TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Stanislav Grebennik 184943IADB

Veebibrauseripõhise heli voogedastuse latentsuse vähendamine klient-server arhitektuuris

Bakalaureusetöö

Juhendaja: Joel Kivi
BSc

Tallinn 2021

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Stanislav Grebennik

16.04.2021

Annotatsioon

Heli voogedastuse latentsuse vähendamine on vajalik nii olemasolevate kõne- ning heliedastusplatvormide arenemiseks, kui ka uute platvormide arendamiseks mille elluviimine on seniks olnud tehniliselt võimatu. Selle töö eesmärk on uurida kõige populaarsemate veebilehitsejate valmidust toetama ülimadala latentsusega helirakendusi. Samuti tähtsaks eesmärgiks on pakkuda lihtne alternatiiv heli voogedastamiseks üle interneti.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 41 leheküljel, 9 peatükki, 16 joonist.

Abstract

Decreasing Latency of Web Browser-Based Digital Audio Streaming in Client-Server Architecture

Reducing the latency of audio streaming is necessary both for the development of existing audio transmission platforms and for the development of new platforms, the implementation of which has so far been technically impossible. The aim of this work is to investigate the readiness of the most popular web browsers to support ultra-low latency audio applications. Another important goal is to propose a simple alternative for audio streaming over the Internet.

The thesis is in Estonian and contains 41 pages of text, 9 chapters, 16 figures.

Lühendite ja mõistete sõnastik

ALPN	*Application-Layer Protocol Negotiation*, turbeprotokolli laiend
API	*Application Programming Interface*, rakendusliides
ARM	*Advanced RISC Machines*, väikese energiakuluga 32-bitiste RISC-protsessorite sari
ASIO	*Audio Stream Input/Output*, helidraiver
BSD license	*Berkeley Software Distribution license*, madala piiranguga avaliku lähtekoodiga tarkvara litsentsi tüüp
CORS	*Cross-Origin Resource Sharing*, HTTP-päisel põhinev mehhanism
DDoS	*Denial-of-service attack*, hajutatud teenusetõkestamise rünne
FIFO	*First-In, First-Out*. Esimesena sisse, esimesena välja
HTML	*HyperText Markup Language*, hüperteksti märgistuskeel
HTTP	*HyperText Transfer Protocol*, hüperteksti edastusprotokoll
HTTPS	*Hypertext Transfer Protocol over SSL*, hüperteksti edastusprotokoll üle turvasoklite kihi
ICMP	*Internet Control Message Protocol*, Interneti kontrollsõnumiprotokoll
IPv4	*Internet Protocol version 4*, IP protokolli neljas versioon
IPv6	*Internet Protocol version 6*, IP protokolli kuues versioon
JACK	*JACK Audio Connection Kit*, helidraiver
JIT compiler	*Just-in-time compiler*, sünkroonkompilaator
MITM	*Man in the middle attack*, küberrünnaku liik
MTU	*Maximum transmission unit*, maksimum-ülekandeühik
NAT	*Network Address Translation*, võrguaadresside teisendamine
P2P	*Peer to Peer*, partnervõrk
PCM	*Pulse-code modulation*, impulss-koodmodulatsioon
QUIC	*Quick UDP Internet Connections*, transpordikihi võrguprotokoll
RTT	*Round-trip time*, pendellevi aeg
SSL	*Secure Sockets Layer*, turvasoklite kiht
STUN	*Session Traversal Utilities for NAT*, standardiseeritud meetodite

	kogum võrguaadresside teisendamiseks
TCP	*Transmission Control Protocol*, edastusohje protokoll
TLS	*Transport Layer Security*, transpordikihi turbeprotokoll
TURN	*Traversal Using Relay NAT*, protokoll võrguaadresside teisendamiseks partnervõrkudes
UDP	*User Datagram Protocol*, kasutajadatagrammi protokoll
URL	*Uniform Resource Locator*, internetiaadress
WASM	*WebAssembly*, binaarne käsuvorming virnapõhisele virtuaalmasinale

Sisukord

1 Sissejuhatus....10
2 Eesmärk ja eelistatav tulemus....12
3 Turul olevate lahenduste analüüs....14
3.1 Avatud lähtekoodiga lahenduste analüüs....14
3.1.1 Jacktrip - tootluse etalon....14
3.1.2 Jacktrip-webrtc....16
3.1.3 Jamulus....17
3.1.4 Sonobus....18
3.2 Suletud lähtekoodiga lahenduste analüüs....19
3.3 Analüüsi kokkuvõtte ning põhiliste probleemide kirjeldus....20
4 Lahenduse ettepanek....23
4.1 Lahenduse põhjendamine....23
4.2 Lahenduse tehniline kirjeldus....28
5 Lahenduse realiseerimine....31
5.1 Klient veebirakendus....31
5.1.1 Ühendamine serveriga....31
5.1.2 Vaikesisendi helivoo hõivamine ja saatmine....32
5.1.3 Sissetuleva helivoo taasesitamine....33
5.2 Serveri rakendus....33
6 Tulemuste analüüs....35
6.1 Lahenduse testimine ja analüüs....35
6.2 Uue lahenduse latentsuse analüüs....36
6.2.1 Kohaliku arvuti heli *RTT* mõõtmise metoodika....36
6.2.2 Heli voogedastamise võrgu *RTT* mõõtmise metoodika....37
6.2.3 Testimises kasutatud seadmed....37
6.2.4 Helivoo edastamise aeg kohtvõrgus....38
6.2.5 Helivoo edastamise aeg üle interneti....39
6.3 Uue lahenduse kasutatavuse analüüs....42

6.4 Lahenduse turbeanalüüs.....42
6.5 Tulemuse võrdlus tänapäeva turul olevate süsteemidega.....42
7 Takistused ja piirangud.....44
7.1 Põhilised takistused.....44
7.2 Lahenduse piirangud.....45
8 Takistuste võimalikud tulevased lahendused.....46
8.1 Helisisendi hõivamise parandamine.....46
8.2 Heli kodeerimise paremaks muutmine.....47
8.3 Targad puhvrid.....48
8.4 Kadunud andmepakettide haldamine.....48
9 Kokkuvõte.....49
Kasutatud kirjandus.....51
Lisa 1– Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks.....55
Lisa 2 – Rakenduse peamised meetodid.....56
Lisa 3 – MicrophoneWorkletProcessor.....60
Lisa 4 – SpeakerWorkletProcessor.....62

Jooniste loetelu

Joonis 1. Olulised *Jacktrip* parameetrid. [6]..15

Joonis 2. *JackTrip-WebRTC* programmi logimise menüü kuvatõmmis...........................16

Joonis 3. *Jamulus* rakenduse klient-server-mudel [13] 17

Joonis 4. *Jamulus* programmi kuvatõmmis..18

Joonis 5. *SonoBus* programmi kuvatõmmis...19

Joonis 6. See diagramm näitab *TURN* serveri toimimist [28] 26

Joonis 7. *TCP* ja *QUIC* protokollide ühenduse loomise võrdlus [31] 27

Joonis 8. *AudioWorkletProcessor*'i skeem, milles kasutatakse kahte rõngapuhvrit *WASM*-funktsiooni jaoks, mis võtab 512 kaadrit sisse ja välja [37] 29

Joonis 9. *connect()* funktsiooni näinäide..32

Joonis 10. Serverirakenduse *AudioStreamHandler* klassi näinäide................................34

Joonis 11. *MacOS* ja *Linux* operatsioonisüsteemide erinevus..38

Joonis 12. *MacOS* test Tallinna serveriga...40

Joonis 13. *MacOS* test Amsterdami serveriga..40

Joonis 14. *Linux* test Tallinna serveriga...41

Joonis 15. *Linux* test Amsterdami serveriga..41

Joonis 16. Erinevate koodekite kvaliteet bitikiiruse funktsioonina [54] 47

1 Sissejuhatus

Äkilise viiruselevikuga põhjustatud karantiinid näitasid kui oluline on pöörata tähelepanu tähtsate töö- ja õppimisprotsesside üleviimisele veebi. Koolid, valitsusasutused ja eraettevõtted olid sunnitud minema üle veebikonverentside lahendustele, mille populaarsus kasvas hüppeliselt aastas 2020. Prognoositakse, et veebikonverentside rakenduste kasutajate baasi edasine kasv on väga tõenäoline. Kahjuks olemasolevad lahendused ei sobi nendele, kelle jaoks heli voogedastamise latentsusaeg on kõige kriitilisem aspekt.

Tänapäeval puuduvad kasutajasõbralikud veebirakendused, mis annaksid muusikutele võimalust mängida koos otseülekandes veebibrauserite kaudu. Muusikud, alates väikestest muusikabändidest ja lõppedes orkestritega, peavad harjutamiseks ja esinemiseks saama kokku ühes ruumis ning mõningatel tingimustel see võib osutada võimatuks. Üksikud ülikoolid arendavad välja enda lahendusi, mis võimaldavad mängida koos üle interneti ülimadala heli latentsusega. Sellised lahendused aga alati kujutavad endast iseseisvaid klientrakendusi, mis on tihti keerulised ning vajavad spetsiifilisi oskusi valdkonnas süsteemi eduka rakendamiseks. Üheks selliseks näiteks on *Jacktrip*. Mõned üritavad lahendada antud probleemi ning teha inimsõbralikumaid rakendusi, nagu näiteks *Jamulus*, *SonoBus*, *Jammr*, *Jamkazam* ja muud samad. Aga iga nimetatud lahendus jällegi kujutab endast iseseisvat arvutile paigaldatavat rakendust, mis on suur limiteeriv faktor juhtudel, kus kasutajatel puuduvad õigused paigaldada arvutile tarkvara või kasutusel on hoopis mõni mittetoetatud platvorm, näiteks *iOS* või *Android* süsteemil opereeriv seade. Samuti see võiks olla probleemiks kasutajatele, kellel puuduvad vajalikud tehnilised oskused. Näiteks võib tuua lastekoori, kus iga lapse arvuti seadistamine esinemiseks ning edaspidine seadistuse korras hoidmine on tihti võimatu ülesanne. Autori arvates probleemi saab lahendada arendades mitmeplatvormilist veebibrauseripõhist rakendust.

Lõputöö jaotub kolmeks osaks. Töö esimene osa – peatükid 2 ja 3 – keskendub probleemi kirjeldamisele, olemasolevate lahenduste analüüsi peale ja nende mittesobivuse põhjustele ning lõputöö eelistatava tulemuse kirjeldusele.

Lõputöö teine osa – peatükid 4 ja 5 – hõimavad autori pakutud lahenduse kirjeldamisele ja realiseerimisele.

Töö kolmas osa – peatükid 6, 7 ja 8 – pühendatud välja arendatud lahenduse analüüsile, tekkinud probleemide ja nende lahenduste kirjeldamisele.

Allikatena on kasutusel valdavalt avalik dokumentatsioon, digitaalse heli ja võrgu voogedastust puudutav kirjandus ning artiklid.

2 Eesmärk ja eelistatav tulemus

Käesolev töö keskendub veebibrauserite helivoo latentsuse vähendamisele. Töö autor uurib, kas madala latentsuse heli voogedastamise probleemile on võimalik arendada välja kliendisõbralik veebibrauseripõhine lahendus, mis potentsiaalselt saab aidata muusikuid, muusikakoole ja ülikoole üle tervet maailma. Lisaks selline tehnoloogia aitab kaasa selliste probleemide lahendamisele nagu suhtlemine mängu ajal veebibrauseri mängudel, helitõlge reaalajas, digitaalse heli andmete analüüs ja muu serveripoolne helitöötlemine, mida tehnilistel põhjustel ei ole võimalik teostada kliendi juures ning mille latentsuse vähendamine toob selgeid jõudluse eeliseid.

Kaasaegsed veebibrauserid on piisavalt arenenud ning autori arvamusel võivad lubada selliste ideede elluviimist. Põhjuseks, miks tänapäevaks keegi pole veel seda teinud osaliselt peitub selles, et arendajatel ei ole õigeid tööriistu väikese latentsusega heli edastamiseks. Eraldiseisvad, operatsioonisüsteemidele paigaldatavad klientrakendused opereerivad *UDP* protokollil, tänu millele saavutatakse nõutud heli voogedastamise kiirus ohverdades sageli turvalisust. Kuigi kasutades raamistikuid nagu *WebRTC* [1] ning *Media Source Extensions* [2] spetsifikatsiooni on selline arhitektuur teostatav, need lahendused toovad kaasa rohkem probleeme kui abi.

WebRTC raamistik pakub arendajatele kõrgetasemeliste *API*’de komplekti, mis kuigi lahendab käsitlevaid probleeme, ei anna ruumi peenhäälestamiseks või teeb häälestamist väga keeruliseks. Tegemist on kohmaka koodikoguga, mille sobitamine arendatava projektiga võib tekitada raskusi arendajate meeskondadele. Lõputöö eesmärk on uurida ning pakkuda kerget ja kiiret alternatiivi digitaalse heli voogesitamiseks veebibrauserist.

Veel üheks tähtsaks töö eesmärgiks on hinnata tänapäeva turul olevad rakendused, uurida millistele tehnoloogiatele tuginetakse kõige tihedamini ning leida selle taga põhjust.

Peamiseks eesmärgiks on kontseptsiooni tõestuseks luua klient-server arhitektuuriga rakendust. Veebirakenduse roll on lehe külastaja mikrofoni heli hõivamine, selle

edastamine serverile, serverist vastu tulnud helivoo hõivamine ning taasesitamine süsteemi heliväljundi kaudu. Heli voogesitamise ajal jälgida ja esitada latentsuse aega. Serveri roll on sissetulnud helivoo peegeldamine tagasi kliendile.

Katsetuste eesmärk on samas regioonis, ehk kui mõlemad klient ja server asuvad Eestis, saavutada veebibrauseripõhist heli edasi-tagasi voo latentsust 30 ms piires, mis on umbkaudne aeg millega heli läbib õhus 10 meetrit.

Väljatöötatav veebirakendus peab olema minimalistlik, võimalusel ilma sõltuvuseta suurtele teekidele. Serveripoolne infrastruktuur peab olema samuti lihtne.

Arendamise jooksul on eesmärgiks analüüsida ning selgitada välja kõik potentsiaalsed probleemid mis võivad takistada rakenduse ellu viimist.

Lõputöö ei kirjelda veebirakenduse kujundust, kasutajasõbralikkuse arendamist ega ei sisalda kasutamiseks valmis toote arendamist. Põhirõhk jääb heli voogedastamise tehnilise rakendamisele, testrakenduse katsetamise ja katsetulemuste analüüsi peale.

Serveri rakenduse arendamine jääb samuti skoobist välja. Töö eesmärgiks ei ole arendada serveri rakendust, mis oskaks panna muusikuid koos mängima. Eesmärk on uurida, kas selline lahendus on tänapäevaste veebibrauserite baasil tehniliselt teostatav, kirjeldada lahendamiseks valitud tehnoloogiad ning põhjendada valikut.

3 Turul olevate lahenduste analüüs

Autor otsis välja püstitatud ülesande lahendamiseks enim sobivaid lahendusi eesmärkiga analüüsida nende töö ning analüüsi tulemustena saada piisavalt teadmisi olemasolevate süsteemide nõrkustest ja tugevustest. Analüüsi peamised sihtmärgid on helivoo edastuse latentsus ning lahenduse kasutatavus. Selle paragrahvi eesmärgiks on leida üldised nõrkuste nimetajad selleks, et kasutada antud informatsiooni uue lahenduse väljatöötamiseks.

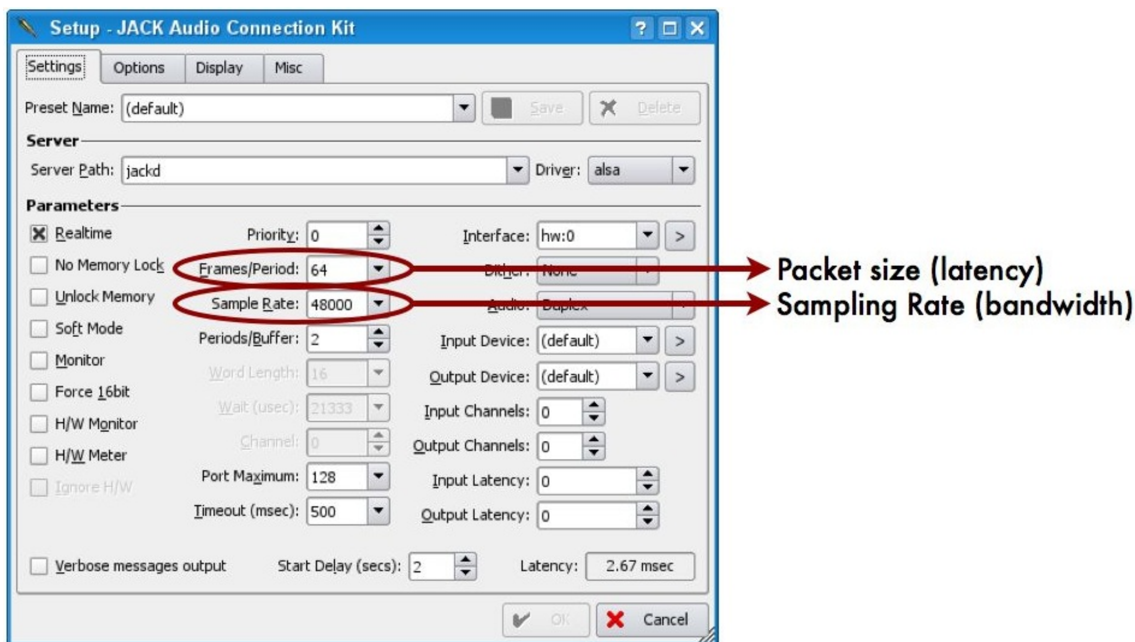
3.1 Avatud lähtekoodiga lahenduste analüüs

Autor analüüsib neli lahendust, mille lähtekood on avatud ning kaetud litsentsiga mis võimaldab koodi taaskasutamist, modifitseerimist ning edasist arendamist.

3.1.1 Jacktrip - tootluse etalon

Jacktrip [3] toetab kahesuunalist ja kvaliteetset heli voogedastust piiramatu kanalite arvuga. Välja töötatud Stanfordi ülikooli muusikaprofessori Chris Chafe ja tema kolleegide poolt [4] *Jacktrip* on käsureaprogramm, mis töötab koos *Jack Audio Connection Kit*'iga [5] . Jacktrip toetab *Windows*, *MacOS* ning *Linux* operatsioonisüsteeme.

Muusikaseanssi alustamiseks tuleb käivitada ning häälestada *Jacktrip* tarkvara. Häälestamiseks kõige olulised parameetrit on kaadrid/periood ja diskreetsagedus.



Joonis 1. Olulised *Jacktrip* parameetrid. [6]

Mida madalam on kaadrid/periood parameeter, seda väiksem on latentsus. Mida suurem on diskreetsagedus, seda suurem on nõuded ribalaiusele. Kõikidel seanssi osalejatel peavad need häälestused olema võrdsed.

Kui seadistatud õigesti, piisavalt kiire internetiühendusega *Jacktrip* võimaldab saavutada ülimadalat heli voogedastuse latentsust, mis jääb alla 30 millisekundi ning sobib suurepäraselt elava muusika esitamiseks.

Jacktrip tehnoloogiat saab samuti kasutada spetsiaalselt seadistatud *Raspberry Pi* [7] seadega, mida kasutatakse kohaliku *Jacktrip* serverina [8] [9] . Selliselt kasutaja ei pea paigaldama *Jacktrip* tarkvara enda arvutile, kogu seadistamine käib üle kohaliku veebilehe kus saab häälestada kõik tööks vajalikud seadistused.

Jacktrip tarkvara kasutatakse peamiselt õppeasutustes ning ka muude teenuste toeks, nagu näiteks teenus muusikute harjutamiseks ning koos mängimiseks [10] .

Kuigi antud programmi saab nimetada jõudluse etaloniks, siiski tegemist on eraldiseisvaga arvutile paigaldatavaga klientrakendusega või *JackTrip Virtual Studio* [8] ning *JackStreamer* [9] toodete puhul iseseisva riistvaraga, mis ei sobi meie püstitatud eesmärgi saavutamiseks. Lisaks sellele, seansside andmevoo ei krüpteerita.

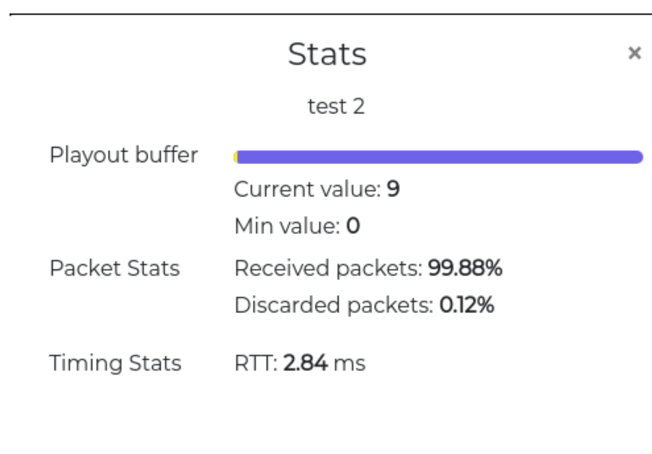
Töö eesmärk on aga uurida veebibrauserite valmidust toetama turvatud heli voogedastamist ilma klient rakenduseta ega riistvaraliste erilahenduseta.

3.1.2 Jacktrip-webrtc

Jacktrip-webrtc kujutab endast *Jacktrip* serveri realiseerimist *HTML 5* tehnoloogias. Veebibrauseripõhine *Jacktrip* server kirjelduse järgi sobib täiuslikult püstitatud ülesande lahendamiseks.

Kahjuks rakenduse lähtekoodi [11] lähemal uurimisel tuli välja, et tegemist on tavalise *Web Audio API* liidesel põhineval veebirakendusega.

Rakendus omab enda jõudluse monitoorimise funktsionaalsust. Kahes arvutis kohtvõrgus katsetades heli andmevoo võrgu *RTT* on 2 kuni 4 ms. Monitooringu näidis on toodud joonisel 2.

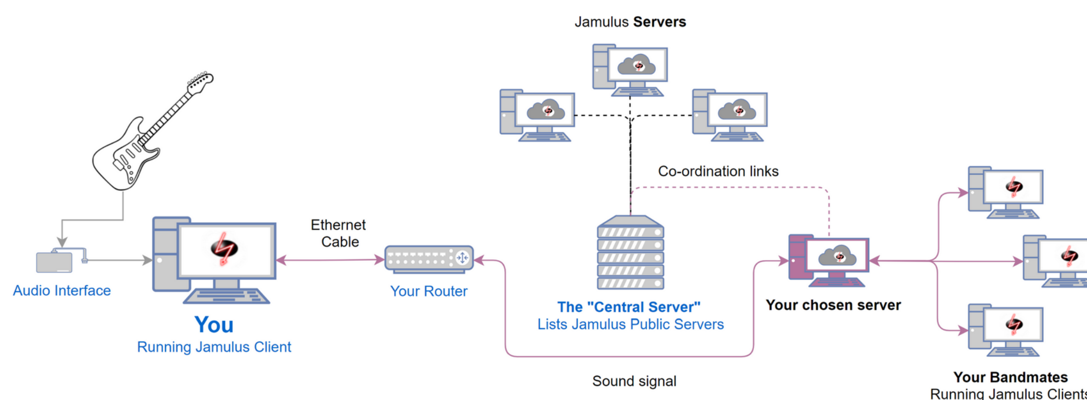


Joonis 2. *JackTrip-WebRTC* programmi logimise menüü kuvatõmmis.

Samas heli latentsus oli kõrva järgi mõõdetud numbrist kordades suurem. Lähtekoodi analüüsides õnnestus tuvastada, et rakenduses mõõdetud *RTT* kirjeldab vaid võrgupakettide edasi-tagasi teekonna aega. Latentsuse mõõtmisel ignoreeritakse veebibrauseri *AudioContext*'i töötamise aega ning ka suure osa rakenduse enda helitöötlemise aega. Programm ei anna võimalust testida tõelist heli *RTT*, seega täpne latentsuse ajavahemik on teadmata. Täpselt saab öelda ainult seda, et autori testimise ajal mängida muusikat nii suure latentsusega ei õnnestunud.

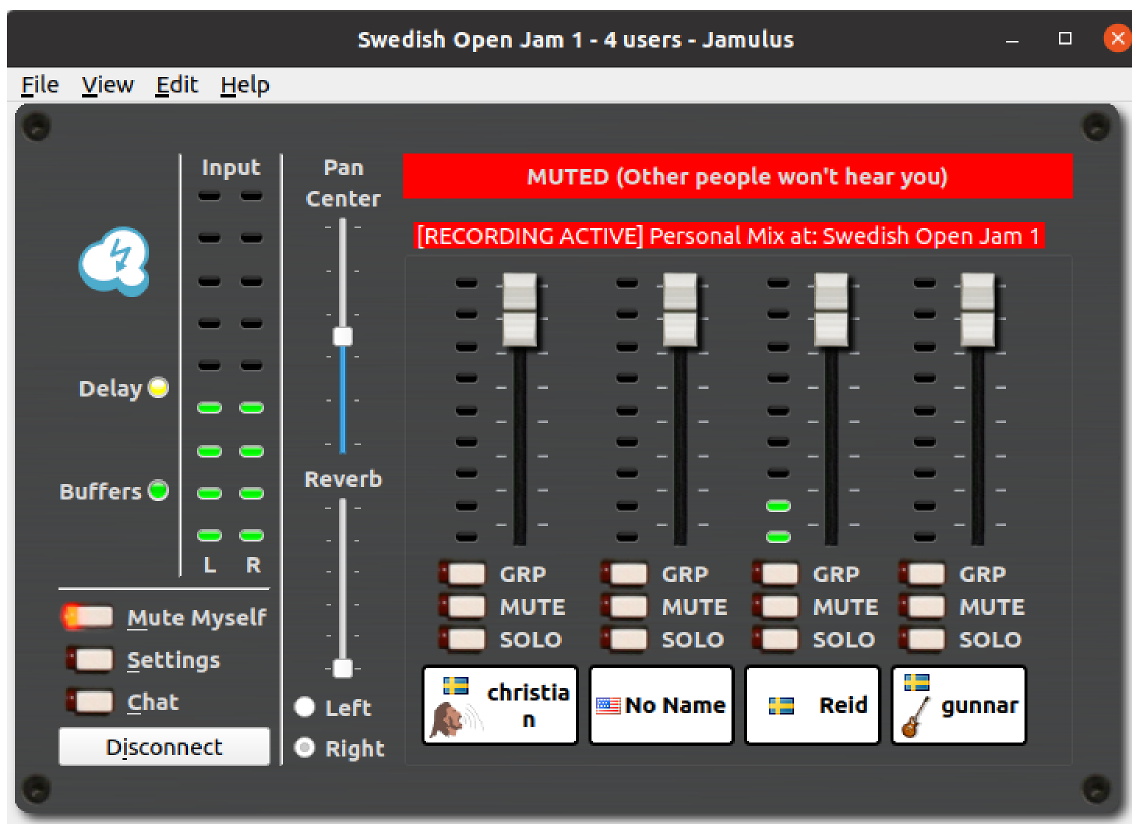
3.1.3 Jamulus

Tegemist on tasuta klient rakendusega mis töötab *JACK* serveri baasil [12] . Programm pakub erinevad serverid teiste muusikutega ühendamiseks. *Jamulus* töötab kliendi-serveri arhitektuuri põhimõttel. Kõikide seanssi osalejate helisisendid saadetakse serverisse, kus helikanalid mikseeritakse ja töödeldakse. Pärast seda saadetakse valmis küpsenud heli igale kliendile tagasi. Kui server tehakse avalikuks ja registreeritakse keskserveris, edastatakse selle teave kõigile klientidele.



Joonis 3. *Jamulus* rakenduse klient-server-mudel [13] .

Programm pakub häid keskkonda muusikutele. Ainsaks nüanssiks jääb programmi paigaldamine ja seadistamine. Selleks, et saada programmi korrektselt tööle, on vaja paigaldada arvutisse mitte ainult klient rakenduse enda, vaid ka spetsiaalse helidraiveri. *Windows*'i puhul *ASIO*, *Linux*'il *JACK* ning ainult *MacOS* operatsioonisüsteemil piisab vaid programmi paigaldamist. Sellega saavutatakse ülimadala heli *RTT* ning üleüldine latentsuse aeg sõltub serveri kaugusest kasutajatest. Samas riigis majutatud serveriga ning hea internetiühendusega alla 30 ms latentsuse saavutamine on garanteeritud.



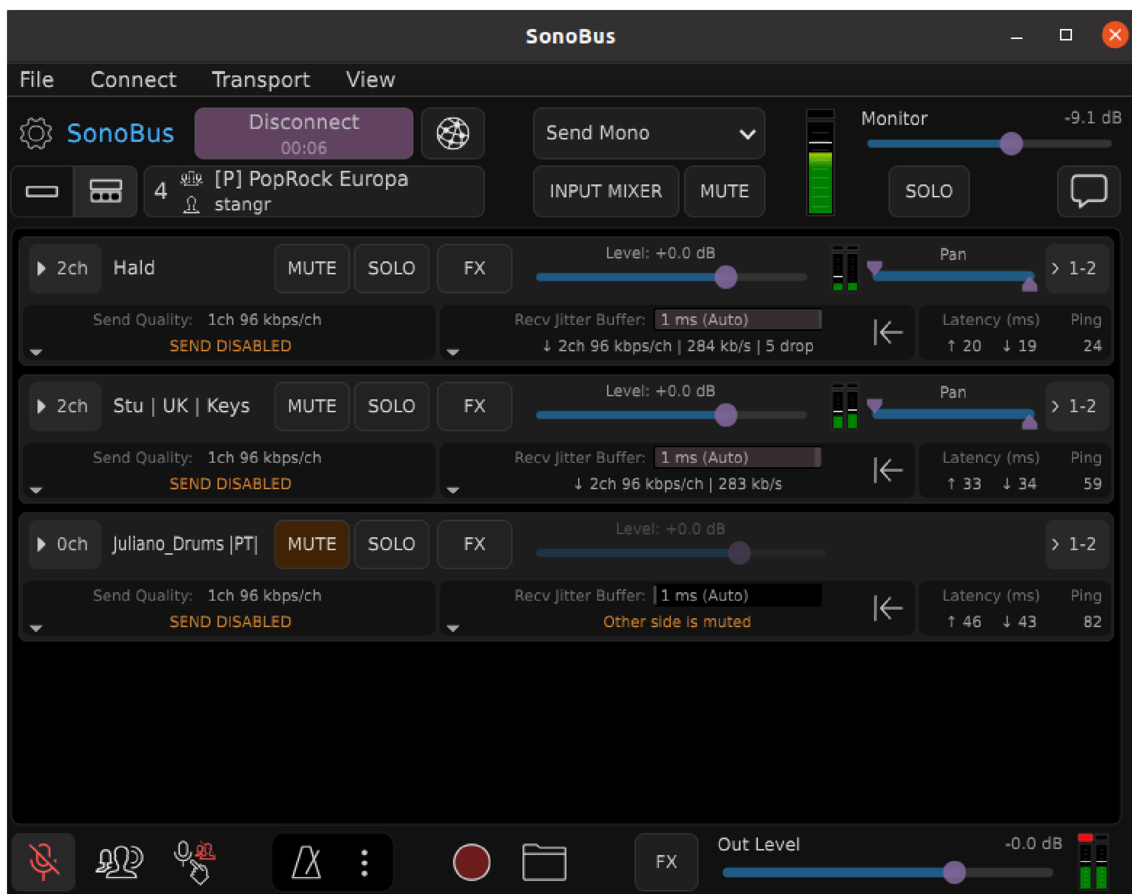
Joonis 4. *Jamulus* programmi kuvatõmmis.

Analüüsi kokkuvõtteks saab öelda, et *Jamulus* on hästi toimiv programm mis täidab oma funktsiooni kuid kasutatavuse poole pealt tegemist pole „ühenda ja mängi" isehäälestuva lahendusega.

3.1.4 Sonobus

SonoBus [14] on hõlpsasti kasutatav klient rakendus kvaliteetse, väikese latentsusega *P2P* heli voogesitamiseks seadmete vahel interneti või kohaliku võrgu kaudu. Programmi tugevad küljed on lisaks arvutitele ka *iOS* ja *Android* tugi, mis muudab programmi tõeliselt universaalseks. *SonoBus*'i jaoks ei ole vaja paigaldada spetsiaalset helidraiverit, kuigi see võib parandada jõudlust ning vähendada latentsust.

Kasutaja kogemuse vaates on autori kallutatud arvamus, et tegemist on kõige parema avatud lähtekoodiga lahendusega, mis toetab kõige suuremat operatsiooni süsteemide ning helidraiverite valikut. Samuti rakendus omab samad ülimadala latentsuse tugevused mida sai leida ülaltoodud *Jamulus* ja *Jacktrip* rakendustes.



Joonis 5. *SonoBus* programmi kuvatõmmis.

Negatiivsetest külgedest, *SonoBus* ei krüpteeri andmesidet ning tegemist on ikkagi klient rakendusega, mille tõttu ei sobi see püstitatud eesmärgi saavutamiseks.

3.2 Suletud lähtekoodiga lahenduste analüüs

Autor analüüsis programme, mille lähtekood on suletud eesmärgiga võrrelda suletud ning tasulisi platvorme ülal toodud tasuta alternatiividega. Kahjuks ükski analüüsitud lahendusest ei sobi eesmärgi saavutamiseks.

Aloha by Elk [15] kirjelduse järgi tundub väga paljulubav lahendus. Tegemist on riistvaralise liidesega, mis asendab kasutaja välise helikaardi. Ettevõte garanteerib, et *Aloha* liides lisab helisignaali hõivamisel ja taasesitamisel ainult 1 ms latentsuse juurde. See tähendab, et kui kahe kasutaja vaheline ühenduse *RTT* on 10 millisekundi, siis *Aloha* liidesega summaarne heli *RRT* jääb 12 millisekundi kanti. Autor ei saanud testida seadet kuna toode pole veel masstoodangus ning ligipääsu tehnoloogiale antakse vaid piiratud inimeste arvule varajase juurdepääsu liitumisprogrammiga.

Soundtrap [16] veebirakendus kuigi täidab kõike digitaalse heli tööjaama funktsioone, ei võimalda muusikutele mängida koos ühes seansis. Selle veebirakendusega saab vaid salvestada ühte helirada korraga ning mikseerida juba salvestatud helirajad. Otse eetris mitu inimest samaaegselt mängida või salvestada muusikat ei saa.

AudioTool [17] on ülaltoodud *Soundtrap*'iga sarnane veebirakendus, täidab samad funktsioonid ning toob samad piirangud, mis ei võimalda kasutada tarkvara töös püstitatud eesmärgi saavutamiseks.

Jamkazam [18] on tasuta, suletud lähtekoodiga lahendus mis pakub nõutud funktsionaalsust. Kahjuks, autoril ei tekkinud võimalust korraldada korraliku testimise seanssi kuna *Jamkazam* ei toeta *Linux* operatsioonisüsteemi. Sai aga välja uuritud, et tegemist on *Jamulus*'iga ja *SonoBus*'iga väga sarnase programmiga, mille korrektse töö tagamiseks peab kasutaja eelkõige kulutama aega enda arvuti helidraiveri seadistamisele.

3.3 Analüüsi kokkuvõtte ning põhiliste probleemide kirjeldus

Püstitatud eesmärgi saavutamise takistuseks on programmide kasutatavus. Kõik analüüsitud platvormid, mis olid võimelised pakkuda ülimadala latentsusega helivoo edastust, olid teostatud klientrakenduste kujul. Selline lähenemine raskendab nii programmide kasutamise, kui ka arendamise protsesse. Näitena toob autor platvormi kasutamist õppeasutustes, kus õpilasel võivad puududa vajalikud oskused klientrakenduste paigaldamiseks isiklikele arvutitele. See asjaolu saab potentsiaalselt takistada platvormi kohandamist.

Arendajate vaatenurgast on eraldiseisvate klientide arendamisprotsess samuti keeruline, kuna see nõuab platvormi sobitamist mitmete operatsioonisüsteemide versioonidega. Mitme operatsioonisüsteemi toetamine suurendab programmivigade esinemise tõenäosust, mis omakorda suurendab riske iga programmi redaktsiooni puhul. Samuti ka operatsioonisüsteemi muudatused võivad muuta arendatud programmi kasutuks. Sellise pretsedenti näitena võib tuua *Apple M1* protsesside avaldamist, mille *ARM* arhitektuuri erinevuse tõttu pidid paljud ettevõtted hakata kohandama oma tarkvara [19] .

Üks analüüsitud lahendusest – *Jacktrip-webrtc* – on veebibrauseri põhine ning sobib paremini meie püstitatud probleemi lahendamiseks. Veebibrauseri põhine platvorm võimaldab parandada kasutatavust, kuna eemaldab vajadust paigaldada arvutile klientrakendust. Iga arvuti kaasaegse veebilehitsejaga on võimeline kasutada süsteemi mis lahendab terve hunnik potentsiaalseid probleeme:

- Heli voogedastamise süsteemi kasutamine arvutil, kus kasutajal puuduvad arvuti administraatori õigused lisatarkvara paigaldamiseks;
- Süsteemi kasutajasõbralikkuse parandamine, kuna töö alustamiseks kasutaja peab vaid minema veebilehele;
- Arendajate töö lihtsustamine, kuna veebirakenduse arendamise ja halduse ulatus on väiksem, kui klientrakenduse arendamine ning mitmete operatsioonisüsteemide toetamine.

Vaatamata selgetele kasutatavuse eelistele *Jacktrip-webrtc* analüüsi käigus autor tuvastas probleeme süsteemi jõudlusega. Testide ajal ei õnnestunud saavutada piisavalt madalat latentsust, mis muutus lahenduse kasutuks.

Veel üks levinud näitaja analüüsitud tarkvarade seas on puudulik võrgu andmepakettide krüpteerimine. Mõned analüüsitud programmid ei krüpteeri andmeid transpordis. Andmepakette saadetakse turvamata kujul üle interneti ning see on suur turvarisk, kuna võimaldab *MITM* rünnaku abil püüda kinni ja dekodeerida andmepakette ning nõndaviisi salvestada pealt helivoo.

Autor oli pannud tähele ka asjaolule, et suur osa analüüsitud tarkvaradest ei toeta *IPv6* kasutamist. Kuna helivoo edastamine tugineb *UDP* peale, latentsuse vähendamiseks ning robustsuse parandamiseks tasub vältida andmepakettide segmenteerimist. *UDP* andmepakettide segmenteerimine toimub siis, kui ühe paketti suurus ületab võrgus lubatud *MTU* suurust. Minimaalne *MTU IPv4* võrgus on 576 oktetti [20] . *IPv6* võrgus on see suurus 1280 oktetti [21] . Need on suurused, mille raames iga võrgusõlm peab garanteerima segmenteerimata *UDP* pakettide edastust.

Heli rakendustes andmepaketisse reeglina kapseldatakse helipuhvrit. Nagu sai välja toodud, minimaalne *MTU IPv6* puhul on ligi kaks korda suurem kui *IPv4* võrkude puhul

mis annab arendajale võimalust mahutada iga andmepaketti sisse rohkem informatsiooni, ehk potentsiaalselt suurema heli puhvri raami. *IPv6* aitab vähendada *UDP* datagrammide arvu ning mõnedel stsenaariumitel aitab parandada võrgu läbilaskevõimet voogedastuse puhul [22] . *IPv6* mitte toetamine tänapäeva programmides võib pidada puuduseks.

4 Lahenduse ettepanek

Turul olevate platvormide analüüsi tulemusena oli autor saanud ülevaadet programmide nõrkustest. Nende nõrkuste kõrvaldamiseks pakub autor lahendust, mis peab parandama programmide kasutatavust ning aitama kaasa arendajate tööd heli platvormide projekteerimisel ning arendamisel. Samuti peab väljatöötav lahendus võimaldama ülimadala latentsusega turvatud heli voogedastust üle interneti.

4.1 Lahenduse põhjendamine

Põhiliseks eesmärgiks on kontseptsiooni tõestamiseks töötada välja veebirakendus, mis lahendab täiesti või aitab vähendada ülal kirjeldatud probleeme. Eesmärgi saavutamiseks välja töötav veebirakendus peab pakkuma järgmised funktsioonid:

- Veebibrauseri põhist heli voogedastust ülimadala latentsusega;
- Vähendatud rakenduse arendamise keerukust;
- Vähendatud infrastruktuuri keerukust rakenduse toetamiseks;
- Helivoo krüpteerimist.

Veebirakenduse arendamiseks autor valis *Javascript* keelt. Kuna *Javascript* on programmeerimise keel mida interpreteeritakse kasutaja veebibrauseris, võimaldab see püüda kinni, taasesitada ning töödelda helivoo kliendi juures, mis aitab kaasa helivoo latentsuse parandamisele ning serveri koormuse vähendamisele. Helitöötlemise näitena võib tuua kaja tühistamist, müra summutamist ja muu taolist ning kõike seda saab teostada kliendi brauseris säästades serveri töötlemisaega ning vähendades koormust.

Heli püüdmiseks ja taasesitamiseks autor valis *HTML 5 Web Audio API* liidest. *Web Audio API* on kõrgetasemeline liides heli töötlemiseks ja sünteesimiseks veebirakenduses. Antud programmiliides on tänapäeval kõige levinud heli töötlemiseks

veebibrauserites ning see on lisatud kõikidesse kaasaegsetesse veebibrauseritesse [23] . Tänu *Web Audio API'*le on kergesti võimalik püüda mikrofoni heli *PCM* andmevoo ning taasesitada *PCM* andmevoo seadme vaike heliväljundi kaudu.

Lisaks *Web Audio API* pakub *AudioWorklet* liidest, mida kasutatakse kohandatud helitöötlusskriptide tarnimiseks. Neid käivitatakse eraldi lõimes pakkudes väga madala latentsusega heli töötlemist [24] .

Helivoo edastamine üle võrgu peab olema kiire ülimadala latentsuse saavutamiseks. *TCP* protokolli puhul iga saadetud paketti peale oodatakse vastuvõtjalt vastust, et see on kohale jõudnud, mis lisab ajakulu igale tehingule. Samuti *TCP* puhul kõik andmepaketid saab vastu võtta vaid selles järjekorras, milles saatja oli saatnud neid välja – *head-of-line* blokeerimine. Mõnikord juhtub, et mõni üksik andmepakett võtab rohkem aega selleks, et jõuda sihtkohani ning isegi kui teised, uuemad andmepaketid on juba sihtkohani jõudnud, protokolli kirjelduse järgi peab ootama kõigepealt kaduma läinud paketti ning ainult siis hakata töötlema uued andmepaketid. Selline käitumine ei sobi reaalaja teenustega.

Võrreldes *TCP'*ga, *UDP* on ühenduseta, mis tähendab, et andmepakette saab saata ning vastu võtta läbirääkimisteta. Samuti puudub *UDP'*l igasugune veajuhtimine. Pakette saab mitte ainult vales järjekorras tarnida, vaid need võivad ka täielikult ära kaduda teenuse töö häirimata. *UDP* on mõeldud rakendustele, mis on rohkem huvitatud teabevoo jätkamisest kui selle veatu oleku tagamisest [25] . See teeb *UDP'*d ideaalseks protokolliks reaalaja teenuste jaoks, nagu näiteks heli- ja video voogedastus.

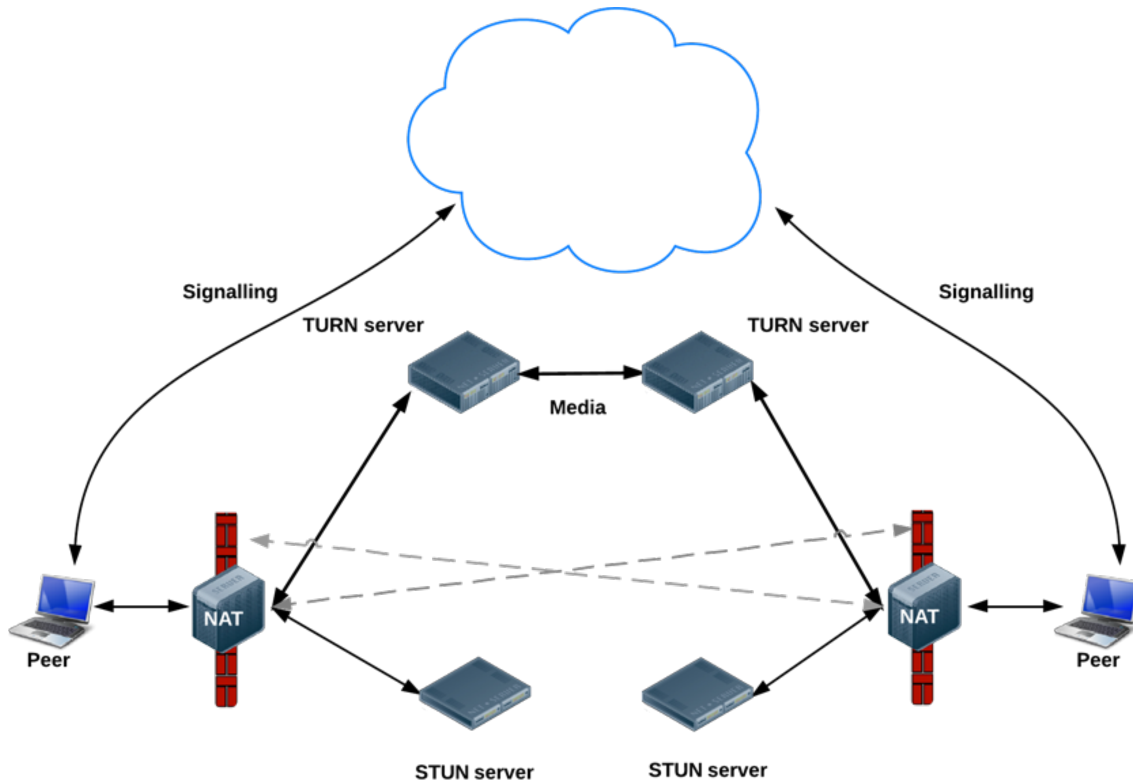
Kahjuks, lõputöö projekti jaoks valitud *UDP* toob kaasa uut väljakutset – *UDP*'ga töötamist veebibrauserites. Veeb on üles ehitatud *TCP* peal ning veebibrauserid ei toeta *UDP* pakettide saatmist ning selleks on mitu põhjust. Mõned neist on [26] :

- Veebisaidid saaksid käivitada *DDoS*-rünnakuid, koordineerides *UDP* pakettide laviine ohvri pihta veebilehe külastajate brauseritest;
- Leiutakse uued turvaaugud. Veebilehtedel töötava *Javascript*'i abil saab luua pahatahtlikke *UDP* pakette, mille abil saab uurida ettevõtte võrkude sisemisi osi ning raporteerida tulemusi tagasi ründajale tavalise *HTTPS'*i kaudu;

- Tavalisi *UDP* pakette ei krüpteerita, nii et ründaja võib nende pakettide abil saadetud andmed pealt lugeda või andmepakettide edastamisel neid isegi muuta;
- Puudub autentimise tugi, seega peaks brauserist saadetud andmepakette lugev server rakendama oma meetodit tagamaks, et ainult kehtivatel klientidel on lubatud sellega ühendust luua. Selle probleemi lahendamine on iseenesest hästi mahukas töö mis nõuab suurt kogemust küberturbe valdkonnas ning enamik veebiarendajad ei ole valmis seda probleemi lahendama.

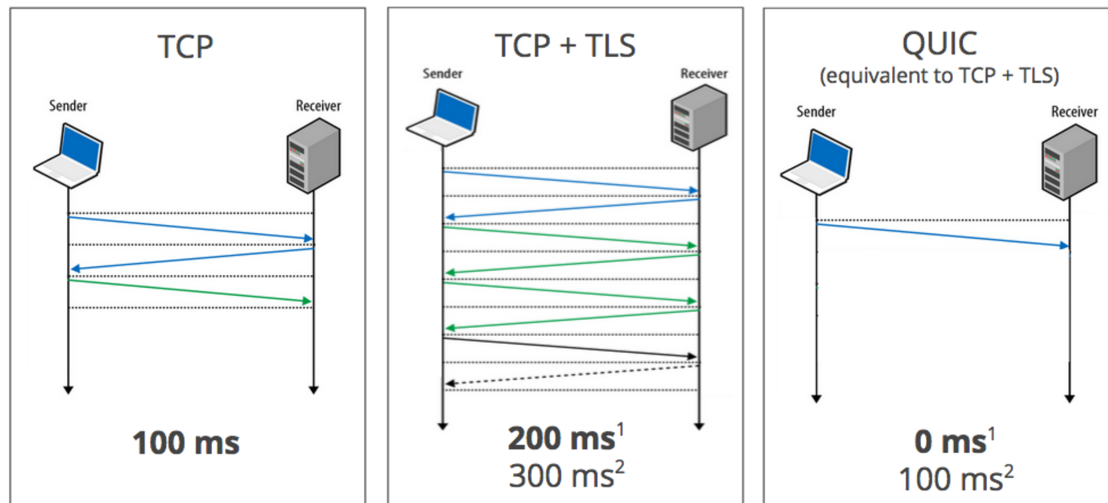
Aegade jooksul oli tehtud mitu katsetust kirjutada liidest, mis annaks arendajatele võimalust kasutada puhta *UDP'*d või *UDP'*ga sarnast loogikat olemasoleva *TCP* baasil mis lahendaks ülaltoodud probleemid. Üks näidetest on *WebSockets* [27] , mis on veebipõhise turbemudeliga ühilduv sõnumipõhine protokoll. Pakutav abstraktsioon on üks, usaldusväärne, järjestatud sõnumivoog ning sellepärast sarnaselt *TCP'*ga kannatab see *head-of-line* blokeerimise all, mis tähendab, et kõik sõnumid tuleb välja saata ning vastu võtta samas järjekorras isegi siis, kui need on sõltumatud. See muudab *WebSockets* lahendust sobimatuks latentsustundlike rakenduste jaoks, mis tuginevad jõudluse osas andmevoo sõltumatusele.

Veel üks selle ala mõjukamatest projektidest on *WebRTC* [1] . *WebRTC* on tasuta, avatud lähtekoodiga projekt, mis pakub veebibrauseritele ja mobiilirakendustele reaalaja sidet lihtsate rakenduste programmeerimisliideste kaudu. Oma disaini pärast *WebRTC* teeb keeruliseks klient-server arhitektuuriga rakenduse arendamist. Projekt on kasvanud üsna suureks ning selle meisterdamine on keeruline. Lõputöö eesmärgiks on aga pakkuda kerget ja kiiret alternatiivi digitaalse heli voogesitamiseks veebibrauserist, ilma keeruka infrastruktuurita *STUN* ja *TURN* serveritega. Infrastruktuuri näide on toodud joonisel 6.



Joonis 6. See diagramm näitab *TURN* serveri toimimist [28] .

Helivoo edastamiseks üle võrgu autor valis *WebTransport API* tehnoloogiat. Tegu on uue tehnoloogiaga mille eeldatav valmimine jääb aastasse 2023 [29] . See põhineb *Google* poolt arendatud *QUIC* protokolli peal, mida kuulutati avalikult välja aastas 2013 [30] . *QUIC* on üldotstarbeline transpordikihi võrguprotokoll mis töötab *UDP* peal ja kavatseb lahendada *TCP*-omased probleeme ning parandada veebirakenduste toimivust. *WebTransport API* võimaldab krüpteeritud datagrammide saatmist veebibrauserist serverisse ning serverist tulnud datagrammide vastu võtmist kliendi juures. Lisaks on uute ühenduste loomisel toimivuse eelised, kuna aluseks olev *QUIC handshake* on kiirem kui *TCP*'l üle *TLS*'i.



Joonis 7. *TCP* ja *QUIC* protokollide ühenduse loomise võrdlus [31] .

WebTransport'i aluseks kasutatud *QUIC* protokoll on tänapäevaks näinud suurt toetust ja seda kasutatakse enamiku *Google Chrome*'i ja *Google*'i infrastruktuuri vaheliste ühenduste jaoks. Mastaapne tehnoloogia testimine aastas 2015 näitas, et *YouTube* videoid vaadates *QUIC*'i üle kasutajad kogevad 30% vähem sisu puhverdamise probleeme [31] .

Lõputöö avaldamise ajaks *WebTransport* pole veel lõpuni küpsenud tehnoloogia. Liideste definitsioonid ja kirjeldused ei ole kinnistatud ning võivad muutuda. Samuti hetkel võib tunda puudust *WebTransport* baasil teostatud rakenduste näidetest veebis. Programmi arendamisel autor tugines avalikule dokumentatsioonile ning selles toodud näidisprogrammile, et saavutada soovitud tulemust [32] .

Kuna *WebTranspot* hõlmab vaid andmete krüpteerimist ning edastamist, annab see suurt paindlikust helirakenduste arendamisel. Lihtsate rakenduste puhul selle tehnoloogiaga ei pea tuginema kolmandate osapoolte sisseehitatud funktsionaalsusele, kõike heliga seotud protsesse saab hoida enda kontrolli all. Siia kuuluvad näiteks töö puhvritega, heli müra summutamine ja muu taoline.

Serveri rakendus on realiseeritud *Python* keeles. Põhjuseks on sarnane asjaolu ülaltooduga – avalik *WebTransport* serveri näide on kirjutatud *Python* keeles [33] . Töö lihtsustamise eesmärgil sai otsustatud väljapakutud näidisprogrammi ümber mitte kirjutada, kuna serveri rakenduse arendamine jääb lõputöö skoobist välja. *Python*'i

serveri näidisrakenduses sai tehtud vaid triviaalne muudatus, mille abil iga sissetulnud *UDP* paketti peegeldatakse tagasi kliendi poole. See annab võimalust lahenduse projekteerimise ajal keskenduda puhtalt heli voogedastuse tootlikkuse peale. Näidisprogrammi muutmine on lubatud *Apache 2.0* litsentsiga.

4.2 Lahenduse tehniline kirjeldus

Kontseptsiooni tõestamiseks sai kirjutatud veebirakendus *Javaskript* keeles. Rakenduse aluseks sai võetud avaliku *BSD* litsentsiga *WebTransport API* näidisprogramm. Näidisprogramm oli lihtsustamise eesmärgil modifitseeritud, sellest said välja lõigatud ühesuunaliste ja kahesuunaliste voogude toetamine. *UnidirectionalStreamsTransport* ja *BidirectionalStreamsTransport* liideste puhul andmed edastatakse järjekorras ning seda tuleb ülalnimetatud põhjustel vältida latentsustundlike rakenduste arendamisel. Pärast modifitseerimist jäid alles vaid datagrammide saatmisega ja vastuvõtmisega tegelevad osad.

Heli töötlemiseks autor otsustas kasutada *Web Audio API AudioWorklet* liidest, kuna see võimaldab jooksutada heli töötlusega seotud koodi eraldi reaalaja lõimes [34] . Sedasi saab rakendada osa helitöötlust otse kliendi brauseri peal, kuna töötluse protsess ei sega ega blokeeri *Javascript'*i pealõimu. Selline disain sai valitud põhjusega parandada rakenduse jõudlust ning lisada tulevikukindlust. Arendatud rakenduses *AudioWorklet* kasutatakse *PCM* heliandmete kodeerimiseks ning dekodeerimiseks enne ja pärast edastamist.

AudioWorklet töötleb korraga 128 kaadrit heliandmeid. Kaader, inglise keeles *frame*, on kõikide heli kanalite diskreetide kogum, mis kirjeldavad kindla ajahetke [35] . Diskreet, inglise keeles *sample*, on analoogsignaali hetkeväärtus, mis saadakse sellele signaalile lühikese diskreetimisimpulsi rakendamisel.

*AudioWorklet'*i kõvakooditud 128 puhvri suurus on heli voogedastuse rakenduste jaoks limiteeriv faktor, kuna ülimadala latentsuse saavutamiseks läheb kasuks programmi puhvri suuruse muutmise võimekus. Puhvri suurust peab muutma selleks, et reguleerida kui kiiresti hakatakse tegelema kogutud heli *PCM* andmete töötlemisega. Mida väiksem puhvri suurus, seda kiiremini see täitub ning seda kiiremini saab hakata puhvri töötlema,

kodeerima ning üle võrgu saatma. Suurte puhvrite puhul on olukord vastupidine. Digitaalses helis üks enim kasutatavatest diskreetimissagedustest on 44,100 Hz [36] , mida võib kohtuda heli kandvates CD-plaatides. Selleks, et seda oleks lihtsam ette kujutada, sellise helikvaliteediga töötades 128 kaadrit on umbes 2.9 millisekundit:

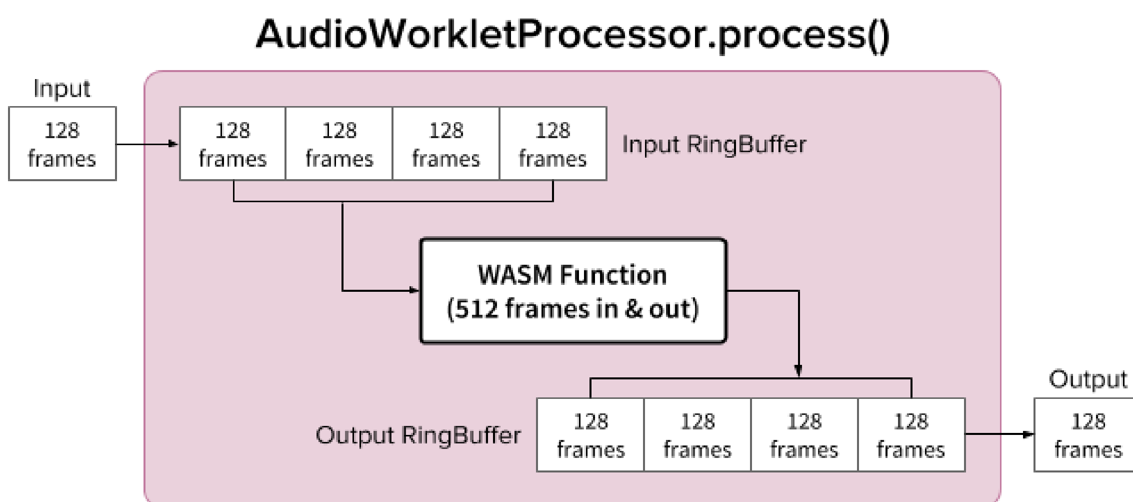
$$128/44100 \approx 0.0029\,s \approx 2.9\,ms$$

Kui hakata saatma veebibrauserist serverisse üle võrgu iga sellise suurusega puhvri eraldi, siis klientide arvu suurendamisega serveri võrk saab väga kiiresti läbustatud. Sekundis peab saatma 344,53 UDP andmepaketi igale poole:

$$44100/128 \approx 344{,}53$$

Kui puhvri suurus oleks 512 kaadrit, tähendaks see juba 86,13 andmepaketi sekundis, mis oluliselt vähendab võrgusõlmede koormust aga suurendab heli latentsuse aega. Samuti võivad esineda ka vastupidised soovid. Näiteks kohtvõrgus, kus ribalaius ei ole limiteeriv faktor, kliendil peab olema võimalus seadistada väiksemad puhvrisuurused latentsuse vähendamiseks. Igal juhul arendatavas rakenduses peab olema võimalus muutma üle võrgu saadetava puhvri suurust.

Seda probleemi aitab lahendada ringpuhver, mida nimetatakse ka rõngapuhvriks või *FIFO'*ks. Joonisel 8 on *AudioWorkletProcessor*'i skeem, milles kasutatakse kahte rõngapuhvrit *WASM* funktsiooni jaoks, mis võtab 512 kaadrit sisse ja välja.



Joonis 8. *AudioWorkletProcessor*'i skeem, milles kasutatakse kahte rõngapuhvrit *WASM*-funktsiooni jaoks, mis võtab 512 kaadrit sisse ja välja [37] .

Arendatav lahendus inspireeritud *GoogleChromeLabs* näidisprogrammist [38] ning *NPM* koodihoidlas olevast *wasm-ring-buffer* pakendi näidisprogrammist [39] .

WASM, ehk *WebAssembly*, on *AudioWorkletProcessor*'i jaoks suurepärane kaaslane. Nende kahe funktsiooni kombinatsioon toob veebi helitöötlusse mitmeid eeliseid, kuid kaks suurimat eelist on olemasoleva *C/C++* helitöötluskoodi viimine *Web Audio* ökosüsteemi ja *Javascript JIT* kompileerimise üldkulude ja mälukoristuse vältimine helitöötluskoodis.

Web Audio API abil saadud heli *PCM* andmeid kodeeritakse *G.711 A-Law* helikodekiga. *A-Law* kodeerimine võimaldab vähendada üle võrgu saadetavate andmete hulga [40] .

Kindlasti on väärt mainida ka asjaolu, et heli sisendi hõivamiseks ja kodeerimiseks ning heli dekodeerimiseks ja taasesitamiseks on kasutatud kahte *AudioWorklet*'i kahe üksteisest sõltumatu puhvritega.

Lõpuks sai ühendatud *Web Audio API* ja *WebTransport API* ühte programmi, mis andis võimeka platvormi edaspidiseks helirakenduse arendamiseks. Samuti on väärt mainimist lahenduse toetamiseks vajaliku infrastruktuuri lihtsus, mis annab eelise teiste lahenduste ees. Süsteemi serveri pool võib olla majutatud iga serverimajutuse teenustepakkuja juures kuna server ei vaja spetsiaalseid seadistusi ega *TURN* või *STUN* servereid.

Kliendipoolset veebirakendust on tänapäevaks võimalik käivitada vaid *Google Chrome* põhistel veebilehitsejatel. Teised veebibrauserid veel ei toeta *WebTransport API*'t kuigi funktsioon on saanud positiivset tagasisidet veebiarendajatelt [41] .

5 Lahenduse realiseerimine

Antud peatükis kirjeldatakse lahenduse tehniliste aspektide realiseerimine koodis, tuuakse koodinäited ning kirjeldatakse nende rollid programmis.

5.1 Klient veebirakendus

Püstitatud eesmärgi saavutamiseks, nimelt helivoo edastamiseks üle *WebTransport API*, sai kirjutatud veebirakendus *Javascript* keeles. Rakendusel on kolm peamist funktsiooni:

- *WebTransport* serveriga ühenduse loomise võime;
- Vaikesisendi helivoo üle võrgu edastamise alustamine ning lõpetamine;
- Serverist üle võrgu sissetulnud helivoo taasesitus vaikeväljundi kaudu.

5.1.1 Ühendamine serveriga

Aluseks sai võetud *WebTransport API* näidiskood litsentsiga *Apache 2.0.* Näidis oli modifitseeritud, et toetada ainult probleemi lahendusega seotud funktsionaalsust. Kõik üleliigne funktsionaalsus oli koodist välja jäetud ning alles jäi ainult datagrammidega töötlev osa.

Ühendamiseks on kasutatud *connect()* funktsioon, mis loob uue *WebTransport()* objekti. Töö kirjutamise ajal *WebTransport()* objekt oli eksperementaalne ning *Chromium*-põhinevatel veebibrauseritel juurdepääsetav ainult *Chrome* päritolukatsete konsooli kaudu saadud tunnuse abil [42] . *WebTransport()* objektile saadetakse korrektne serveri *URL*. Ühenduse serveriga luuakse siis, kui server aktsepteerib ühenduse.

```
async function connect() {
  const url = document.getElementById('url').value;
  try {
    var transport = new WebTransport(url);
    addToEventLog('Initiating connection...');
  } catch (e) {
    addToEventLog('Failed to create connection object. ' + e,
'error');
    return;
  }
}
```

Joonis 9. *connect()* funktsiooni näinäide.

Peale ühenduse loomist *connect()* funktsioon algab sissetulevate datagrammide kuulamise protsessi funktsiooniga *readDatagrams()* ning kutsub funktsiooni *sendDatagrams()*, mis hakkab saatma andmed serverile.

5.1.2 Vaikesisendi helivoo hõivamine ja saatmine

Vaikesisendi helivoo hõivamine toimub funktsioonis nimega *sendDatagrams()*. Selle põhiline eesmärk on luua soovitud parameetritega heli konteksti, kust võetakse vaikesisendi heli *PCM* andmevoo ning edastatakse selle *microphone-worklet-processor* nimelisele *AudioWorkletNode*’ile.

AudioWorklet on eraldiseisev liides mis võimaldab jooksutada heli töötlusega seotud koodi eraldi reaalaja lõimes. Antud juhul *AudioWorkletNode* tegeleb heli *PCM* algandmete kodeerimisega *G.711 A-Law* helikodeki abil. Näide on toodud lisas 3.

Kodeeritud andmed pannakse *SharedArrayBuffer()* puhvrisse, mida saab kasutada jagatud mällu vaadete loomiseks [43] . *SharedArrayBuffer* mängib funktsionaalsuse toimimises kriitilist rolli. Vaatamata sellele, et nii *Worker* kui ka *AudioWorkletProcessor* on varustatud asünkroonse sõnumsidega *MessagePort,* pole see reaalaja helitöötluseks kõige optimaalne lahendus korduva mälu jaotamise ja sõnumside latentsuse tõttu. Seega võimaliku latentsuse likvideerimiseks eraldame ette mäluploki, millele pääseb kiirelt juurde mõlemast lõimest kahesuunaliseks andmeedastuseks. Näide on toodud lisas 2.

Viide *SharedArrayBuffer*’ile *AudioWorkletNode*’is kodeeritud andmevooga saadakse *Javascript* pealõimele *MessagePort* liidese kaudu ning saadetakse serverile *WebTransport API DatagramWriter’*i abil.

5.1.3 Sissetuleva helivoo taasesitamine

Ennekõike luuakse uue heli konteksti nimega *speakerAudioContext*. Siis luuakse *SpeakerWorklet*, mis kujutab endast *speaker-worklet-processor* nimelist *AudioWorkletNode’*i. Seda kasutatakse helivoo dekodeerimiseks ning taasesitamiseks arvuti vaikeväljundi kaudu.

Serverist tulnud andmete püüdmiseks kasutatakse *readDatagrams()* funktsiooni. Andmed pannakse *SharedArrayBuffer()* puhvrisse ning viide puhvrile edastatakse ülalnimetatud *Worklet*’ile.

Enne heli taasesitamist tuleb kõigepealt andmed dekodeerima. Andmevoo dekodeerimine toimub *SpeakerWorklet*’is *G.711 A-Law* helikodeki abil. Pärast dekodeerimist helivoo edastatakse otse *speakerAudioContext’*i vaikeväljundisse. Näide on toodud lisas 4.

5.2 Serveri rakendus

Serveri rakenduse realiseerimiseks sai samuti taaskasutatud *WebTransport Python*’i näidisrakendus litsentsiga *Apache 2.0* [33] , kust oli eemaldatud kõik ülesanne lahendamiseks mittevajalik funktsionaalsus. *WebTransport* ühendused on hallatud *QuicTransportProtocol* klassis.

Server kuulab *UDP* porti 4433. Ühenduse protokoll kasutab *TLS’*i, mis pakub liikluse konfidentsiaalsust ja terviklikkust.

Meie helirakenduse loogika, ehk andmepakettide peegeldamine tagasi saatjale, on rakendatud *AudioStreamHandler* klassis, mis asendab näidisrakenduses olevat *CounterHandler* klassi. Näide on toodud joonisel 10.

```
class AudioStreamHandler:
    def __init__(self, connection) -> None:
        self.connection = connection

    def quic_event_received(self, event: QuicEvent) -> None:
        if isinstance(event, DatagramFrameReceived):
            self.connection.send_datagram_frame(event.data)
```

Joonis 10. Serverirakenduse *AudioStreamHandler* klassi näinäide.

6 Tulemuste analüüs

6.1 Lahenduse testimine ja analüüs

Teostatud lahendus saab hästi hakkama heli voogedastusega. Kontseptsioon sai tõestatud, heli edastamine kodeeritud *WebTransport API UDP* datagrammidega on võimalik ja töötab.

Testimise ajal tulid välja mõned probleemid realiseerimisega. Valede puhvri suuruste ja diskreetimissageduste valimisel kannatab heli kvaliteet. Liiga väikeste või liiga suurte puhvrite puhul ilmuvad heli moonutused ja müra. Diskreetimissagedus samuti peab olema võrdne kasutaja arvuti helikaardi vaike sagedusega, et vältida võimalikke jõudluse üldkulusi diskreetimissageduse konverteerimisest.

Tuleb samuti pidada meeles, et kuna iga puhver võrdub ühe *UDP* datagrammiga, siis puhvri suurus ei tohi ületada serveri poolt seadistatud *MTU* väärtust, muidu server lihtsalt ei võta andmepaketi vastu. Samuti, kuna minimaalsete *MTU* väärtused on *IPv4* puhul 576 ning *IPv6* puhul 1280 baiti, tasub hoida *UDP* datagrammide suurust selles piires vastavalt valitud võrguprotokollile. Sellega saab vältida *UDP* andmepakettide segmenteerimist ning parandada andmevoo tõhusust.

See fakt paneb selged piired puhvri suuruse ja diskreetimissageduse seadetele. Puhvri suurus peab olema arvutatud automaatselt klient veebirakenduse poolt võttes arvesse kasutaja arvuti helikaardi omadusi ja võrgu ühenduse kvaliteedi. Nende parameetrite kasutajale kättesaadavaks tegemine tekitab segadust ning toob kaasa helivoo kvaliteedi alandamist.

6.2 Uue lahenduse latentsuse analüüs

Paremaks latentsuse analüüsiks on õigem kasutada veebilehitsejatesse sisseehitatud jõudluse mõõtmise funktsionaalsust. *WebRTC* puhul on *Chromium*-baseerivatel brauseritel olemas tööriist *chrome://webrtc-internals*, mis pakub monitooringu mõõdikuid latentsuse põhjalikuks analüüsiks. *WebTransport API* jõudluse mõõtmise jaoks sarnast tööriista pole veel olemas, seega granulaarne latentsuse analüüs osutub võimatuks. Võrgu edastamise latentsus sai mõõdetud otse programmis, kasutades reaalaja mõõtmiseks *Javascript*'i *performance.now()* funktsiooni [44] .

Heli töötlemises kõige tähtsamad mõõdikud on heli sisendi hõivamise, kodeerimise, dekodeerimise ning helivoo taasesitamise kiirused. Nende mõõdikute summat saab nimetada veebibrauseri kohalikuks edasi-tagasi helivoo latentsuseks. *Web Audio API* pakub *AudioContext* liidese kirjutuskaitstud atribuuti *baseLatency*, mis näitab töötlemisviivituse arvu sekundites, kui kaua *AudioContext* kannab *AudioDestinationNode'*ilt helipuhvrit kasutaja arvuti heli alamsüsteemile taasesitamiseks [45] . Seda ainsat mõõdikut aga ei piisa täieliku ning tõelise pildi saamiseks.

Helivoo tõelise latentsuse mõõtmiseks autor genereerib lühikesi heliimpulssi arvuti kõlaritest, neid impulssi hõivab mikrofon ning impulssi genereerimise ja selle hõivamise ajalist vahet peetakse helisisendi hõivamise latentsuseks.

Seejärel heli kodeeritakse, impulssi edastatakse serverisse, serverist selle peegeldatakse tagasi ning veebirakenduses dekodeeritakse. Selle impulssi tagasi kätte saades saame arvutada kodeerimise, dekodeerimise ning võrgu edasi-tagasi andmevoo latentsust.

Iga sammu mõõtmisel kasutatakse *Javascript performance.now()* funktsiooni.

6.2.1 Kohaliku arvuti heli *RTT* mõõtmise metoodika

Tõelise kohaliku latentsuse mõõtmise metoodika:

1. Genereeritakse ning esitatakse määratud intervalliga heliimpulssi arvuti kõlaritest, vahetult peale seda salvestatakse heliimpulssi esitamise aega *t1*;

2. Püüakse mikrofoniga esitatud heliimpulssi analüüsides mikrofoni helivoo kaadrid. Salvestatakse aeg *t2* kui sisendi heli andmevoos tuvastatakse heliimpulss. Heliimpulssi tuvastamiseks vaadatakse helitugevust. Impulssi hetkel sisendi helitugevuse keskmine väärtus on suurem ning impulss loetakse püütuna, kui helitugevuse väärtus ületab ette seadistatud väärtust. Lävendiks on kasutatud *-12 dB* piir;

3. Kahe ajamõõdiku vahe *t2 - t1* võrdub kohaliku heli *RTT*'ga. See aeg näitab tõelist heli hõivamisele ning taasesitamisele kulutatud aega.

6.2.2 Heli voogedastamise võrgu *RTT* mõõtmise metoodika

Helivoo üle võrgu saatmise aega mõõtmise metoodika:

1. Ette määratud intervalliga salvestatakse veebirakenduses üks täidetud puhver vahetult enne selle edastamist üle võrgu *UDP* datagrammina. Salvestatakse aeg *t1*;

2. Serverirakendus peegeldab kõiki sissetulnud pakette tagasi saatjale;

3. Veebirakendus võrdleb iga sissetulnud andmepaketti salvestatud puhvriga. Kui sissetulnud puhvri datagramm võrdub salvestatud näidisega, salvestatakse aeg *t2;*

4. Kahe ajamõõdiku vahe *t2 - t1* võrdub helivoo võrgu *RTT*'ga. See aeg näitab kui palju aega võtab datagrammidel, et ületada edasi-tagasi teekonna kasutaja ja serveri vahel.

6.2.3 Testimises kasutatud seadmed

Testimiseks oli server majutatud kahes asukohas:

1. Eestis, Tallinnas aadressil Ädala 29 olevas Zone Media LLC serveripargis, *ping* < 5 ms;

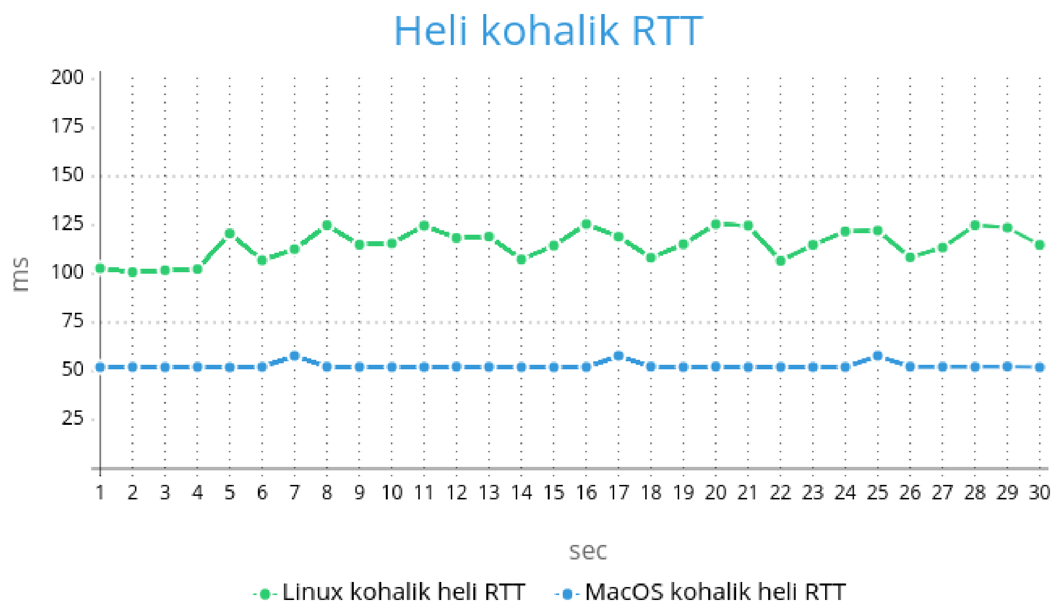
2. Hollandis, Amsterdamis aadressil Duivendrechtsekade 80A olevas Zone Media LLC serveripargis, *ping* < 38 ms.

Testimine oli korraldatud kahe arvutiga:

1. HP Elitebook 840 G5, protsessor Intel® Core™ i5-8350U, operatsioonisüsteem Ubuntu 20.04.2 LTS, veebibrauser Chromium Version 90.0.4430.72;
2. Apple iMac Retina 5K 2019, protsessor Intel® Core™ i5-8600, operatsioonisüsteem macOS Big Sur 11.2.2 (20D80), veebibrauser Chrome Version 89.0.4389.128.

6.2.4 Helivoo edastamise aeg kohtvõrgus

Kohalikus võrgus püstitatud serveri ja kliendi vaheline *ping* jääb alla 1 ms. See annab võimalust keskenduda välja arendatud süsteemi jõudluse testimise peale, ilma võrgutransporti üldkuludeta.



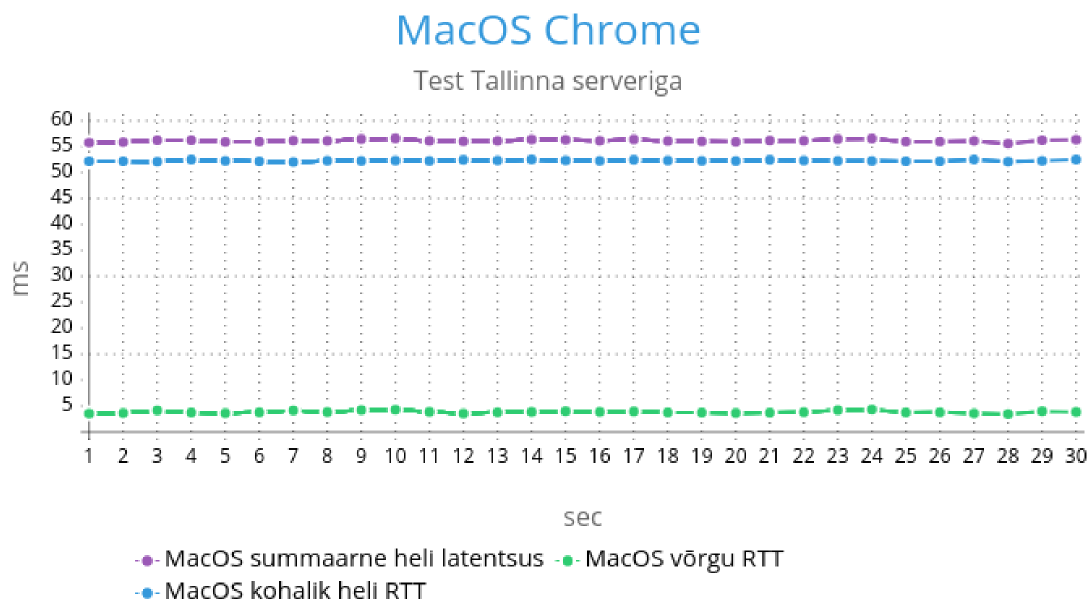
Joonis 11. *MacOS* ja *Linux* operatsioonisüsteemide erinevus.

Testimine näitas, et heli latentsus sõltub suurel määral operatsioonisüsteemilt. *MacOS* operatsioonisüsteem suudab heli töödelda peagi kaks korda kiiremini, kui *Linux* operatsioonisüsteem. Testimise ajal *CPU* ei olnud ülekoormatud. See tähendab, et tõenäoliselt latentsuse erinevus sõltub veebibrauseri võimekusest efektiivselt kasutada helidraiverid ning operatsioonisüsteemi madala taseme liidesed.

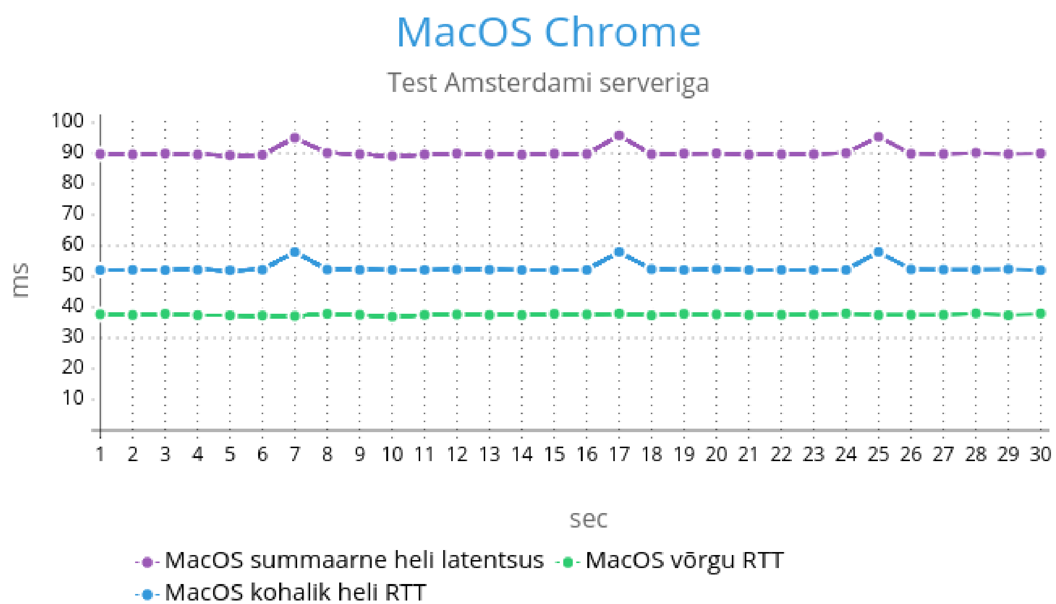
Testi tulemus samuti näitab, et kaasaegsete veebibrauserite *Web Audio API* liides ei ole veel piisavalt arenenud, et pakkuda ülimadala latentsuse heli salvestamisel ja taasesitamisel.

6.2.5 Helivoo edastamise aeg üle interneti

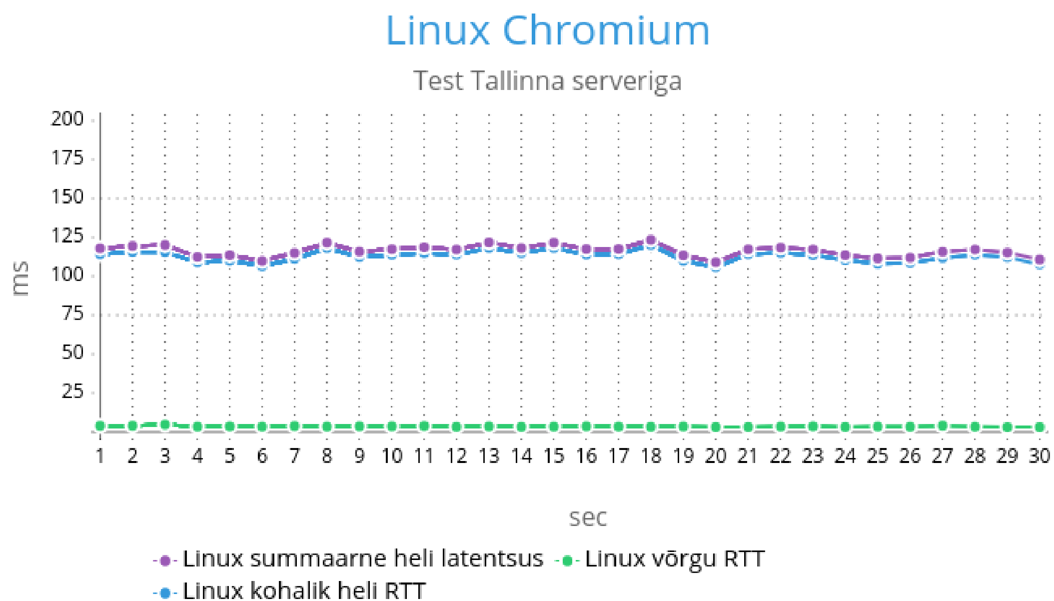
Testid näitavad, et kõige suurem ja olulisem kulu tuleb kohalikust *Web Audio API* jõudlusest. Heli andmepakettide edastamine üle võrgu on väga sarnane *ping* ajaga. See tähendab, et Eestis majutatud serveriga on võimalik saavutada võrguedastuse aega alla 5 ms. Töös püstitatud eesmärgi kontekstis, see annab 25 ms heli töötlemiseks serveris ja kliendi juures.



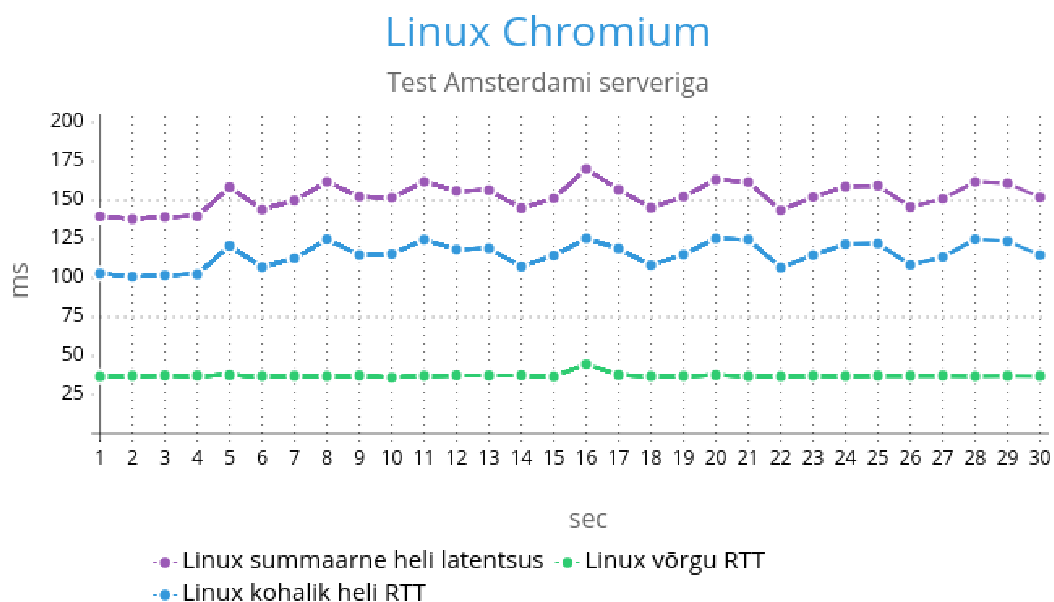
Joonis 12. *MacOS* test Tallinna serveriga.



Joonis 13. *MacOS* test Amsterdami serveriga.



Joonis 14. *Linux* test Tallinna serveriga.



Joonis 15. *Linux* test Amsterdami serveriga.

Kahjuks tuli välja, et standardne, modifitseerimata Chrome veebibrauser *Web Audio API* liidesega ei ole tänaseks võimeline pakkuda piisavalt head jõudlust, et saavutada nõutud heli kohaliku edasi-tagasi edastamise kiirust.

6.3 Uue lahenduse kasutatavuse analüüs

Veebirakenduse kasutaja vaatenurgast lahendus hakkab olema mugavam siis, kui osad *Web Audio API* funktsioonidest ja *WebTransport API* väljuvad eksperimentaalsest olekust. Siis rakendust saab hakata kasutama iga kaasaegse veebibrauseriga. Senikaua rakendus töötab vaid *Chromium* põhinevatel veebibrauseritel.

Veel üheks segadust tekitavaks faktoriks on heli puhvri suuruste ja diskreetimissageduste konfigureerimise võimalus. Kuigi see annab kasutajale paremat kontrolli tehniliste aspektide üle, funktsionaalsus tekitab segadust ning lisab keerukust kuna nõuab kasutajalt teadmisi valdkonnas. Valede parameetritega heli kvaliteet langeb või heli lihtsalt kaob.

6.4 Lahenduse turbeanalüüs

QUIC põhiprotokoll on vaikimisi täielikult krüpteeritud, välja arvatud avalike lippude bait, 8-baidine ühenduse *ID* ja pakettide järjekorranumber [46] . Ühendus serveriga kasutab *TLS* protokolli andmete krüpteerimiseks, mis pakub liikluse konfidentsiaalsust ja terviklikkust. *WebTransport API* kasutab *TLS*'iga sama sertifikaadi kontrollimehhanismi, tuginedes seega kaugserveri autentimiseks samale avaliku võtme infrastruktuurile. *WebTransport*'is on sertifikaatide kontrollimise vead saatuslikud ning ei luba mööda minna sertifikaadi valideerimisest.

Server on alati teadlik, et kõnealune ühendus pärineb veebirakendusest. See on vajalik protokollidevaheliste rünnakute vältimiseks. *WebTransport* kasutab selleks *ALPN*'i [47] .

6.5 Tulemuse võrdlus tänapäeva turul olevate süsteemidega

Kuna teostatud lahendus vaikimisi tugineb *TLS* krüpteerimise peale, datagrammidele põhinevate rakenduste arendamine on arendajate vaatenurgast nüüd kordades lihtsamad. Võrreldes teiste datagrammide üle võrgu saatmise võimalustega, nagu näiteks *WebRTC*, *WebTransport*'i lähenemine pakub kerget alternatiivi helirakenduste toetamiseks seansside ja andmevoogude turvalisust kahjustamata.

Välja töötatud rakendus toetab *IPv6*, mida ei saa öelda mitmete analüüsitud rakenduste kohta. Datagrammidele tuginevatele rakenduste kontekstis *IPv6* eelis on suuremate segmenteerimata pakettide toetamine võrreldes *IPv4* protokolliga. Kui pidada meeles, et välja töötatud rakenduse disainis üks *UDP* datagramm võrdub ühele täitunud puhvrile, *IPv6* pakub võimalust kasutada suuremaid puhvri suurusi mis võib parandada heli kvaliteedi aeglasematel arvutitel või võrkudel. Mida suurem on heli puhvri suurus, seda rohkem aega on arvuti protsessoril heli töötlemiseks, salvestamiseks ja taasesitamiseks.

Järgmiseks eeliseks üle teiste analüüsitud platvorme on lihtne infrastruktuur rakenduse toetamiseks. Heaks näiteks on *WebRTC* tehnoloogial tuginevad rakendused. *WebRTC* oli projekteeritud pidades silmas *P2P* lahendusi. Tavaline *WebRTC* tehnoloogiale tuginev infrastruktuur koosneb nii rakenduse serverist, kui ka *TURN* või *STUN* serverist mis raskendab süsteemi seadistamist ja haldust. *WebRTC* klient-server rakenduse arendamisel aga võib tekkida tunne, et lihtsa probleemi lahenduseks, nimelt datagrammide edasi-tagasi saatmiseks, tuleb oma projekti lisada sõltuvust ebamõistlikult suurele koodikogule ning kasutada vaid selle murdosa. Siin on selgelt näha miks *WebTransport* võib olla väga atraktiivseks tehnoloogiaks arendajatele, kes töötavad lihtsate veebirakenduste peal ning kes tahavad vältida projekti üleliigset keerukust. *WebTransport*'i puhul infrastruktuur koosneb ühest rakenduse serverist. Teostatud näidisrakenduse puhul on tegemist *Python* serveriga *aioquic* [48] teekiga.

7 Takistused ja piirangud

7.1 Põhilised takistused

Rakenduse ja lõputöö kirjutamise ajal *WebTransport API* liides on olnud ligipääsetav ainult *Chromium'*il põhinevatel veebibrauseril. Testimine oli läbi viidud *Chromium* brauseris. *Safari*, *Firefox*, *Opera* ning muud veebibrauserid veel ei toetanud antud funktsionaalsust.

Testimise ajal selgus, et põhiliseks takistuseks ülimadala latentsuseni jõudmiseks on *Web Audio API*, mille helitöötlemise kiirus jätab soovida paremat. Kõige põhilisemad funktsioonid, nagu mikrofoni heli hõivamine ning helivoo taasesitamine võtab liiga palju aega, et sobida muusikutele orienteeritud rakendusele. *Web Audio API'*ga tekitav latentsus ületab maksimaalse sobiliku latentsuse piiri mitmekordselt. Ainus asi, mida saab latentsuse vähendamiseks teha, on sundida brauserit keelama kaja tühistamist ning müra vähendamist, kuid selle kasu on minimaalne. Selliselt, isegi helisisendi ühendamine otse heliväljundiga kohalikus veebibrauseris, helivoo üle võrgu saatmata, tekitab liiga suurt latentsust, mis ei sobi helisalvestuse ega muu taolistele helitöötlus rakendustele.

Puhvritega töötamine samuti tekitab raskusi, kuna *Web Audio API* ja selle *AudioWorklet* liides ei toeta puhvrite suuruste muutmist. *AudioWorklet* vaikimisi töötleb täpselt 128 kaadrit korraga ning seda saab pidada puuduseks kui rakenduse väljatöötamisel on vajalik paindlikus puhvri suuruste valimisel. Samuti on oluline välja tuua võimetus integreerida *WASM'*i puhvri otse *AudioWorklet'*i süsteemi [49] .

Web Audio API disaini segamine *WebAssembly'*misega võib olla keeruline *WASM'*i kuhjaga seotud mäluhalduse tõttu. *WASM'*i sisenevad ja sealt väljuvad andmed tuleb pidevalt kloonida ning kuigi mäluhalduse lihtsustamiseks saame kasutada klassi *HeapAudioBuffer*, lisab antud protsess jõudluse üldkulusid. Tulevikus arutletakse mõtet kasutada kasutaja eraldatud mälu üleliigse andmete kloonimise vähendamiseks [50] .

Võttes arvesse fakti, et helisisendi hõivamiseks ja heli taasesitamiseks on kasutatud kaks eraldi *AudioWorklet*'i, ehk heli hõivamiseks ja taasesitamiseks kasutakse kahte üksteisest sõltumatut puhvrit, võib ülal kirjeldatud probleemi mõju korrutada kahega.

Kindlasti on väärt mainimist *SharedArrayBuffer'*i kasutamise fakt, mis nõuab rist päritolu eraldamist – C*ross-origin isolation.* See keelab veebilehele päritolupärast sisu laadida, kui ressurss seda selgesõnaliselt ei võimalda selliste *CORS'*i päiste abil nagu „*Access-Control-Allow-**" [51] .

7.2 Lahenduse piirangud

Rääkides veebirakendusest muusikutele, mille abil saaksid inimesed mängida muusikat koos reaalajas üle veebi, kõige suuremaks piiranguks jääb süsteemi kohandamine. Testid näitasid, et *WebTransport*'i abil helivoo üle võrgu saatmise *RTT* on praktiliselt võrdne *ping* ajaga mis tähendab, et *WebTransport* ei lisa suurt kulu andmevoo töötlemiseks, s.h krüpteerimiseks ja dekrüpteerimiseks. See on suurepärane tulemus ning kiirte võrkude puhul võimaldab saavutada reaalaja heli voogedastust. Testides Eestis olles aga ühendust serveriga, mis on majutatud Saksamaal, oli näha *UDP* pakettide *RTT* latentsust 45 ms kanti. Muusika maailmas see tekitab juba märgatava kaja heli taasesitusel.

Platvormi kohandamisel peab alati pidama meeles, et selline lähenemine ei võimalda erinevates regioonides asuvate inimeste ühendamist ühte seansi. Selleks, et muusika mängimise seanss toimuks, peale head internetiühendust peavad kõik seanssi liikmed olema serveriga füüsiliselt lähedal ehk samas geograafilises regioonis.

8 Takistuste võimalikud tulevased lahendused

8.1 Helisisendi hõivamise parandamine

Ootamatult eesmärgi saavutamiseks kõige suuremaks takistuseks oli *Web Audio API* liides, mille heli hõivamise ja taasesitamise kiirused on liiga suured selleks, et kasutada antud liidest latentsus tundlikes helirakendustes. Järgmine süsteemi parandamise samm peab olema töö veebilehe külastaja arvuti helikaardi sisend-väljundiga.

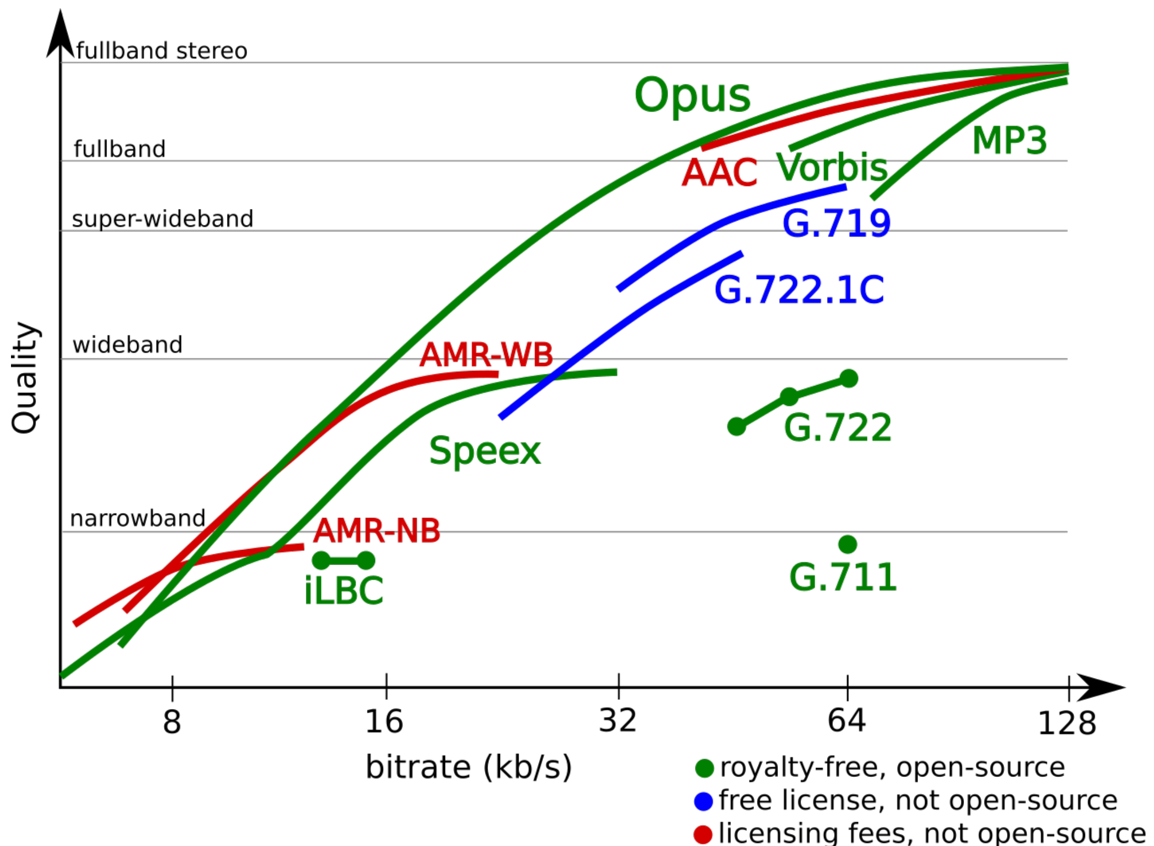
Professionaalsed digitaalse helirakendused jätavad tavaliselt arvuti süsteemi kõrgetasemelised *API*'d vahele ja kasutavad *ASIO* või muid kohandatud draiveriliideseid, nii et nad ei pea muretsema selle üle, kuidas operatsioonisüsteem käsitleb helivoo latentsust. Samuti kaasaegsetel analoog-digitaal konverteritel on hästi kirjeldatud väljendid ning operatsioonisüsteem ei pea mikrofoni sisendis järeltöötlust läbi viima. Piltlikult ette kujutades, professionaalsed helirakendused on võimelised töötada helisisenditega otse helidraiveri kaudu jättes vahele kõiksugused operatsioonisüsteemi järeltöötlused. Tänapäeval veebibrauseritel sellist võimekust pole.

Kõige olulisem on nüüd uurida võimalust kasutada *WASM*'i heli sisend-väljundi töötlemiseks ning muu võimalusi saada ühendust otse helidraiveriga, möödudes *Web Audio API* liidest. Lahendus aga peab toetama kõike enamkasutatud veebibrausereid ning operatsioonisüsteeme.

Üks võimalustest oleks lisada veebibrauseritesse *Jackaudio* tugi. *Jackaudio* serveri tugi lahendaks latentsusprobleemid, ühenduvusprobleemid, toetatud seadmete ning litsentsimise probleemid *MacOS*, *Windows* ja *Linux* operatsioonisüsteemidel. Samuti toetab *Jackaudio* kõiki peamisi draivereid: *ASIO, ASLA, WASAPI, Direct Sound, MME.* Kahjuks, veebibrauserite arendate prioriteetide nimekirjas *Jackaudio* korralik rakendamine ei ole esiridades [52] [53] .

8.2 Heli kodeerimise paremaks muutmine

Näidisrakenduses on kasutatud avatud lähtekoodiga *G.711 A-Law* helikodek. *A-Law* on triviaalne kodek mis sai kasutatud kontseptsiooni tõestuseks oma lihtsuse pärast. Helikvaliteedi parandamiseks tasub võtta kasutusele *Opus* helikodeki. *Opus* on kadudega helikodeerimise formaat mis on võimeline pakkuda kaadri suurust vahemikus 2,5 ms kuni 60 ms. Selle tõttu sobib see ideaalselt reaalaja latentsus tundlike helirakenduste jaoks.



Joonis 16. Erinevate koodekite kvaliteet bitikiiruse funktsioonina [54] .

Töö kodekitega saab samuti optimeerida uue eksperementaalse *WebCodecs API* tehnoloogia abil [55] , mis võimaldab arendajatele enda koodekite rakendamist vahele jätta ning kasutada veebibrauserites sisseehitatud kodekid.

8.3 Targad puhvrid

Veebirakenduse kasutatavuse parandamiseks võib lisada loogikat, mis otsustaks ise millist puhvri suurust tuleb millistel ühendustel ja millistel arvutitel kasutada. See vabastab kasutajaid vajadusest seadistada puhvri suuruse iseseisvalt, kuna puhvri suuruse arvutamine nõuab teatud tehnilist teadmist. Kui veebirakendus oskaks seadistada puhvri suuruse automaatselt võttes arvesse helikaardi ja võrguühenduse omadusi, see lihtsustaks rakenduse kasutaja tööd ning parandaks kasutajakogemust.

Samuti tasub analüüsida võimalusi puhvrite optimeerimiseks, kuna hetkel heli töötlemise ajal mõnikord ilmub müra erinevate puhvrite suuruste ja diskreetsageduse seadistamisel. Üheks võimalikuks lahenduseks oleks rakendada puhvrite järjekorda. Põhimõte oleks panna lehe külastaja veebibrauserit võtma heli esitamiseks *PCM* andmete puhvrit järjekorrast, kus igal hetkel saab olla ette määratud puhvrite arv. Selliselt, programmi täitmise ajal ei teki heli *PCM* andmete puudust, mis aitab kõrvaldada võimalikke heliartefakte.

8.4 Kadunud andmepakettide haldamine

Andmete voogedastamisel *UDP* kaudu võib juhtuda, et mõned datagrammid lähevad lihtsalt kaduma. Reaalaja heli voogedastamise rakenduste iseloomujoon on võime jätkata katkematult tööd datagrammi rikkumise või täieliku mitte ilmumise puhul. Kasutaja vaatepilgu pealt selline paketikadu näeb välja kui heli hakimine või moonutamine. Näiteks 512 baiti puhvri suurusega diskreetsagedusega 44.1 kHz, kadunud andmepaket võrdub 11.6 ms kadunud helivooga:

$$512/44100 \approx 0.0116\,s = 11.6\,ms$$

Rakenduse edaspidisel arendamisel tasub uurida võimalusi heli hakkimise leevendamiseks paketikadu puhul.

9 Kokkuvõte

Lõputöö eesmärgiks on olnud uurida võimalusi ning pakkuda lahendus heli voogedastamise latentsuse vähendamiseks veebibrauseri rakendustes. Lisaks heale jõudlusele arendatav lahendus peab pakkuma andmete konfidentsiaalsust ja terviklikkust üle interneti edastamisel. Pakutud lahendus peab sobima arendajatele, kelle soov on leida lihtsat ja kerget asendust *WebRTC* raamistikule.

Probleemi lahendamiseks on autor teinud tänapäeva turul olevate lahenduste analüüsi. Selle analüüsi abil kogutud informatsioon levinumatest probleemidest oli võetud arvesse uue lahenduse projekteerimisel ning tehnoloogia valimisel. Uue lahenduse välja töötamisel oli eesmärk keskenduda veebibrauseri põhistel tehnoloogiatel. Põhirõhk on olnud heli voogedastamise latentsuse vähendamise ning veebirakenduse kasutatavuse peal.

Tulemuseks on arendatud veebirakendus mis on võimeline turvaliselt ja väikse ajakuluga edastada kasutaja vaikesisendi helivoo serverisse, võtta serverist sisse tulnud helivoo vastu ning taasesitada selle vaike heliväljundi kaudu. *WebTransport API* võimaldab mitteblokeerival viisil andmete turvalist edastamist üle interneti üliväikese aja lisakuluga. Testid näitavad, et tehnoloogia sobib suurepäraselt voogedastamiseks üle võrgu ning lihtsate helirakenduste puhul saab asendada *WebRTC* raamistiku.

Samuti selgus, et tänapäevaste veebibrauserite *Web Audio API* liides ei ole veel piisavalt arenenud, et toetada ülimadala latentsusega helirakendusi. Eesmärk on olnud saada heli latentsust 30 ms piiril. Testid näitasid, et heli kohalik edasi-tagasi teekonna aeg on heal juhul 10 - 20 ms pikem püstitatud eesmärgist. See tähendab, et ülimadala latentsusega reaalaja helirakenduste toeks peab otsima muid võimalusi töötama arvuti heli sisend-väljundiga.

Lõputöö planeerimise ning projekti arendamise ajal suurim eeldatav takistus on olnud seotud heli andmevoo edastamisega üle interneti. Valmis töötatud lahenduse testimisel selgus, et sellele probleemile sai leitud lahendus ning helivoo edastamine üle võrgu

nüüd lisab vaid marginaalse ajakulu. Nii kaua kui server ja kasutajad asuvad üksteisest füüsiliselt lähedal, reaalaja helirakenduse arendamisel ei pea enam keskenduma võrgu transporti peale. Lõputöö autor on leidnud ja analüüsinud arendatud süsteemi nõrkused ning jõuab järelduseni, et kõige suurem neist on arvuti kohalik heli hõivamine ja taasesitamine.

Autor näeb vajadust töö edasiarenduses ning pakub eesmärgi saavutamise tekkinud takistuste võimalikud lahendused. Lõpliku lahenduse leidmine annab suure hoogu heli veebirakenduste arendajatele ning toob palju kasu mitte ainult heli voogedastamise rakendustele, vaid aitab ka parandada kõike muude reaalaja klient-server-mudeliga rakenduste jõudlust.

Kasutatud kirjandus

[1] “WebRTC – Real-time communication for the web”, [Online]. Available: https://webrtc.org/. [Accessed 17 March 2021].

[2] “Media Source Extensions”, [Online]. Available: https://www.w3.org/TR/media-source/. [Accessed 17 March 2021].

[3] “JackTrip – A System for High-Quality Audio Network Performance over the Internet”, [Online]. Available: https://ccrma.stanford.edu/software/jacktrip/. [Accessed 20 March 2021].

[4] Juan-Pablo Caceres and Chris Chafe, “JackTrip: Under the hood of an engine for network audio”, Center for Computer Research in Music and Acoustics (CCRMA), Stanford University, Nov 2010.

[5] “JACK Audio Connection Kit”, [Online]. Available: https://jackaudio.org/. [Accessed 20 March 2021]

[6] “JackTrip Documentation”, [Online]. Available: https://ccrma.stanford.edu/groups/soundwire/software/jacktrip/. [Accessed 20 March 2021]

[7] “Raspberry Pi”, [Online]. Available: https://www.raspberrypi.org/. [Accessed 25 March 2021].

[8] “JackTrip Virtual Studio”, [Online]. Available: https://help.jacktrip.org/hc/en-us/categories/360004519513-JackTrip-Virtual-Studio. [Accessed 25 March 2021].

[9] “JackStreamer”, [Online]. Available: https://ccrma.stanford.edu/docs/common/JackStreamer.html. [Accessed 25 March 2021].

[10] “Musicians Together Apart”, [Online]. Available: https://musicianstogetherapart.com/. [Accessed 25 March 2021].

[11] “JackTrip WebRTC”, [Online]. Available: https://github.com/jacktrip-webrtc/jacktrip-webrtc. [Accessed 26 March 2021].

[12] “Jamulus – Play music online. With friends. For free.”, [Online]. Available: https://jamulus.io. [Accessed 30 March 2021].

[13] “Getting Started with Jamulus”, [Online]. Available: https://jamulus.io/wiki/Getting-Started. [Accessed 30 March 2021].

[14] “Sonobus – High Quality Network Audio Streaming”, [Online]. Available: https://sonobus.net/. [Accessed 30 March 2021].

[15] "Aloha By Elk”, [Online]. Available: https://alohabyelk.com/. [Accessed 2 April 2021].

[16] “Soundtrap – Make music online”, [Online]. Available: https://www.soundtrap.com/. [Accessed 2 April 2021].

[17] “Audiotool – Free Music Software”, [Online]. Available: https://www.audiotool.com/. [Accessed 2 April 2021].

[18] “JamKazam – Live, In-Sync Music Jamming Over the Internet”, [Online]. Available: https://jamkazam.com/. [Accessed 2 April 2021].

[19] “List of apps that are not working with M1”, [Online]. Available: https://isapplesiliconready.com/for/unsupported. [Accessed 17 April 2021].

[20] “Internet Protocol – Darpa Internet Program Protocol Specification”, p. 12, Information Sciences Institute, University of Southern California, September 1981. [Online]. Available: https://tools.ietf.org/html/rfc791. [Accessed 17 April 2021].

[21] S. Deering and R. Hinden, “Internet Protocol, Version 6 (IPv6) Specification”, p. 24, Internet Engineering Task Force (IETF), July 2017. [Online]. Available: https://tools.ietf.org/html/rfc8200. [Accessed 17 April 2021].

[22] Stefan Schmid, Andrew C. Scott, David Hutchison, and Konrad Froitzheim "QoS-based real-time audio streaming in IPv6 networks", Proc. SPIE 3529, Internet Routing and Quality of Service, 16 December 1998.

[23] “Web Audio API – Browser compatibility”, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API#browser_compatibility. [Accessed 18 April 2021].

[24] “AudioWorklet documentation”, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/AudioWorklet. [Accessed 18 April 2021].

[25] “UDP Versus TCP for VoIP”, [Online]. Available: https://www.onsip.com/voip-resources/voip-fundamentals/udp-versus-tcp-for-voip. [Accessed 18 April 2021].

[26] Glenn Fiedler, “Why can't I send UDP packets from a browser?”, [Online]. Available: https://gafferongames.com/post/why_cant_i_send_udp_packets_from_a_browser/. [Accessed 19 April 2021].

[27] I. Fette and A. Melnikov, “The WebSocket Protocol”, Internet Engineering Task Force (IETF), December 2011. [Online]. Available: https://tools.ietf.org/html/rfc6455. [Accessed 19 April 2021].

[28] Sam Dutton, “Build the backend services needed for a WebRTC app”, [Online]. Available: https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/. [Accessed 19 April 2021].

[29] "WebTransport Working Group Charter”, [Online]. Available: https://www.w3.org/2020/06/proposed-webtransport-charter.html. [Accessed 19 April 2021].

[30] “Experimenting with QUIC”, [Online]. Available: https://blog.chromium.org/2013/06/experimenting-with-quic.html. [Accessed 19 April 2021].

[31] “A QUIC update on Google’s experimental transport”, [Online]. Available: https://blog.chromium.org/2015/04/a-quic-update-on-googles-experimental.html. [Accessed 19 April 2021].

[32] “WebTransport specification, Complete example”, [Online]. Available: https://w3c.github.io/webtransport/#example-complete. [Accessed 19 April 2021].

[33] “Google Chrome Samples, Python QUIC server example”, [Online]. Available: https://github.com/GoogleChrome/samples/blob/gh-pages/webtransport/quic_transport_server.py. [Accessed 13 March 2021].

[34] “Web APIs, AudioWorkletGlobalScope”, MDN Web Docs, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/AudioWorkletGlobalScope. [Accessed 13 March 2021].

[35] “Basic concepts behind Web Audio API. Audio buffers: frames, samples and channels”, MDN Web Docs, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API/Basic_concepts_behind_Web_Audio_API#audio_buffers_frames_samples_and_channels. [Accessed 13 March 2021].

[36] John Watkinson, “The Art of Digital Audio”, 2nd edition, p. 104, September 2002.

[37] Hongchan Choi, “Audio Worklet Design Pattern”, [Online]. Available: https://developers.google.com/web/updates/2018/06/audio-worklet-design-pattern#handling_buffer_size_mismatch. [Accessed 13 March 2021].

[38] “Web Audio API Samples, Ring buffer worklet processor example”, Google Chrome Labs, [Online]. Available: https://github.com/GoogleChromeLabs/web-audio-samples/blob/master/audio-worklet/design-pattern/wasm-ring-buffer/ring-buffer-worklet-processor.js. [Accessed 13 March 2021].

[39] “Web Assembly Ring Buffer package”, [Online]. Available: https://npm.io/package/wasm-ring-buffer. [Accessed 13 March 2021].

[40] H. Schulzrinne and S. Casner, “RTP Profile for Audio and Video Conferences with Minimal Control”, p. 28, Network Working Group, July 2003. [Online]. Available: https://tools.ietf.org/html/rfc3551. [Accessed 13 March 2021].

[41] “Feature: WebTransport”, Chrome Platform Status, [Online]. Available: https://chromestatus.com/feature/4854144902889472. [Accessed 13 March 2021].

[42] "Chrome Origin Trials”, [Online]. Available: https://developers.chrome.com/origintrials. [Accessed 3 April 2021].

[43] “Javascript Standard built-in objects, SharedArrayBuffer”, MDN Web Docs, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/SharedArrayBuffer. [Accessed 3 April 2021].

[44] “Web APIs, performance.now()”, MDN Web Docs, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Performance/now. [Accessed 3 April 2021].

[45] “Web APIs, AudioContext.baseLatency”, MDN Web Docs, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/AudioContext/baseLatency. [Accessed 3 April 2021].

[46] “QUIC Overview”, [Online]. Available: https://peering.google.com/#/learn-more/quic. [Accessed 3 April 2021].

[47] S. Friedl, A. Popov, A. Langley and E. Stephan, “Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension”, Internet Engineering Task Force (IETF), July 2014. [Online]. Available: https://tools.ietf.org/html/rfc7301. [Accessed 3 April 2021].

[48] “Aioquic library for the QUIC network protocol in Python”, [Online]. Available: https://github.com/aiortc/aioquic. [Accessed 3 April 2021].

[49] “AudioWorklets integration with WebAssembly”, Web Audio issue, [Online]. Available: https://github.com/WebAudio/web-audio-api-v2/issues/5. [Accessed 10 April 2021].

[50] “Bring Your Own Buffer style of memory management”, Web Audio issue, [Online]. Available: https://github.com/WebAudio/web-audio-api-v2/issues/4. [Accessed 10 April 2021].

[51] Jake Archibald, “SharedArrayBuffer updates in Android Chrome 88 and Desktop Chrome 91”, [Online]. Available: https://developer.chrome.com/blog/enabling-shared-array-buffer/. [Accessed 10 April 2021].

[52] “Resolve Audio Latency by adding jackaudio server support”, Chromium issue, [Online]. Available: https://bugs.chromium.org/p/chromium/issues/detail?id=451138. [Accessed 10 April 2021].

[53] “JACK audio output support”, Mozilla issue, [Online]. Available: https://bugzilla.mozilla.org/show_bug.cgi?id=783733. [Accessed 10 April 2021].

[54] “Opus Interactive Audio Codec, comparison”, [Online]. Available: https://opus-codec.org/comparison. [Accessed 10 April 2021].

[55] “WebCodecs API”, [Online]. Available: https://github.com/w3c/webcodecs. [Accessed 10 April 2021].

Lisa 1– Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Stanislav Grebennik

1 Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Veebibrauseripõhise heli voogedastuse latentsuse vähendamine klient-server arhitektuuris” mille juhendaja on Joel Kivi.

   1.1 reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

   1.2 üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2 Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.

3 Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

26.04.2021

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Rakenduse peamised meetodid

```
let currentTransportDatagramWriter;
let microphoneWorklet;
let speakerWorklet;
let microphoneAudioContext;
let speakerAudioContext;
let sampleRate = 44100;
let audioBufferSize = 512;

async function connect() {
  const url = document.getElementById('url').value;

  sampleRate = Number(document.getElementById('sampleRate').value)
  if (Number.isNaN(sampleRate)) {
    addToEventLog('Wrong sample rate value: ' +
      document.getElementById('sampleRate').value, 'error');
    return;
  }

  audioBufferSize =
Number(document.getElementById('audioBufferSize').value);
  if (Number.isNaN(audioBufferSize)) {
    addToEventLog('Wrong audio buffer size value: ' +
      document.getElementById('audioBufferSize').value, 'error');
    return;
  }

  try {
    var transport = new WebTransport(url);
    addToEventLog('Initiating connection...');
  } catch (e) {
    addToEventLog('Failed to create connection object. ' + e, 'error');
    return;
  }

  try {
    await transport.ready;
    addToEventLog('Connection ready.');
  } catch (e) {
    addToEventLog('Connection failed. ' + e, 'error');
    return;
  }

  transport.closed
    .then(() => {
      addToEventLog('Connection closed normally.');
    })
    .catch(() => {
```

```
    addToEventLog('Connection closed abruptly.', 'error');
   });

  try {
    currentTransportDatagramWriter = transport.datagramWritable.getWriter();
    addToEventLog('Datagram writer ready.');
  } catch (e) {
    addToEventLog('Sending datagrams not supported: ' + e, 'error');
    return;
  }

  readDatagrams(transport);
  speakerContext();
  sendDatagrams();

  document.getElementById('connect').disabled = true;
  document.getElementById('disconnect').disabled = false;
}

async function disconnect() {
  currentTransportDatagramWriter.abort();
  microphoneAudioContext.close();
  speakerAudioContext.close();

  document.getElementById('connect').disabled = false;
  document.getElementById('disconnect').disabled = true;
}

async function sendDatagrams() {
  let AudioContextConstraints = {
    audio: {
      echoCancellation: false,
      autoGainControl: false,
      noiseSuppression: false,
      latency: 0
    }
  };

  microphoneAudioContext = new AudioContext({
    latencyHint: 'interactive',
    sampleRate: sampleRate
  });

  microphoneAudioContext.audioWorklet
    .addModule('lib/microphone-worklet-processor.js')
    .then(() => {
      navigator.mediaDevices
        .getUserMedia(AudioContextConstraints)
        .then(stream => {
          const microphoneStream =
microphoneAudioContext.createMediaStreamSource(stream);
          microphoneWorklet = new AudioWorkletNode(
            microphoneAudioContext,
            'microphone-worklet-processor',
            {
              channelCount: 1,
```

```
            processorOptions: {
              bufferSize: audioBufferSize,
              channelCount: 1,
            },
          },
        );
        microphoneWorklet.port.onmessage = ({ data }) => {
          if(data && data.payload && currentTransportDatagramWriter){
            let payload = new Uint8Array(data.payload)
            currentTransportDatagramWriter.write(payload);
          }
        };
        microphoneStream.connect(microphoneWorklet)
      })
      .catch(e => {
        addToEventLog('Microphone audio context error: ' + e, 'error');
      });
    })
    .catch(e => {
      addToEventLog('Microphone worklet processor error: ' + e, 'error');
    });
}

async function readDatagrams(transport) {
  try {
    var reader = transport.datagramReadable.getReader();
    addToEventLog('Datagram reader ready.');
  } catch (e) {
    addToEventLog('Receiving receiver datagrams not supported: ' + e, 'error');
    return;
  }

  try {
    while (true) {
      const { value, done } = await reader.read();
      if (done) {
        addToEventLog('Done reading datagrams!');
        return;
      }

      const sharedPayload = new Uint8Array(new
SharedArrayBuffer(value.length));
      sharedPayload.set(value, 0);
      speakerWorklet.port.postMessage(sharedPayload);
    }
  } catch (e) {
    addToEventLog('Error while reading datagrams: ' + e, 'error');
  }
}

async function speakerContext() {
  speakerAudioContext = new AudioContext({
    latencyHint: 'interactive',
    sampleRate: sampleRate
  });
```

```
  let audioBuffer = speakerAudioContext.createBuffer(1, audioBufferSize,
sampleRate);
  let speakerAudioSource = speakerAudioContext.createBufferSource();

  speakerAudioSource.buffer = audioBuffer;
  speakerAudioSource.loop = true;

  speakerAudioContext.audioWorklet
    .addModule('lib/speaker-worklet-processor.js')
    .then(() => {
      speakerWorklet = new AudioWorkletNode(
        speakerAudioContext,
        'speaker-worklet-processor',
        {
          channelCount: 1,
          processorOptions: {
            bufferSize: audioBufferSize,
            channelCount: 1,
          },
        },
      );

speakerAudioSource.connect(speakerWorklet).connect(speakerAudioContext.de
stination);
    }).catch(e => {
      addToEventLog('Speaker worklet processor error: ' + e, 'error');
  })
}

function addToEventLog(text, severity = 'info') {
  let log = document.getElementById('event-log');
  let mostRecentEntry = log.lastElementChild;
  let entry = document.createElement('li');
  entry.innerText = text;
  entry.className = 'log-' + severity;
  log.appendChild(entry);

  if (mostRecentEntry != null &&
    mostRecentEntry.getBoundingClientRect().top <
    log.getBoundingClientRect().bottom) {
    entry.scrollIntoView();
  }
}
```

Lisa 3 – MicrophoneWorkletProcessor

```
import Module from './variable-buffer-kernel.wasmmodule.js';
import { HeapAudioBuffer, RingBuffer, LOG_TABLE } from './audio-helper.js';


class MicrophoneWorkletProcessor extends AudioWorkletProcessor {
  constructor(options) {
    super();
    this.bufferSize = options.processorOptions.bufferSize;
    this.channelCount = options.processorOptions.channelCount;
    this.inputRingBuffer = new RingBuffer(this.bufferSize, this.channelCount);
    this.heapInputBuffer = new HeapAudioBuffer(Module, this.bufferSize,
this.channelCount);
    this.heapOutputBuffer = new HeapAudioBuffer(Module, this.bufferSize,
this.channelCount);
    this.kernel = new Module.VariableBufferKernel(this.bufferSize);
  }

  float32ToInt16(float32array) {
    let l = float32array.length;
    const buffer = new Int16Array(l);
    while (l--) {
      buffer[l] = Math.min(1, float32array[l]) * 0x7fff;
    }
    return buffer;
  }

  alawEncode(sample) {
    let compandedValue;
    sample = sample === -32768 ? -32767 : sample;
    const sign = (~sample >> 8) & 0x80;
    if (!sign) {
      sample *= -1;
    }
    if (sample > 32635) {
      sample = 32635;
    }
    if (sample >= 256) {
      const exponent = LOG_TABLE[(sample >> 8) & 0x7f];
      const mantissa = (sample >> (exponent + 3)) & 0x0f;
      compandedValue = (exponent << 4) | mantissa;
    } else {
      compandedValue = sample >> 4;
    }
    return compandedValue ^ (sign ^ 0x55);
  }

  linearToAlaw(int16array) {
```

```
    const aLawSamples = new Uint8Array(new
SharedArrayBuffer(int16array.length));
    for (let i = 0; i < int16array.length; i++) {
      aLawSamples[i] = this.alawEncode(int16array[i]);
    }
    return aLawSamples;
  }

  process(inputs) {
    const input = inputs[0];

    this.inputRingBuffer.push(input);

    if (this.inputRingBuffer.framesAvailable >= this.bufferSize) {
      this.inputRingBuffer.pull(this.heapInputBuffer.getChannelData());

      this.kernel.process(
        this.heapInputBuffer.getHeapAddress(),
        this.heapOutputBuffer.getHeapAddress(),
        this.channelCount,
      );
      const channelData = this.heapOutputBuffer.getChannelData();

      if (channelData && !!channelData.length) {
        const float32array = channelData[0];
        const int16array = this.float32ToInt16(float32array);
        let payload = this.linearToAlaw(int16array);
        this.port.postMessage({ payload, time });
      }
    }
    return true;
  }
}

registerProcessor(`microphone-worklet-processor`,
MicrophoneWorkletProcessor);
```

Lisa 4 – SpeakerWorkletProcessor

```
import Module from './variable-buffer-kernel.wasmmodule.js';
import { HeapAudioBuffer, RingBuffer, ALAW_TO_LINEAR } from './audio-
helper.js';

class SpeakerWorkletProcessor extends AudioWorkletProcessor {
  constructor(options) {
    super();
    this.payload = null;
    this.bufferSize = options.processorOptions.bufferSize;
    this.channelCount = options.processorOptions.channelCount;
    this.inputRingBuffer = new RingBuffer(this.bufferSize, this.channelCount);
    this.outputRingBuffer = new RingBuffer(this.bufferSize, this.channelCount);
    this.heapInputBuffer = new HeapAudioBuffer(Module, this.bufferSize,
this.channelCount);
    this.heapOutputBuffer = new HeapAudioBuffer(Module, this.bufferSize,
this.channelCount);
    this.kernel = new Module.VariableBufferKernel(this.bufferSize);
    this.port.onmessage = this.onmessage.bind(this);
  }

  alawToLinear(incomingData) {
    const outputData = new Float32Array(incomingData.length);
    for (let i = 0; i < incomingData.length; i++) {
      outputData[i] = (ALAW_TO_LINEAR[incomingData[i]] * 1.0) / 32768;
    }
    return outputData;
  }

  onmessage(event) {
    const { data } = event;
    if (data) {
      this.payload = this.alawToLinear(new Uint8Array(data));
    } else {
      this.payload = null;
    }
  }

  process(inputs, outputs) {
    const output = outputs[0];
    if (this.payload && this.payload.length > 0) {
      this.inputRingBuffer.push([this.payload]);
      if (this.inputRingBuffer.framesAvailable >= this.bufferSize) {
        this.inputRingBuffer.pull(this.heapInputBuffer.getChannelData());
        this.kernel.process(
          this.heapInputBuffer.getHeapAddress(),
          this.heapOutputBuffer.getHeapAddress(),
          this.channelCount,
```

```
        );
        this.outputRingBuffer.push(this.heapOutputBuffer.getChannelData());
      }
      this.outputRingBuffer.pull(output);
    }
    return true;
  }
}

registerProcessor(`speaker-worklet-processor`, SpeakerWorkletProcessor);
```