 **UNIVERZITET U ZENICI**

**Politehnički fakultet Softversko inženjerstvo**

**Komunikacija kao app**

**Predmet**: Multimedijalni sistemi i aplikacije

**Profesor:Studenti:**

doc.dr. Nermin Goran Amar Ćatović, Vedad Karalić

Zenica, novembar 2022

Sadržaj

[**Uvod** 3](#_Toc119623018)

[WebRTC 3](#_Toc119623019)

[Princip rada WebRTC 3](#_Toc119623020)

[Implementacija 5](#_Toc119623021)

[**Reference** 23](#_Toc119623022)

[**Zaključak** 24](#_Toc119623023)

**Uvod**

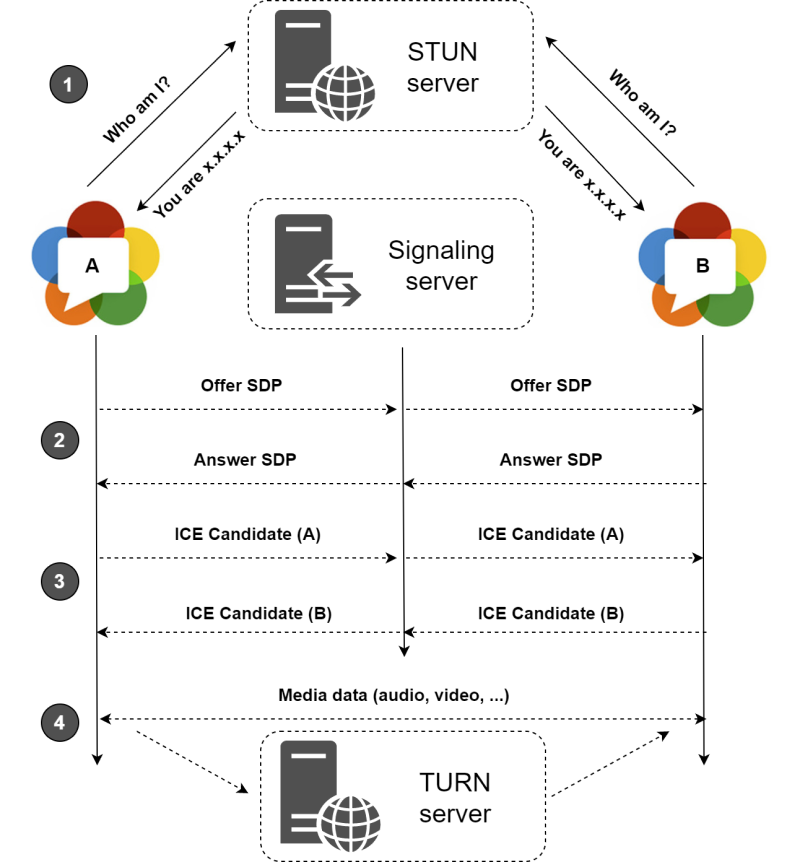
U svijetu web aplikacija, svi vole scenarij u kojem jedna aplikacija zahtijeva akciju, a druga aplikacija naknadno odgovara na taj zahtjev. Ovaj osnovni, ali efikasan obrazac je, u većini slučajeva, sve što je vašim web aplikacijama potrebno za pravilno funkcioniranje. Međutim, što se događa ako jedna aplikacija ne može čekati da druga aplikacija završi svoju radnju, a zatim pošalje odgovor? U razmjeni trenutnih poruka i video konferencijskim pozivima, na primjer, obje aplikacije moraju biti savršeno sinhronizirane jedna s drugom kako bi mogle komunicirati u realnom vremenu. Ovo je mjesto gdje komunikacijske web tehnologije u realnom vremenu dolaze u centar pažnje.

# WebRTC

Web tehnologija komunikacije u realnom vremenu (RTC) je termin koji se koristi za bilo koju web tehnologiju koja omogućava živu komunikaciju između dva ili više vršnjaka preko Interneta bez kašnjenja u prijenosu. U posljednje vrijeme, lider u ovoj oblasti je jedna moderna, perspektivna i moćna tehnologija pod nazivom WebRTC. WebRTC je besplatna tehnologija otvorenog koda koja pretraživačima i mobilnim aplikacijama pruža mogućnosti komunikacije u realnom vremenu (RTC) putem jednostavnih sučelja za programiranje aplikacija (API). Omogućava direktnu peer-to-peer audio i video komunikaciju, eliminirajući potrebu za instaliranjem dodatnih dodataka ili izvornih aplikacija. Stvorio ga je Google u maju 2011. godine kao tehnologiju otvorenog koda za komunikaciju u realnom vremenu zasnovanu na pretraživaču. Trenutno je WebRTC potpuno stabilan, njegovo posljednje izdanje (v1.0) napravljeno je u maju 2018. Standardizirano je kroz World Wide Web Consortium (W3C) i Internet Engineering Task Force (IETF), a podržavaju ga Apple, Google, Microsoft , Mozilla i Opera.

## Princip rada WebRTC

WebRTC dolazi s unaprijed definiranim algoritmom za razmjenu poruka, koji se mora konfigurirati u softverskom rješenju kako bi WebRTC komunikacija mogla započeti i normalno funkcionirati. Slika ispod ilustruje šta se dešava kada se uspostavi WebRTC veza.

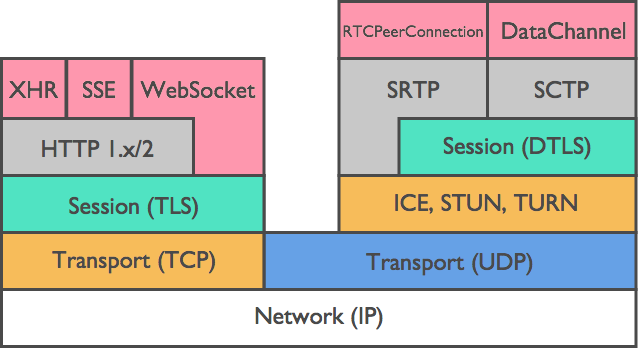


Slika 1. Shema WebRTC

Da sumiramo, postoje četiri glavna koraka u pokretanju WebRTC sesije:

* Peer-i treba da pronađu svoju javnu IP adresu koristeći STUN – Session Traversal Utilities za NAT – server.
* Nakon što vršnjaci pronađu svoju javnu IP adresu, klijent A treba da pošalje 'Ponudi SDP' klijentu B preko signalnog servera, nakon čega klijent B treba da potvrdi ponudu i vrati 'SDP za odgovor' klijentu A. SDP – Session Description Protocol – je format za opisivanje parametara komunikacije striming medija koji se koriste za najavu sesije i validaciju njenih parametara.
* Nakon što je sesija najavljena i potvrđena, klijent A treba da pošalje 'ICE kandidate' klijentu B preko signalnog servera, nakon čega klijent B treba da potvrdi ICE kandidate i odgovori istom vrstom poruke. ICE – Interactive Connectivity Establishment – ​​je metoda koja se koristi za preuzimanje svih dostupnih kandidata (IP adresa), koji se zatim prosljeđuju klijentima.
* Kada je sesija u potpunosti potvrđena i ICE kandidati spremni, WebRTC audio i video komunikacija može započeti, bilo direktno ili putem TURN – Traversal koristeći releje oko NAT – servera.

Ispod haube, WebRTC koristi nekoliko različitih protokola za pružanje pouzdane audio i video komunikacije.



Slika 2. WebRTC protocol

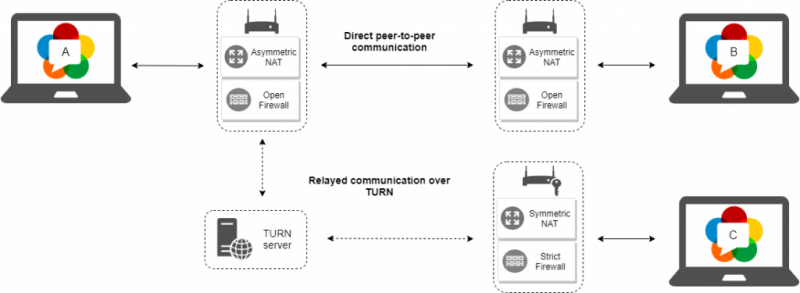
Kao što slika iznad pokazuje, WebRTC koristi UDP – User Datagram Protocol – na transportnom sloju, uglavnom zato što je UDP protokol osnova za komunikaciju u realnom vremenu u modernim pretraživačima. Osim toga, WebRTC koristi pomenute ICE, STUN i TURN protokole za uspostavljanje i održavanje peer-to-peer konekcije preko UDP-a. Konačno, SCTP – Transportni protokol za kontrolu toka – i SRTP – Secure Real-Time Transport Protocol – koriste se za multipleksiranje različitih tokova, obezbjeđivanje sigurnosti i zagušenja i obezbjeđivanje djelomično pouzdane isporuke na vrhu UDP-a.

# Implementacija

Kao što se vidi u objašnjenju WebRTC algoritma za razmjenu poruka, da bi se WebRTC implementirao u softversko rješenje, potrebno je konfigurirati tri glavne stvari:

* Signalni server
* STUN i TURN serveri
* Aplikacije na strani klijenta (web ili mobilne)

Server za signalizaciju je neophodna aplikacija na strani servera koju klijentske aplikacije koriste za međusobno komuniciranje. Koristi tehnologiju mrežnih utičnica (WebSocket, socket.io ili slično rješenje) kako bi organizirao klijente u određene grupe (obično se zovu sobe), kako bi preslušao njihove poruke i preusmjerio ih na pravog primatelja. Implementacija signalnog servera je obično prilično jednostavna, ali mora biti sinkronizirana s algoritmom za razmjenu poruka implementiranim u klijentskim aplikacijama.



Slika 3. Shema STUN i TURN servera

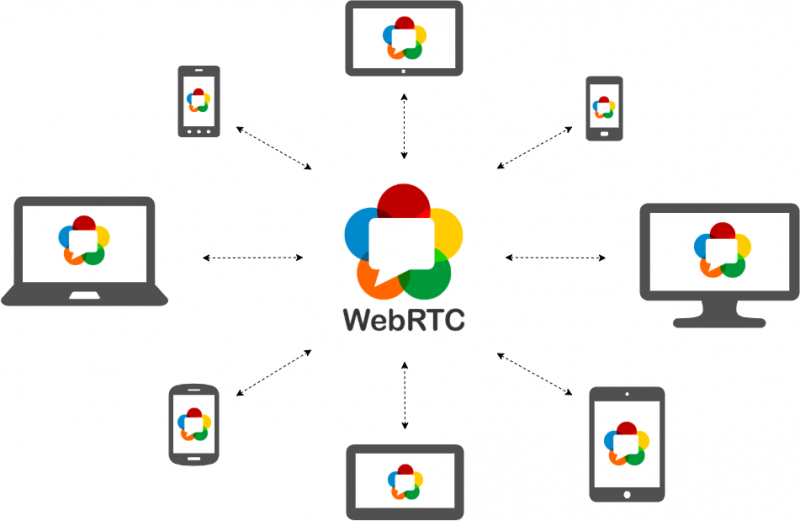
TURN serveri se mogu implementirati po narudžbi (bilo u potpunosti ili korištenjem rješenja otvorenog koda, kao što je coTURN) ili se mogu kupiti kao usluga (npr. Xirsys). Preporučuje se da u WebRTC konfiguraciju uključite nekoliko STUN servera, kao i nekoliko TURN servera. Zapravo, što više servera koristite, to je veza stabilnija. Međutim, također je vrijedno zapamtiti da, nakon određene točke, korištenje previše može usporiti proces povezivanja.

STUN serveri su lagani serveri koji se koriste da pomognu klijentima da pronađu svoju javnu IP adresu. Nije ih tako teško postaviti i također su uobičajeno dostupni besplatno (npr. Google STUN serveri su potpuno besplatni za korištenje). Kada vršnjaci saznaju svoju javnu IP adresu, mogu nastaviti s procesom slanja poruka, nakon čega se uspostavlja veza i može započeti prijenos medija. A šta je sa TURN serverima, pitate se? U savršenom svijetu, STUN serveri bi uvijek bili dovoljni za uspostavljanje WebRTC veze. Međutim, u oko 30 posto slučajeva pronalaženje javne IP adrese vršnjaka preko STUN servera nije dovoljno za uspostavljanje veze. Ovo se uglavnom dešava zbog strogih pravila zaštitnog zida ili simetričnog NAT-a (prevod mrežnih adresa) koji se koristi u mrežnoj konfiguraciji kolega. U tim slučajevima, TURN server je neophodan da uspostavi vezu i da služi kao relej prilikom emitovanja medijskih podataka između vršnjaka.

Posljednji korak u implementaciji WebRTC-a je konfiguracija aplikacija na strani klijenta (web ili mobilne aplikacije). U programiranju aplikacije na strani klijenta, potrebno je konfigurirati dvije glavne stvari da bi WebRTC u potpunosti funkcionirao:

* Dobivanje korisničkog medija specifičnog za klijenta – sastoji se od korištenja nekoliko unaprijed definiranih WebRTC API-ja u programskom kodu. Programeri WebRTC-a su zaista olakšali programerima implementaciju ovog dijela, tako da bi nekoliko redova koda po tehnologiji trebalo učiniti trik.
* Definisanje komunikacije putem utičnica – znači kreiranje algoritma za razmjenu poruka u programskom kodu, koji će omogućiti svakom ravnopravnom uređaju da prođe kroz svaki korak prethodno opisanog WebRTC algoritma za razmjenu poruka u pravom redoslijedu.

Kada je komunikacija preko utičnica ispravno definirana i sinkronizirana s implementacijom signalnog servera, klijent se samo treba pridružiti grupi utičnica, dobiti korisnički medij sa uređaja i pokrenuti obrazac za razmjenu poruka preko signalnog servera. Ako je sve ispravno konfigurirano, WebRTC sesija bi se trebala automatski inicijalizirati i poziv može započeti.



Slika 4. Konceptualna shema aplikacije

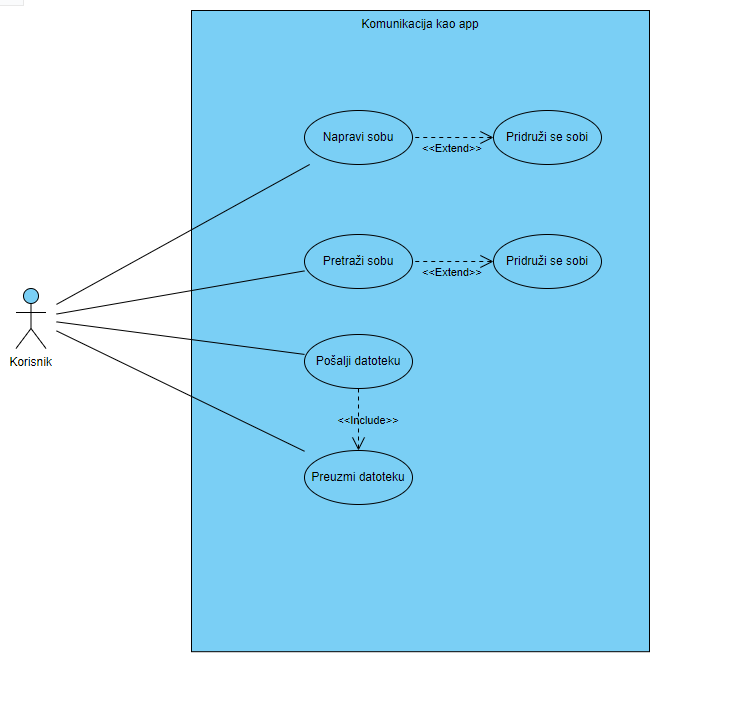
Microsoft SignalR je biblioteka otvorenog koda koja pojednostavljuje dodavanje web funkcionalnosti u realnom vremenu velikom broju aplikacija. Omogućava kodu na strani servera da trenutno prosljeđuje sadržaj klijentima, podržavajući tako scenarije kao što su obavještenja u realnom vremenu i instant poruke, baš kao što WebRTC zahtijeva. Međutim, ono što SignalR prevazilazi slične tehnologije je njegova jednostavnost, skalabilnost i sigurnost. SignalR je vrlo jednostavan za implementaciju, što će biti prikazano u primjeru kasnije. Štaviše, njegova skalabilnost je uglavnom izgrađena na činjenici da u svojoj bazi ima .NET/.NET Core, koji u kombinaciji sa uslugom Microsoft Azure SignalR može podržati čak i najzahtjevnija okruženja. Konačno, njegova sigurnost je također zasnovana na .NET praksi, tako da je sve što je podržano u standardnoj .NET/.NET Core aplikaciji također dostupno u već pripremljenom SignalR-u.

Za implementaciju kompletnog WebRTC rješenja korištenjem navedenih tehnologija potrebno je implementirati dva glavna dijela – serversku aplikaciju za signalizaciju i aplikaciju web klijenta. Iako je izgradnja STUN/TURN servera spomenuta u prošlom članku, za ovaj primjer ćemo koristiti besplatne javne STUN servere od Googlea:

* stun:stun.1.google.com:19302
* stun:stun1.l.google.com:19302

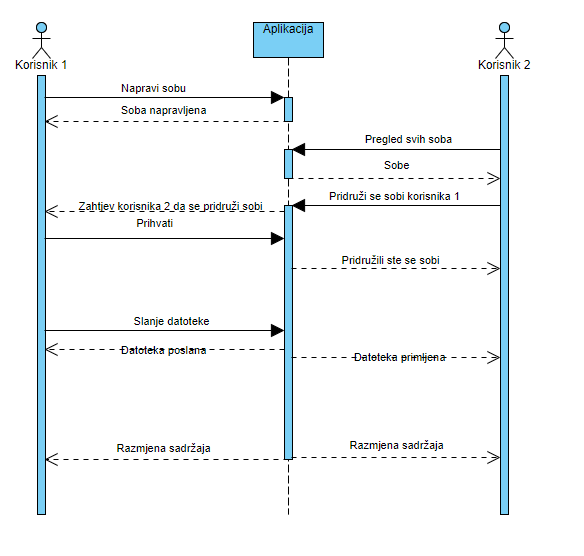
Samo rješenje će biti izgrađeno što jednostavnije, ali sa implementiranim i funkcionalnim svim funkcionalnostima. Ideja je da se pokaže kako koristiti te tehnologije za izgradnju sigurnog i funkcionalnog WebRTC rješenja bez previše upuštanja u napredne WebRTC funkcije. Svi dijelovi koda dati u nastavku, kao i drugi dijelovi koji će biti referencirani, ali neće biti prikazani, mogu se naći u Github repozitorijumu ovog projekta.

**UML**



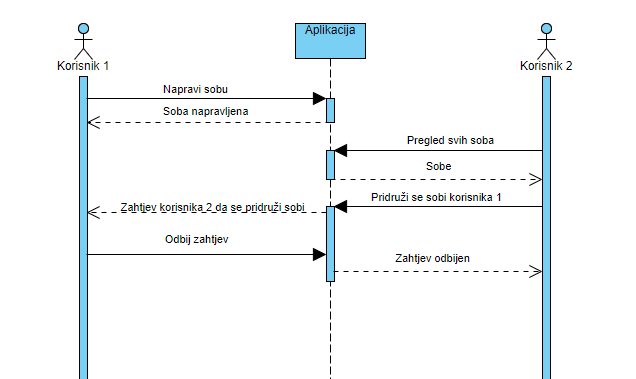
Slika . Use Case dijagram

Na slici 5. prikazan je use case dijagram za aplikaciju. Korisnik prilikom dolaska na Home page staranice ima mogućnost da napravi sobu, pretraži sobi i istoj se pridruži. Također, korisnik može da pošalje datoteku i preuzme istu. Prilikom pridruživanja sobi, šalje se zahtjev za poziv koji korisnik može da prihvati ili odbije.



Slika . Sekvencijalni dijagram prihvatanja poziva i komunikacije

Na slici 6. je prikazan sekvencijalni dijagram za uspostavljenu komunikaciju. Kada korisnik napravi sobu, drugi korisnici će je moći vidjeti, te poslati zahtjev za pridruženje istoj. Zatim kosinik koji je napravio sobu prihvata zahtjev te se komunikacija uspostavlja. U toku trajanja komunikacije, korisnici mogu da razmjenjuju datoteke. Kada se datoteka uploaduje, drugi korisnik ima mogućnost istu da preuzme.



Slika . Sekvencijalni dijagram odbijana poziva

Na slici 7. je prikazan slučaj kada se poziv odbije. Korisniku kojem je odbijen poziv će biti ispisana poruka da je njegov zahtjev odbijen od korisnika koji je napravio sobu.

#### Programski kod.

Struktura aplikacije je prikazana na sljedećem stablu:

|  |
| --- |
| 📦Komunikacija\_kao\_app  ┣ 📂Documentation  ┃ ┣ 📜Dokumentacija.docx  ┃ ┗ 📜~$kumentacija.docx  ┣ 📂Hubs  ┃ ┗ 📜WebRTCHub.cs  ┣ 📂Installations  ┃ ┣ 📜aspnetcore-runtime-2.2.8-win-x64.exe  ┃ ┗ 📜dotnet-runtime-2.2.8-win-x64.exe  ┣ 📂Pages  ┃ ┣ 📂Shared  ┃ ┃ ┣ 📜\_CookieConsentPartial.cshtml  ┃ ┃ ┣ 📜\_Layout.cshtml  ┃ ┃ ┗ 📜\_ValidationScriptsPartial.cshtml  ┃ ┣ 📜About.cshtml  ┃ ┣ 📜About.cshtml.cs  ┃ ┣ 📜Contact.cshtml  ┃ ┣ 📜Contact.cshtml.cs  ┃ ┣ 📜Error.cshtml  ┃ ┣ 📜Error.cshtml.cs  ┃ ┣ 📜Index.cshtml  ┃ ┣ 📜Index.cshtml.cs  ┃ ┣ 📜Privacy.cshtml  ┃ ┣ 📜Privacy.cshtml.cs  ┃ ┣ 📜\_ViewImports.cshtml  ┃ ┗ 📜\_ViewStart.cshtml  ┣ 📂Properties  ┃ ┗ 📜launchSettings.json  ┣ 📂wwwroot  ┃ ┣ 📂css  ┃ ┃ ┣ 📜site.css  ┃ ┃ ┗ 📜site.min.css  ┃ ┣ 📂images  ┃ ┃ ┣ 📜banner1.svg  ┃ ┃ ┣ 📜banner2.svg  ┃ ┃ ┗ 📜banner3.svg  ┃ ┣ 📂js  ┃ ┃ ┣ 📜site.js  ┃ ┃ ┣ 📜site.min.js  ┃ ┃ ┗ 📜webrtc.js  ┃ ┣ 📂lib  ┃ ┃ ┣ 📂bootstrap  ┃ ┃ ┃ ┣ 📂dist  ┃ ┃ ┃ ┃ ┣ 📂css  ┃ ┃ ┃ ┃ ┃ ┣ 📜bootstrap-theme.css  ┃ ┃ ┃ ┃ ┃ ┣ 📜bootstrap-theme.css.map  ┃ ┃ ┃ ┃ ┃ ┣ 📜bootstrap-theme.min.css  ┃ ┃ ┃ ┃ ┃ ┣ 📜bootstrap-theme.min.css.map  ┃ ┃ ┃ ┃ ┃ ┣ 📜bootstrap.css  ┃ ┃ ┃ ┃ ┃ ┣ 📜bootstrap.css.map  ┃ ┃ ┃ ┃ ┃ ┣ 📜bootstrap.min.css  ┃ ┃ ┃ ┃ ┃ ┗ 📜bootstrap.min.css.map  ┃ ┃ ┃ ┃ ┣ 📂fonts  ┃ ┃ ┃ ┃ ┃ ┣ 📜glyphicons-halflings-regular.eot  ┃ ┃ ┃ ┃ ┃ ┣ 📜glyphicons-halflings-regular.svg  ┃ ┃ ┃ ┃ ┃ ┣ 📜glyphicons-halflings-regular.ttf  ┃ ┃ ┃ ┃ ┃ ┣ 📜glyphicons-halflings-regular.woff  ┃ ┃ ┃ ┃ ┃ ┗ 📜glyphicons-halflings-regular.woff2  ┃ ┃ ┃ ┃ ┗ 📂js  ┃ ┃ ┃ ┃ ┃ ┣ 📜bootstrap.js  ┃ ┃ ┃ ┃ ┃ ┣ 📜bootstrap.min.js  ┃ ┃ ┃ ┃ ┃ ┗ 📜npm.js  ┃ ┃ ┃ ┣ 📜.bower.json  ┃ ┃ ┃ ┗ 📜LICENSE  ┃ ┃ ┣ 📂jquery  ┃ ┃ ┃ ┣ 📂dist  ┃ ┃ ┃ ┃ ┣ 📜jquery.js  ┃ ┃ ┃ ┃ ┣ 📜jquery.min.js  ┃ ┃ ┃ ┃ ┗ 📜jquery.min.map  ┃ ┃ ┃ ┣ 📜.bower.json  ┃ ┃ ┃ ┗ 📜LICENSE.txt  ┃ ┃ ┣ 📂jquery-validation  ┃ ┃ ┃ ┣ 📂dist  ┃ ┃ ┃ ┃ ┣ 📜additional-methods.js  ┃ ┃ ┃ ┃ ┣ 📜additional-methods.min.js  ┃ ┃ ┃ ┃ ┣ 📜jquery.validate.js  ┃ ┃ ┃ ┃ ┗ 📜jquery.validate.min.js  ┃ ┃ ┃ ┣ 📜.bower.json  ┃ ┃ ┃ ┗ 📜LICENSE.md  ┃ ┃ ┣ 📂jquery-validation-unobtrusive  ┃ ┃ ┃ ┣ 📜.bower.json  ┃ ┃ ┃ ┣ 📜jquery.validate.unobtrusive.js  ┃ ┃ ┃ ┣ 📜jquery.validate.unobtrusive.min.js  ┃ ┃ ┃ ┗ 📜LICENSE.txt  ┃ ┃ ┗ 📂signalr  ┃ ┃ ┃ ┗ 📂dist  ┃ ┃ ┃ ┃ ┗ 📂browser  ┃ ┃ ┃ ┃ ┃ ┣ 📜signalr.js  ┃ ┃ ┃ ┃ ┃ ┗ 📜signalr.min.js  ┃ ┗ 📜favicon.ico  ┣ 📜.gitattributes  ┣ 📜.gitignore  ┣ 📜appsettings.Development.json  ┣ 📜appsettings.json  ┣ 📜Komunikacija\_kao\_app.csproj  ┣ 📜Komunikacija\_kao\_app.csproj.user  ┣ 📜Komunikacija\_kao\_app.sln  ┣ 📜libman.json  ┣ 📜Program.cs  ┣ 📜README.md  ┗ 📜Startup.cs |

U folderu dokumentacija, nalazi se trenutni dokument koji sadrži dokumentaciju projekta. Folder Installations sadrži instalaciju za .NET Core 2.2 framework. Program počinje sa izvršavanjem u Program.cs klasi, koja zatim poziva Startup.cs klasu u kojoj se nalaze postavke za SignalR:

|  |
| --- |
| services.AddSignalR();  ...  app.UseSignalR(routes =>              {                  routes.MapHub<Hubs.WebRTCHub>("/WebRTCHub");              }); |

Ove postavke govore ASP.NET frameworku da će aplikacija koristiti SignalR server koji je implementiran u WebRTCHub.cs klasi koja se nalazi unutar Hubs foldera.

|  |
| --- |
| public class WebRTCHub : Hub      {          private static RoomManager roomManager = new RoomManager();          public override Task OnConnectedAsync()          {              return base.OnConnectedAsync();          }          public override Task OnDisconnectedAsync(Exception exception)          {              return base.OnDisconnectedAsync(exception);          }          public async Task CreateRoom(string name)          {              RoomInfo roomInfo = roomManager.CreateRoom(Context.ConnectionId, name);              if (roomInfo != null)              {                  await Groups.AddToGroupAsync(Context.ConnectionId, roomInfo.RoomId);                  await Clients.Caller.SendAsync("created", roomInfo.RoomId);              }              else              {                  await Clients.Caller.SendAsync("error", "Greška prilikom pravljenja sobe.");              }          }          public async Task Join(string roomId)          {              await Groups.AddToGroupAsync(Context.ConnectionId, roomId);              await Clients.Caller.SendAsync("joined", roomId);              await Clients.Group(roomId).SendAsync("ready");              if (int.TryParse(roomId, out int id))              {                  roomManager.DeleteRoom(id);              }          }          public async Task LeaveRoom(string roomId)          {              await Clients.Group(roomId).SendAsync("bye");          }          public async Task GetRoomInfo()          {              await NotifyRoomInfoAsync(true);          }          public async Task SendMessage(string roomId, object message)          {              await Clients.OthersInGroup(roomId).SendAsync("message", message);          }          public async Task NotifyRoomInfoAsync(bool notifyOnlyCaller)          {              List<RoomInfo> roomInfos = roomManager.GetAllRoomInfo();              var list = from room in roomInfos                         select new                         {                             RoomId = room.RoomId,                             Name = room.Name,                             Button = "<button class=\"joinButton\">Pridruži se!</button>"                         };              var data = JsonConvert.SerializeObject(list);              if (notifyOnlyCaller)              {                  await Clients.Caller.SendAsync("updateRoom", data);              }              else              {                  await Clients.All.SendAsync("updateRoom", data);              }          }      } |

Klasa WebRTCHub.cs sadrži metode za slanje i primanje poruka prema klijent aplikaciji. U prethodnoj tabeli prikazana je samo implementacija klase bez pomoćne klase RoomManager radi obimnosti. Metode OnConnectedAsync i OnDisconnectedAsync su metode iz bazne klase Hub koje je bilo potrebno preopteretiti. Metoda CreateRoom služi da se napravi soba za video chat. Ona svim klijentima šalje poruku da je soba kreirana, te ukoliko neki od klijenata pošalje poruku Join, ista se šalje onome koji je napravio sobu te se uspostavlja WebRTC komunikacija. Na isti princip radi i LeaveRoom metoda. SendMessage metoda služi za slanje datoteka, dok NotifyRoomInfoAsync klijentima šalje podatke o dostupnim sobama.

U folderu wwwroot se nalaze HTML, CSS i JavaScript datoteke. Glavna HTML datoteka je Index.cshtml:

|  |
| --- |
| @page  @{      ViewData["Title"] = "Početna";  }  @section StyleSheets{      <link rel="stylesheet" href="https://cdn.datatables.net/1.10.20/css/jquery.dataTables.min.css" type="text/css" />      <link rel="stylesheet" href="~/css/site.css" type="text/css" />  }  <div class="mt-3" style="margin: 100px">      <div class="roomCreate">          <label for="name">Ime sobe:</label>          <input type="text" id="roomNameTxt" />          <button id="createRoomBtn">Napravi sobu</button>      </div>      <div>          <table id="roomTable" class="display" style="width:100%;">              <thead>                  <tr>                      <th>ID Sobe</th>                      <th>Ime</th>                      <th></th>                  </tr>              </thead>              <tbody>              </tbody>          </table>      </div>      <div class="borderLine"></div>      <div>          <div class="connectionStatus">              <p id="connectionStatusMessage">\*Napravite sobu ili se pridružite nekoj.</p>          </div>          <h5>Video chat</h5>          <div class="videoArea">              <video id="localVideo" autoplay playsinline></video>              <video id="remoteVideo" autoplay playsinline></video>          </div>          <h5>Dijeljenje datoteka</h5>          <div id="fileShare">              <input type="file" id="fileInput">              <button id="sendFileBtn" class="sendFileBtn">Pošalji</button>          </div>          <table id="fileTable" class="fileTable">              <thead>                  <tr>                      <th>Podijeljene datoteke</th>                  </tr>              </thead>              <tbody>              </tbody>          </table>      </div>  </div>  @section Scripts{      <script type="text/javascript" src="~/lib/signalr/dist/browser/signalr.js"></script>      <script type="text/javascript" src="https://webrtc.github.io/adapter/adapter-latest.js"></script>      <script type="text/javascript" src="https://cdn.datatables.net/1.10.20/js/jquery.dataTables.min.js"></script>      <script type="text/javascript" src="~/js/webrtc.js"></script>  } |

Ova datoteka sadrži skeleton same aplikacije, te importuje signalr.js datoteku:

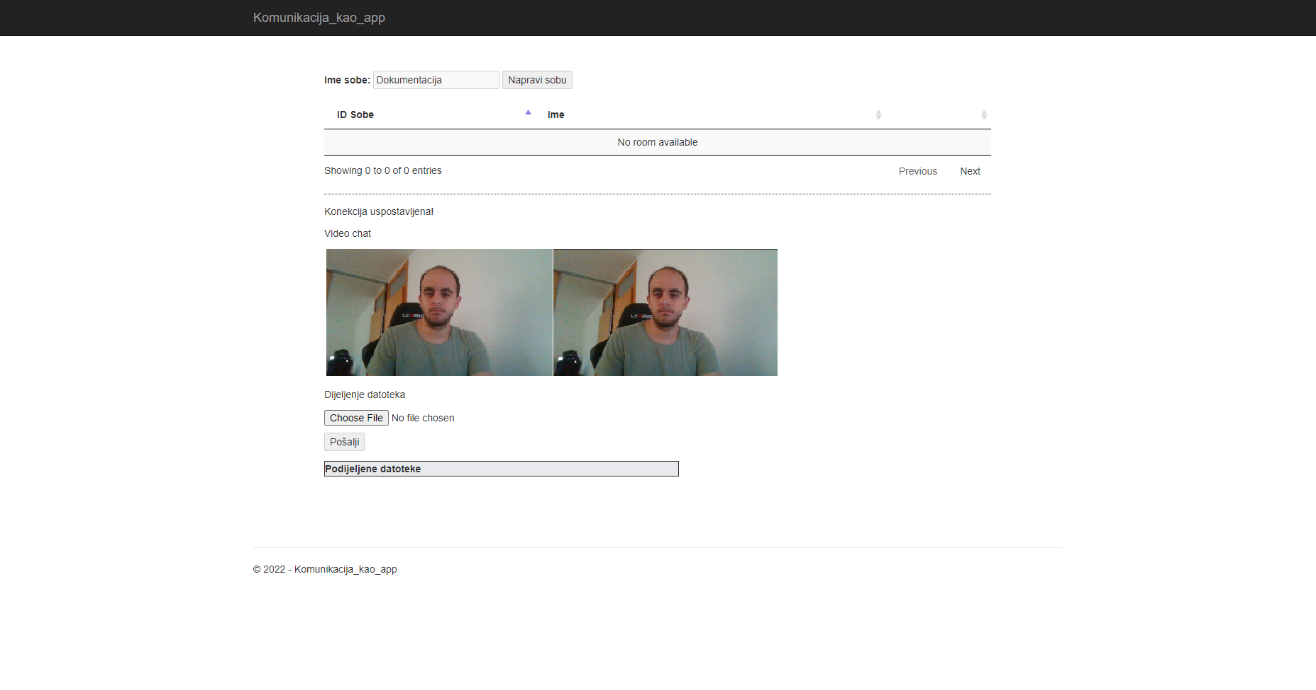
|  |
| --- |
| var connection = new signalR.HubConnectionBuilder().withUrl("/WebRTCHub").build();  // Postavke  const configuration = {      'iceServers': [{          'urls': 'stun:stun.l.google.com:19302'      }]  };  const peerConn = new RTCPeerConnection(configuration);  ...  // WebRTC postavke  function grabWebCamVideo() {      navigator.mediaDevices.getUserMedia({          audio: true,          video: true      })          .then(gotStream)          .catch(function (e) {              alert('getUserMedia() error: ' + e.name);          });  } |

U prethodnoj tabeli prikazane su glavne funkcije za konfiguraciju. Najprije se uspostavlja komunikacija sa backendom, to jeste, sa SirnalR-om. Zatim se uspostavlja konfiguriše RTCPeerConnection koristeći google STUN server. Zatim se dohvata korisnikov zvuk i video pomoću getUserMedia metode.

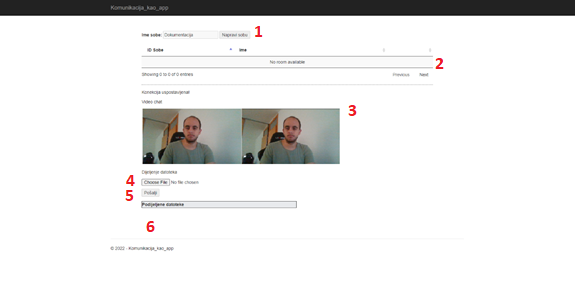
|  |
| --- |
| function createPeerConnection(isInitiator, config) {      peerConn.onicecandidate = function (event) {          console.log('icecandidate event:', event);          if (!event.candidate) {              sendMessage(peerConn.localDescription);          }      };      peerConn.ontrack = function (event) {          remoteVideo.srcObject = event.streams[0];      };      if (isInitiator) {          dataChannel = peerConn.createDataChannel('sendDataChannel');          onDataChannelCreated(dataChannel);          peerConn.createOffer(onLocalSessionCreated, logError);      } else {          peerConn.ondatachannel = function (event) {              dataChannel = event.channel;              onDataChannelCreated(dataChannel);          };      }  } |

Prethodna funkcija, createPeerConnection, se poziva kada se uspostavi veza između dva korisnika. Koristi se peerConn objekat tipa RTCPeerConnection da se inicira početak komunikacije.

Ostale metode se nalaze na GitHub repozitoriju koji sadrži mnogo funkcija, dok su u ovom dokumentu opisane one najbitnije, to jeste, opisani su najbitniji koraci.

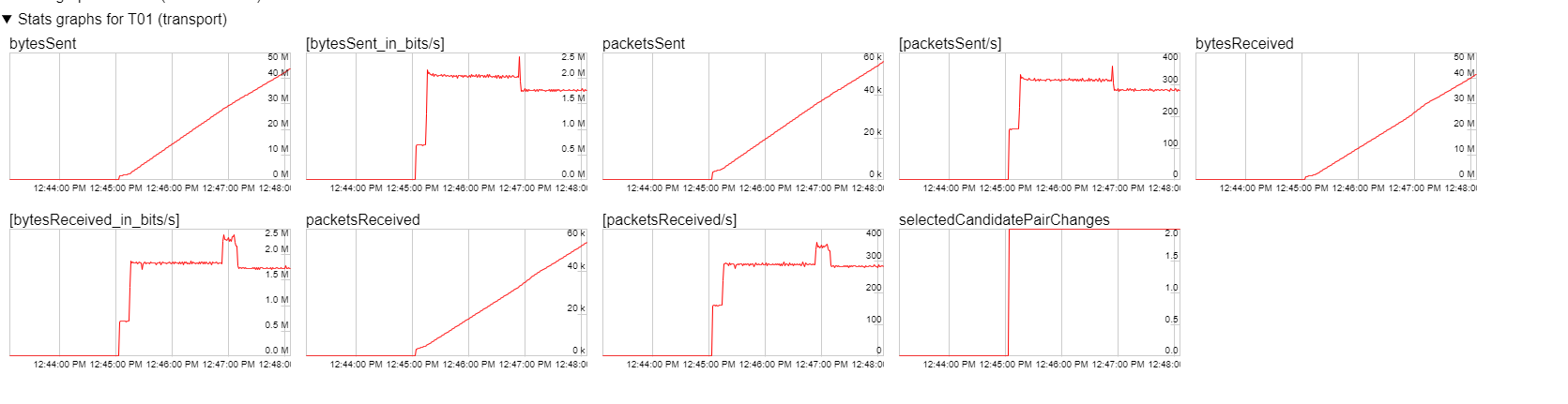


Slika 8. Izgled aplikacije



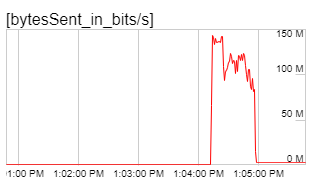
Slika . Upute za korištenje

1. Pravljenje sobe – unošenje imena u input polje, te klik na dugme “Kreiraj sobu” će napraviti sobu s unesenim imenom.
2. Pridruži se sobi – klikom na dugme “Pridruži se”, šalje se zahtjev za pridruženje i uspostavu poziva.
3. Uspostava poziva
4. Upload datoteke – odabir datoteke kroz file system
5. Upload datoteke – klikom na dugme “Pošalji” datoteka se šalje drugom korisniku
6. Download datoteke – datoteka se downloaduje od strane drugog korisnika

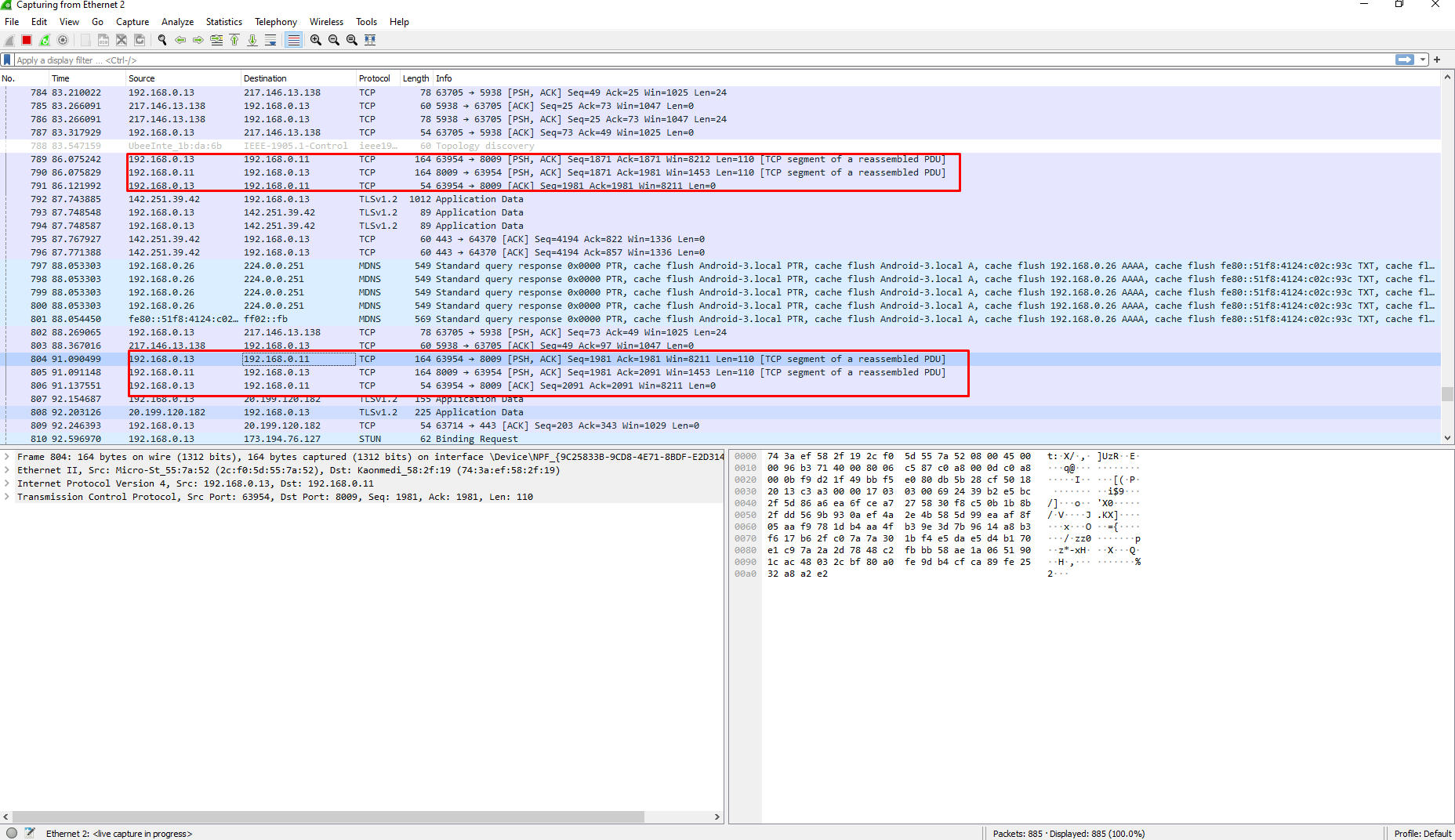


Slika 10. WebRTC analizator, transport

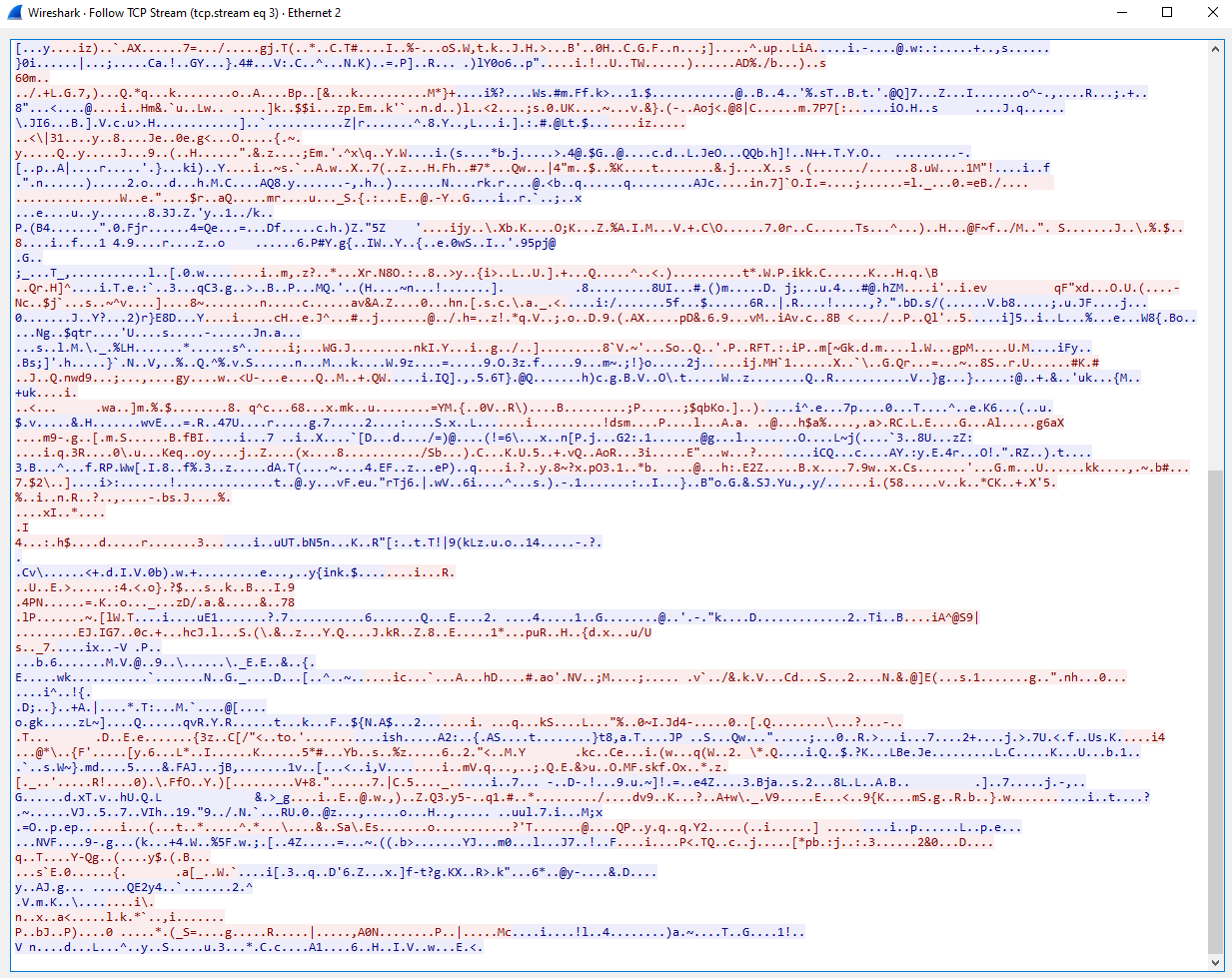
U 12:45 dolazi do uspostave konekcije između dva klijenta. Primjetno je da aktivnost na svim dijagramima skače sa nule, na neku drugu vrijedost. Eksponencijalno raste broj poslanih i primljenih bajta i paketa. U zavisnosti od komunikacije, primjetno je u sekciji bytesSent\_in\_bits/s da se kriva mijenja u zavisnosti od toka komunikacije.



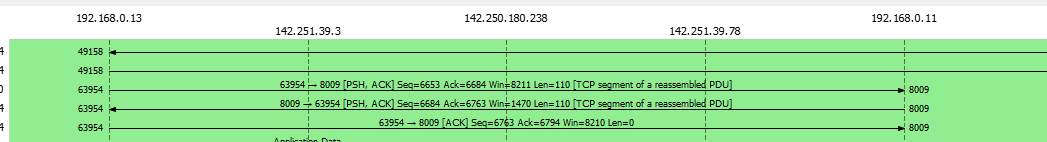
Slika 11. Slanje video sadržaja



Slika 12. WireShark analiza razmjene paketa



Slika 13. Praćenje stream-a



Slika 14. Flow Graph

Na slikama 8, 9 i 10 nalazi se analiza sa programa WireShark. Na slici 8. se analiziraju primljeni i poslani paketi između dva klijenta. WireShark nudi opciju da se podaci analiziraju u same detalje. Moguće je pratiti stream između dva klijenta i tačno analizirati podatke koji se šalju, te je moguće vidjeti dijagram toka između dva klijenta.

Online WebRTC analizator je samo grafički prikaz razmjene podataka, dok WireShark nudi enterprise analizu. WebRTC analizator je dobar za programere koji ne trebaju ulaziti u srž, te koji imaju potrebu samo grafički da analiziraju tok slanja i primanja podataka.

**Reference**

1 .https://ekobit.com/blog/building-a-modern-and-secure-webrtc-solution-with-microsoft-signalr-and-angular/

2 .<https://webrtc.org/>

3. https://www.w3.org/TR/webrtc/

**Zaključak**

Imajući u vidu sve do sada rečeno, možemo zaključiti da WebRTC definitivno nije slučajno lider u komunikacijskim tehnologijama u realnom vremenu. Glavne prednosti WebRTC-a u odnosu na slične tehnologije istog tipa su njegova jednostavnost implementacije, unaprijed definirana sigurnost na više slojeva, mogućnost implementacije cross-platform rješenja povezivanja i web pretraživača i mobilnih operativnih sistema i na kraju, ali ne i najmanje važno, jednostavnost upotrebe od strane krajnjih korisnika. Bilo da se radi o jednostavnom web chatu između dva klijenta na istoj platformi ili o kompliciranom komunikacijskom rješenju u realnom vremenu između više klijenata na različitim platformama, WebRTC je s razlogom prvi izbor velikog broja IT stručnjaka i bit će vrlo zanimljivo vidjeti kako će dalji razvoj ove tehnologije poboljšava razumijevanje komunikacije u realnom vremenu kakvu danas poznajemo.