Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kierunek: **Informatyka techniczna**
Specjalność: **Grafika i Systemy Multimedialne**

PRACA DYPLOMOWA INŻYNIERSKA

Wideokomunikator internetowy implementujący nowoczesne kodeki wideo w języku Rust

Internet videochat application implementing modern video codecs using Rust programming language

Marcel Guzik

Opiekun pracy
dr inż. Marek Woda

WROCŁAW, 2022

Opracował: Tomasz Kubik <tomasz.kubik@pwr.edu.pl>
Data: grudzień 2021

Streszczenie

Streszczenie w języku polskim powininno zmieścić się na połowie strony (drugą połowę powinien zająć abstract w języku angielskim).

Lorem ipsum dolor sit amet eleifend et, congue arcu. Morbi tellus sit amet, massa. Vivamus est id risus. Sed sit amet, libero. Aenean ac ipsum. Mauris vel lectus.

Nam id nulla a adipiscing tortor, dictum ut, lobortis urna. Donec non dui. Cras tempus orci ipsum, molestie quis, lacinia varius nunc, rhoncus purus, consectetuer congue risus.

Słowa kluczowe: raz, dwa, trzy, cztery

Abstract

Streszczenie in Polish should fit on the half of the page (the other half should be covered by the abstract in English).

Lorem ipsum dolor sit amet eleifend et, congue arcu. Morbi tellus sit amet, massa. Vivamus est id risus. Sed sit amet, libero. Aenean ac ipsum. Mauris vel lectus.

Nam id nulla a adipiscing tortor, dictum ut, lobortis urna. Donec non dui. Cras tempus orci ipsum, molestie quis, lacinia varius nunc, rhoncus purus, consectetuer congue risus.

Keywords: one, two, three, four

Spis treści

Spis rysunków

Spis tabel

Spis listingów

Skróty

OGC (ang. *Open Geospatial Consortium*)
XML (ang. *eXtensible Markup Language*)
SOAP (ang. *Simple Object Access Protocol*)
WSDL (ang. *Web Services Description Language*)
UDDI (ang. *Universal Description Discovery and Integration*)
GIS (ang. *Geographical Information System*)
SDI (ang. *Spatial Data Infrastructure*)
ISO (ang. *International Standards Organization*)
WMS (ang. *Web Map Service*)
WFS (ang. *Web Feature Service*)
WPS (ang. *Web Processing Service*)
GML (ang. *Geography Markup Language*)
SRG (ang. *Seeded Region Growing*)
SOA (ang. *Service Oriented Architecture*)
IT (ang. *Information Technology*)

Rozdział 1

Wstęp

1.1. Wprowadzenie

Pandemia COVID-19 oraz zdobywające coraz większą popularność zdalne formy zatrudnienia obrazują jak ważna jest rola połączeń wideo we współczesnym społeczeństwie. W wielu przypadkach kontakt "twarzą w twarz"jest preferowalny, a nawet niezbędny do realizacji pewnych zadań. W takich wypadkach niezawodność oraz efektywność transmisji wideo stają się bardzo ważnymi problemami.

Celem niniejszej pracy jest analiza połączeń wideo czasu rzeczywistego w każdym ich etapie, badanie procesów składających się na nie, i wreszcie utworzenie internetowego komunikatora wideo wykorzystującego poznane koncepty i rozwiązania. Tworzony komunikator będzie aplikacją okienkową na systemy Linux, będzie nawiązywał połączenia wideo peer-to-peer, a także będzie wykorzystywał kodek AV1 do kompresji wideo.

W rozdziale 2 omówione zostaną koncepty i metody związane ze strumieniowaniem wideo. Przedstawiony zostanie uproszczony opis procesu transmisji wideo w czasie rzeczywistym, od pobrania klatki obrazu przez kamerę internetową, do wyświetlenia tejże klatki na monitorze rozmówcy. Poruszone zostaną problemy związane z transmisją wideo przez sieć internetową, np. problem ustanowienia kanału P2P pomiędzy klientami za siecią NAT, negocjowanie sesji pomiędzy klientami, adaptive bitrate, etc.

W rozdziale 3 omówiony zostanie projekt WebRTC i jego protokoły składowe, czytelnik dowie się również w jaki sposób WebRTC rozwiązuje problemy poruszone we wcześniejszym rozdziale i tym samym niezwykle upraszcza tworzenie multimedialnych aplikacji webowych.

W rozdziale 4 zaprezentowana zostanie aplikacja webowa wykorzystująca WebRTC, nastąpi ogólny przegląd wykorzystywanych technologii oraz oprogramowania, zaprezentowane zostaną fragmenty kodu źródłowego realizujące kluczowe procesy nawiązywania połączenia poruszone we wcześniejszym rozdziale. Wykonana zostanie dokładna analiza ruchu sieciowego pomiędzy hostami i uzupełniony zostanie proces nawiązywania połączenia WebRTC poruszony we wcześniejszym rozdziale, zobrazowany konkretnym przykładem.

W rozdziale 5 omówione zostaną problemy, generalne mechanizmy i koncepty związane z kompresją wideo, kontenery, strumienie oraz ich muxowanie do kontenerów, rodzaje kodeków wideo, proces enkodowania/dekodowania, sprzętowe oraz programowe implementacje koderów.

W rozdziale 6 poruszone zostaną problemy i rozwiązania związane ze strumieniowaniem AV1. Zrealizowane zostanie "zejście na niższy poziom", co pozwoli na głębsze zapoznanie się z krokami realizowanymi celem przygotowania strumienia wideo do transmisji oraz wyświetlenia przychodzącego wideo na ekranie monitora. Zrealizowane zostanie min.: zastąpienie całości lub części stosu WebRTC rozwiązaniami natywnymi, oferowanymi przez sprzęt lub system operacyjny, wyeliminowanie abstrakcji oferowanych przez przeglądarkę, wybór optymalnego kodera

oraz jego parametrów, wybór technologii GUI, zaplanowanie i omówienie procesu strumieniowania z rozdziału 1szego, wraz z elementami które będą go realizowały.

W rozdziale 7 zaprezentowana zostanie wykonana aplikacja desktopowa wykorzystująca biblioteki oraz inne rozwiązania udostępniane przez system operacyjny. Omówienie architektury, wykorzystywanych technologii i fragmentów kodu aplikacji okienkowej. Zaprezentowanie wybranych etapów procesu strumieniowania realizowanych przez aplikację.

Rozdział 2

Internetowe strumienie wideo

Jakieś wprowadzenie o wideo ogólnie, w jakich kontekstach występuje (pliki wideo, streaming np. Youtube/Netflix, strumienie czasu rzeczywistego np. Twitch/Discord/Teams), czym te konteksty się różnią.

Możemy wyróżnić 3 rodzaje wideo:

- lokalne pliki wideo - pliki wideo na dysku w kontenerze mp4, mkv, lub innym
- streamowanie plików wideo (np. Youtube albo Netflix) - mamy przygotowane segmenty pliku wideo w różnych rozdzielczościach i wysyłamy je po kolei, dobieramy rozdzielczość wg. dostępnego pasma pomiędzy serwerem a klientem
- strumieniowanie w czasie rzeczywistym - priorytetem jest opóźnienie, kodujemy na bieżąco przechwytywane klatki i wysyłamy je najszybciej jak się da; przez to niektóre mechanizmy kompresji są niedostępne (np. B-klatki, które wykorzystują dane z następnej klatki aby zmniejszyć wielkość klatki)

Zidealizowany obraz rozmowy wideo w internecie:

1. Komputery są publicznymi hostami w internecie i program komunikatora słucha na danym porcie
2. Strona nawiązująca połączenie łączy się do hosta odbiorcy po tym porcie, sygnalizuje chęć nawiązania połączenia
3. Strona odbierająca akceptuje
4. Kamera oraz mikrofon nadawcy przechwytują najlepszy możliwy obraz i dzwięk, i przesyłają je do komputera
5. Strumienie wideo i audio są łączone i synchronizowane
6. Komputer wysyła strumień audio-wideo wcześniej ustanowionym kanałem

Natomiast pojawiają się problemy:

1. Komputery znajdują się w sieciach domowych, za NATem, nie można się do nich bezpośrednio połączyć
2. Nieskompresowane wideo jest zbyt duże by wysłać je przez internet, potrzebny jest jakiś mechanizm kompresji
3. Komputery mogą mieć różne możliwości przetwarzania wideo: znajdować się w sieciach o znacząco różnej szybkości, mieć różniące się szybkością procesory, kamery zapisujące klatki w różnych formatach

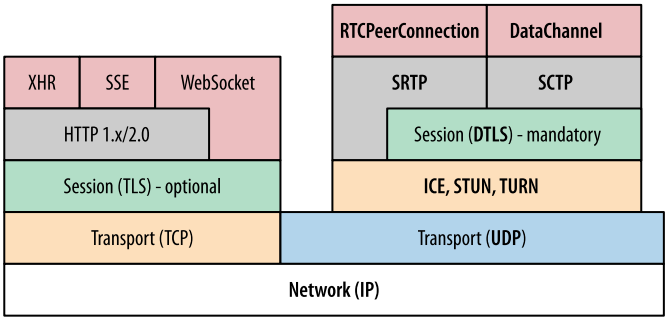
Rozdział 3

WebRTC

3.1. Co to jest, z czego się składa?

[1]

WebRTC jest projektem zapewniającym przeglądarkom możliwości komunikacji czasu rzeczywistego, dzięki czemu możliwe jest budowanie komunikatorów internetowych w całości w obrębie zapewnianego przez przeglądarkę javascriptowego API. Aby funkcjonalność ta była możliwa do zrealizowania, niezbędne było użycie wielu protokołów sieciowych, które pokazano na rysunku 3.1.



Rys. 3.1: Stos protokołów w WebRTC - wzięty z HPBN - do zmiany

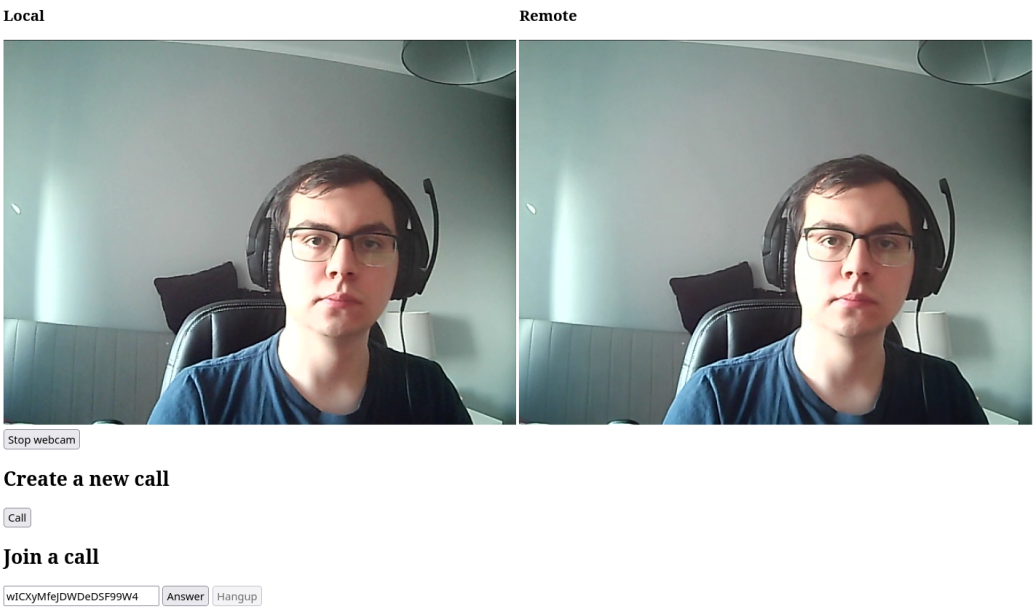
3.2. Sygnalizacja nawiązania połączenia

3.3. Interactive Connectivity Establishment (ICE)

3.4. Opis API WebRTC

Rozdział 4

Aplikacja webowa z użyciem WebRTC



Rys. 4.1: Zrzut ekranu aplikacji WebRTC podczas połączenia na tym samym komputerze

4.1. Architektura, budowa aplikacji, omówienie kodu

Aplikacja używa WebRTC do nawiązywania połączenia, Firebase do sygnalizowania out of band.

Aby nawiązać połączenie, trzeba wykonać następujące kroki:

1. Użytkownicy 1 i 2 wciskają przycisk **Start webcam** aby udostępnić obraz z kamery aplikacji.
2. Użytkownik 1 wciska przycisk **Call**, tworząc nowe połączenie.
3. Użytkownik 1 odczytuje ID połączenia które pojawiło się w polu tekstowym i przekazuje je użytkownikowi 2.
4. Użytkownik 2 wprowadza ID połączenia uzyskane od użytkownika 1 w to samo pole tekstowe i wciska przycisk **Connect**.

4.1.1. Przechwytywanie wideo i audio

Listing 4.1: Przechwytywanie wideo i audio z komputera

```
let localStream: MediaStream;
const webcamButton = document.getElementById('webcamButton');

webcamButton?.addEventListener('click', async () => {
    if (localVideo.srcObject) {
        localVideo.srcObject = null;
        return;
    }

    localStream = await navigator.mediaDevices.getUserMedia({ video: true,
    ↪ audio: true, });
    webcamButton.innerHTML = 'Stop webcam';

    remoteStream = new MediaStream();
    remoteVideo.srcObject = remoteStream;


    localStream.getTracks().forEach((track) => { peerConnection.addTrack(
    ↪ track, localStream); });
    peerConnection.addEventListener('track', event => {
        event.streams[0].getTracks().forEach(track => {
            remoteStream.addTrack(track);
        });
    });

    localVideo.srcObject = localStream;
    remoteVideo.srcObject = remoteStream;

    callButton.disabled = false;
    answerButton.disabled = false;
});
```

4.1.2. Tworzenie połączenia

Aby utworzyć połączenie WebRTC, musimy najpierw skomunikować się z drugim klientem przez jakiś inny kanał, aka. out-of-band, wykorzystamy do tego celu bazę danych czasu rzeczywistego Firebase. Przygotujemy zatem uchwyt do bazy danych:

Listing 4.2: Inicjalizacja Firebase

```
import { initializeApp } from "firebase/app";
import { getFirestore, collection, addDoc, getDoc, doc, setDoc, onSnapshot,
    ↪  updateDoc } from "firebase/firestore";


// Your web app's Firebase configuration
const firebaseConfig = {
    apiKey: "AIzaSyCr12-OQV5bgdQPFoexd44O9Ubmht966pw",
    authDomain: "piperchat-2eacd.firebaseapp.com",
    projectId: "piperchat-2eacd",
    storageBucket: "piperchat-2eacd.appspot.com",
    messagingSenderId: "172730710087",
    appId: "1:172730710087:web:3dabdb9a62bee44e095962"
};

// Initialize Firebase
```

```
const app = initializeApp(firebaseConfig);
const db = getFirestore(app);
```

Następnie, do przycisku **Call** tworzącego połączenie podpinamy handler:

Listing 4.3: Tworzenie połączenia

```
callButton?.addEventListener('click', async () => {
  const callDoc = doc(collection(db, "calls"));
  const offerCandidates = collection(callDoc, "offerCandidates");
  const answerCandidates = collection(callDoc, "answerCandidates");

  callInput.value = callDoc.id;

  peerConnection.onicecandidate = (event) => {
    if (event.candidate) {
      addDoc(offerCandidates, event.candidate.toJSON());
    }
  }

  const offerDescription = await peerConnection.createOffer();
  await peerConnection.setLocalDescription(offerDescription);

  const offer = {
    sdp: offerDescription.sdp,
    type: offerDescription.type
  };

  await setDoc(callDoc, { offer });

  onSnapshot(callDoc, (snapshot) => {
    const data = snapshot.data();
    if (!peerConnection.currentRemoteDescription && data?.answer) {
      const answerDescription = new RTCSessionDescription(data.answer);
      peerConnection.setRemoteDescription(answerDescription);
    }
  });

  onSnapshot(answerCandidates, (snapshot) => {
    snapshot.docChanges().forEach((change) => {
      if (change.type === "added") {
        const candidate = new RTCIceCandidate(change.doc.data());
        peerConnection.addIceCandidate(candidate);
      }
    })
  })
});
```

4.1.3. Dołączanie do połączenia

Listing 4.4: Dołączanie do połączenia połączenia

```
answerButton?.addEventListener("click", async () => {
  const callId = callInput.value;
  const callDoc = doc(db, "calls", callId);
  const answerCandidates = collection(callDoc, "answerCandidates");
  const offerCandidates = collection(callDoc, "offerCandidates");

  peerConnection.onicecandidate = (event) => {
    if (event.candidate) {
```

```
      addDoc(answerCandidates, event.candidate.toJSON());
    }
  }

  const callData = (await getDoc(callDoc)).data();
  if (!callData) {
    console.error("Call document no longer exists");
    return;
  }
  const offerDescription = callData.offer;
  await peerConnection.setRemoteDescription(new RTCSessionDescription(
    ↪ offerDescription));

  const answerDescription = await peerConnection.createAnswer();
  await peerConnection.setLocalDescription(answerDescription);

  const answer = {
    type: answerDescription.type,
    sdp: answerDescription.sdp,
  };

  await updateDoc(callDoc, { answer });

  onSnapshot(offerCandidates, (snapshot) => {
    snapshot.docChanges().forEach((change) => {
      if (change.type === "added") {
        const data = change.doc.data();
        peerConnection.addIceCandidate(new RTCIceCandidate(data));
      }
    })
  });
});
```

4.2. Omówienie działania aplikacji na przykładzie

4.2.1. Śledzenie procesu nawiązywania połączenia

W poniższym rozdziale zostaną omówione dane wymieniane pomiędzy stronami na rzecz ustanowienia połączenia WebRTC.

Aby ustanowić połączenie peer-to-peer, należy rozwiązać poniższe problemy: [1]

- Należy powiadomić drugą stronę że chcemy ustanowić do niej połączenie, aby wiedziała ona żeby rozpocząć nasłuchiwanie.
- Należy zidentyfikować ścieżki trasowania dla połączenia peer-to-peer i uzgodnić jedną pomiędzy obiema stronami.
- Należy wymienić niezbędne informacje o używanych przez peerów parametrach połączenia - jakich protokołów, kodeków, ustawień, etc. użyć.

Sygnalizowanie negocjacji sesji

W aplikacji webowej, użytkownik 1 tworzący nowe połączenie tworzy dokument w kolekcji calls, a następnie nasłuchuje na zmiany tego dokumentu. ID rozmowy to ID dokumentu utworzonego w bazie Firestore. Po tym kroku, w bazie pojawia się następujący dokument:

Listing 4.5: Dokument połączenia po utworzeniu przez użytkownika 1

```
{
```

```
    id: "YHARFdJoA4lAd8nRu3Uw",
    offerCandidates: [],
    answerCandidates: []
}
```

4.2.2. Analiza pakietów protokołu SDP

Rozdział 5

Mechanizmy kompresji wideo

5.1. Koncepty kompresji

5.2. Kontenery

5.3. Kodeki

5.3.1. H.264

5.3.2. H.265

5.3.3. VP9

5.3.4. AV1

Rozdział 6

Architektura projektu Piperchat

6.1. Serwer

6.2. Klient

Rozdział 7

Aplikacja desktopowa w języku Rust

7.1. GUI framework

7.2. ffmpeg

Bibliografia

[1] I. Grigorik, “High performance browser networking”, en, w Sebastopol, CA: O’Reilly Media, wrz. 2013, rozd. WebRTC.

Rozdział 8

Instrukcja wdrożeniowa

Jeśli praca skończyła się wykonaniem jakiegoś oprogramowania, to w dodatku powinna pojawić się instrukcja wdrożeniowa (o tym jak skompilować/zainstalować to oprogramowanie). Przydałoby się również krótkie how to (jak uruchomić system i coś w nim zrobić – zademonstrowane na jakimś najprostszym przypadku użycia). Można z tego zrobić osobny dodatek,

Rozdział 9

Opis załączonej płyty CD/DVD

Tutaj jest miejsce na zamieszczenie opisu zawartości załączonej płyty. Należy wymienić, co zawiera.