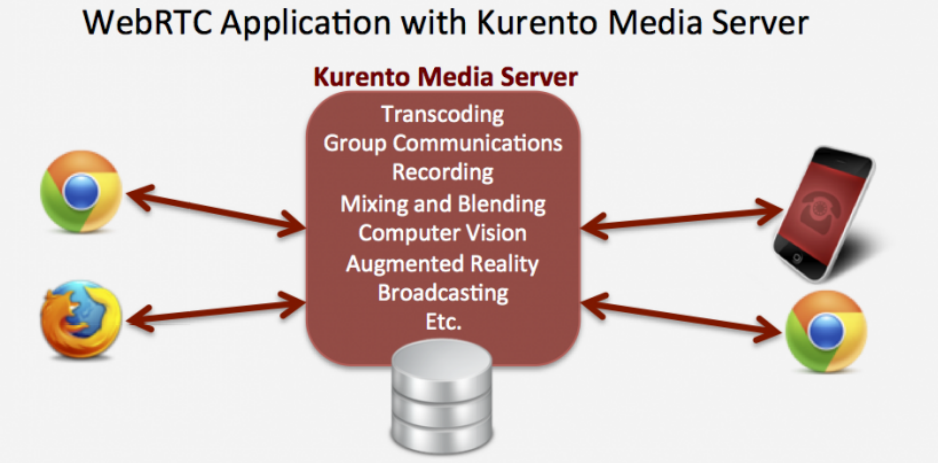
3.6 Bower

Sledenje vsem komponentam aplikacije (ogrodja, knjižnice, itd.) in zagotoviti, da bodo vse komponente med seboj pravilno nameščene, je zelo zapleten postopek. Tukaj nastopi na pomoč Bower.
Bower lahko upravlja s paketi, ki vsebujejo HTML, CSS, JavaScript, različne pisave in celo slikovne datoteke. Bower ne združuje ali spreminja kode, preprosto namesti pravilne različice paketov, ki jih potrebujemo in njihovo medsebojno odvisnost. Bower deluje na način pridobivanja in nameščanja paketov tako, da nam sam poišče, prenaša in shranjuje potrebne pakete za delovanje programa. Informacije o paketih so shranjene kot manifest v datoteki bower.json. Bower najde potrebne pakete, ki jih je kasneje potrebno pravilno združiti. Bower potrebujemo za pravilno delovanje Kurento medijskega serverja. [3]

4 Predstavitev Kurento medijskega strežnika

Kurento je odprtokodna programska oprema napisana v programskem jeziku C/C++ in sestoji se iz različnih modulov, ki skupaj sestavljajo medijski strežnik. Kurento je WebRTC medijski strežnik z naborom API-jev za preprost razvoj naprednih video aplikacij za svetovni splet in pametne platforme. Lastnosti Kurento medijskega strežnika vključujejo skupinsko komuniciranje, kodiranje, snemanje, mešanje, oddajanje in usmerjanje avdiovizualnih tokov. Kurento medisjki strežnik je komponenta, ki spremeni običajni način povezovanja WebRTC-ja med vozlišči. Namenjen je vmesnim operacijam nad medijskimi tokovi, ki se izvedejo na prenosni poti. Tradicionalno enotno povezavo med vozliščema preusmerimo preko medijskega strežnika.
Ob zagonu aplikacije oddaljenega nadzora, bo stekla povezava z medijskim strežnikom in ustvarila IP naslov na katerega se lahko uporabniki povežejo. Vsi podatki bodo sedaj šli preko medijskega strežnika. Osnovna ideja WebRTC protokola je uporaba uporabnikove računalniške moči pri dodatni obdelavi video vsebin. S tem WebRTC prokotol olajša delo razvijalcem aplikacij. Vsa povezava in pretok vsebine, se pri protokolu WebRTC, odvija na uporabnikovih računalnikih. Takšen način uporabe WebRTC protokola je smiselen, kadar je obdelava pretočnih vsebin minimalna. Pri sistemu oddaljenega nadzorja pa potrebujemo dodatno računalniško moč in s tem posledično potrebujemo strežnik/računalnik med uporabniki, ki nam bo takšne obdelave, brez motenj omogočil.
Razvoj Kurento medijskega strežnika se je začel leta 2010 na univerzi v Madridu. Projekt je imel cilj zagotoviti celovito rešitev na področju multimedijske komunikacije med sistemi v realnem času. Odprtokodna programska oprema prinaša hitro rastoč sistem, ki ga redno posodabljajo raziskovalci, posamezni uporabniki, podjetja in celo končni uporabniki iz celega sveta. [8]
Poleg že naštetih funkcionalnosti, Kurento medijski strežnik omogoča tudi napredne rešitve, ki vključujejo računalniški vid, video indeksiranje, navidezno resničnost in analizo govora, kar je prikazano na sliki 6. Modularnost Kurento arhitekture omogoča enostavno integracijo algoritmov iz drugih virov, poleg uporabe že vgrajenih funkcij. [7]
Multimedijska komunikacija v realnem času je že v osnovni zelo kompleksen problem,

kjer se je zaradi različnih interesov, standardov, tehnologij in vizije, tekmovalo in usmerjalo v različne smeri. Prihod WebRTC je prinesel konvergenco in odprtost, kar je omogočilo izoblikovanje Kurento medijskega strežnika. Ideja Kurento medijskega strežnika je bila preprosta: ustvariti odprt medijski strežnik, ki temelji na standardih, ki lahko zagotavljajo dovoljšno moč medijske obdelave z uporabo modulov in preprostih API-jev, ki jih tudi začetniki v svetu računalništva razumejo. [9]
Beseda Kurento izhaja iz Esperanto (mednarodni umetni pomožni sporazumevalni jezik), ki v angleškem prevodu pomeni "stream"ali v slovenščini "tok". Beseda se zgleduje po Espranto in ponazarja: enostavnost, odprtost in univerzalnost, kar Kurento tudi želi biti. [9] Kurento medijski strežnik ni del brskalnika, je program, zato ga je po-

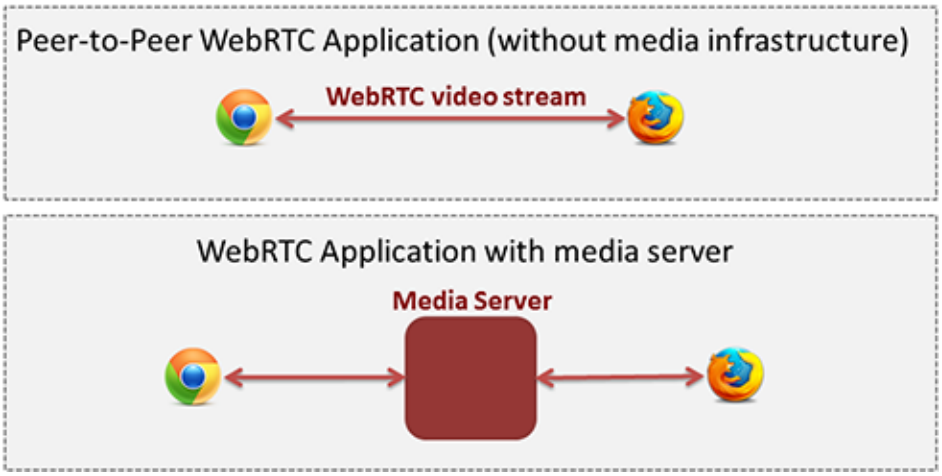


Slika 6: WebRTC aplikacije s Kurento medijskim strežnikom [7].

trebno ločeno namestiti. Kurento medijski strežnik je potrebno namestiti na napravo, ki bo delovala kot strežnik, preko katerega bo potekala komunikacija. Vse medijske povezave bodo potekale preko medijskega strežnika in ne več neposredno med končnimi napravami. Po namestitvi Kurento medijskega strežnika (podrobneje opisano v naslednjem poglavju) je potrebno strežnik zagnati, ki bo deloval do preklica ali posodobitve strežnika z novimi moduli.

4.1 WebRTC medijski strežniki

WebRTC je zasnovan kot peer-to-peer tehnologija, kjer ne potrebujemo posredovanja kakršne koli druge infrastrukture. Zasnovan model je dovolj za ustvarjanje osnovnih aplikacij, če pa želimo uporabiti naprednejše funkcionalnosti kot je naprimer manipulacija video vsebine, potrebujemo vmesno infrastrukturo. V takšnih primerih nam priskoči na pomoč medijski strežnik [10]. Razlika je grafično prikazana na sliki 7.



Slika 7: P2P WebRTC pristop proti WebRTC preko medijskega strežnika. [10].

Konceptualno je medijski strežnik neke vrste "multimedijski vmesnik"(vmesnik vrinjen v komunikacijo), kjer medijski podatki potujejo od vira do destinacije. Medijski strežniki lahko predelajo medijske podatke in ponujajo različne možnosti izbolšave prenosa medijskih podatkov. Medijski strežnik nam omogoči obdelavo video posnetkov, skupinsko komunikacijo, prekodiranje, snemanje, itd.

4.2 Kurento medijski strežnik

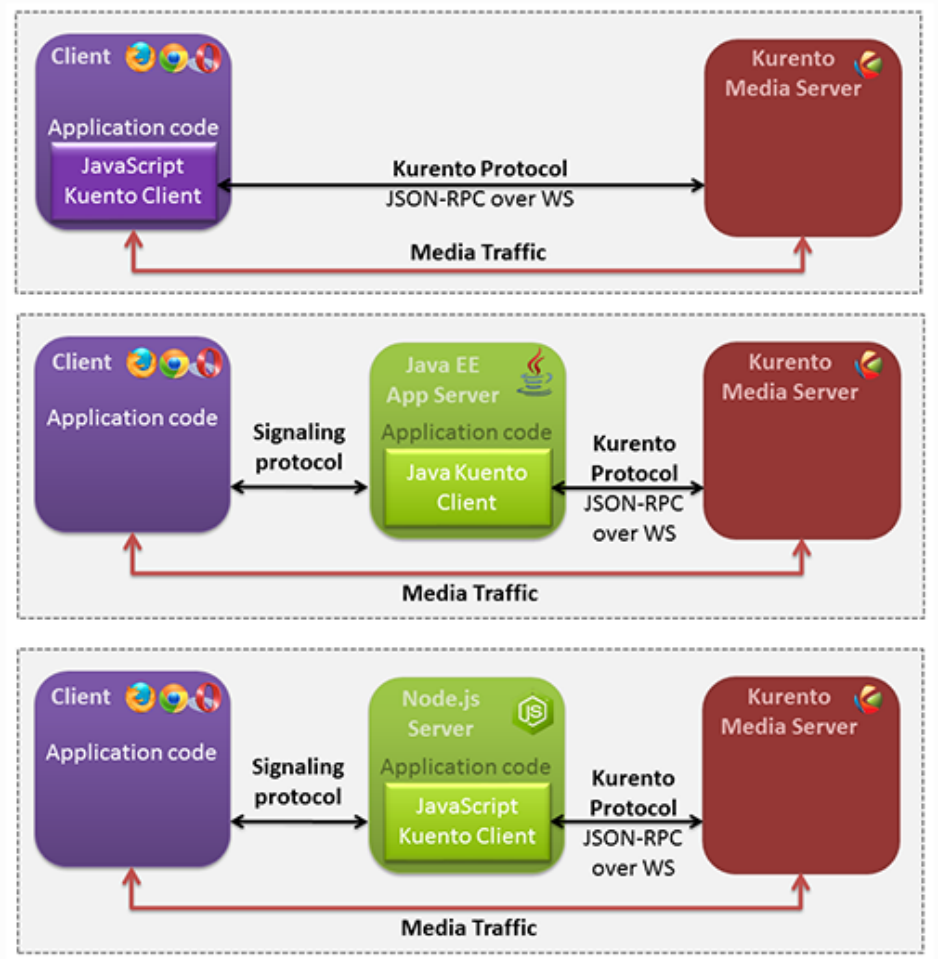
V jedru Kurento arhitekture je medijski strežnik z imenom Kurento Media Server (KMS). Funkcionalnost Kurento arhitetkture lahko obogatimo z dodajanjem in odstavnjevanjem modulov. Vsak uporabnik lahko razvije svoj modul ter ga priključi k medijskem strežniku. S takšnim načinom delovanja lahko dinamično dodajamo nove funkcionalnosti.
Osnovne funkcionalnosti Kurento medijskega strežnika so skupinska komunikacija, prekodiranje, snemanje, alfa korekcije in predvajanje medijskih vsebin. Poleg tega nam ponujajo demonstacijske module, s pomočjo katerih lahko razvijamo dodatne module za svoje potrebe, in s tem strežnik še dodatno nadgradimo. Na voljo imamo napredne module za medijsko obdelavo, vključno z računalniškim vidom, razširjeno resničnost, itd. [10].

4.3 Kurento API, Odjemalci in Protokol

Kurento medijski strežnik lahko izboljšamo in konfiguriramo s pomočjo Kurento API-jev, ki jih razvijalci lahko uporabljajo za svoje aplikacije. API-ji se izvajajo s pomočjo

knjižnic imenovanih Kurento Clients. Kurento ponuja dve možnosti uporabe njihovih API-jev. Prvi s programskim jezikom Java, drugi JavaScript. V primeru, da želimo uporabiti drugi programski jezik, uporabimo Kurento Protokol, ki nam omogoča nadzor nad Kurento medijskim strežnikom in ga implementiramo z našim programskim jezikom. Glede na izbiro našega načina uporabe Kurento medijskega strežnika lahko (Glej sliko 8): [10]

- Uporabimo Kurento JavaScript Client neposredno z brskalnikom, ki podpira WebRTC protokol
- Uporabimo Kurento Java Client v kombinaciji z Java EE aplikacijskim strežnikom
- Uporabimo Kurento JavaScript Client z Node.js strežnikom (slednjega sem uporabil pri zaključni nalogi)



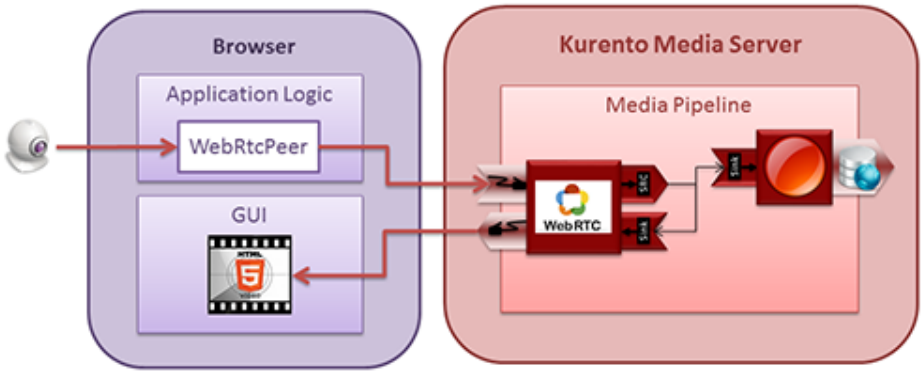
Slika 8: Možne povezave s Kurento medijskem strežnikom. [10].

Kurento Client API temelji na konceptu medijskega elementa (Media Element). Je funkcionalna enota, ki opravlja določeno dejanje nad medijskimi podatki. Medijski

element vsebuje specifične medijske sposobnosti, ki so predstavljene kot samostojni elementi, s katerimi se izognemo razumevanju nizkonivojskih podrobnostih pri njihovem delovanju.
Na primer, medijski element WebRtcEndpoint ima sposobnosti za pošiljanje in prejemanje WebRTC medijskih podatkov. Imamo tudi medijski element RecorderEndpoint, ki nam omogoča snemanje medijskiih podatkov in pa medijski element FaceOverlayFilter, ki zaznava obraz in obdela video vsebino, tako da doda predefinirano sliko preko videa. Poleg že naštetih nam Kurento medijski server ponuja tudi druge zanimive medijske elemente. [10]

4.4 Aplikacije s Kurento medijskim strežnikom

Z vidika razvijanja aplikacij, so medijski elementi kot Lego kocke. Potrebno je zgolj vzeti željene elemente in jih potem pravilno med seboj povezati v celoto. V Kurento medijskem strežniku se graf povezanih medijskih elementov imenuje Media Pipeline. Za primer vzemimo aplikacijo, katera snema WebRTC pretok vsebin. Za medijske elemente bomo potrebovali WebRtcEndpoint in RecorderEndpoint. Pri povezovanju uporabnika, bo potrebno sprva aktivirati WebRtcEndpoint, ki bo nato podatke poslal naprej do RecorderEndpoint. [10]
Da bi poenostavili uporabo WebRTC komunikacije na strani odjemalca, Kurento zagotavlja pripomoček, ki se imenuje WebRtcPeer. WebRtcPeer povezuje standardne WebRTC API-je kot so UserMedia, RTCPeerConnection, itd.. S tem pripomočkom je možno povezati WebRtcEndpoint z standardnimi WebRTC API-ji. [10]

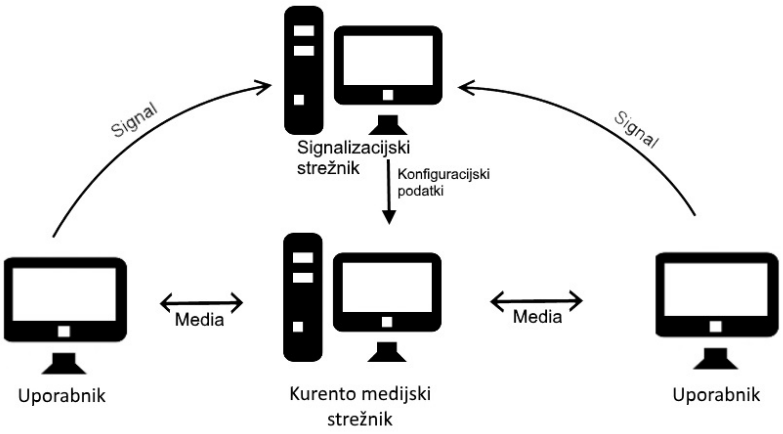


Slika 9: Delovanje in povezovanje aplikacije z Kurento medijskim strežnikom [10].

5 Izvedba sistema za oddaljeni nadzor

5.1 Ideja delovanja sistema

Sistem za oddaljeni nadzor je aplikacija, ki za grafični vmesnik izkorišča spletno tehnologijo preko spletnega brskalnika. Sistem deluje s pomočjo medijskega strežnika, ki pa je pred uporabnikovimi očmi skrit. Sistem deluje po protokolu WebRTC. Vse kar uporabnik potrebuje je speltni brskalnik in s tem posledničnо ni potrebe po dodatni programski opremi. WebRTC protokol temelji na povezavi P2P (peer-to-peer),



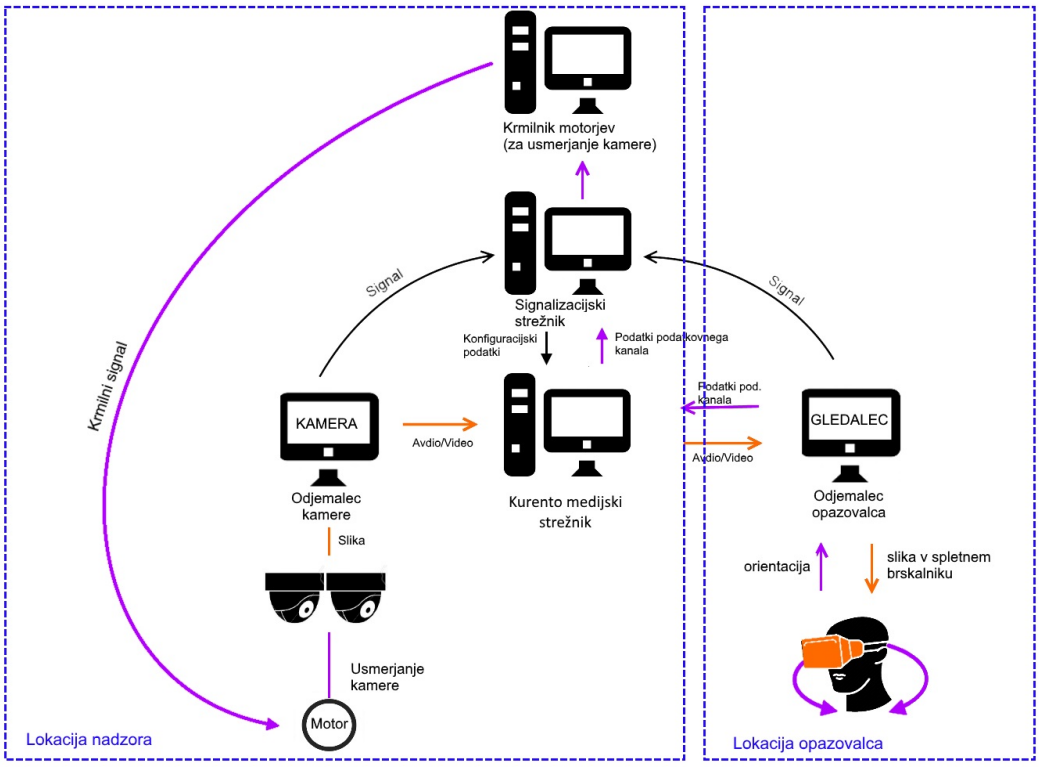
Slika 10: Ilustracija ideje delovanja sistema za oddaljen nadzor.

ker pa bo naš sistem ponujal dodatno obdelavo medijskih podatkov je med povezavo, uporabnikom in aplikacijo vrinjen Kurento medijski strežnik. Strežnika uporabniki ne zaznajo, je zgolj vmesni del za predelavo medijskih podatkov. Komunikacija prinaša neposredno povezavo med uporabniki in s tem posledničnо hitrejši in bolj učinkovit način komuniciranja. Zastavili smo si cilj, da bomo vzpostavili sistem za oddaljen nadzor v realnem času (glej sliko 10). Potrebno je pripraviti okolje v katerem bo lahko naš

sistem deloval.

5.2 Arhitektura sistema

Sistem sestavljata dve kameri, ki delujejo v stereo načinu, te so povezane na strežnik kamor pošiljajo video vsebino. Uporabnik se lahko poveže na strežnik in tako dostopa do video vsebine. Za pristenejšo izkušnjo lahko uporabi Google Cardboard očala za navidezno resničnost. S pomočjo očal in mobilne naprave tako spremljamo premik glave uporabnika. Podatki o gibanju se pošiljajo na strežnik preko podatkovnega kanala. Na strežniku, v ozadju deluje program, ki sprejema UDP pakete s strani strežnika. Ta posreduje informacije naprej do programa v ozadju, program nato pošlje informacije kamerama, te se posledično obrneta glede na pridobljene informacije. Prikaz delovanja sistema in toka podatkov je prikazano na sliki 11.



Slika 11: Ilustracija arhitekture sistema za oddaljen nadzor.

5.2.1 Razširitev kamer v stereo način

Sistem želimo obogati z uporabo navidezne resničnosti, zato potrebujemo dve kameri, ki bosta delovali v stereo načinu. Na strani uporabnika Kamera bi bile postavljene

dve kameri, ki bi oddajale sliko v živo. Sliki obeh kamer je zato potrebno uskladiti in sinhonizirati med seboj. Na voljo imamo več možnosti kako lahko to storimo. Kurento medijski strežnik nam omogoča prikazovanje video vsebine iz več kamer hkrati kar nam olajša delo pri signalizaciji. Pri postopku signalizacije tako ni potrebe po dodatni signalizaciji za dodatno kamero. Medijske tokove pridobljene iz kamer je potrebno na strežniku prilagoditi navidezni resničnosti. Odločimo se lahko, da bomo pošiljali vsak medijski tok posebej naprej do opazovalca ali pa bomo medijske podatke pridobljene iz kamer združili v en medijski tok, ga obdelali in poslali naprej do uporabnika. Slednji način je bolj primer, saj za obdelavo in sihronizacijo uporabimo kar medijski strežnik. Uporabniku tako pošljemo na novo sestavljene in obdelane medijske podatke. Pri realizaciji slednjega bomo uporabili Hub - medijski element Kurento medijskega strežnika. Hub je medijski objekt, ki je odgovoren za upravljanje večih medijskih tokov v tekoči povezavi. Hub (razdelilnik) nam ponuja več priključnih mest, kamor so povezane medijske vsebine. Kurento medijski strežnik nam nudi sledeče Hub-e:

- Composite je razdelilnik, ki meša zvočni tok podatkov vhodnih povezav in ustvari mrežo videov vhodnih tokov.
- DispatcherOneToMany je razdelilnik, ki pošlje vhodni podatek vsem povezanim na Hub vrata.
- Dispatcher je razdelilnik, ki omogoča samovoljno usmerjanje vhodno-izhodnih parov na Hub vratih.

Za naš primer združevanja video vhodnih tokov podatkov v edinstven video izhodni tok podatkov, bi tako uporabili Composite [14].

5.3 Priprava programskega okolja

Vsak sistem zahteva programsko okolje v katerem se bo lahko izvajalo. Zaradi tega je potrebno namestiti programsko opremo, ki nam bo omogočila delovanje našega sistema. Naš sistem bo temeljil in uporabljal funkcionalnosti Kurento medijskega strežnika, zato bo to prva stvar, ki jo bomo nastavili. Namestitev je možna v Linux okolju, kjer potrebujemo vpisati nekaj ukazov v ukazno vrstico terminala. [12]

5.3.1 Strežniška platforma

Strežniška platforma je okolje/program, ki nam nudi strežniške storitve. S storitvami lahko uporabnikom omogočimo uporabo aplikacij, kot je v zaključni nalogi sistem za oddaljen nadzor v realnem času. Za potrebe naloge bomo uporabili Kurento medijski

strežnik.
Kurento medijski strežnik deluje zgolj v okolju Linux. Najboljše je podprta distribucija Ubuntu 14.04 LTS. Pri namestitvi je potrebno dodati pot namestitvenih datotek v Ubuntu repozitorij. To naredimo na sledeči način: [11]:

```
echo "deb http://ubuntu.kurento.org trusty kms6" | sudo tee
    /etc/apt/sources.list.d/kurento.list
wget -O - http://ubuntu.kurento.org/kurento.gpg.key | sudo apt-key add -
sudo apt-get update
sudo apt-get install kurento-media-server-6.0
```

Medijski strežnik je nameščen in lahko ga zaženo z naslednjim ukazom:

```
sudo service kurento-media-server-6.0 start
```

Kurento medijski strežnik je lahko lociran za NAT sistemom. V tem primeru je potrebno temu primerno nastaviti STUN in TURN strežnik.
Sedaj imamo strežniško platformo pripravljeno, zato lahko nadaljujemo z nastavitvijo strežniškega izvajalnega okolja. [11]

5.3.2 Strežniško izvajalno okolje

Strežniško izvajalno okolje (prikazano na sliki 10) je okolje na strani strežnika, ki nam omogoča izvajanje spletnih aplikacij. V navezi z Kurento medijskim strežnikom bomo za potrebe naloge uporabili Node.js in Bower. S pomočjo Node.js se vzpostavi signalizacija med dvema vozliščema. Lahko ga namestimo kar iz Ubuntu repositorija, medtem kot Bower namestimo z NPM ukazom:

```
curl -sL https://deb.nodesource.com/setup | sudo bash -
sudo apt-get install -y nodejs
sudo npm install -g bower
```

Npm je ukaz, ki nam pomaga pri namestitvi paketov in njihovim odvisnostnih za programski jezik JavaScript. [11]
Z naslednjim ukazom pridobljene pakete namestimo.

```
npm install
```

NPM nam bo namestil, povezal in združil vse lokalne datoteke, ki se nahajajo v direktoriju node_modules. Celotno aplikacijo pa lahko zaženemo z ukazom:

```
npm start
```

5.4 Razumevanje problema

V sistemu za oddaljeni nadzor imamo dve vrsti uporabnikov. Prvi uporabnik je tisti, ki pošilja sliko (recimo mu Kamera) in drugi uporabnik je tisti, ki video sprejme (recimo mu Gledalec). Glede na razpored uporabnikov je Media Pipeline sestavljen iz Kamere in enega Gledalca (1+1) ali ene Kamere in N Gledalcev (1+N). Kamera in Gledalec sta medsebojno povezana z WebRtcEndpoint. Za izvajanje aplikacije je potrebno vzpostaviti Media Pipeline z WebRtcEndpoint na način 1+1. Uporabnik Kamera pošlje podatke, ki so potem na voljo Gledalcu za sprejem.
Sistem za oddaljen nadzor je spletna aplikacija, zato sledi arhitekturi odjemalec-strežnik. Na strani odjemalca se logika izvaja v JavaScript. Na strani strežnika se pa uporablja Kurento JavaScript Client, ki nam omogoča povezavo s Kurento medijskim strežnikom (KMS). Kurento JavaScript Client je zgolj poimenovanje modula na strani Kurento medijska strežnika s pomočjo katera lahko vzpostavimo povezavo. Skupna logika sistema je trislojna. Za komunikacijo med entitetami potrebujemo dva WebSocket-a. Prva komunikacija poteka med odjemalcem in Node.js strežnikom za potrebe signalizacije. Drugi nivo komunikacije se odvija med Kurento JavaScript Client, ki vrši ukaze nad Node.js in Kurento medijskim strežnikom. Da pa stvar v celoti deluje poskrbi Kurento Protocol, kateri komunicira z Kurento medijskim strežnikom in je baziran na WebSocket-u. [11]
Gledalec in strežnik aplikacije komunicirata preko signalizacijskega protokola, ki temelji na sporočilih JSON preko Kurento WebSocket-ov.

Normalna signalizacija med Gledalcem in strežnikom poteka tako:

1. Kamera(pošiljatelj) vstopi v sistem. Obstajati mora samo en uporabnik Kamera (ki bi potencialno lahko imel priklopljeni dve kameri) v celotni trenutni seji. Če je Kamera že prisotna, potem vsak naslednji, ki želi pristopit dobi sporočilo o napaki.

2. Gledalec se poveže s Kamero. Če ni prisotne Kamere, se pošlje sporočilo o napaki.

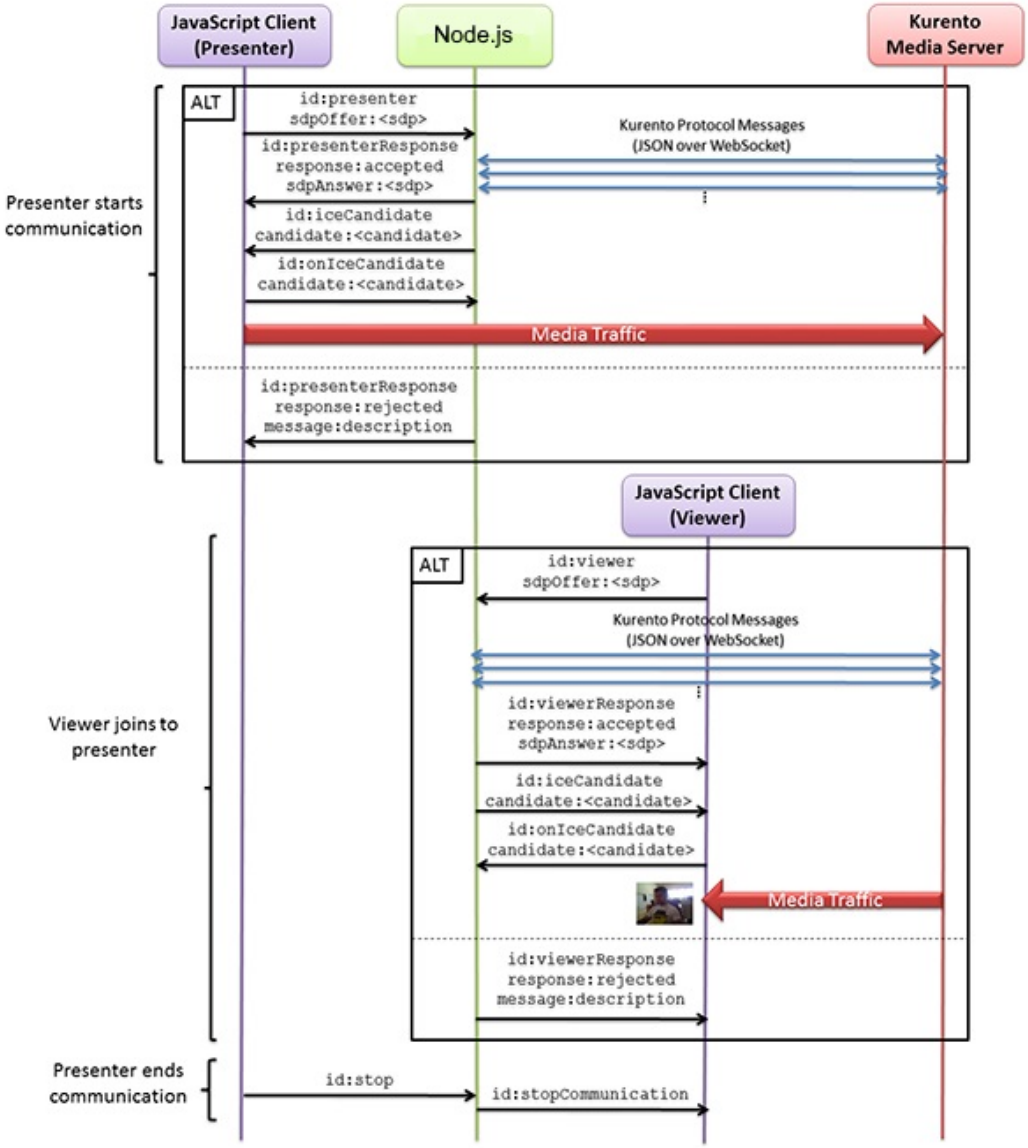
3. Gledalec lahko zapusti sejo kadarkoli.

4. Ko Kamera zapusti sejo, se zaprejo vse povezave z Gledalci.

Kot lahko vidimo na sliki, je potrebna izmenjava SDP in ICE kandidatov med odjemalcem in strežnikom za vzpostavitev povezave na Kurento medijskem strežniku.

5.4.1 Delovanje strežnika

Logika strežnika aplikacije je realizirana s pomočjo Express ogrodja za Node.js okolje. Za pošiljanje sporočil med JavaScript odjemalcem in Node aplikacijo se uporablja



Slika 12: Sekvenčni diagram signalizacije med Kamero(Presenter) in Gledalcem(Viewer) s pomočjo Node.js in KMS [11].

WebSocket. Dohodna sporočila v WebSocket (spremenljivka ws v kodi) se uporabljajo s protokolom signalizacije med Gledalcem in Kamero. Pri signalizaciji se prenašajo sporočila kot so message, stop, error, close, text, control in onIceCandidate. Vsaka od sporočil sproži določeno dejanje, ki je povezano s sporočilom [11].
Node.js bo služil trem namenom kot:

- signalizacijski strežnik,
- aplikacijski strežnik,
- posrednik podatkov podatkvonega toka.

Za nadzor nad medijskimi zmogljivostmi, ki nam jih ponuja Kurento medijski strežnik, potrebujemo primerek KMS objekta "KurentoClient" v Node.js aplikaciji. Pri ustvarjanju "KurentoClient", moramo določiti lokacijo Kurento medijskega strežnika. V primeru našega sistema je to kar localhost na vratih 8888. Uporabniki pa se povezujejo na IP naslov localhost-a na vratih 8443 [11].
Enkrat, ko je Kurento Client inicializiran, lahko steče komunikacija z Kurento medijskim strežnikom. Prva operacija je ustvariti Media Pipeline in medijske elemente, ter jih medseboj povezati. V našem sistemu potrebujemo en WebRtcEndpoint za Kamero in enega za Gledalca. S tem omogočimo pretok podatkov iz enega vozlišča v drugega. Fukncije se kličejo takrat, ko se izvede klic za prošnjo po Kameri ali Gledalcu.
Kurento medijski strežnik za pogajanje uporablja ICE kandidate. Tej so izbrani na podlagi primernosti za vzpostavitev povezave. WebRtcEndpoint pridobi vse kandidate s pomočjo ICE. Pridobljene kandidate shrani za morebitno kasnejšo uporabo. Ob izbiri primernega kandidata, se ga doda k pripadajočemu medijskemu elementu z klicem addIceCandidate [11].

5.4.2 Delovanje na strani odjemalca

Brez odjemalca, ki koristi funkcionalnost strežnika, je sistem oddaljenega nadzora, brez pomena. Odjemalec, v našem primeru Gledalec, se mora povezati s strežnikom in vzpostaviti povezavo, da lahko prične s prenosom slike. To nam omogoča WebSocket, ki se poveže z WebSocket-om na strani strežnika. Kurento medijski strežnik vso stvar še bolj poenostavi z uporabo Kurento JavaScript knjižnic. Kurento-utilis.js je knjižnica, ki nam olajša delo pri WebRTC interakciji s strežnikom. Knjižnica temelji na adapter.js, ki je JavaScript WebRTC pripomoček, vzdrževan s strani Googla. Knjižnica nam omogoča odstranitev vseh preprek glede različnosti spletnih brskalnikov.
V spodnjem odrezku kode lahko vidimo kreiranje WebSocket-a.
Z uporabo "onmessage"funkcije WebSocket-a, implementiramo JSON signalizirni protokol na strani Gledalca. V kodi lahko opazimo, da obstaja 6 dohodnih sporočil k Gledalcu: kameraResponse, gledalecResponse, iceCandidate, stopCommunication, text, control [11].
Pomembna stvar pri komunikaciji, kot je sistem za oddaljen nadzor, je način kako se podatki pošiljajo med odjemalcem in strežnikom. V našem sistemu Kamera uporablja zgolj WebRtcPeerSendonly funkcijo, medtem ko Gledalec uporablja WebRtcPeerRecvonly funkcijo. Pri Kameri omejimo podatke zgolj na pošiljanje, pri Gledalcu pa na sprejemanje.

5.4.3 Uporaba medijskih filtrov

Zaključna naloga potrebuje tudi obdelavo slike v realnem času, s katero prilagodimo sliko potrebam odjemalca. V mislih imamo predvsem implementacije združevanja več slik in radialnega popačenja, ki je potrebno za prikazovalnike virtualne resničnosti. Pomagamo si lahko z dodatnimi moduli, ki nam jih ponuja Kurento medijski strežnik. V praksi bi morali filtre narediti sami glede na naše potrebe. Za namen demonstracije, pa sem uporabil že integrirane filtre. Kurento medijski strežnik nam pri namestitvi ponuja osnovne funkcionalnosti gstreamer programske knjižnice, face overlay in ZBar filtra.
Delovanje filtrov sem prikazal z uporabo face overlay filtra. Filter ima to sposobnost, da zazna obraz, ki je postavljen pred kamero in na njegovo glavo postavi določeno sliko. V mojem primeru je to pokrivalo.
Filter nima pomena za sistem za oddaljeni nadzor, vendar sem z njim hotel pokazati možnost uporabe filtrov za rešitev zgoraj navedenih problemov pri uporabi sistema za prikazovalnike virtualne resničnosti.

5.4.4 Premikanje oddaljene kamere

Pristnost uporabe sistema za oddaljeni nadzor skozi virtualna očala lahko prikažemo le če se oddaljena kamera premika usklajeno glede na uporabnikovo premikanje glave. Ena od možnih implementacij pošiljanja podatkov oddaljeni kameri je uporaba UDP ali TCP paketov. S takim načinom pošiljanja podatkov naletimo na težavo. Težava se pojavi kadar smo povezani na prekinjajočo se brezžično omrežje. Neposlani paketi se zato nabirajo v polnilniku pošiljatelja in se ob prvi priložnosti pošljejo na strežnik. Teh paketov se lahko nabere zelo veliko in strežnik vse te informacije sprejme in posreduje kameri. Kamera vse ukaze izvede v bistveno krajšem času in privede do tega, da se začne kamera sunkovito premikati.
Ena rešitev problema je pošiljanje podatkov o premiku kamere preko WebRTC podatkovnega kanala. Z njim zagotovimo zanesljivo pošiljanje podatkov. Podatki se od odjemalca pošljejo na strežnik, kjer se lokalno, preko UDP paketov, posreduje programskemu modulu za krmiljenje pogonov naprave za usmerjanje kamere. Prejete podatke se potem uporabi za mehanski premik kamere. Druga rešitev bi lahko potencialno bilo časovno žigosanje prejetih paketov.

5.4.5 Delovanje podatkovnega kanala

Pri uporabi sistema za oddaljen nadzor je potrebna tudi komunikacija med odjemalcem in pošiljateljem. Zato je potrebno ustvariti podatkovni kanal, po katerem se bojo

informacije prenašale. S podatkovnim kanalom želimo pridobiti možnost pošiljanja informacij o premiku kamere med dvema vozliščema.
WebRTC podatkovni kanal ustvarimo z vmesnikom RTCDataChannel. Namenjen je predvsem prenosu datotek, deljenju namizja in igranju spletnih iger. Za svoje delovanje potrebuje že delujočo RTCPeerConnection povezavo. Prednost vmesnika je neposredna izmenjava vseh podatkov v realnem času. Ključna značilnost RTCDataChannel je, da podpira prilagodljive vrste podatkov. Zasnovan je bil tako, da posnema dvosmerno funkcionalnost WebSocket-a in podpira nize, dobro znane binarne vrste, kot so Blob, ArrayBuffer in ArrayBufferView. S sposobnostjo uporabe SCTP protokola, nam RTCDataChannel nudi možnost določanja izmenjave podatkov med dvema brskalnikoma. Protokol omogoča večjo prilagodljivost od tradicionalnih protokolov kot sta TCP in UDP.
Razumevanje delovanja podatkovnega kanala lahko razložimo s spodnjo kodo:

```
var peerConnection = new RTCPeerConnection();
var dataChannel =
  peerConnection.createDataChannel("test", dataChannelOptions);
dataChannel.onerror = function (error) {
  console.log("Data Channel Error:", error);
};
dataChannel.onmessage = function (event) {
  console.log("Got Data Channel Message:", event.data);
};
dataChannel.onopen = function () {
  dataChannel.send("Hello World!");
};
dataChannel.onclose = function () {
  console.log("The Data Channel is Closed");
};
```

V izrezku kode, spletni brskalnik ustvari povezavo. Nato se ustvari podatkovni kanal in se ob vzpostavitvi pošlje sporočilo s funkcijo send(). Z onmessage funkcijo pa pridobimo poslano sporočilo. DataChannel objekt lahko ustvarimo na že vzpostavljeni povezavi. Lahko je ustvarjen pred ali po signalizaciji. S pomočjo oznake lahko podatkovne kanale med seboj razlikujemo in jim dodajamo dodatne parametre.

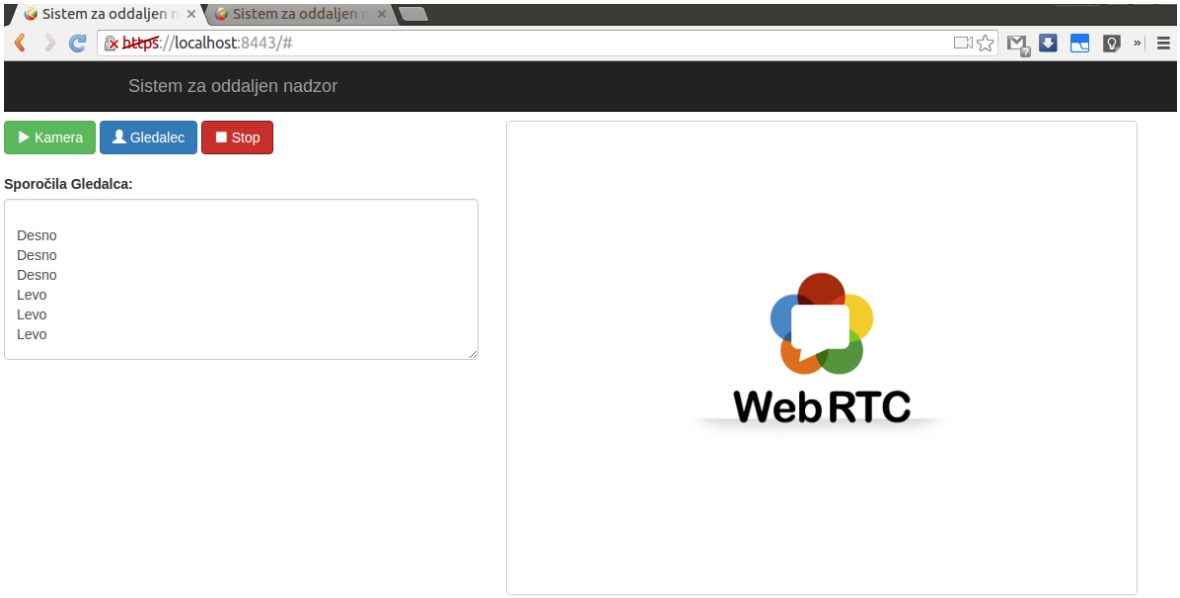
5.5 Uporaba sistema za oddaljen nadzor

Sistem je pripravljen za uporabo, ko so vse komponente in njigove odvisnosti pravilno povezane. Iz uporabnikovega vidika sistem deluje kot spletna aplikacija, ki jo zaženemo preko spletnega brskalnika s pravilnim IP naslovom in vrati. Uporabniku se odpre okno,

kjer ima na izbiro 3 različne gumbe: Kamera, Gledalec, Stop. Glede na izbiro gumba se posledično aktivira dejanje. Ob pritisku na Kamera, uporabnik postane pošiljatelj svoje video vsebine drugim uporabnikom. S pritiskom na gumb Gledalec se uporabnik poveže z aktivnim pošiljateljem video vsebin. V primeru, da ni aktivnega pošiljatelja, uporabnik dobi sporočilo, da nobeden ne oddaja svojih video vsebin. Tretji gumb, Stop, služi za prekinitev procesa, ne glede na to al si povezan kot pošiljatelj ali gledalec.

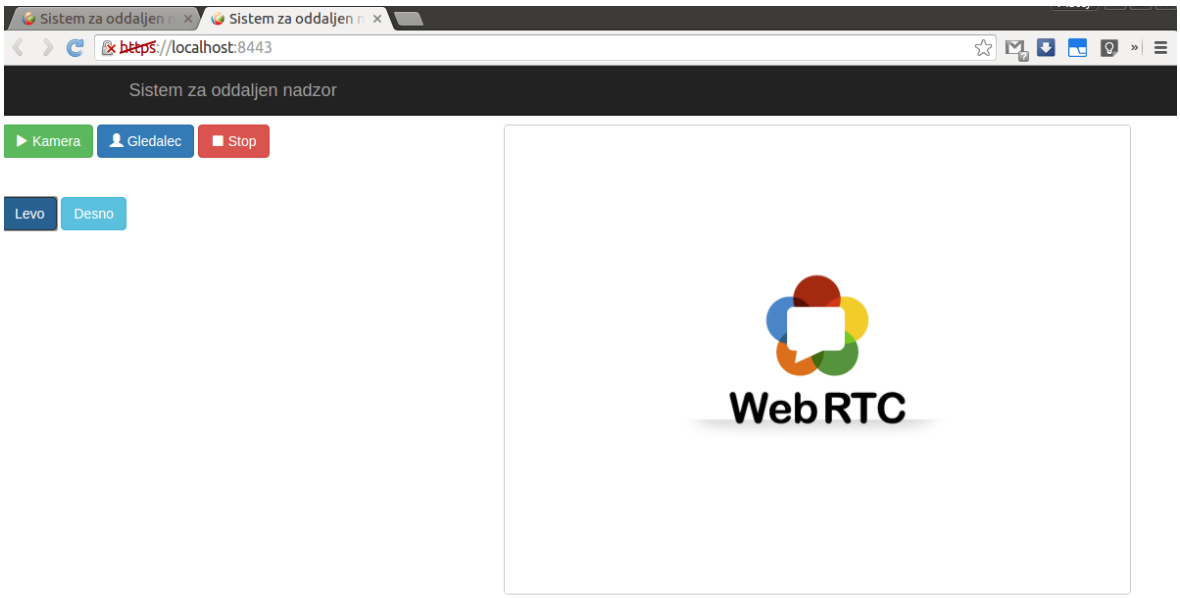
V načinu Kamera se uporabniku odpre novo tekstovno polje, kjer se sporočila s strani Gledalca izpisujejo. Namen sporočil, je simulacija premikanja Gledalca.

V načinu Gledalec se prikažeta dva nova gumba z napisom Levo in Desno. Ob pritisku



Slika 13: Pogled uporabinika Kamera na aplikacijo oddaljenga nadzora.

enega od gumbov se njegova oznaka pošlje preko podatkovnega kanala uporabiku Kamera, kjer se v tekstovnem polju izpiše pritisnjena oznaka gumba. Poleg že omenjenega se dobljene podatke na strežniku lokalno pošilja naprej preko UDP kanala nad katerim bedi program za usmerjanje kamere. S pomočjo virtualnih očal na strani uporabnika in mehanskih posodobitev kamere na strani pošiljatelja, bi lahko sistem nadgradili do točke, kjer bi uporabniku/gledalcu omogočili naraven in pristen vpogled v drug prostor. Uporabnik bi se tako lahko s svojim gibanjem obračal po prostoru, ki bi ga videl skozi očala.



Slika 14: Pogled uporabinika Gledalec na aplikacijo oddaljenga nadzora.

5.6 Razširitev s prilagojenimi Kurento moduli

Sistem za oddaljen nadzor ima veliko prostora za nadgradnjo. Pri zaključni nalogi smo za demonstracijo predlagane rešitve uporabili enega od integriranih modulov, ki jih ponuja Kurento medijski server. Na razpolago pa jih imamo še veliko. Poleg že ponujenih modulov Kurenta medijskega serverja imamo možnost integracije novih modulov. [13].

Uporabimo lahko module, ki so osnovani na OpenCV ali GStreamer. OpenCV je odprtokodna knjižnica z BSD licenco za računalniški vid in strojno učenje programske opreme. OpenCV nam zagotavlja platformo aplikacije računalniškega vida za svoje delovanje. OpenCV module lahko ustvarimo s pomočjo orodja, ki ga ponuja Kurento medijski server.

Najprej, je potrebno pripraviti strukturo modula. Za to nalogo, lahko uporabimo orodje *kurento-module-scaffold*. Orodje nam ustvari datotečno drevo, ki vsebujejo vse potrebne datoteka za uporabo OpenCV modula. Ustvarijo se datoteke s .kmd končnico. Enkrat, ko imamo .kmd datoteke lahko ustvarimo kodo. Uporabimo orodje *kurento-module-creator*, ki nam kodo zgenerira in združi. Ustvarijo se nam štiri datoteke [13]:

```
ModuleNameImpl.cpp
ModuleNameImpl.hpp
ModuleNameOpenCVImpl.cpp
ModuleNameOpenCVImpl.hpp
```

Prve dve datoteke ne smemo spreminjati. Zadnji dve datoteki določata delovanje našega modula. Datoteka ModuleNameOpenCVImpl.cpp vsebuje funkcije za uporabo metod in parametrov. Potrebno je dodatno implementirati postopek obdelave posamezne slike. V datoteki je metoda z imenom process. Klic funkcije se bo izvajala z vsakim prenosom medijskih podatkov preko vzpostavljene povezave. Znotraj te metode je potrebno implementirati postopek obdelave slike.
OpenCV modul lahko uporabimo šele, ko datoteke združimo v celoto in jih namestimo v naš Kurento medijski strežnik. Slednje lahko naredimo na sledeči način:
Ustvarimo lahko Debian pakete

```
debuild -us -uc
```

in nato jih namestimo

```
dpkg -i
```

Možnost imamo sami definirati spremenljivke v datoteki:

```
\etc\default\kurento:
KURENTO_MODULES_PATH= module\build\src
GST_PLUGIN_PATH=module\build\src
```

Na strani strežnika je sedaj vse pripravljeno. Sledi ustvarjanje kode, ki se bo uporabljala v Kurento medijskem strežniku in bo aktivirala svoj učinek na strani odjemalca. Pri tem si pomagamo z naslednjim ukazom

```
cmake .. -DGENERATE_JS_CLIENT_PROJECT= TRUE
```

S tem ukazom ustvarimo *.js* datoteko s kodo odjemalca. Enkrat, ko je *.js* datoteka pripravljena, jo ročno dodamo v direktorij naše aplikacije. Potrebno je še nastaviti odvisnosti med paketi, pri čemer si pomagamo z NPM ukazom, ki nam to omogoči sam [13].
Po končanem postopku se lahko vse OpenCV funckije implementira v našo kodo in jih uporabilja.

6 Zaključek

V zaključni nalogi smo se posvetili uporabi tehnologije WebRTC. Prikazali smo možnosti uporabe pri oddaljenem nadzoru, ki vključuje tako vzpostavitev videopovezave, kot tudi podatkovnega prenosa. Osredotočili smo se na rešitve, ki omogočajo sprotno obdelavo medijskih tokov v smislu obdelave slike. Prikazali smo možne vidike uporabe WebRTC protokola. Opisali smo protokole, programske jezike in progamske vmesnike. Opisali smo celoten postopek delovanja sistema za oddaljen nadzor s pomočjo protokola WebRTC in Kurento medijska strežnika. Med samo izdelavo zaključne naloge smo spoznali nove tehnologije in se bolje spoznali z programskim jezikom JavaScript ter omrežnim povezovanjem naprav. Že na začetku izdelave zaključne naloge smo se spopadali z problemi. WebRTC je novost v svetu računalništva in zato ne podpira veliko različnih operacijskih sistemov in brskalnikov. Zaradi tega se je bilo treba prilagoditi in uporabljati stvari, kjer je bila podpora za WebRTC čim večja.
Spoznavanje s tehnologijo WebRTC je zelo dobrodošla. V prihodnosti bo ta tehnologija imela velik vpliv na spletne kot tudi na namizne aplikacije. Z WebRTC tehnologijo imamo odprta vrata pri razvijanju aplikacij, ki ponujajo prenos podatkov, avdio in video vsebin, brez kakršnih koli omejitev glede programske ali strojne opreme. V kombinaciji z Kurento medijskim strežnikom, postane vse skupaj zelo uporabno in zanimivo za prihodnost. Vedno več poudarka je nad navidezno resničnostjo in življenjem v navideznem svetu. Brez primerne tehnologije v to ne bi bilo izvedljivo, zato ima tukaj veliko prednost WebRTC tehnologija v navezi z Kuretno medijskim strežnikom. Zaradi svoje odprtokodnosti jo popravlja nešteto ljudi po vsem svetu in pripomore k boljšemu jutri za to tehnologijo.
Zaključna naloga ima še veliko prostora za napredek. Ena od dobrodošlih nadgradnj bi bila uporaba dveh kamer v stereo načinu in tako bi naš sistem še obogati. S kamerami v takšen načinu lahko naš sistem uporabimo v navidezni resničnosti. Uporabniki bi sistem za oddaljen nadzor uporabljali s pomočjo VR očal. Meni, da bi bila to zelo zanimiva izkušnja. Sistem bi lahko uporabili npr. v muzejih, kjer bi naš sistem bil postavljen na premikajočega se robota s katerim bi se lahko iz domačega naslonjala premikala po muzejskih sobah in si ogledovali umetnine kar od doma.
Pri sistemu za oddaljeni nadzor bi bilo potrebno ustvariti in uporabiti dodatni OpenCV filter za Kurento medijski strežnik. Uporabili bi ga pri naši stereo kameri, da bi v slike

vnesli geometrijsko popačenje in s tem poizkušali doseči čim pristnejšo uporabniško izkušnjo.

Glede izvedbe zaključne naloge sem zelo zadovoljen, saj sem se naučil nekaj novega in spoznal, da je WebRTC tehnologija, ki jo je dobro znati in bo v prihodnosti zaželjeno poznavanje te tehnologije. Cilj zaključne naloge je bil do dobra spoznati WebRTC tehnologijo skozi razvijanje sistema, ki ni nov, vendar ima dosti prostora za napredek.

7 Literatura in viri

[1] Andrew Prokop, *Avaya*, https://www.avaya.com/blogs/archives/2014/09/writing-your-first-webrtc-application-part-3.html. (Datum ogleda: 20. 5. 2017.) *(Citirano na strani 4.)*

[2] Andrew Prokop, *Avaya*, https://www.avaya.com/blogs/archives/2014/08/introduction-to-webrtc-signaling.html. (Datum ogleda: 20. 5. 2017.) *(Ni citirano.)*

[3] Bower, *Bower*, https://bower.io. (Datum ogleda: 1. 8. 2016.) *(Citirano na strani 17.)*

[4] Codelabs, *Codelabs*, https://codelabs.developers.google.com/codelabs/webrtc-web. (Datum ogleda: 1. 8. 2016.) *(Citirano na straneh 8, 9, 11, 12 in 14.)*

[5] Dan Ristic, *Html5rocks*, https://www.html5rocks.com/en/tutorials/webrtc/datachanne
(Datum ogleda: 25. 5. 2017.) *(Ni citirano.)*

[6] Harald Alvestrand, *Google Groups*, https://groups.google.com/forum/!-topic/discuss-webrtc/I0GqzwfKJfQ. (Datum ogleda: 2. 8. 2016.) *(Citirano na strani 4.)*

[7] Kurento.org, *Kurento.org*, https://www.kurento.org/whats-kurento. (Datum ogleda: 1. 8. 2016.) *(Citirano na straneh 1, 18 in 19.)*

[8] Kurento.org, *Kurento.org*, https://www.kurento.org/about. (Datum ogleda: 1. 8. 2016.) *(Citirano na strani 18.)*

[9] Kurento.org, *Kurento.org*, https://media.readthedocs.org/pdf/doc-kurento-room/latest/doc-kurento-room.pdf. (Datum ogleda: 15. 8. 2016.) *(Citirano na strani 19.)*

[10] Kurento.org, *Kurento.org*, https://doc-kurento.readthedocs.io/en/stable/introducing-kurento.html. (Datum ogleda: 15. 8. 2016.) *(Citirano na straneh 19, 20, 21 in 22.)*

[11] Kurento.org, *Kurento.org*, https://doc-kurento.readthedocs.io/en/stable/tutorials/node/tutorial-one2many.html. (Datum ogleda: 15. 8. 2016.) *(Citirano na straneh 26, 27, 28 in 29.)*

[12] Kurento.org, *Kurento.org*, https://doc-kurento.readthedocs.io/en/stable/installation-guide.html. (Datum ogleda: 15. 8. 2016.) *(Citirano na strani 25.)*

[13] Kurento.org, *Kurento.org*, https://doc-kurento.readthedocs.io/en/stable/mastering/develop-kurento-modules.html. (Datum ogleda: 19. 8. 2016.) *(Citirano na straneh 33 in 34.)*

[14] Kurento.org, *Kurento.org*, https://doc-kurento.readthedocs.io/en/stable/mastering (Datum ogleda: 25. 5. 2017.) *(Citirano na strani 25.)*

[15] Sam Dutton, *Google Groups*, http://www.html5rocks.com/en/tutorials/webrtc/infrastructure/. (Datum ogleda: 1. 8. 2016.) *(Citirano na straneh 7, 10 in 11.)*

[16] StrongLoop, *Express*, https://expressjs.com/. (Datum ogleda: 1. 8. 2016.) *(Citirano na strani 14.)*

[17] Tutorials point, *Tutorials point*, http://www.tutorialspoint.com/nodejs/nodejs-introduction.htm. (Datum ogleda: 1. 8. 2016.) *(Citirano na straneh 15, 16 in 17.)*

[18] Wikipedia, *Wikipedia*, https://sl.wikipedia.org/wiki/JavaScript. (Datum ogleda: 1. 2. 2017.) *(Citirano na strani 13.)*

[19] Wikipedia, *Wikipedia*, https://en.wikipedia.org/wiki/WebSocket. (Datum ogleda: 1. 2. 2017.) *(Citirano na strani 14.)*

[20] Wikipedia, *Wikipedia*, https://en.wikipedia.org/wiki/Interactive-Connectivity-Establishment. (Datum ogleda: 20. 5. 2017.) *(Ni citirano.)*

[21] Wikipedia, *Wikipedia*, https://en.wikipedia.org/wiki/Traversal-Using-Relays-around (Datum ogleda: 20. 5. 2017.) *(Citirano na strani 11.)*

[22] Wikipedia, *Wikipedia*, https://en.wikipedia.org/wiki/JSON. (Datum ogleda: 20. 5. 2017.) *(Citirano na strani 14.)*

Priloge

A Spletni repozitorij

Spletni repozitorij s celotno programsko kodo: https://github.com/MatejZem/Using-WebRTC-technology-for-remote-control.git