UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

LUCRARE DE LICENŢĂ

Distribuția streamurilor audio-video în sesiuni WebRTC

Conducător științific
Lect. dr. Sterca Adrian-Ioan

Absolvent
Haja Florin-Gabriel

2021

ABSTRACT

This thesis is an in-depth study of the communication protocols used by the WebRTC project and the way the audio-video content can be distributed across multiple clients. As people are using the Internet productively more and more, it is crucial to ensure the information is transmitted as fluidly as possible. Since some Internet connection may be limited in bandwidth, it is more difficult to handle video call sessions with many users.

We can opt for using a media server that handles the data delivery for us, or to connect each client with the other, in a mesh topology fashion. When attending a class, for example, we are noticing a broadcasting scenario, as the teacher is providing the content and the students are only looking at it, thus the data can be streamed in one direction, and the mesh topology having no real purpose. Instead of serving all the students at once, we can serve a few, which will forward the stream to others, repeatedly streaming until everyone is able to watch what is the teacher showing. We notice a tree topology. A media server is no longer required, meaning cheaper maintenance costs, but sacrificing the low bandwidth each client is using.

Cuprins

Listă de figuri

Capitolul 1

Introducere

În zilele noastre, comunicarea este îndeosebi importantă. Iar pentru că distanța între persoane poate să fie mare, trebuie apelat la soluțiile de telecomunicații. Din fericire, tehnologiile au evoluat continuu și au devenit din ce în ce mai accesibile, în așa fel încât oricine care dispune de un telefon sau de un laptop, dar și de o conexiune la Internet, poate să ia legătura cu oricine, să vorbească, să se vadă și să colaboreze la diversele idei pe care le au.

De asemenea, impunerea unei multitudini de standarde a dus la facilitarea creării de noi platforme de comunicare, în așa fel încât nici nu mai este nevoie de a instala local vreo aplicație.

Însă datorită creșterii populației și a cererii crescânde în acest domeniu, într-o gamă largă de contexte (personale, profesionale, academice, preuniversitare, chiar și guvernamentale), sunt inevitabile discrepanțele între calitatea diverselor conexiuni la Internet și a performanțelor pe care fiecare dispozitiv le are, așadar resursele trebuie distribuite cât mai eficient, cu scopul de a evita compromisurile în privința calității imaginii și a sunetului.

În această lucrare, voi prezenta o posibilă optimizare într-un context des întâlnit în mediul școlar, în care o singură persoană transmite conținut audio-video, pe care îl distribuie unui grup (mai restrâns sau mai larg), în așa fel încât să nu fie dezavantajați cei care dispun de conexiuni mai slabe la Internet. Voi pune, de asemenea, la dispoziție o aplicație prin care voi demonstra această optimizare, folosind standarde din industrie precum WebRTC și WebSocket, implementată cu framework-uri moderne precum React și Socket.IO. Deoarece această optimizare presupune o topologie proprie, voi pune la dispoziție și serverul prin care va fi posibilă legătura între participanți, implementat cu Node.js, o tehnologie modernă și accesibilă.

În primul capitol, acesta, este prezentată tema lucrării și posibilul său potențial.

Al doilea capitol prezintă istoria videoconferințelor, care conturează momente

importante din evoluția videotelefoniei, precum și tranziția către digital și standarde în industrie.

Al treilea capitol descrie detaliat protocoalele prin care proiectul WebRTC a fost făcut posibil, precum și interfața prin care dezvoltatorii pot construi aplicații folosind protocoalele puse la dispoziție.

Capitolul patru ilustrează modalitatea prin care aplicații populare precum Microsoft Teams și Zoom implementează aceste tehnologii.

În capitolul cinci este inclus un studiu de caz, prin care se analizează în profunzime topologiile pe care aplicațiile moderne le folosesc pentru o comunicare eficientă, avantajele și dezavantajele lor.

Capitolul șase conține o descriere a aplicației puse la dispoziție de mine, cazuri de utilizare, precum și topologia pe care o folosesc pentru optimizare.

Capitolul șapte încheie lucrarea cu observații asupra posibilelor îmbunătățiri ale aplicației personale, precum și cu concluzii asupra distribuției streamurilor audio-video în sesiuni WebRTC.

Capitolul 2

Istoria conferințelor video

2.1 Era analogică

Primele concepte ale transmiterii de imagini datează încă din secolul XIX, contemporane cu brevetarea telefonului de către Alexander Graham Bell în data de 14 februarie 1876 și prezentarea acestuia la Expoziția Centenară de la Philadelphia în același an. Un articol publicat în 30 martie 1877 de către The New York Sun menționează termenul *electroscop* în acest context, prin care prezintă posibilele aplicații ale unui dispozitiv prin care se pot observa imagini din orice colț al lumii. De asemenea, scriitori precum Abbé Moigno și Louis Figuier, prezintă un concept foarte similar, creând termenul *telectroscop*, atribuindu-i-l eronat lui Graham Bell [Lan02, Lan20].

Însă primul prototip funcțional a fost demonstrat public în 7 aprilie 1927 într-o aulă din New York. Herbert Hoover, pe atunci secretarul comerțului al Statelor Unite, a apărut pe ecran din Washington D.C., fiind observat de către oficialii companiei AT&T [Uen20]. Apelul era unidirecțional pentru imagine și bidirecțional pentru sunet, datorită liniilor telefonice deja existente. Mai târziu, la Expoziția Mondială de la New York din 1964, AT&T a prezentat primul

Figura 2.1: Primul model de Picturephone, care a fost instalat în cabine în 3 orașe din Statele Unite

Picturephone, la care publicul putea să efectueze apeluri bidirecționale către o cabină telefonică similară montată la Disneyland, folosind conexiune VHF sau UHF prin cablu [Avo]. Acesta a fost indisponibil pentru uz casnic până în 1970, când aceeași companie a lansat un Picturephone îmbunătățit, disponibil printr-un serviciu lunar. Din cauza costului ridicat de 160 de dolari pe lună (echivalent a 1000 de dolari în 2020), nu a prezentat interes, serviciul fiind desființat în 1973 [Uen20].

2.2 Era digitală și tranziția către Internet

În anii '80, diverse companii, printre care Mitsubishi, au comercializat diverse videotelefoane care puteau transmite imagini statice prin intermediul rețelei telefonice deja existente, fapt care permitea folosirea modemurilor existente cu rate de transfer între 2,4 și 9,6 kilobiți pe secundă. Evoluția algoritmilor digitali de compresie a imaginilor și a lățimii de bandă, a permis ca AT&T să mai încerce o dată cu VideoPhone 2500, care, de asemenea, nu a avut succes, costând inițial 1500 de dolari în 1992 [Bor].

Apariția protocoalelor Internet și a tehnicilor și mai avansate de compresie audio-video a permis ca imaginea și sunetul să fie transmise în pachete mult mai mici, rezultând în costuri mult mai reduse. Simultan cu dezvoltarea rețelei ARPANET (precursoarea Internetului), destinată inițial pentru transfer de date, s-a discutat problema comunicării în timp real prin voce [RFC77]. Așadar, s-a propus în decembrie 1973 implementarea Network Voice Protocol (NVP), care a venit cu următoarele considerente: separarea semnalelor de control de traficul de date, evitarea retransmiterii pachetelor pierdute, adaptabilitate la condițiile variabile ale rețelei, precum și independența gestionării resurselor de către fiecare sistem, dar și de către protocoalele de nivel mai jos[RFC77].

PictureTel a jucat un rol important în evoluția comunicațiilor video. Cei doi fondatori ai săi, Brian L. Hinman și Jeffrey G. Bernstein, absolvenți ai MIT și buni prieteni, au fondat PicTel în 1984 (a fost ulterior redenumită pentru a evita confuzia cu termenul *pixel*) cu sprijin financiar din partea lui Robert Sterling. În 1986, după ce compania a devenit publică, a dezvoltat algoritmul Motion Compensated Transform (MCT), care a permis reducerea unei transmisii audio-video de calitate de la 768 Kb/s la 224. Pe baza acestui algoritm, au lansat codecul C-2000 în luna iulie al aceluiași an, cu ajutorul căruia PictureTel a devenit lider în domeniul său. În 1988, un nou algoritm de compresie video a fost lansat, Hierarchical Vector Quantizing (HVQ), folosit în codecul C-3000, precum și în sistemul V-2100 [Roo00]. Cu ajutorul acestuia, lățimea de bandă folosită a fost redusă la jumătate, comparativ cu algoritmul MCT. În anii '90, a colaborat la crearea standardului H.320 pentru videoconferințe prin rețeaua ISDN, precum și a standardului H.323

pentru comunicare audio-video prin Internet pe baza protocolullui TCP, cel din urmă folosit pentru a crea produsul software LiveLan [Roo00].

O dată cu migrarea videoconferințelor către Internet, au început să apară soluții gratuite, potrivite și consumatorilor de rând, care să nu ceară echipament auxiliar, pe lângă un computer și o cameră web. Una din primele aplicații influente de acest fel este realizată de Tim Dorcey la Universitatea Cornell și se numește CU-SeeMe, apărut prima dată în 1992 pentru sistemele Macintosh 2.2. Apelurile cu doi participanți pot fi inițiate de către unul din ei conectându-se direct la celălalt, în timp ce apelurile cu mai mulți participanți necesită conectarea fiecăruia la un *reflector* CU-SeeMe [Dor95]. *Reflectorul* este un software destinat sistemelor UNIX ce are menirea de a a redistribui fluxurile de pachete, principala sa motivație fiind lipsa abilităților de multicast pe Macintosh de la vremea respectivă [Dor95]. Pentru transmisie, imaginea se împarte în pătrate de dimensiune 8x8, acestea fiind selectate pentru trimitere doar în cazul în care gradul de similitudine este suficient de ridicat [Dor95]. Acest grad este calculat ca sumă a tuturor diferențelor absolute între fiecare pixel de pe aceeași poziție, la care se aplică o penalizare multiplicativă pentru diferențele între pixelii apropiați [Dor95]. Mai târziu, a adoptat standardul H.323, care s-a regăsit și pe alte produse, precum Microsoft NetMeeting (începând cu versiunea 2) [Vid, Per00].

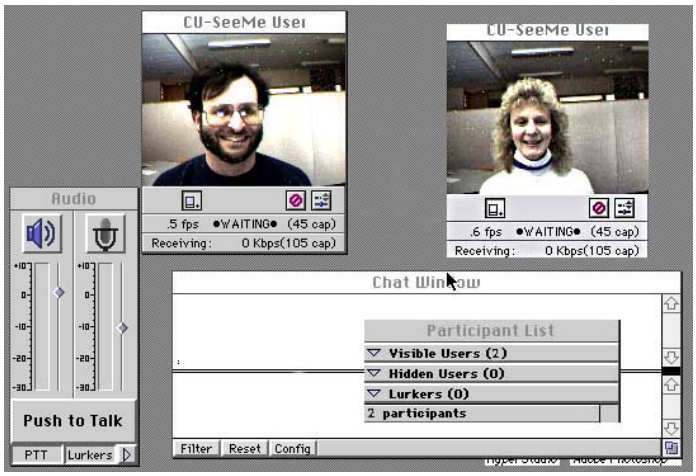


Figura 2.2: Aplicația CU-SeeMe rulând pe Macintosh

În 1994, un moment important a fost lansarea primei camere web destinate consumatorilor, Connectix QuickCam 2.3 [Wol19]. Putea captura imagini doar în 16 culori la 15 cadre pe secundă și a fost inițial compatibilă doar cu Macintosh [Wol19]. Un an mai târziu, a fost lansată și varianta pentru Windows. Însă calitatea redusă a imaginii nu a fost un impediment pentru persoanele aflate la distanță de a se putea simți aproape una de cealaltă, de a putea interacționa, de a trăi

evenimente aflate în diverse colțuri ale lumii [Qui]. Ulterior, au apărut variante îmbunătățite de QuickCam, cu captură de imagini color la rezoluții mai mare, dar și cu conectivitate paralelă și USB.

Figura 2.3: Connectix QuickCam, prima cameră web comercială

În paralel s-au dezvoltat o multitudine de standarde menite de a reduce și mai mult lățimea de bandă folosită, precum și de a ușura crearea de noi platforme de comunicare. În 1997 a fost creat Session Initiation Protocol (SIP), prin care utilizatori pot fi invitați la participarea într-o sesiune multimedia, sau chiar servere prin care se poate înregistra sesiunea [SSRH99]. Este o combinație dintre Session Invitation Protocol și dintre Simple Conference Invitation Protocol, prin care se încearcă potrivirea lui SIP pe o infrastructură mai flexibilă care include și alte protocoale, precum SDP, HTTP și RTSP [SSRH99]. Poate funcționa atât peste TCP, cât și peste UDP, pentru o performanță îmbunătățită [SSRH99].

Codecurile video au continuat de-a lungul timpului să evolueze. În 2003, Unitatea Internațională a Telecomunicațiilor (ITU - Internation Telecommunication Union) a făcut public standardul H.264, destinat codării audio-video în aplicații precum televiziunea prin cablu și satelit, videoconferințe prin Internet, dar și filme stocate pe medii de stocare optice [H.203]. Folosește tehnici precum partiționarea imaginilor în macroblocuri, precum și codificare predictivă bidirecțională. A fost dezvoltat împreună cu ISO MPEG (Moving Picture Experts Group) [H.203].

Între timp, au apărut platforme de mesagerie instantă precum AOL Instant Messenger (AIM) în 1997, Yahoo! Messenger în 1998, MSN Messenger în 1999 [Wol19]. Toate au suportat apeluri video în 2003, folosind, în principiu, SIP pentru a iniția apelurile și RTP pentru transportul pachetelor. Apple a lansat în 2002 un client pentru AIM numit iChat, folosind implementarea oficială a protocolului furnizată de America Online [Wol19]. iChat a primit suport pentru codecul H.264 în 2004, oferind calitate mai bună decât predecesorul său, H.263 [Roy07].

Însă mult mai popular pentru apelurile video a devenit Skype, care, când a fost lansat în 2003 de KaZaa, permitea apeluri gratuite în grup de 25 de persoane.

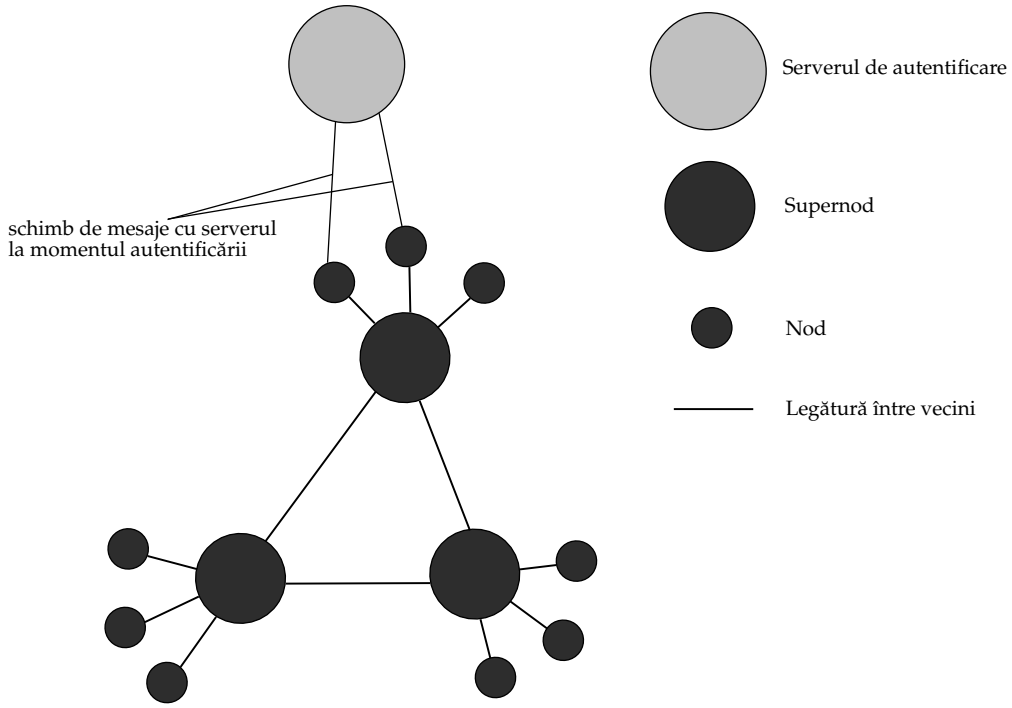


Figura 2.4: O reprezentare a topologiei de rețea folosită de Skype

Promitea, la momentul respectiv, calitate a vocii superioară față de alternativele Yahoo și MSN, dar și o traversare facilă a NAT-urilor și firewall-urilor [Bas04]. Singurul rol al serverului central era doar de login, informațiile utilizatorilor fiind păstrate într-o manieră descentralizată, prin intermediul nodurilor și supernodurilor 2.4 [Bas04]. Un nod ce dispune de suficientă putere de procesare și lățime de bandă fi candidat pentru a fi supernod [Bas04]. Se presupunea că fiecare nod folosește o variație a protocolului STUN pentru a comunica pe rețeaua suprapusă Skype [Bas04]. De asemenea, fiecare nod își păstra o listă de supernoduri cu care să facă legătura - această listă este denumită *host cache* și este păstrată în registrul sistemului de operare Windows [Bas04]. Ulterior, Skype a fost cumpărat de Microsoft în 2011 și a renunțat la topologia cu supernoduri pentru o scalabilitate mai bună, folosind, în schimb, serverele Microsoft, cu toate că au existat suspiciuni privind securitatea serviciului [Whi13].

Apoi, a urmat înglobarea unei colecții de protocoale precum SDP, STUN, TURN și RTP, precum și a unor codecuri precum VP8, G.711 și iLBC într-un proiect numit WebRTC, care vine inclus cu majoritatea browserelor moderne și pe baza căruia s-au construit majoritatea aplicațiilor moderne precum Microsoft Teams, Zoom, Jitsi Meet, OBS.Ninja și inclusiv FaceTime, care s-au dovedit esențiale în mediul profesional și academic pe parcursul epidemiei de coronavirus începută în 2019 și care continuă și în 2021.

Capitolul 3

WebRTC

Acest capitol are ca scop introducerea în principiile de bază și în componentele proiectului WebRTC, cu menirea de a putea înțelege conceptele următoare. Acoperă API-urile comune, precum și principiile de rețelistică folosite pentru comunicare calitativă, cu latență mică.

WebRTC a fost dezvoltat ca standard de către World Wide Web Consortium (W3C) și de către Internet Engineering Task Force (IETF), fiind o colecție open-source de standarde și protocoale. Tehnologiile necesare, precum codecuri și algoritmi de anulare a ecoului, au fost dezvoltate de către o companie suedeză numită Global IP Solutions (GIPS), care a fost mai târziu cumpărată de Google în mai 2010 [EAST18]. Google a făcut publică implementarea acest proiect în mai 2011 și a propus către IETF standardizarea acestuia. Similar, alți lideri ai industriei precum Mozilla, Ericsson, Microsoft și Apple au furnizat implementări pentru WebRTC, în 2013 fiind realizată interoperabilitatea între browserele Chrome și Firefox [Nym13]. Standardul a fost finalizat în ianuarie 2021 și este cunoscut ca RFC 8825.

3.1 Arhitectura

3.1.1 Stabilirea conexiunilor

La începuturile videoconferințelor pe Internet, protocolul dominant pentru stabilirea conexiunilor între clienți era SIP (Session Initiation Protocol). Însă acesta nu vine inclus cu browserele moderne, așadar este nevoie de a furniza o implementare precum SIP.js [EAST18]. Așadar, WebRTC oferă liberatea dezvoltatorului de a stabili modalitatea prin care doi clienți pot iniția un apel.

Mecanismul prin care se face legătura între aceștia se numește *signaling* (engl. *semnalizare*). Poate fi implementat folosind cereri HTTP, însă prin natura stateless a protocolului HTTP, prin care nu se păstrează informațiile de la cererile anterioare,

trebuie trimise continuu cereri, chiar dacă nu se primește sau transmite niciun mesaj nou [EAST18]. Această abordare duce la risipă de lățime de bandă și la o latență ridicată [EAST18]. O soluție ar fi de a inițializa o conexiune WebSocket, care păstrează deschisă legătura cu serverul. Singura responsabilitate a dezvolatorului rămâne de a asigura realizarea schimbului de pachete SDP între clienți și a candidaților ICE, despre care voi discuta în subcapitolul următor. Ca observație, oricare din clienți poate să schimbe sursa conținutului de transmis, organizată pe track-uri separate pentru audio și pentru video - ce vor fi șterse - moment în care se vor renegocia pachetele SDP.

3.1.2 Topologii de comunicare

WebRTC este conceput pentru a permite comunicarea directă între participanți, fără a fi nevoie de a transporta pachetele multimedia printr-un server specializat. Totuși, există situații în care această modalitate este dezavantajoasă, acest lucru putându-se specifica la momentul creării conexiunii.

Mesh

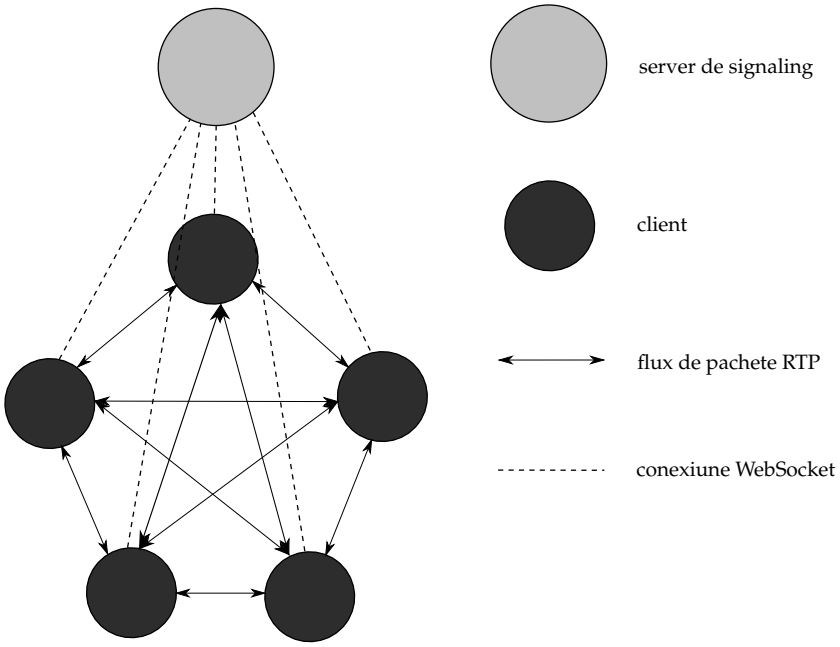


Figura 3.1: Topologie mesh într-o sesiune WebRTC

Cea mai simplă topologie este cea mesh (engl. *pânză*) 3.1, în care toți clienții sunt interconectați. Într-o conversație de n utilizatori, fiecare utilizator va avea $n-1$ conexiuni deschise, în total fiind $n*(n-1)$ conexiuni. Această metodă este potrivită doar în pentru n mic, deoarece fiecare nod va trebui să decodeze $n-1$ fluxuri de

date audio-video, ceea ce crește radical consumul resurselor de către procesor, cu atât mai mult dacă imaginile sunt de calitate înaltă. De asemenea, lățimea de bandă necesară este ridicată, datorită multiplelor fluxuri. În schimb, nu este nevoie de server intermediar, ceea ce reduce costurile de implementare.

Se poate particulariza acest scenariu în cazul în care nu toți partajează conținut. Așadar, unele conexiuni nu mai sunt necesare, iar topologia va deveni, după caz, arborescentă. Mai departe, se poate modela în așa fel încât gradul fiecărui nod să fie redus, iar lățimea de bandă folosită să fie scăzută.

Stea

O altă topologie des întâlnită în implementările aplicațiilor WebRTC este cea de tip stea. În acest caz, fiecare pereche de noduri va fi intermediată de către un server media. Acest server poate manipula conținutul de la fiecare nod, să îl unească într-un singur set de date, sau să îl redistribuie fiecăruia așa cum este. În primul caz, avem de a face cu un multipoint control unit (MCU) 3.2, iar în al doilea cu un selective forwarding unit (SFU) 3.3.

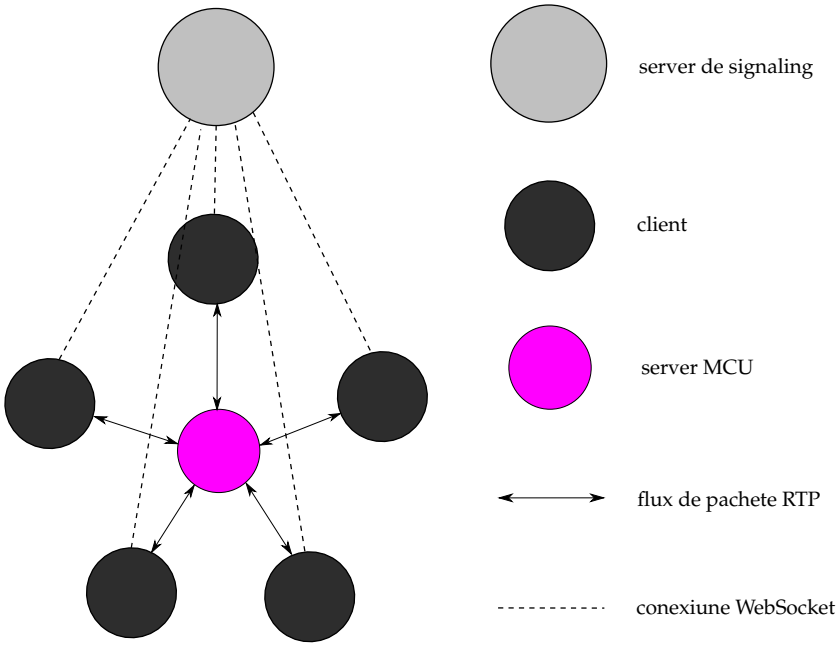


Figura 3.2: Topologie de tip stea cu server MCU

Un avantaj major în cazul topologiei cu server MCU este că fiecare nod trimite și primește câte un singur flux audio-video, ceea ce duce la o economisire de resurse utilizate și de lățime de bandă. Dar există și dezavantaje, și anume, că serverul media va fi consumatorul de resurse, deoarece lui i se atribuie sarcina de a decoda

și multiplexa conținutul, ceea ce duce la costuri mai mari de întreținere. Fiind o singură imagine primită, aceasta va fi dificil de prelucrat de către client.

Se observă că serverul de signaling poate exista independent de cel media, datorită independenței responsabilităților sale și a posibilității de a specifica mai multe servere media din fiecare nod.

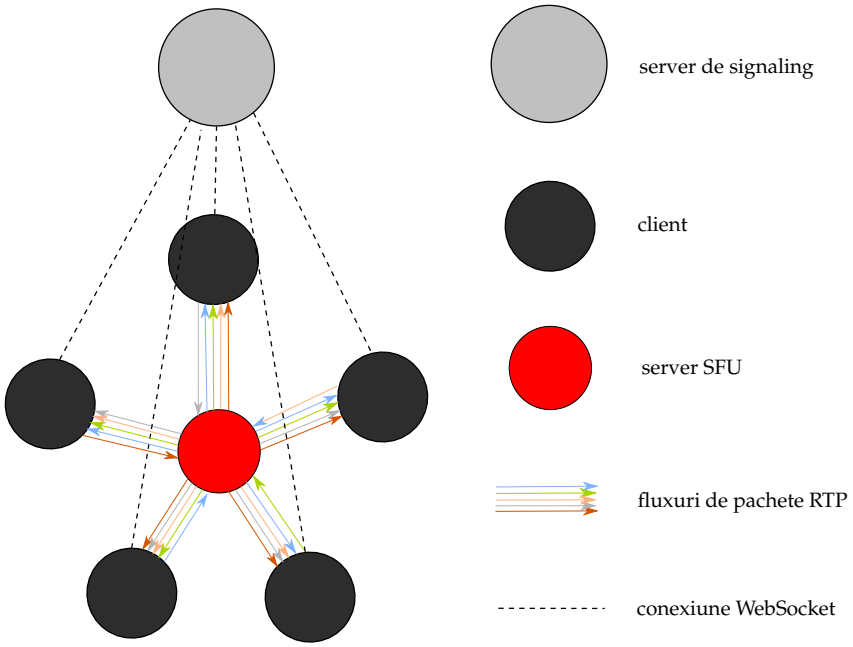


Figura 3.3: Topologie de tip stea cu server SFU

Serverul SFU din topologia de mai sus va fi mai puțin costisitor de întreținut față de unul MCU, deoarece consumul de resurse este redus, nemaifiind necesară procesarea datelor. Însă, într-o sesiune cu n participanți, fiecare va recepționa $n - 1$ fluxuri de date și va trasmite unul, ceea ce crește lățimea de bandă considerabil față de MCU, dar oferă un avantaj clar față de topologia mesh. O îmbunătățire ar putea reprezenta simulcast SFU, prin care fiecare client va transmite imaginea la calități diferite (în condițiile în care toți suportă același codec), dar va cere de la serverul SFU imaginea de calitate potrivită lățimii sale de bandă [SFU20]. Astfel, clienții care nu dispun de o lățime de bandă largă nu vor compromite calitatea imaginii celor care dispun [SFU20]. De exemplu, avem patru participanți, doi dintre ei folosind un telefon conectat la date mobile, iar ceilalți conectați de pe laptopuri la rețeaua de acasă. Cei conectați de pe telefoane vor transmite doar imagini de calitate joasă, iar cei de pe laptopuri vor transmite imagini și de calitate înaltă. Cei din urmă își vor vedea imaginile la calitate înaltă, iar ceilalți vor vedea imaginile de la laptopuri la calitate joasă.

Hibrid

Se pot combina avantajele celor două topologii prezentate anterior, rezultând o distribuție mai eficientă a resurselor utilizate 3.4. Implicarea unor topologii de tip mesh unite între ele într-o manieră de tip stea are un compromis pe partea latenței, deoarece datele vor avea un drum mai lung de parcurs până la fiecare destinație, însă libertatea de a decide conexiunile directe între participanți este a dezvoltatorului, iar optimizările pe care acesta le poate realiza sunt numeroase. Folosind algoritmi de flux, se poate rezolva problema lățimii de bandă utilizate, iar prin algoritmi de distanță minimă (ex. Bellman-Kalaba), și cea a latenței.

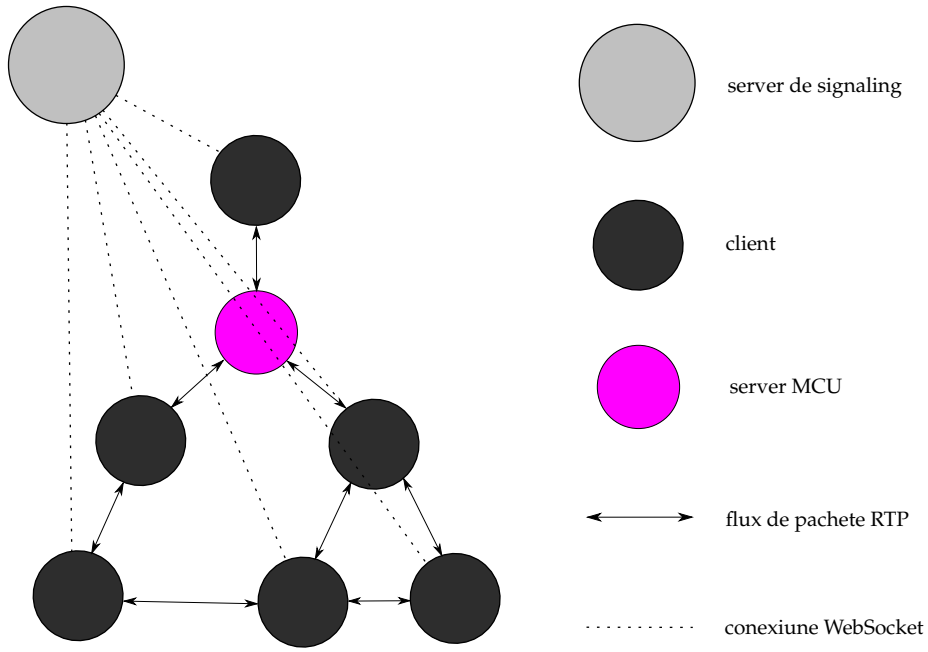


Figura 3.4: Topologie de tip hibrid cu server MCU

3.2 ICE și NAT traversal. Servere STUN și TURN

După realizarea schimbului de descriptori SDP, participanții trebuie să stabilească modul de comunicare între ei, deoarece pe traseu poate traversa unul sau mai multe NAT-uri (Network Address Translator), iar adresele lor IP dintr-o rețea nu vor mai fi valide pe altă rețea [KHR18]. Prin urmare, transportul de date nu ar mai putea fi posibil în mod direct. Pentru aceasta, s-a realizat un protocol numit Interactive Connectivity Estabilishment (ICE). Ideea principală a ICE este că se va folosi un server intermediar (STUN sau TURN), pe care fiecare agent (nod) are o varietate de adrese candidat [KHR18].

Teoretic, oricare adresă candidat al unui agent poate fi folosită pentru a comunica cu oricare din adresele candidat ale altui agent. În realitate, majoritatea

din combinațiile de adrese nu vor funcționa. De exemplu, dacă amândouă nodurile se află în spatele unor NAT-uri, prin adresele interfețelor atașate direct (de pe partea publică a NAT-urilor) nu vor putea comunica direct [KHR18]. ICE are tocmai acest rol: de a încerca fiecare pereche posibilă de adrese până găsește una sau mai multe care funcționează.

3.2.1 Tipuri de candidați

Pentru a face posibil schimbul de conexiuni, un agent trebuie să colecteze una sau mai multe adrese candidat. Adresele candidat sunt perechi de adresă IP și de port pe un protocol de transport specific (de obicei UDP). Există patru tipuri de candidați:

- *host* (engl. *gazdă*), care poate comunica direct; este obținut direct de la interfețele locale de rețea (Ethernet, Wi-Fi);
- *server-reflexive*, candidat obținut de la un server STUN; este adresa publică a NAT-ului, împreună cu portul pe care NAT-ul îl atribuie când un client trimite cererea de alocare (binding request) către serverul STUN sau TURN [Ros10];
- *peer-reflexive*, candidat obținut de la un server STUN; este adresa publică a NAT-ului, împreună cu portul pe care NAT-ul îl atribuie când clientul primește răspunsul la un binding request de la server, în fazele mai târzii ale ICE;
- *relay* (engl. *releu*), candidat obținut de la un server TURN; va conține adresa serverului TURN care va fi responsabil de a retrimite conținutul către cealaltă parte [Ros10].

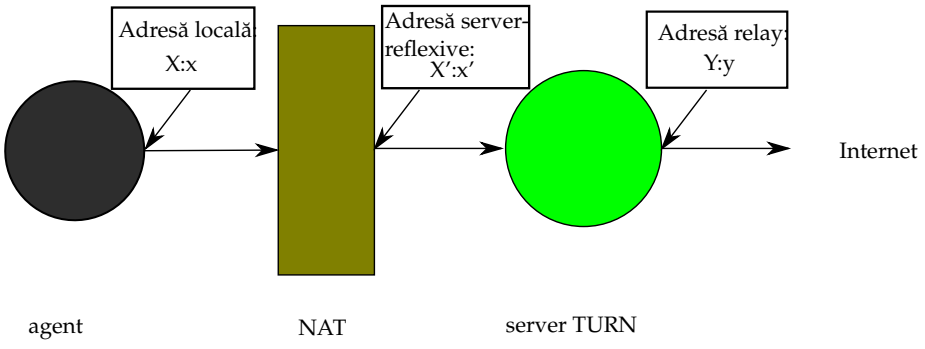


Figura 3.5: Diagrama adreselor în traseul către serverul TURN

De observat este că serverele STUN (Session Traversal Utilities for NAT) sunt mai simple decât cele TURN (Traversal Using Relays around NAT), deoarece ele nu au responsabilitatea de a primi pachete de la clienți și de a le redistribui. Cererea ce

trebuie trimisă către STUN se numește *binding request* (engl. *cerere de legare*), iar cea către TURN se numește *allocate request* (engl. *cerere de alocare*), cea din urmă având rolul de a aloca un relay pe serverul TURN.

O altă observație este că serverele STUN nu pot fi folosite în cazul unui NAT simetric, deoarece clientului din spatele NAT-ului i se va atribui un port diferit dacă va trimite către o destinație diferită, deși adresa și portul lui sursă rămân neschimbate [MRWM08].

De asemenea, nu se confunda serverele TURN cu cele SFU, deși responsabilitățile lor sunt similare. Clienții trebuie să se conecteze cu serverul SFU ca și cum s-ar conecta cu un alt client, așadar s-ar putea să fie nevoie de un server TURN pe traseul către SFU. Serverul TURN este folosit doar în cazul în care nu se pot conecta la candidații găsiți prin STUN, este un intermediar între clienți sau între clienți și un server media, deci nu poate înlocui un SFU și invers.

3.2.2 Obținerea candidaților

Înainte de a proba fiecare conexiune, un agent ICE are nevoie de o listă de candidați. În primă fază, el listează adresele fiecărei interfețe de rețea existente (fizică precum Ethernet, Wi-Fi sau USB, sau virtuale, precum mecanisme de tunelare ca VPN) [KHR18]. Agentul obține fiecare candidat host prin legarea la un port UDP de pe adresa IP a fiecărei interfețe listate [KHR18].

Candidatul host este mereu asociat cu o componentă pentru care este candidat [KHR18]. Fiecare componentă are asociat un ID. Acest *component ID* va avea valoarea 2 pentru un flux de date (stream) RTCP (RTP Control Protocol) dacă nu este multiplexat pe același port UDP [KHR18]. În caz contrar, va avea valoarea 1. Pentru RTP (Real-time Transport Protocol) va fi mereu valoarea 1 [KHR18]. În cazul în care agentul are K adrese IP și folosește candidați separați pentru RTP și RTCP, va avea $2 * K$ candidați host.

Urmează căutarea candidaților server-reflexive și a celor relay. Pentru aceasta, este nevoie de interogarea serverelor STUN și TURN specificate în listă. În unele situații, totuși, nu este nevoie de utilizarea lor [KHR18]. În acest caz, este recomandat să se implementeze această funcționalitate și să fie doar lăsată dezactivată, deoarece, în timp, se pot schimba cerințele [KHR18].

Când sunt specificate mai multe servere STUN sau TURN, agentul ar trebui să returneze candidații pentru cel puțin unul din ele [KHR18]. Va realiza perechi între fiecare candidat host și fiecare server STUN sau TURN, iar pentru fiecare pereche, va trimite un binding request sau un allocate request [KHR18]. Agentul va primi un binding sau allocate response [KHR18]. În cazul unei cereri de alocare, va primi un candidat server-reflexive și un candidat relay [KHR18]. În cazul în care această

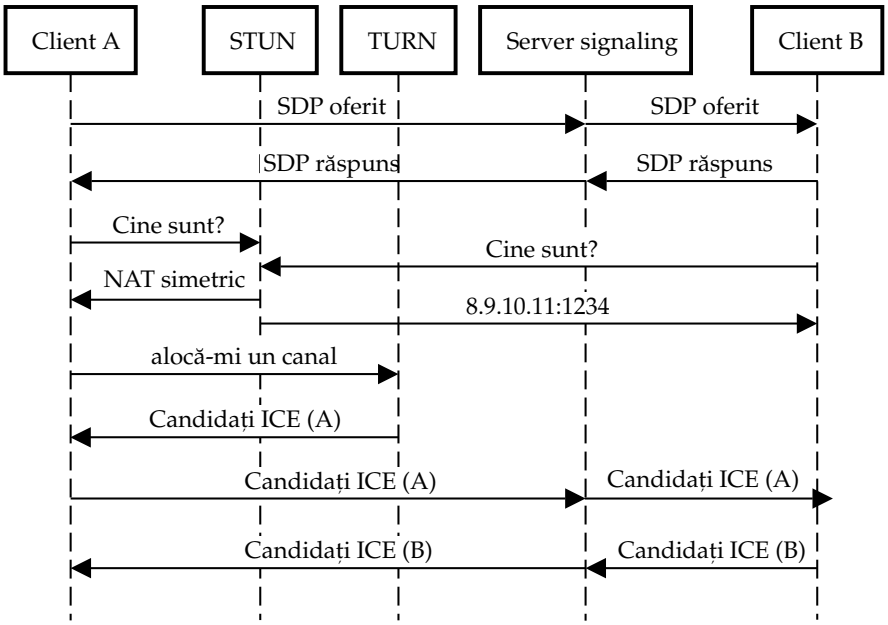


Figura 3.6: Schimbul de candidați ICE în cadrul unei sesiuni WebRTC

cerere eșuează din cauza resurselor insuficiente pe server, va încerca un binding request , care va returna doar candidatul server-reflexive [KHR18].

Procesul de căutare este controlat de un contor, Ta [KHR18]. De fiecare dată când expiră acest contor, agentul poate iniția o nouă cerere către STUN sau TURN. Poate reiniția cererile și în cazul în care acestea returnează o eroare, însă nu la un interval mai scurt de Ta [KHR18].

3.2.3 Calcularea parametrilor pentru candidați

După obținerea candidaților, acesta va calcula un *foundation* pentru fiecare. Acest *foundation* este identic pentru candidații care au aceeași adresă IP (chiar dacă porturile sunt distincte), sunt de același tip, au același protocol de transport (TCP sau UDP), sau dacă adresele IP ale serverelor STUN sau TURN de pe care au fost obținuți sunt identice [KHR18].

Agentul va calcula și prioritatea fiecărui candidat [KHR18]. Formula recomandată după care se calculează prioritatea este:

$$prioritate = 2^24 * preferințăTip + 2^8 * preferințăLocală + 2^0 * (256 - componentID) \tag{3.1}$$

$preferințăTip$ va fi o valoare între 0 și 126, distinctă pentru fiecare tip de candidat și neapărat mai mare pentru tipul peer-reflexive decât cel server-reflexive. $preferințăLocală$ va avea valoarea între 0 și 65535 dacă există adrese IP distincte între candidați sau 65535 dacă există doar una singură. $componentID$ va fi ID-ul componentei asociate [KHR18].

3.2.4 Schimbul de informații despre candidați

Agenții ICE trebuie să stabilească modalitatea prin care vor realiza schimbul de informații despre candidați (pentru WebRTC se va folosi, de obicei, WebSocket prin serverul de signaling). De asemenea, trebuie să transmită următoarele informații:

- lista de candidați. Pentru fiecare candidat (în această ordine):
  - *foundation*-ul ca secvență de cel mult 32 de caractere;
  - component ID-ul;
  - protocolul de transport;
  - coeficientul de prioritate, așa cum este menționat mai sus 3.1;
  - adresa IP și portul specific protocolului de transport;
  - tipul candidatului (*host*; *srflx* - server-reflexive; *prflx* - peer-reflexive; *relay*);
  - adresa și portul relative (opționale);
  - alte atribute (ex.: adresa server-reflexive pentru candidații relay).
- Tipul Lite sau Full al candidatului.
- Valoarea de stimulare a verificării conexiunii (opțională dacă agentul optează pentru valoarea implicită).
- Extensii la nivel de flux media sau sesiune (opțiuni ICE) [KHR18].

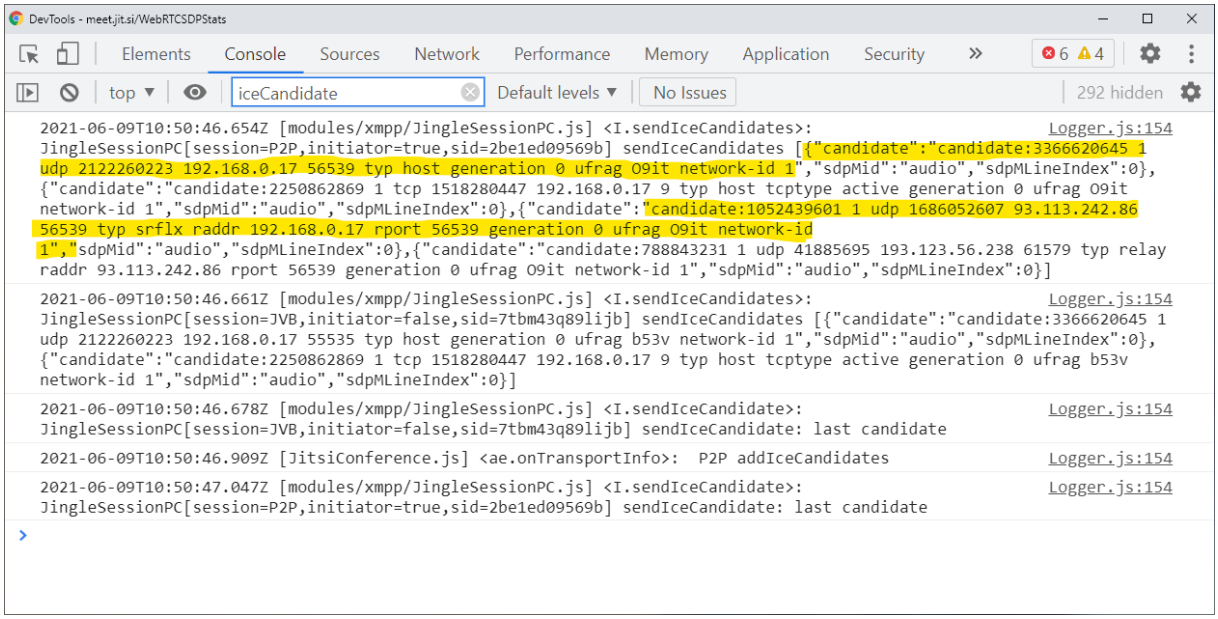


Figura 3.7: O listă de candidați ICE dintr-o conversație pe Jitsi Meet

Observând figura de mai sus, se remarcă exact parametrii menționați anterior. Pentru al doilea candidat evidențiat cu galben, foundation-ul său, `1052439601`, este unic, deoarece candidatul este distinct. Component ID-ul său este `1`, așadar acesta are cu o componentă de tip RTP atașată. Protocolul de transport ales este `udp`, iar `1686052607` reprezintă coeficientul de prioritate . Urmează adresa și IP-ul (`93.113.242.86`, respectiv `56539`), atribuite, în acest caz, de către NAT. Următorii parametri sunt cei sunt descriși ca perechi proprietate-valoare. Este menționat `typ srflx`, deci se specifică faptul că este vorba de tip, respectiv că este server-reflexive. Mai mult, pentru `raddr 192.168.0.17`, `raddr` reprezintă mențiunea adresei IP reale a agentului, iar `192.168.0.17` reprezintă adresa în sine.

3.2.5 ICE mismatch

În unele cazuri, poate apărea fenomenul de mismatch (engl. *nepotrivire*), în care un intermediar, cum ar fi un application-level gateway, poate altera informațiile despre candidații ICE [KHR18]. În această situație, agentul cu care se face schimbul trebuie să poată detecta această alterare și să îl informeze [KHR18].

3.3 Streaming. Protocoalele SDP și RTP

Pentru ca streaming-ul de date să fie posibil, clienții trebuie să stabilească modalitatea prin care ei vor comunica, precum și detalii asupra conținutului pe care îl vor transmite. Pentru acesta, s-a creat protocolul SDP (Session Description Protocol). Este specificat în RFC 4566 și este strict un protocol pentru descriere de sesiuni. Nu incorporează mecanisme de transport, așadar, pentru transmiterea descrierilor, se va folosi alt protocol precum SIP sau WebSocket [PHJ06]. De asemenea, nu suportă negociere de conținut, deoarece se consideră a fi un concept separat de cel al descrierii sesiunilor [PHJ06].

3.3.1 Structura unui descriptor de sesiune SDP

SDP are ca scop oferirea de informații despre streamurile multimedia ce vor permite noilor participanți să se alăture unei sesiuni. [PHJ06]. Streamurile pot fi multicast, așadar vor exista mai mulți clienți care vor transmite unui grup mai larg. În acest caz, SDP deservește și rolul de a comunica existența unei sesiuni [PHJ06].

O descriere SDP conține:

- Numele sesiunii și scopul;

- Timpul la care sesiunea este activă;
- Conținutul multimedia asociat sesiunii;
- Informația necesară pentru a primi conținutul (adrese, porturi, formate, codecuri etc.) [PHJ06].

De asemenea, este indicată menționarea lățimii de bandă folosită de sesiune și a informațiilor de contact despre persoana responsabilă de sesiune, deoarece resursele participanților pot fi limitate [PHJ06].

Conținutul SDP este descris prin tipul `application/sdp`, pentru o interpretare corectă de către application-level gateway-uri prin oricare protocol de transport [PHJ06].

Structura sa va fi sub forma unor perechi `tip=valoare`, delimitate pe rânduri separate și **fără spațiu** în fața sau în spatele egalului. Fiecare câmp de tip va conține o singură literă mică din alfabetul englez. Fiecare valoare poate conține restul de caractere din setul ISO 10646, cu excepția denumirilor tipurilor de atribute [PHJ06]. Va conține următoarele câmpuri (în această ordine):

- `v=` (versiunea protocolului)
- `o=` (originea și identificatorul sesiunii)
- `s=` (numele sesiunii)
- `i=*` (informații despre sesiune)
- `u=*` (URL-ul descrierii)
- `e=*` (adresă e-mail)
- `p=*` (număr de telefon)
- `c=*` (informații despre conexiune; opțional dacă sunt incluse în descriptorii media)
- `b=*` (informații despre lățimea de bandă)
- Unul sau mai mulți descriptori temporali
- `z=*` (variațiuni ale fusului orar)
- `k=*` (cheie de criptare)
- `a=*` (una sau mai multe linii; atribute despre sesiune)
- Zero sau mai mulți descriptori media

  `*` - parametrul este opțional

Descriptorii temporali vor avea următorul format:

- `t=` (intervalul de timp la care sesiunea este activă)
- `r=*` (numărul de repetări: zero sau mai mult)

Descriptorii media, în cazul în care există, vor avea următorul format:

- `m=` (numele fluxului media și adresa de transport)

- `i=*` (titlul fluxului)
- `c=*` (informații despre conexiune; opțional dacă este inclus la nivel de sesiune)
- `k=*` (cheia de criptare)
- `a=*` (una sau mai multe linii; atribute despre media)

Setul de tipuri nu este conceput să fie extensibil, așadar, dacă întâlnește un câmp de tip pe care nu îl recunoaște, va ignora întregul descriptor [PHJ06]. Pentru aceasta, este pus la dispoziție mecanismul `a=`, al cărui rol este de a extinde SDP și de a-l face potrivit aplicațiilor particulare [PHJ06].

```
v=0
o=jdoe 2890844526 2890842807 IN IP4 10.47.16.5
s=SDP Seminar
i=A Seminar on the session description protocol
u=http://www.example.com/seminars/sdp.pdf
e=j.doe@example.com (Jane Doe)
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
a=recvonly
m=audio 49170 RTP/AVP 0
m=video 51372 RTP/AVP 99
a=rtpmap:99 h263-1998/90000
```

Listing 3.1: Exemplu de descriptor SDP [PHJ06]

Analizând exemplul de mai sus, observăm că, în special pentru câmpurile `o=`, `c=` și `m=`, avem informații ce par criptice la prima vedere. Luând pe bucăți, observăm că `o=` va conține următorii parametri: numele de utilizator (`jdoe`), identificatorul sesiunii (`2890844526`), versiune a sesiunii (`2890842807`), tipul rețelei (`IN`, prescurtarea pentru Internet), tipul adresei (`IP4`, IP versiunea 4), precum și adresa inițiatorului (`10.47.16.5`).

Continuând cu câmpul `c=`, observăm informații despre conexiune. `IN` se va regăsi și aici și are aceeași semnificație precum la `o=`, asemenea și tipul `IP4`. Însă adresa IP arată diferit. Asta se datorează faptului că este o adresă multicast, la care trebuie atașat și TTL-ul (time to live) separat cu o bară oblică.

Pentru `t=`, cele două valori reprezintă numărul de secunde trecute începând cu 1 ianuarie 1900, specific protocolului NTP (Network Time Protocol), reprezentate pe 64 de biți. De observat este că, în anul 2036, se va atinge limita reprezentării în NTP, dar nu va reprezenta o problemă, deoarece SDP folosește reprezentare pe număr arbitrar de cifre.

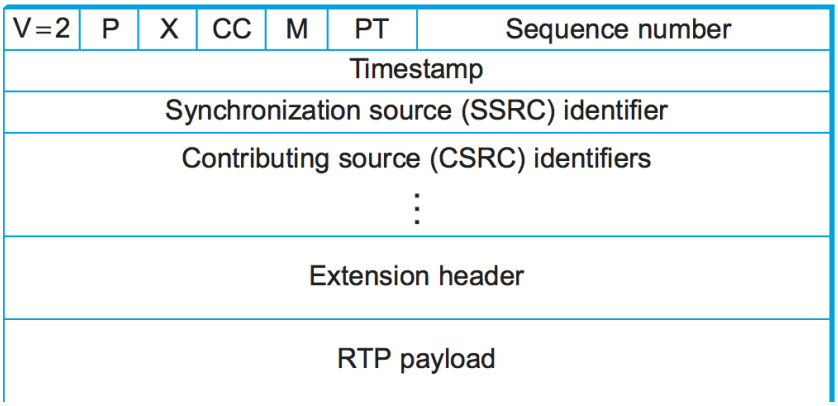
Pentru cele două câmpuri `m=`, reprezentând descriptorii media, vom avea tipurile de media `audio` și `video`. Valorile `text`, `application` și `message` sunt valide. Urmează porturile `49170`, respectiv `51372` pe care se va realiza transmisia. Protocolul de transmisie este `RTP/AVP`, însemnând că RTP folosește profilul audio-video. Se poate întâlni și `RTP/SAVP`, care se distinge prin faptul că RTP este criptat. Următoarele numere, `0` și `99` reprezintă indecși de descriere media și vor avea câte un câmp `a=` asociat, iar, pentru că se folosește RTP pentru transport, se va regăsi `rtpmap`. Putem avea mai mulți indecși, prin care putem menționa transmisia a mai multe tipuri de media pe același port. Este menționat codecul `h263-1998`, împreună cu clock rate-ul (engl. *rata de ceas*) de `90000`, standard pentru acest codec.

3.3.2 RTP

După stabilirea adreselor IP prin care se va realiza transportul conținutului multimedia, trebuie găsită o convenție pentru a-l transporta. Cel mai des folosit protocol de transport este RTP și a fost prima dată specificat în RFC 1889 în ianuarie 1996. Are o strânsă legătură cu RTCP (RTP Control Protocol), destinat transmiterii de feedback asupra calității stream-ului RTP, ceea ce îl face complementar [SCFJ03].

RTP nu oferă mecanisme de garantare a transmiterii, lăsând responsabilitatea protocoalelor de nivel mai jos precum UDP, însă folosește numere de secvență pentru ca receptorul să poată reordona corect pachetele sau să determine poziția corectă a pachetului [SCFJ03]. Va fi atașat, de obicei, unui port cu număr par între 16384 și 32767 și va forma pereche cu un stream RTCP aflat pe portul imediat următor, cu număr impar.

Pachetele RTP respectă următoarea structură:



De observat este că fiecare rând din tabel va reprezenta câte 4 octeți (32 de biți). De asemenea, există o varietate de parametri ce au următoarea semnificație:

- `V=2` - versiunea 2 a protocolului RTP, va ocupa 2 biți;
- `P` (padding) - bitul de aliniere; va simboliza faptul că pachetul RTP va conține octeți adiționali pentru a-l alinia la o anumită dimensiune, necesară unor algoritmi de criptare; ultimul octet de aliniere va conține numărul de octeți ce trebuie ignorați, inclusiv pe el însuși;
- `X` (extension) - bitul de extensie; va menționa existența unui header (engl. *antet*) de extensie ce se va afla imediat după antetul fix al pachetului RTP;
- `CC` (CSRC Count) - numărul de identificatori CSRC prezenți după antet; va ocupa 4 biți;
- `M` (marker): bit a cărui interpretare variază în funcție de profilul RTP;
- `PT` (payload type) - tipul de conținut transmis în pachetul RTP, va ocupa 7 biți;
- sequence number - numărul pachetului RTP; de menționat este că valoarea sa inițială este aleatoare, pentru a îngreuna atacurile criptografice; va ocupa 32 de biți;
- timestamp - folosit pentru a marca temporal locul unde ar trebui să fie redat pachetul RTP; asemenea sequence number-ului, valoarea sa inițială trebuie să fie aleatoare;
- SSRC (synchronization source) identifier - identificatorul sursei, destinat sincronizării și evitării coliziunilor în cazul în care va primi pachete de la altă sursă;
- CSRC (contributing source) identifiers - lista surselor care vor contribui la transmiterea pachetului RTP [SCFJ03].

3.4 Implementare în browser

WebRTC este furnizat cu fiecare browser web popular, precum și cu framework-urile derivate (ex. Electron). Expune mai multe API-uri JavaScript ușor de folosit, care ascund detaliile de implementare și simplifică munca dezvoltatorului. Fiecare API este responsabil de setul său specific de funcții, precum: cameră și microfon, partajarea conținutului ecranului și conexiuni peer-to-peer.

3.4.1 Capturarea conținutului media

Pentru a putea realiza streaming-ul audio-video cu un scop, o sursă de conținut este necesară. Media Devices este un numitor comun pentru camerele și microfoanele conectate. În JavaScript, aceste device-uri sunt accesibile prin intermedul interfeței *navigator.mediaDevices*, prin care toate dispozitivele conectate pot fi enumerate, urmărite pentru schimbări, precum și deschise pentru a obține o instanță de MediaStream [Weba].

Acest API este folosit prin apelul funcției *mediaDevices.getUserMedia()*, care returnează un promise al cărei funcție de resolve conține un parametru pentru MediaStream-ul asociat dispozitivului. Aceasta cere furnizarea ca parametru un obiect de tip MediaStreamConstraints, prin care constrângeri asupra sursei audio și video se pot specifica, precum și, opțional, identitatea peer, singura care are are acces la stream, unde conținutul este protejat ca și cum regulile CORS cross-origin ar fi în aplicare.

O altă opțiune este de a partaja conținutul ecranului. Aceasta este posibilă prin intermediul funcției *mediaDevices.getDisplayMedia()*, foarte similară cu *getUserMedia*, care de asemenea returnează un promise ce rezolvă un MediaStream. Constrângerile sunt diferite: alegerea între multiple opțiuni de a afișa cursorul (mereu, doar când este în mișcare, sau niciodată) și de a afișa zona capturată (întregul ecran, doar o fereastră, sau o filă din browser).

Înregistrarea conținutului unui MediaStream este posibilă prin API-ul MediaRecorder, dar în această teză, ne vom concentra doar pe streaming.

Vizionarea streamului o chestiune de a crea un elementul HTML *video*, setarea obiectului sursă ca fiind MediaStream-ul și de a furniza o funcție care să redea streamul când metadatele sunt încărcate, ca event handler în proprietatea *onloadedmetadata* din elementul *video*.

3.4.2 Inițializarea unei conexiuni peer

Conexiunile peer sunt o parte a specificațiilor WebRTC care ajută la stabilirea unei conexiuni între două aplicații pe două calculatoare diferite pentru a comunica printr-un protocol peer-to-peer. Acestea pot transmite audio, video, sau date bine (cât timp clienții suportă API-ul RTCDataChannel) [Webb].

Fiecare conexiune peer este gestionată de un obiect RTCPeerConnection, care este instanțiată prin constructorul său specificat ce primește ca parametru un RTCConfiguration, care definește modul în care conexiunea peer este pregătită și care ar trebui să conțină informații despre serverele ICE folosite.

Pentru inițierea comunicării, este prioritară crearea unei cereri sau unui răspuns SDP, în funcție dacă este vorba despre peer-ul ce apelează, sau despre peer-ul ce

răspunde. Apelantul trebuie să trimită obiectul SDP pe care l-a creat peer-ului de la distanță printr-un canal de signaling, diferit de cel prin care se vor transmite datele. Procedura numită signaling nu are o definiție standard.

Inițierea unui RTCPeerConnection din apelant cere crearea unui descriptor local, obiect de tip RTCSessionDescription prin metoda *createOffer()* din instanța de RTCPeerConnection. Va fi setat ca fiind descriptor local prin *setLocalDescription()* și va fi trimis la apelat prin canalul de signaling. Apelatul, când primește descriptorul sesiunii, va apela *setRemoteDescription()* și va crea un răspuns la cererea primită cu *createAnswer()* care de asemenea va fi trimis prin canalul de signaling. Inițiatorul apelului va primi răspunsul și îl va seta ca fiind descriptorul remote.

3.4.3 Adăugarea stream track-urilor

După crearea unei instanțe de RTCPeerConnection, track-urile de stream trebuie adăugate. Revenind la dispozitivele media și la obiectele sale MediaStream, track-urile sunt conținute într-o listă accesibilă din funcția *getTracks()*. Acestea pot fi adăugate la conexiunea peer cu *addTrack*, care cere instanța streamului [Webc].

Peer-ul care răspunde nu conține o instanță proprie de MediaStream care să conțină track-urile, așadar trebuie să creeze una și să adauge track-urile remote la ea. Un handler de evenimente numit *ontrack* trebuie adăugat. Gestionează câte un track odată. În acest caz, handler-ul va adăuga la instanța nou creată de *MediaStream*, care va fi setată ca obiectul sursă al elementului *video*.

3.4.4 Obținerea candidaților ICE

Schimbul de informație de conectivitate este obligatoriu înainte ca doi peers pot comunica. Condițiile rețelei pot fi variabile în funcție de un număr de factori (ex. ascunderea adreselor sale IP prin NAT), așadar un intermediar este folosit pentru a descoperi candidații pentru conectarea la peer. Acesta este un serviciu extern și se numește ICE.

Imediat după setarea descriptorului local, se va iniția căutarea candidaților ICE, în următoarea ordine: cei host, listând fiecare interfață de rețea (fizică sau virtuală) instalată pe computer, apoi interogând serverele STUN și/sau TURN specificate în câmpul *iceServers* al obiectului de tip RTCConfiguration folosit la crearea obiectului RTCPeerConnection.

Fiecare instanță de RTCPeerConnection conține un handler de evenimente pentru noii candidați, numit *onicecandidate*. Va fi apelat de fiecare dată când găsește un candidat nou. Este responsabilitatea dezvoltatorului să îl preia fiecare prin event handler și să îl trimită celuilalt participant la apel, urmând ca acela să îl adauge folosind metoda *addIceCandidate()*.

Capitolul 4

Aplicații WebRTC

4.1 Microsoft Teams

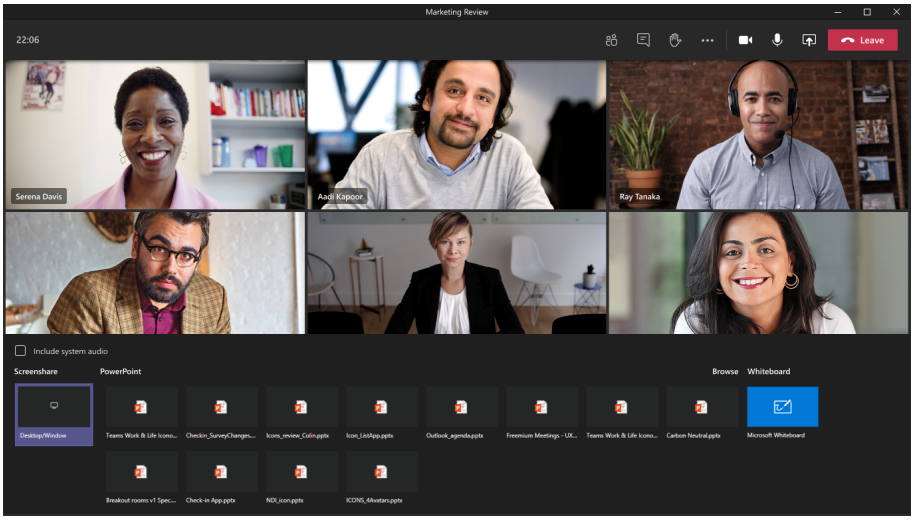


Figura 4.1: Conversație pe Teams

Aplicația propriu-zisă a fost implementată inițial folosind framework-ul Angular. În 2018, Abhik Mitra a propus migrarea treptată către React [Mit18], astfel că dezvoltatorii pot crea în zilele noastre aplicații pentru Teams folosind componente React puse la dispoziție de Microsoft.

Teams este conceput pentru a fi folosit în mediul profesional, în care există posibilitatea ca angajații să folosească VPN-uri, ca unii clienți să folosească Skype for Business și/sau să participe dintr-o rețea externă companiei sau chiar de pe o linie telefonică obișnuită, prin rețeaua PSTN. Pentru a putea face conversația în aceste cazuri posibilă, Microsoft a implementat o serie de mecanisme de redistribuire a streamurilor media. Transport relay-ul este un server STUN, folosit pentru a obține candidații ICE, și este parte a suitei Microsoft 365. Va funcționa ca

releu media, asemenea serverului TURN, în cazul în care un participant nu se află în aceeași rețea. Protocolul de signaling folosit este preponderent HTTPS, prin servicii REST [Tea], însă va folosi și SIP pentru a interacționa cu SBC-uri (session border controller).

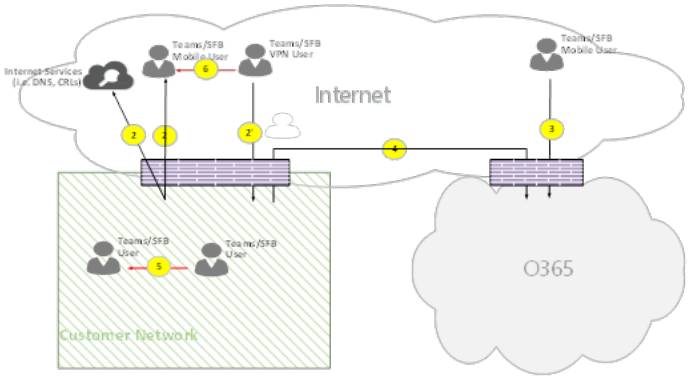


Figura 4.2: Topologia de rețea folosită în apelurile Microsoft Teams în care toți participanții folosesc Teams

Pentru a face posibile conversațiile pe Teams prin linia telefonică, Microsoft pune la dispoziție așa numitul Phone System prin abonamentul Microsoft 365, care este un private branch exchange (PBX), o centrală telefonică.

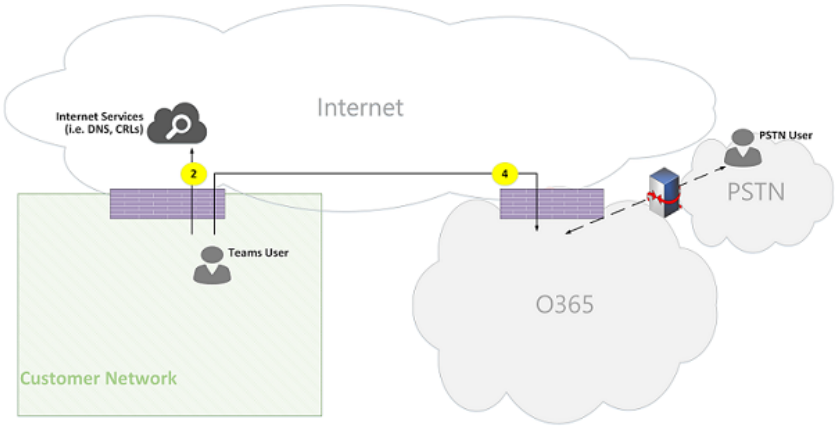


Figura 4.3: Topologia unei conversații între un utilizator Teams și un utilizator al liniei telefonice obișnuite

4.2 Jitsi Meet

Jitsi Meet este o aplicație open-source implementată, de asemenea, cu WebRTC. Asociat acestei aplicații este Jitsi Videobridge, un server SFU ce suportă simulcast și care este considerat mai eficient decât un MCU. Oricine poate să îl instaleze local. Signaling-ul va fi realizat folosind un server Prosody prin protocolul XMPP [Jit].

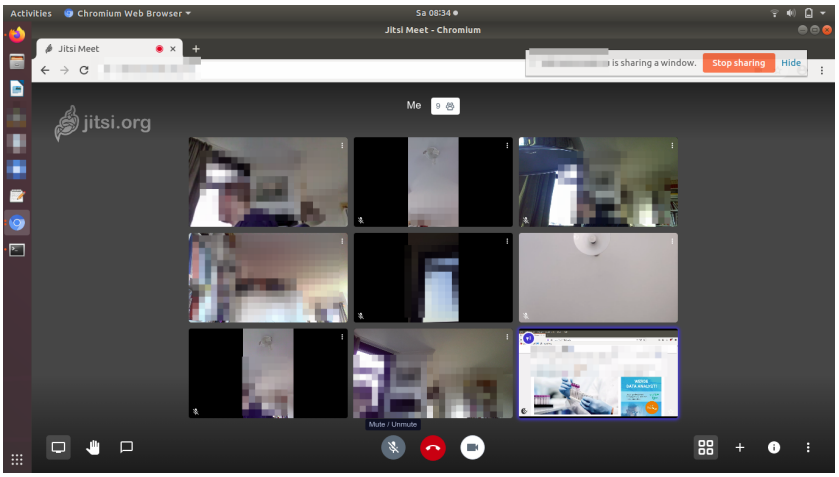


Figura 4.4: Conversație pe Jitsi

Teste de performanță au fost făcute pe o configurație cu un procesor quad-core Intel Xeon E5-1620 v2 la 3,7 GHz. S-a observat că o dată cu creșterea lățimii de bandă folosite, consumul de resurse a crescut liniar 4.6. Folosind 33 de clienți (load generators), serverul a distribuit 1056 de streamuri, iar lățimea de bandă folosită a ajuns la 550 Mbps 4.5 [GI]. 20% din capacitatea procesorului a fost folosită, conform datelor furnizate de procesul *top* [GI].

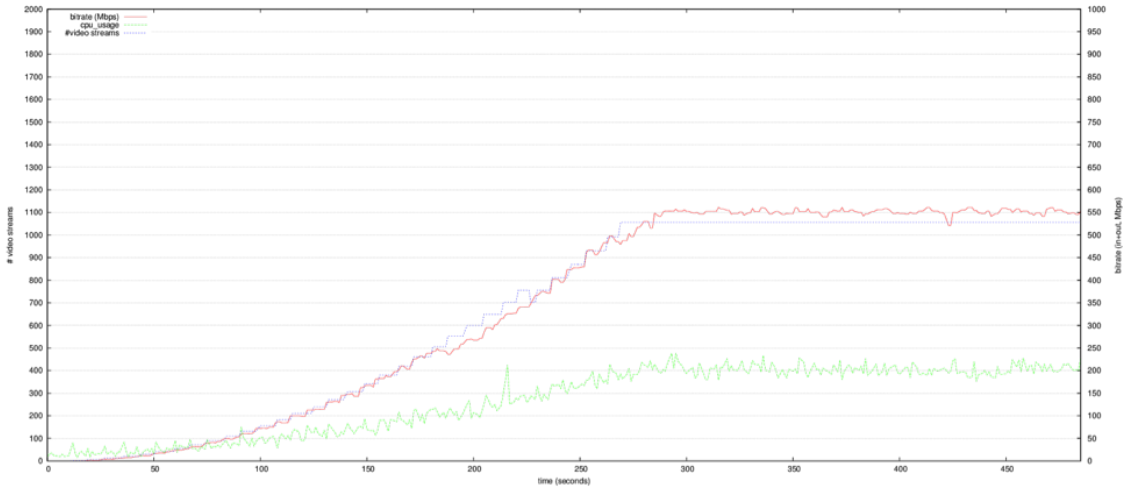


Figura 4.5: Grafic pentru lățimea de bandă folosită direct proporțională cu numărul de streamuri

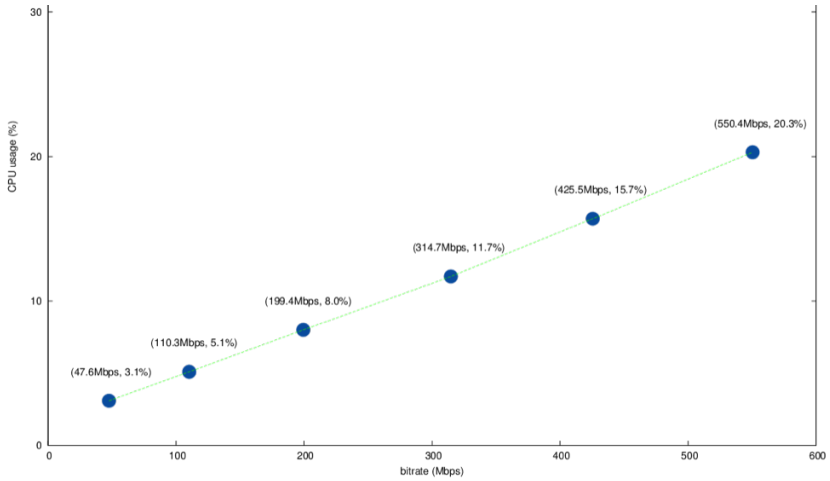


Figura 4.6: Grafic al capacității procesorului folosite în funcție de lățimea de bandă

Deoarece pot fi mulți participanți într-o conferință, s-a creat Jicofo (Jitsi Conference Focus), care are rolul de load balancing între fiecare participant și Videobridge [Jit].

În cazul participanților din diferite zone geografice, aceștia se vor conecta la cel mai apropiat SFU (bridge), urmând ca serverele la care s-au conectat clienții să își transmită reciproc stream-urile [Ivo18]. Această tehnică se numește bridge cascading [Ivo18].

4.3 Zoom

Figura 4.7: Conversație pe Zoom

Spre deosebire de alte platforme de videoconferințe, Zoom folosește o implementare care se bazează pe RTCDataChannel și nu pe protocolul RTP [Han19]. Conținutul video folosește codecul H.264 Annex G, care permite o scalabilitate mai bună datorită codării stratificate și pe care browserul Chrome nu îl suportă nativ [Sac20, Han19]. așadar, include furnizează un codec special compilat în WebAssembly [Han19].

Zoom folosește un router multimedia care separă conținutul ce trebuie transcodat și procesat de cel care trebuie pur și simplu redistribuit, ceea ce îi oferă un avantaj real în privința scalabilității și a consumului de resurse [Sac20].

Nu folosește server STUN sau TURN, dovada fiind lista *iceServers* goală din configurația obiectului RTCPeerConnection [Han19]. În cazul în care nu se poate folosi UDP (ex.: este blocat pe firewall), va folosi implementarea veche pe bază de WebSocket, încercând să instanțieze un nou obiect RTCPeerConnection la fiecare 10 secunde [Han19].

```
▼ Connection:14232-1 URL: https://zoom.us/wc/...
Configuration: "{ iceServers: [], iceTransportPolicy: all, bundlePolicy: balanced,
Signaling state: => SignalingStateHaveLocalOffer => SignalingStateStable
ICE connection state: => checking => connected
```

Figura 4.8: Configurația folosită de clientul web Zoom

Zoom, de asemenea, manipulează descriptorul creat folosind funcția *createOffer* înainte să îl seteze ca fiind cel local. Va schimba username fragment-ul candidaților (prezent în SDP ca `a=ice-ufrag`) dintr-unul scurt, generat de Chrome, cu un UUID lung [Han19]. Folosește specificația *ice-lite*, o versiune minimală a protocolului ICE, marcată în SDP ca `a=ice-lite`.

Capitolul 5

Studiu de caz

Capitolul 6

Aplicația practică

6.1 Specificație

6.2 Design

6.3 Utilizare și testare

Capitolul 7

Concluzii

Prin această lucrare am demonstrat importanța standardizării protocoalelor ce contribuie la distribuția eficientă a streamurilor audio-video în videoconferințe, precum și a alegerii unei topologii potrivite scenariului de utilizare. Videotelefonia are o istorie îndelungată, timp în care a reușit să devină accesibilă oricui, dovedindu-și utilitatea mai ales în zilele noastre.

Prin analiza pachetelor ICE, SDP și RTP, prin descrierea API-ului JavaScript și prin multitudinea de topologii posibile am observat că WebRTC este o librărie maleabilă și foarte apreciată atât de marile corporații, pe cât și de dezvoltatorii independenți. Iar prin implementarea aplicației punând în aplicare o topologie înlănțuită împreună cu tehnica de a redistribui stream-ul, am realizat o optimizare utilă în cazul existenței multor participanți într-un apel, care ajută la evitarea compromisului de a reduce calitatea apelului.

O posibilă îmbunătățire ar fi extinderea topologiei de distribuție înlănțuite într-una arborescentă, ce ar profita de lățimea de bandă largă pe care unii participanți o au la dispoziție și de lungimea mai scurtă a unui drum de la broadcaster la clienții cei mai depărtați din topologie, așadar de o latență redusă.

Bibliografie

[Avo] A History of Video Conferencing. https://www.videonations.co.uk/resources/blog/a-history-of-video-conferencing/.

[Bas04] Henning Baset, Salman A. Schulzrinne. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. *CoRR*, abs/cs/0412017, 2004.

[Bor] David E. Borth. Videophone. *Encyclopædia Britannica*.

[Dor95] Dorcey, Timothy. CU-SeeMe Desktop VideoConferencing Software. *Connexions*, 9(3), 1995.

[EAST18] Naktal Edan, Ali Al-Sherbaz, and Scott Turner. Design and Implement a Hybrid WebRTC Signalling Mechanism for Unidirectional & Bi-directional Video Conferencing. *International Journal of Electrical and Computer Engineering*, 8:390–399, 02 2018.

[GI] Boris Grozev and Emil Ivov. Jitsi videobridge performance evaluation. https://jitsi.org/jitsi-videobridge-performance-evaluation/.

[H.203] ITU-T Recommendation H.264: Advanced video coding for generic audiovisual services. Technical report, 05 2003.

[Han19] Philipp Hancke. How Zoom's web client avoids using WebRTC (DataChannel Update). https://webrtchacks.com/zoom-avoids-using-webrtc/, 2019.

[Ivo18] Emil Ivov. Jitsi Meet, now with geographical bridge cascading. https://jitsi.org/blog/jitsi-meet-now-with-geographical-bridge-cascading/, 2018.

[Jit] Architecture. https://jitsi.github.io/handbook/docs/architecture.

[KHR18] Ari Keränen, Christer Holmberg, and Jonathan Rosenberg. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal. RFC 8445, July 2018.

[Lan02] Lange, André. L'histoire de la télévision commence par un canular: l'électroscope de 1877. `https://www.histv.net/electroscope-1878`, 2002.

[Lan20] Lange, André. L'attribution imaginaire de l'invention du télectroscope à Graham Bell par l'Abbé Moigno et Louis Figuier (1877-1878). `https://www.histv.net/telectroscope1877`, 2020.

[Mit18] Abhik Mitra. Moving a huge AngularJS App to React, progressively at Microsoft Scale. `https://hasgeek.com/jsfoo/2019-pune/sub/moving-a-huge-angularjs-app-to-react-progressively-X3igB6hPLD` 2018.

[MRWM08] Philip Matthews, Jonathan Rosenberg, Dan Wing, and Rohan Mahy. Session Traversal Utilities for NAT (STUN). RFC 5389, October 2008.

[Nym13] Robert Nyman. Hello Chrome, it's Firefox calling! *Mozilla Hacks*, 02 2013.

[Per00] Christine Perey. Microsoft expands platform for voice and video on the LAN. *Computer World*, 2000.

[PHJ06] Colin Perkins, Mark J. Handley, and Van Jacobson. SDP: Session Description Protocol. RFC 4566, July 2006.

[Qui] Connectix QuickCam. `http://people.vcu.edu/~aplinas/opticnerve/quickcam.html`.

[RFC77] Specifications for the Network Voice Protocol (NVP). RFC 741, 11 1977.

[Roo00] Root, Timothy D. Audio communications product strategy for the 21st century. Master's thesis, Massachusetts Institute of Technology, 2000.

[Ros10] Jonathan Rosenberg. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. RFC 5245, April 2010.

[Roy07] Greg Royal. Apple iChat Next Wave of VoIP. *Network World*, 06 2007.

[Sac20] Vikram Sachdeva. Zoom — Video conf app at scale. *Medium*, 2020.

[SCFJ03] Henning Schulzrinne, Stephen L. Casner, Ron Frederick, and Van Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, July 2003.

[SFU20] An Introduction to Selective Forwarding Units. `https://voximplant.com/blog/an-introduction-to-selective-forwarding-units`, 2020.

[SSRH99] Henning Schulzrinne, Eve Schooler, Jonathan Rosenberg, and Mark J. Handley. SIP: Session Initiation Protocol. RFC 2543, March 1999.

[Tea] Microsoft Teams call flows. `https://docs.microsoft.com/en-us/microsoftteams/microsoft-teams-online-call-flows`. Online; accessed 21 May 2021.

[Uen20] Uenuma, Francine. Video Chat Is Helping Us Stay Connected in Lockdown. But the Tech Was Once a 'Spectacular Flop'. *Time*, 2020.

[Vid] Distance Multimedia: 4 score & more. `https://vidconf.net/`.

[Weba] Getting started with media devices. `https://webrtc.org/getting-started/media-devices`. Online; accessed 13 April 2021.

[Webb] Getting started with peer connections. `https://webrtc.org/getting-started/peer-connections`. Online; accessed 13 April 2021.

[Webc] Getting started with remote streams. `https://webrtc.org/getting-started/remote-streams`. Online; accessed 13 April 2021.

[Whi13] Zack Whittaker. Skype ditched peer-to-peer supernodes for scalability, not surveillance. *ZDNet*, 2013.

[Wol19] Erinn Wolfe. The History of Video Conferencing from 1870 to Today. *Lifesize*, 2019.