

Bridging WebRTC and SIP using GStreamer & SIPjs

Sanchayan Maity

- ▶ Open source consulting firm based out of Bangalore and Toronto.
- ▶ Building high-quality, low-level systems software.
- ▶ Most of our work revolves around audio/video using GStreamer and PipeWire.

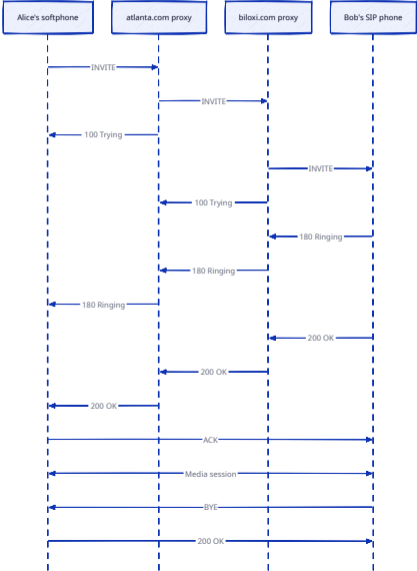
- ▶ `getUserMedia` - gets access to data streams such as the user's camera and phone
- ▶ `RTCPeerConnection` - enables audio or video calling with facilities for encryption and bandwidth management
 - ▶ `RTCDataChannel` - enables peer-to-peer communication of generic data
- ▶ Works in conjunction with other standards involved.
 - ▶ Session Description Protocol (SDP)
 - ▶ Interactive Connectivity Establishment (ICE)
 - ▶ Session Traversal Utilities for NAT (STUN)/Traversal Using Relays around NAT (TURN)
 - ▶ Real-time Transport Protocol (RTP)

- ▶ RFC 3261
- ▶ Signalling Protocol
- ▶ Managing session (initiate, maintain, stop)
- ▶ Works in conjunction with other standards involved.
 - ▶ Session Description Protocol (SDP)
 - ▶ Interactive Connectivity Establishment (ICE)
 - ▶ Session Traversal Utilities for NAT (STUN)/Traversal Using Relays around NAT (TURN)
 - ▶ Real-time Transport Protocol (RTP)
- ▶ Resources on network (user agents, routers) are identified by a Uniform Resource Identifier (URI)
- ▶ Carried by transport layer protocols like TCP or UDP

Why SIP?

- ▶ SIP is how phones connect to the Internet
- ▶ Dial in and dial out
- ▶ Conference room equipment interoperates using SIP

SIP session setup



- ▶ Library for talking to SIP servers from JavaScript
 - ▶ Uses WebSockets for SIP signalling (RFC 7118)
 - ▶ Uses `RTCPeerConnection` for media
 - ▶ Primarily used on browser

```
const transportOptions = {
  server: "wss://atlanta.com:8443"
};
const uri = UserAgent.makeURI("sip:alice@atlanta.com");
const userAgentOptions: UserAgentOptions = {
  authorizationPassword: 'secretPassword',
  authorizationUsername: 'authorizationUsername',
  transportOptions,
  uri
};
const userAgent = new UserAgent(userAgentOptions);
userAgent.start().then(() => {
  registerer.register();
});
```

¹SIPjs guides - user agent

SIPjs - make or receive a call²³

```
// userAgent defined elsewhere
userAgent.start().then(() => {
  const target = UserAgent.makeURI("sip:bob@biloxi.com");

  const inviter = new Inviter(userAgent, target);
  // make a call
  inviter.invite();
});

// receive call
function onInvite(invitation) {
  invitation.accept();
}
```

²SIPjs guides - make call

³SIPjs guides - receive call

```
// Assumes you have a media element on the DOM
const mediaElement = document.getElementById('mediaElement');

const remoteStream = new MediaStream(); // getUserMedia
function setupRemoteMedia(session: Session) {
  session.sessionDescriptionHandler.peerConnection.getReceivers().
    forEach((receiver) => {
      if (receiver.track) {
        remoteStream.addTrack(receiver.track);
      }
    });
  mediaElement.srcObject = remoteStream;
  mediaElement.play();
}
```

⁴SIPjs guides - attach media

```
export interface SessionDescriptionHandler {
  getDescription(
    options?: SessionDescriptionHandlerOptions,
    modifiers?: Array<SessionDescriptionHandlerModifier>
  ): Promise<BodyAndContentType>;

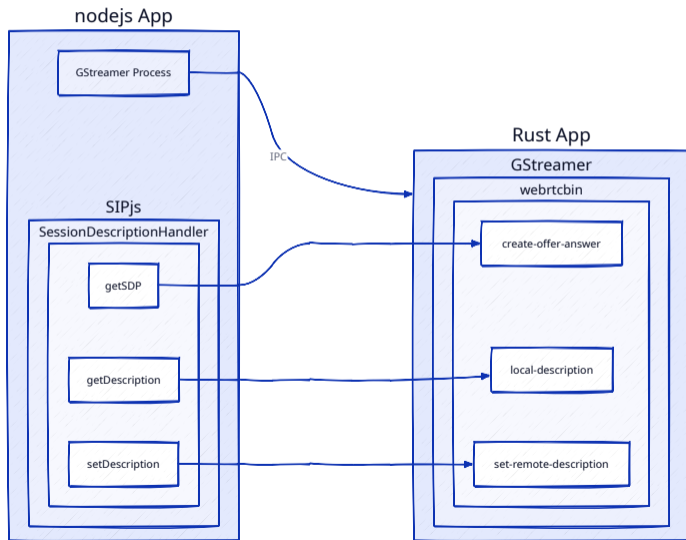
  hasDescription(contentType: string): boolean;

  setDescription(
    sdp: string,
    options?: SessionDescriptionHandlerOptions,
    modifiers?: Array<SessionDescriptionHandlerModifier>
  ): Promise<void>;
}
```

```
export class SessionDescriptionHandler
  implements SessionDescriptionHandlerDefinition {
  protected _localMediaStream: MediaStream;
  protected _remoteMediaStream: MediaStream;
  protected _dataChannel: RTCDataChannel;
  protected _peerConnection: RTCPeerConnection;

  private iceGatheringCompletePromise: Promise<void>;
  private iceGatheringCompleteTimeoutId: number;
  private iceGatheringCompleteResolve: ResolveFunction;
  private iceGatheringCompleteReject: RejectFunction;
  private localMediaStreamConstraints: MediaStreamConstraints;
  private onDataChannel: ((dataChannel: RTCDataChannel) => void);
}
```

- ▶ SIPjs provides hooks to override `RTCPeerConnection` usage
 - ▶ GStreamer has a `webrtc` implementation with an API similar to `RTCPeerConnection`
 - ▶ Use this to call out to a GStreamer-based implementation leveraging `webrtcbin`
 - ▶ that's SIPjs (Node) -> GStreamer app (Rust)



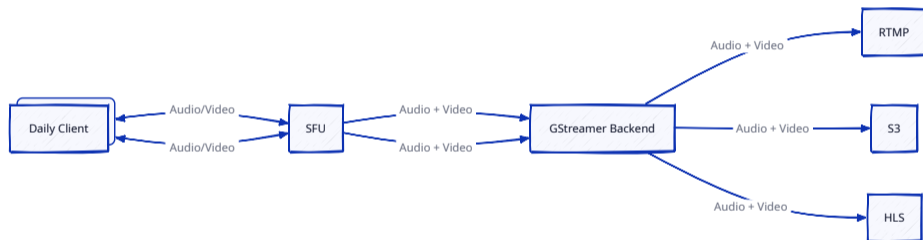
```
class GstSessionDescriptionHandler implements SessionDescriptionHandler {
  gstProcess: ChildProcess;

  private async sendGst(msg: JsMessage): Promise<GstMessageSdp> {
    this.gstProcess.send(msg);
    return new Promise((resolve, reject) => {
      this.gstProcess.stdout.on('data', (message: string) => {
        let msg: GstMessage = JSON.parse(message);
        resolve(msg);
      });
    });
  }

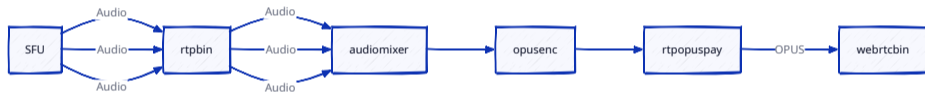
  hasDescription(contentType: string): boolean {
    return contentType === 'application/sdp';
  }
}
```

```
async getDescription(): Promise<BodyAndContentType> {
  let msg =
    await this.sendGst({ code: JsMessageCode.GET_LOCAL_DESCRIPTION });
  return Promise.resolve({
    body: msg.sdpBody,
    contentType: 'application/sdp',
  });
}

async setDescription(
  sdp: string,
): Promise<void> {
  await this.sendGst({ code: JsMessageCode.SET_REMOTE_DESCRIPTION,
    sdpBody: sdp });
}
```

⁵GStreamer for your back-end services

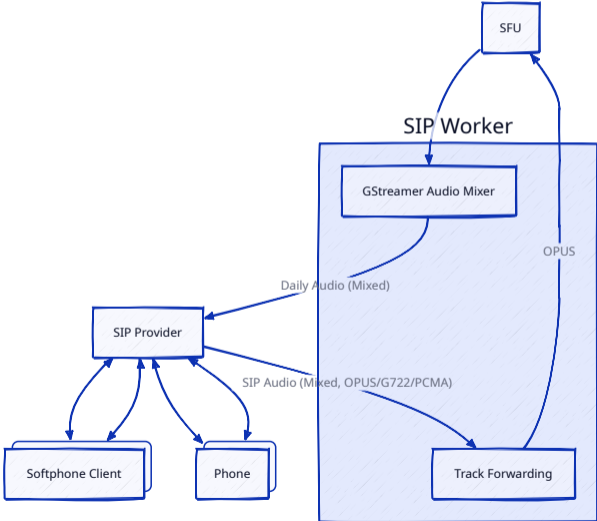


SIP -> WebRTC

asymptotic



SIP Architecture



- ▶ Refactoring as bins
- ▶ NodeJS & Rust interop alternative approaches
 - ▶ Library style wrapper/FFI
 - ▶ Develop `RTCPeerConnection` like API
- ▶ Minor `webrtcbin` changes
 - ▶ Handling of non bundle media with `webrtcbin`
 - ▶ No default direction specified in SDP
- ▶ Media/Codec incompatibilities
- ▶ Latency

- ▶ Upstream webrtcbin patches
- ▶ Open source webrtcbin and SIPjs demo
- ▶ Implement matrix/audio mixing ourselves
- ▶ Add video support

Questions?

asympt^{otic}

Thank You.