Implementação e avaliação de cenário de convergência telefonia-rede integrando serviços de VoIP e video-chamada com o uso de WebRTC

Rafael Ossamu Togo

Orientador: Prof. Arliones Hoeller

Julho 2014

Sumário

Abstract

O WebRTC *Web Real Time Communication* permite a transmissão de áudio vídeo e compartilhamento de dados entre usuários que utilizem um simples navegador, de maneira nativa. Como um conjunto de normas, WebRTC proporciona a qualquer navegador a capacidade de compartilhar dados e realizar teleconferência peer to peer, sem a necessidade de instalar plugins ou software de terceiros. WebRTC não vem adicionar novos serviços à rede, mas sim tornar a sua utilização mais fácil

Este documento aborda as tecnologias necessárias para o desenvolvimento de uma aplicação, que tem como objetivo realizar a integração de serviços de VoIP, com o uso do WebRTC. Apontar as vantagens e desvantagens frente as atuais tecnologias, e avalia o desempenho do sistema utilizando esta tecnologia.

1 Introdução

Entendemos que a Internet revolucionou a forma como as pessoas vivem e se relacionam, porém a maneira em que hoje a enxergamos a Web (World Wide Web), demandou um certo tempo de amadurecimento, no que diz respeito a organização, estruturação e padronização.

Em 12 de março de 1989, Tim Berners-Lee divulgou no interior da Organização Europeia para a Pesquisa Nuclear (CERN) um projeto baseado no conceito de hipertexto para facilitar o compartilhamento e a atualização de informações apenas entre os pesquisadores da própria organização. Deu-se então o surgimento do que viria a se tornar a World Wide Web (WWW).

Com a Web em processo de evolução, em 1994 surge a W3C World Wide Web Consortium que torna-se o órgão oficial para a gestão da Internet e padrões da web como HTML (HyperText Markup Language) e CSS (Cascading Style Sheets), em todo o mundo, tornando a página web melhor estruturada e organizada.

Com o passar do tempo, diversas diretrizes, recomendações e normas da W3C foram surgindo, tornando uma página Web um ambiente maduro para desenvolvimento, permitindo a diversos desenvolvedores criarem suas próprias aplicações.

Diretamente no ano de 2012, com a ajuda da W3C e da The Internet Engineering Task Force (IETF) foi possível criar uma tecnologia que permite comunicação em áudio e vídeo diretamente entre web browsers, resultando no chamado *Web Real Time Communication*(WebRTC). Esta tecnologia permite que haja comunicação em tempo real, com voz, vídeos e dados, utilizando apenas navegadores web, não havendo necessidade de instalação de plugins ou softwares, como é o caso do Skype, eBuddy e Numbuzz, nem da intermediação de servidores para troca de dados.

Em uma época onde temos cada vez mais serviços rodando na nuvem, essa tecnologia tem tudo para se tornar a base de soluções de comunicação em tempo real, fazendo com que muitas empresas hoje consolidadas busquem pelo mesmo tipo de solução para não perderem espaço no mercado.

1.1 Motivação

O fato do WebRTC permitir estabelecer comunicação por som e imagem (video-conferência), de browser para browser, sem necessidade de nenhum outro plugin ou software, já o torna atraente para as futuras aplicações. Ou seja, a tendência para as próximas aplicações que utilizarem esta tecnologia, é tornar a comunicações entre os usuário, cada vez mais prática e eficaz. É por esse tipo de solução que os usuários tem buscado fortemente.

Este trabalho também busca dar continuidade a um TCC desenvolvido anteriormente no IFSC-SJ por Felipe Borges intitulado *Webrtc: Estudo e análise do projeto* [1].

1.2 Objetivos

O objetivo deste trabalho é implementar e avaliar um cenário típico de convergência telefonia-rede integrando serviços de VoIP e video-chamadas, utilizados a partir de navegadores Web com suporte ao WebRTC. Neste cenário será desenvolvido uma aplicação onde um usuário encontrará outros que estarão disponíveis (online) em sua lista de contatos, para então iniciar uma chamada.

1.2.1 Objetivos Específicos

- Implementar test-bed de infraestrutura de controle de VoIP e Videoconferencia com Asterisk.
- Desenvolver/buscar um mecanismo de divulgação de usuários disponiveis.
- Realizar testes utilizando a rede do Campos IFSC/SJ.
- Criar uma interface amigável e intuitiva aos usuários, para lidar com a aplicação desenvolvida.
- Avaliar o desempenho do sistema.

1.3 Metodologia

Através das linguagens HTML, CSS e Javascript, é necessário criar uma interface amigável e intuitiva aos usuários, para que estes possam interagir entre sí, sem grandes dificuldades. Essa interação ocorrerá de três maneiras: chamadas de áudio, video e texto.

Após a criação da aplicação, haverá a integração do sistema com o Asterisk, pois toda a sinalização será através de SIP. É preciso criar um método que irá coletar todos os usuários que estão registrados, e mostrar quem está *online*.

Será necessário o uso de máquinas virtuais para simulação dos usuário, do servidor Asterisk e outra para a própria aplicação. Serão testados os protocolos ICE, STUN, TURN, protocolos que auxiliarão vários clientes trocarem informação mídia através de NAT.

Com o cenário já montado, serão realizados testes para avaliar o desempenho e qualidade de serviço implementado.

2 *Fundamentação Teórica*

Este capítulo apresenta a fundamentação teórica do trabalho, inclui o estudo e descrição das tecnologias que serão importantes para o desenvolvimento do projeto.

2.1 WebRTC

O WebRTC é um projeto de código aberto que permite aplicações de comunicação em tempo real, com voz, vídeos e dados, utilizando apenas navegadores web, não havendo necessidade de instalação de plugins ou softwares.

Oferece aos desenvolvedores meios para desenvolver aplicações em tempo real de maneira prática e auxilia na construção de uma plataforma RTC *Real-Time Communications* que funciona em diversos navegadores através de múltiplas plataformas.

O papel fundamental do WebRTC é disponibilizar funções e recursos para que o desenvolvedor web possa implementar uma aplicação que funcione como um telefone ou uma unidade de vídeoconferência, pois o WebRTC nada mais é que um *média engine*[1] que faz interações com API's Javascript e com o próprio sistema operacional do usuário, para ter acesso aos recursos de câmera e microfone.

Outras funções que são de responsabilidade de um navegador com suporte a WebRTC, é a codificação/decodificação das mídias que serão utilizadas, bem como funções de processamento de mídia como cancelamento de eco entre outras.

2.2 Arquitetura do WebRTC

A arquitetura do WebRTC é constituida por duas camadas. A primeira é denominada Web API, onde é disponibilizada funções para que os desenvolvedores Web possam desenvolver

[1] Responsável por realizar o tratamento da mídia

suas aplicações. A segunda é o WebRTC C++ API, a qual é utilizada para o desenvolvimento de browsers. A Figura 2.1 demonstra a arquitetura do WebRTC, com suas camadas e blocos.

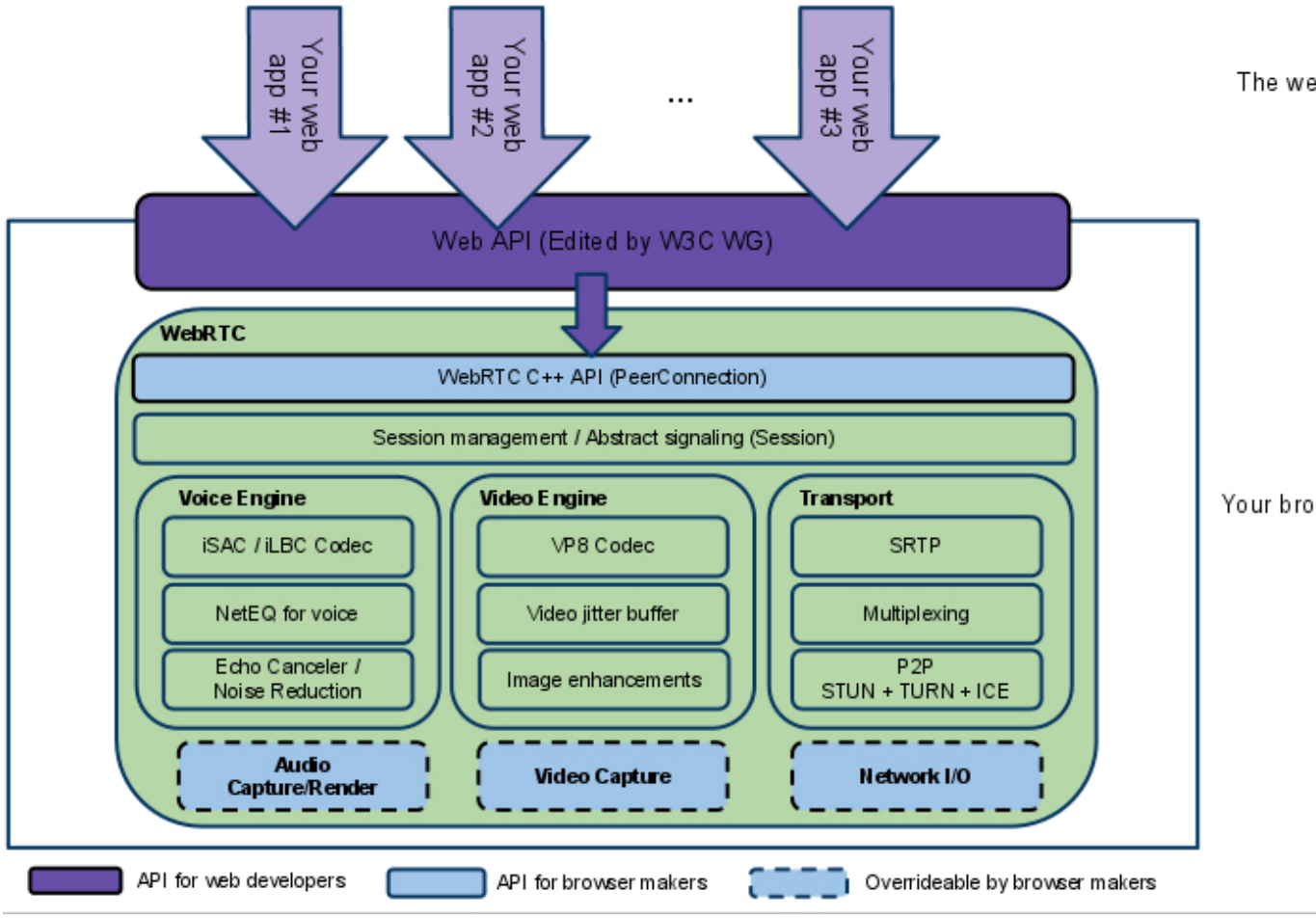

Figura 2.1: Arquitetura do WebRTC

2.2.1 Web API

Utilizando o Javascript, esta API permite o desenvolvimento de aplicações de comunicação em tempo real em web browsers. Podemos dividir em três partes: *MediaStream*, *PeerConnection* e *DataChannels*.

MediaStream

Utilizando a função *getUserMedia()*, esta API é responsável por solicitar acesso ao microfone e câmera do usuário para a geração de uma *stream* de dados. Esta *stream* será enviada utilizando o protocolo de transporte *Secure Real-time Transport Protocol* (SRTP), no qual é umas das exigências no uso do WebRTC. Exemplo de funcionamento na figura 2.2.

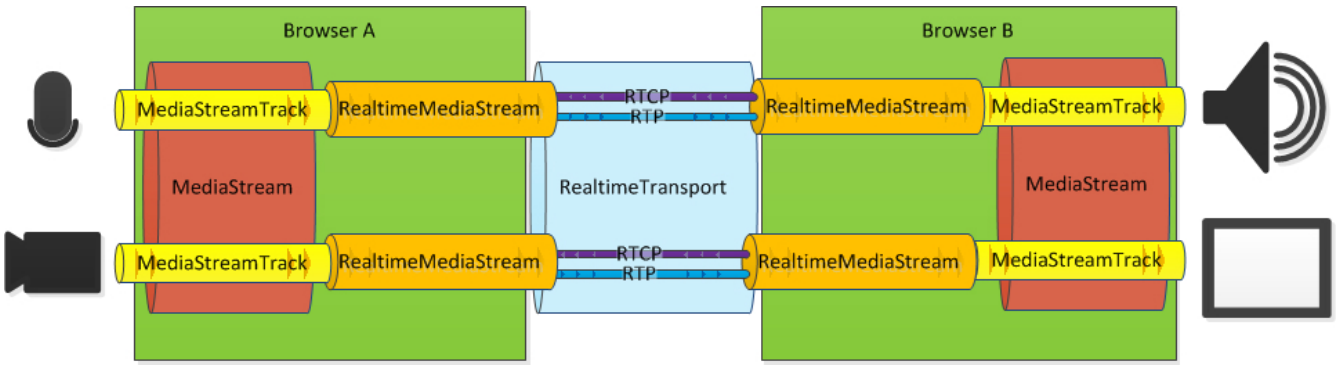

Figura 2.2: Captura e envio da stream para outro usuário

O código abaixo (Figura 2.3), mostra de maneira simples, como é feita a captura da mídia de um usuário. Neste exemplo, é solicitado apenas áudio para ser enviado para outro destino.

```
function gotStream(stream) {
    window.AudioContext = window.AudioContext ||
window.webkitAudioContext;
    var audioContext = new AudioContext();

    // Create an AudioNode from the stream
    var mediaStreamSource =
audioContext.createMediaStreamSource(stream);

    // Connect it to destination to hear yourself
    // or any other node for processing!
    mediaStreamSource.connect(audioContext.destination);
}

navigator.getUserMedia({audio:true}, gotStream);
```

Figura 2.3: Função de captura de mídia

Na última linha, o método *navigator.getUserMedia()*, solicita o acesso ao microfone do usuário. Logo em seguida ele chama a função *gotStream()* para gerar uma stream de dados, que pode ser enviado a algum destino específico.

PeerConnection

Esta API é quem estabelece uma conexão entre dois browsers, faz o controle do codec que será utilizado, criptografia e gerenciamento de banda. Ou seja, protege os desenvolvedores web de diversas complexidades que existem, ao se fazer uma conexão *browser-to-browser* Para estabelecer esta conexão, é necessário um canal para a sinalização. Este é implementando num servidor, utilizando WebSockets ou XML-HttpRequest.

DataChannel

Para a transferência de dados diretamente de um ponto a outro, utilizamos a API DataChannel. Entende-se como dados:

- Transferência de arquivos
- Mensagens instantâneas
- Compartilhamento de tela

Tem como expectativa o desempenho e baixa latência de conexão entre dois clientes. Pode-se utilizar tanto TCP *Transmission Control Protocol*, causando velocidades mais lentas porém garantido, ou UDP *User Datagram Protocol*, para velocidades mais rápidas com possível perda de pacotes.

2.2.2 WebRTC C++ API

Esta API é voltada exclusivamente para o desenvolvimento de browsers, na implementação do WebRTC. A WebAPI é open source e está disponível para qualquer desenvolvedor que queira realizar a integração de seus produtos com os padrões WebRTC. Este trabalho não trabalhará em cima desta API.

2.2.3 Voice Engine

Como foi visto na imagem 2.1, o Voice Engine é um dos mecanismos que formam o WebRTC. É responsável desde a captura do áudio da placa de som, tratamento e envio para a interface de rede. Cancelamento de eco, redução de ruído e uso de codecs específicos.

2.2.4 Video Engine

Responsável pela captura da imagem da *WebCam*, para que esta seja enviada pela *web*. Também é responsável pelo seu tratamento, reduzindo a quantidade de ruído da imagem. Até hoje não foi definido qual codec de vídeo que será utilizado, pois ainda existe uma grande disputa comercial em torno deste assunto. A disputa hoje está entre o VP8 e o H.264.

Em questão de qualidade e consumo de bateria, o H.264 é considerado superior, pois os processadores de vídeo atuais possuem hardwares dedicados para codifica-los, ao contrário do

VP8, onde a codificação é feita através de software, consumindo uma quantidade considerável da bateria. Em favor do VP8 e uma das premissas do WebRTC desde o seu princípio, é que este está livre de *royalties*[1].

2.2.5 Transport

Em relação ao transporte da mídia, o WebRTC faz uso do SRTP, que tem como função criptografar a mídia para que esta não seja criptografada por terceiros.

Para clientes que estão por trás de NAT, há necessidade da utilização dos protocolos ICE, STUN e TURN para auxiliar no envio da mídia.

Interactive Connectivity Establishment(ICE)

O ICE é um protocolo para travessia de NAT para *streams* de mídia, estabelecidas sob o modelo Oferta e Resposta. Surgiu com o intuito de servir com uma solução geral para diversas topologias de rede, resolvendo o problema de desenvolvedores e administradores de rede que precisam fazer suposições e estudo a respeito das topologias onde seus sistemas serão implantados[2].

O modelo de Oferta e Reposta apresenta algumas dificuldades em operar através de NAT, pois para estabelecer um fluxo de mídia entre dois usuários é trocado entre eles mensagens de endereços IP e portas, que não são tratados corretamente pelos dispositivos de NAT.

Para viabilizar o estabelecimento de conexões multimídia entre dois usuários, o protocolo ICE pode tirar proveito ou compor as funcionalidades de outros protocolos, neste caso o STUN e TURN.

Em relação ao WebRTC, este protocolo permite a troca de mídia entre dois clientes que estejam por trás de NAT.

Session Traversal Utilities for NAT (STUN)

O STUN, juntamente com o protocolo ICE, apresentam uma solução integral para a questão de travessia de NAT. Ele permite a um agente descobrir qual porta e endereço IP externo está mapeado para seu endereço IP e porta interno, ou seja, permite que a ligação NAT permaneça ativa[2].

Na tecnologia WebRTC, o STUN é utilizado para estebelecer a sessão entre dois clientes,

onde o *web browser* funciona como um cliente STUN e o servidor web como servidor STUN, e são enviados pacotes de testes para mapear as portas e endereços IP por trás de NAT. Exemplo na figura 2.4.

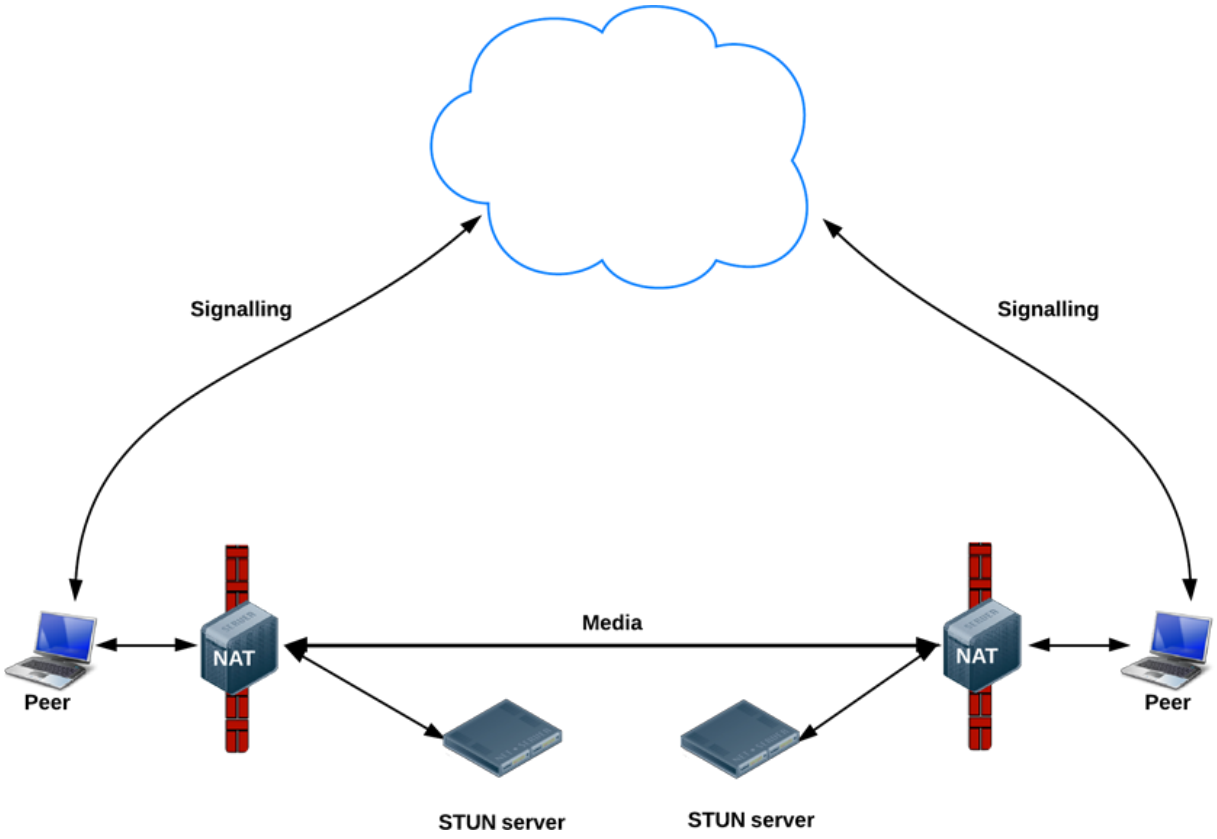

Figura 2.4: Cenário utilizando o protocolo STUN

Traversal Using Relays around NAT (TURN)

Quando não há possibilidade de estabelecer um canal direto entre dois agentes posicionados atrás de um NAT, é necessário recorrer a um host intermediário, que irá agir como um *relay*. Este normalmente se encontra na rede pública e tem a função de retransmitir os pacotes entre os agentes[2]. Exemplo de utilização na figura 2.5.

2.3 Sinalização

Para estabelecer uma chamada entre os usuários do serviço, é necessário um mecanismo de sinalização para negociação de parâmetros da sessão. No WebRTC, a sinalização dos recursos (áudio, vídeo ou dados) fica a critério do próprio desenvolvedor da aplicação. É ele quem define a sinalização mais adequada ao projeto.

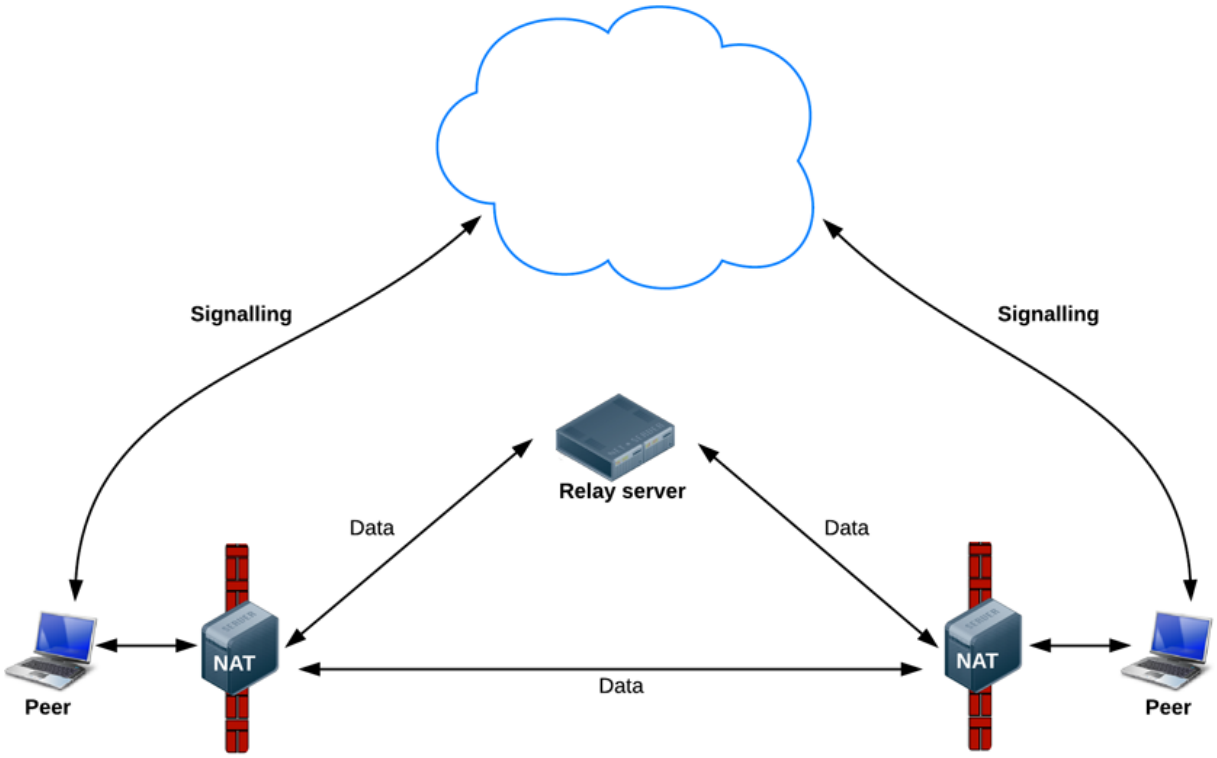

Figura 2.5: Cenário utilizando o protocolo TURN

2.3.1 SIP

O SIP *Session Initiation Protocol* é um protocolo para estabelecimento de chamadas, manutenção e finalização de sessões multimídia. Foi proposto pela IETF (Internet Engineering Task Force), RFC 3261. Sua concepção foi baseada nos protocolos HTTP *Hypertext Transfer Protocol* (modelo de pedido e resposta) e SMTP *Simple Mail Transfer Protocol* (mensagens em texto puro). [3] Habilidades do SIP:

- Localização do usuário
  - Determina o atual dispositivo do usuário
- Disponibilidade do usuário
  - Indica se o usuário está disponível para uma comunicação
- Habilidades do usuário
  - Determina quais os codecs o usuário possui
- Estabelecimento de sessão
  - Determina parâmetros como as portas usadas

- Gerenciamento de sessão
  - Transferência de sessão e modificação dos parâmetros

Uma rede SIP é composta por entidades SIP lógicas, na qual possuem papeis distintos, podendo ser clientes (originam pedidos), servidores(recebem pedidos) ou ambos. As entidades logicas são:

- Agente Usuário (*User Agente - UA*): Trata-se de um ponto ponto final de uma comunicação SIP (Telefones IP, *softphones*, ATAs). São formado por clientes (UAC) que originam pedidos de conexão e servidores (UAS) que recebem e respondem a estes pedidos.
- Servidor Proxy (*Proxy Server*): É a entidade intermediária que atua como cliente e servidor, com o objetivo de originar pedidos em nome de outros clientes. Pode interpretar os pedidos, reescrever e encaminha-los caso haja necessidade.
- Servidor de Redirecionamento (*Redirect Server*) : Aceita pedidos SIP e faz a correspondência do endereço destino com os endereços finais.
- Servidor de Registro (*Registrar Server*): Geralmente usado em conjunto com o *Proxy Server* e *Redirect Server*, possibilita o registro dos usuário e sua localização na rede.

Para iniciar uma sessão SIP, um usuário envia uma mensagem de INVITE com todos os dados necessários para o início de sessão no cabeçalho. Caso o outro usuário aceitar o INVITE, este responderá com uma mensagem de 200 OK. Na figura 2.6 um exemplo de uma chamada SIP.

2.3.2 WebSockets

A web tem sido construída com base no mecanismo de pedido e resposta de HTTP, ou seja, quando um usuário acessa uma página, nada irá acontecer até que ele clique em outra página para atualizar as informações desta. Mais tarde, o AJAX veio para deixar a web mais dinâmica. porém, toda a comunicação HTTP continuava sendo direcionada pelo cliente, o que ainda exigia interação do usuário ou temporizador que atualizasse a página.

A tecnologia que permite o servidor enviar dados atualizados para os clientes, e que foi utilizado por algum tempo, é conhecido como sondagem longa. Um dos principais problemas deste tipo de solução era o overhead de HTTP, que poderia prejudicar soluções que não são tolerantes a atrasos, como jogos on-line. Para solucionar este problema, surgiu então o WebSocket.

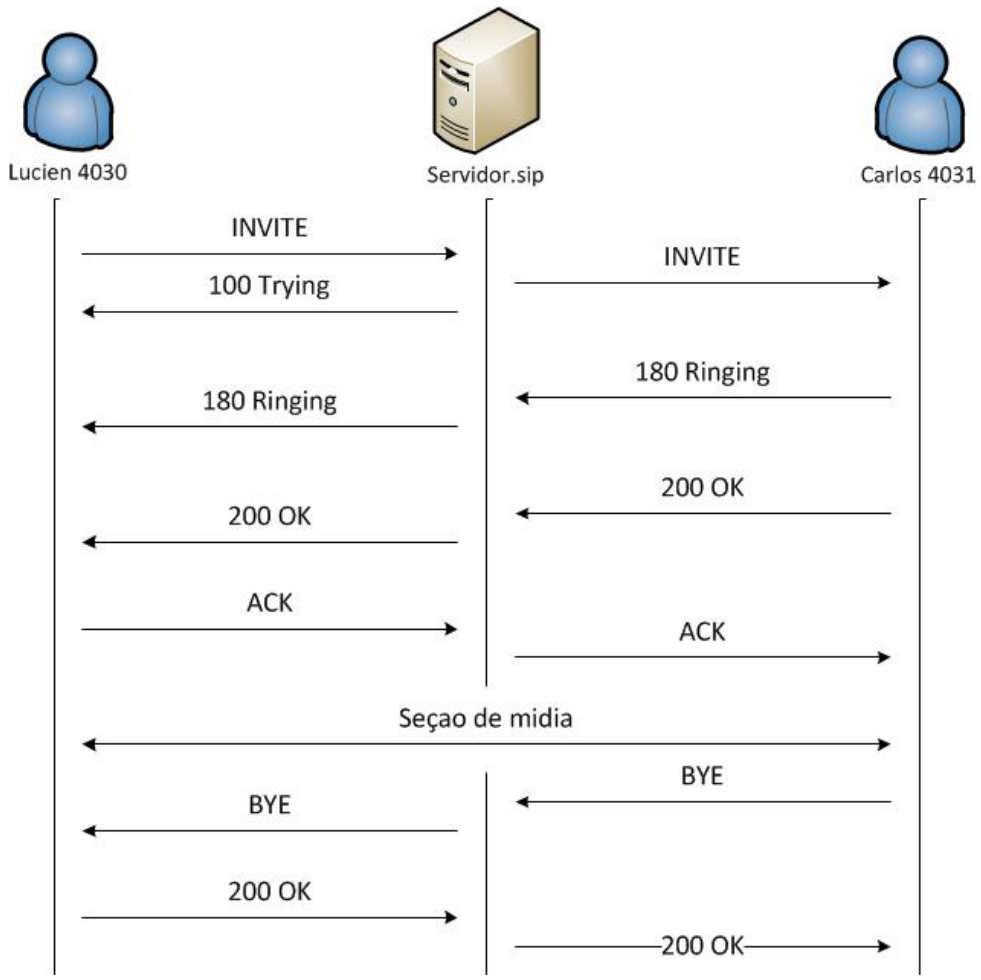

Figura 2.6: Exemplo de chamada SIP

WebSocket é uma tecnologia que permite a comunicação full-duplex sobre um único soquete TCP, entre cliente e servidor web. Ou seja, há uma conexão persistente onde ambas as partes podem começar a enviar dados a qualquer momento

2.3.3 SIP over WebSockets

SIP over Websockets é uma especificação que define um subprotocolo de Websocket, permitindo a troca de mensagens SIP entre um cliente e um servidor web. Assim como o SIP, ela é composta por algumas entidades [4]. As duas principais são:

- *SIP WebSocket Client*: entidade responsável por abrir ligações de *websockets* de saída para o servidor e que se comunica por *WebSocket SIP subprotocol*.
- *SIP WebSocket Server*: responsável por escutar ligações de entrada dos usuários e que se comunica por *WebSocket SIP subprotocol*.

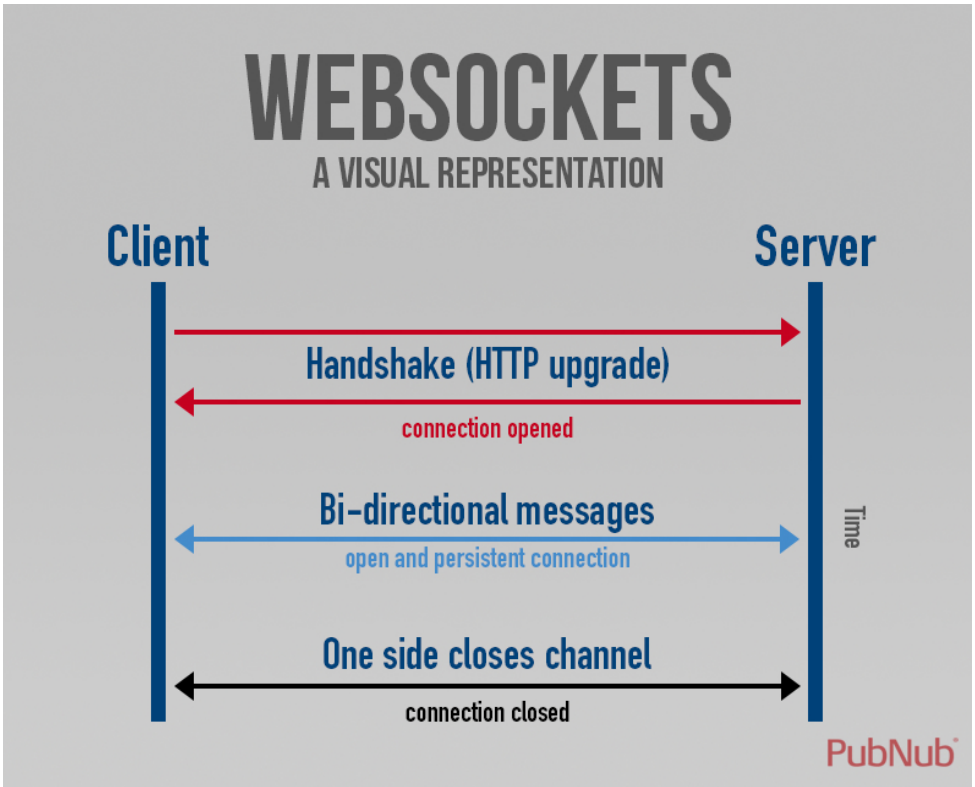

Figura 2.7: WebSockets

2.4 Asterisk

O Asterisk é um software de código aberto, que implementa os recursos encontrados em um PABX convencional, utilizando a tecnologia VoIP. Foi desenvolvido e ainda é mantido pela empresa Digium (surgida em 1999). Atualmente, diversas empresas de pequeno a grande porte tem adotado este tipo de solução, pois permite o desenvolvimento de multiprotocolos a aplicações de comunicações em tempo real com voz e vídeo. Por ser código aberto, o software pode ser modelado de acordo com a necessidade da empresa.

Na versão 11 do Asterisk, foi adicionado o suporte ao WebRTC e criado o módulo res_http_websocket que permite programadores Javascript desenvolverem soluções em que haja interação e comunicação do WebRTC com o Asterisk. Dentro do módulo chan_sip, foram adicionados WebSockets para permitir o uso de SIP como protocolo de sinalização. [4].

2.5 Tecnologia auxiliares

Atualmente já existem algumas soluções WebRTC com funcionalidades diferentes que podem ser usadas e servir de base para outras aplicações. Uma delas é o SIPML5, que permitem

a interoperabilidade entre soluções web e clientes SIP.

2.5.1 sipML5

O sipML5 é uma aplicacao desenvolvida pela Doubango Telecom, que implementa a pilha do protocolo SIP em Javascript [1]. Pode ser usado em qualquer web browser, para uma ligação a uma rede SIP permitindo realizar e receber chamadas vídeo ou voz. Alguma das funções suportadas:

- chamadas vídeo e áudio;
- mensagens instantâneas;
- presença;
- transferência de chamadas;
- chamada em espera;

Seu uso, é recomendável no Google Chrome e Firefox stable. Nos demais navegadores, é necessário instalar a extensão *webrtc4all*. Exemplo na figura 2.8.

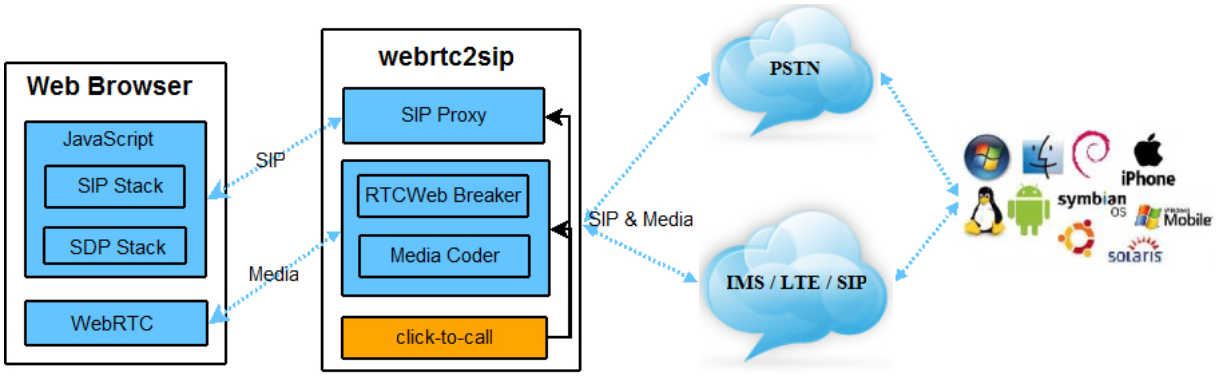

Figura 2.8: Uso do sipML5

É importante ressaltar, que a partir da versão 11, o Asterisk tem suporte nativo ao WebRTC, dispensando o uso de um gateway como é o caso do *WebRTC2SIP* que realiza a conversões dos pacotes enviados pelo endpoint que utiliza webRTC, no caso o SIPML5, para o PABX IP. Na figura 2.9 um pedaço do código responsável pela chamada entre dois usuários.

```
SIPml.init(
    function(e){
        var stack =  new SIPml.Stack({
                        realm: 'example.org',
                        impi: 'bob',
                        impu: 'sip:bob@example.org',
                        password: 'mysecret',
                        events_listener: {
                            events: 'started', listener: function(e){
                                var callSession = stack.newSession('call-audiovideo', {
                                        video_local: document.getElementById('video-local'),
                                        video_remote: document.getElementById('video-remote'),
                                        audio_remote: document.getElementById('audio-remote')
                                    });
                                callSession.call('alice');
                            }
                        }
                    });
        stack.start();
        }
);
```

Figura 2.9: Inicialização da maquina, da pilha SIP e uma chamada de Bob para Alice no sipML5

2.5.2 Node.js

O node.js é uma plataforma orientada a eventos, do lado do servidor, baseadas no V8 Javascript Engine, com o objetivo de criar aplicações de redes rápidas e escaláveis.

Servidores que utilizam o Java ou PHP que possui muitos acessos, necessitam de uma grande quantidade de memória, pois a cada conexão, é alocada cerca de 2MB do sistema. Ou seja, mais conexões, mais memória.

O node resolve esta questão trocando a maneira de como cada conexão é tratada no servidor. Cada conexão cria um processo que não requer que o bloco de memória o acompanhe. Node não permite bloqueios e alega que pode suportar dezenas de milhares de conexões simultânea.

Integrado ao Node.js, o Node Package Manager (npm), implementa um repositório de módulos, onde cada um possui uma funcionalidade particular. Projetos utilizando Node, podem fazer uso desse sistema. Neste TCC, utilizaremos apenas o Node Package Manager em nosso projeto, pois ele disponibiliza a utilização de um módulo essencial para o projeto, o socket.io. Mais adiante, será explicado este módulo.

* V8 Javascript Engine: É o interpretador de Javascript open source implementado pelo Google em C++ e utilizado pelo Google Chrome.

2.5.3 Sails.js

Sails.js é um framework que utiliza o conceito MVC(explicar), com suporte para os requisitos de aplicativos modernos: APIs orientadas a dados com uma arquitetura orientada a serviços escaláveis. Indicados para construção de serviços em tempo real.

O Sails roda em cima do Node.sj, e o utilizaremos neste TCC, pois em seu projeto base, já nos fornece toda uma estrutura de projeto, sem nos preocuparmos em começar do zero. Nos entrega também algumas dependências que será necessário para capturar alguns dados do Asterisk. A principal delas é o Socket.js

2.5.4 Socket.io

O Socket.IO permite realizar comunicação bidirecional utilizando uma das APIs de transporte na seguinte ordem: WebSockets, FlashSockets, AJAX long polling, AJAX multipart streaming, Forever Iframe ou JSONP Polling. O motivo dessa ordem é garantir compatibilidade cross-browser.

Largamente utilizado para geração de relatórios em tempo real, chats, jogos multiplayer online, aplicações de interação em tempo real com diversos usuários

É através do Socket.io que iremos coletar informações do Asterisk, para a atualização das informações em nosso sistema SIPML5.

2.5.5 NodeJS Asterisk Manager

2.5.6 Asterisk Manager API (AMI)

2.6 Linguagens utilizadas no projeto

2.6.1 HTML5

Criada por Tim Barners Lee na década de 1990, e com suas especificações controladas pela W3C *World Wide Web Consortium*, o HTML *HyperText Markup Language*, é uma linguagem de marcação utilizada para produção de páginas na *web*, que permite a criação de documentos que podem ser lidos em praticamente qualquer navegador.

Para escrever documentos HTML é necessário de apenas um editor de texto e conhecimento sobre a linguagem. Esta são compostas por tags servem para indicar a função de cada elemento

da página *web*. Os tags funcionam como comandos de formatação de textos, formulários, links, imagens, tabelas, entre outros.

O HTML versão 5, adiciona várias novas funções sintáticas, incluindo as tags de video e áudio que são indispensáveis para o funcionamento o WebRTC. Na figura 2.10 um simples exemplo utilizando as principais tags do HTML5.

```
<video controls>
  <source src="foo.ogg" type="video/ogg">
  <source src="foo.mp4" type="video/mp4">
  Seu navegador não suporta o elemento <code>video</code>.
</video>
```

```
<audio src="audio.ogg" controls autoplay loop>
<p>Seu navegador não suporta o elemento audio </p>
</audio>
```

Figura 2.10: Utilização de vídeo e áudio em uma página *web*

2.6.2 CSS

O CSS *Cascading Style Sheets* é uma linguagem para estilos que define o layout de documentos HTML, ou seja, ela controla os tipos de fontes, cores, alturas e larguras, posicionamento, entre outros elementos de uma página web. Foram criadas basicamente para deixá-las mais elegantes e atrativas aos usuários.

2.6.3 Javascript

Criada por Brendan Eich e lançado pela primeira vez na versão beta do navegador NetScape 2.0 em 1995, o Javascript é uma linguagem programação interpretada no lado cliente, ou seja, é processada pelo próprio navegador. Com o JavaScript podemos criar efeitos especiais para nossas páginas na Web, além de proporcionar uma maior interatividade com nossos usuários. [5] Hoje o Javascript é considerado a principal linguagem para programação client-side em navegadores web e todos os exemplos de códigos aqui apresentados, o utilizam.

3 *LevantamentoDeRequisitos*

Após algumas pesquisas envolvendo projetos utilizando WebRTC, iniciou-se a fase de desenvolvimento do projeto. Foram definidas os requisitos para que esta fosse implementada de acordo com o pretendido.

Neste capítulo, será apresentado como a aplicação deverá funcionar, descrevendo a funcionalidade dos módulos e como todo o sistema deverá interagir.

3.1 Requisitos Funcionais

O objetivo deste projeto é desenvolver uma aplicação WebRTC+SIP robusta, onde um usuário consegue interagir com outros de maneira prática, e mostrar o desempenho da aplicação em diferentes plataformas. Esta seção apresenta os requisitos que foram definidos inicialmente no projeto.

3.1.1 Estabelecimento de chamada

O sistema deve permitir que um usuário consiga estabelecer uma chamada através de video, áudio ou ambos, com outro usuário que estiver conectado no sistema. Em caso onde os ramais conectados estejam em ligação, ausente ou não conectados o sistema deverá informar o porque da ligação não ter sido completada com sucesso.

3.1.2 Estado de presença e Lista de contato

É um serviço que se encontra na maior parte de aplicações de *chat*. É necessário um banco de dados que contenha todas as informações dos usuários que se encontram na lista de contatos, para quando necessário, exibir o estado de presença em que uma pessoa se encontra. (Ocupado, Disponível, Ausente, *Offline*). Sempre que o estado de um usuário for alterado, é preciso notificar a todos que estão conectados, o estado deste, em tempo real.

3.1.3 Video, chat/mensagem instantâneas

Outro item desejável é no sistema, é um serviço de *chat*. Cada usuário poderá abrir um canal de comunicação com outro, interagindo via mensagens instantâneas. A solução sugerida foi o uso do framework Socket.io. O usuário envia sua mensagem para o servidor, e este se encarrega de encaminhar as mensagens para outros ou um cliente específico.

3.2 Requisitos Não Funcionais

Esta seção está relacionada ao uso da aplicação em termos de desempenho, usabilidade, confiabilidade, tecnologias envolvidas. Será descrita o que se deve esperar do sistema desenvolvido,

3.2.1 Desempenho

- O sistema deverá atualizar assim que o estado de um usuário for alterado. Não deverá ultrapassar mais que 5s.

3.2.2 Usabilidade

A interface deverá ser intuitiva. Desde a localização de um usuário disponível no sistema até a realização e encerramento da chamada.

3.2.3 Segurança

Nenhum agente externo poderá ter acesso aos dados trafegados.

3.2.4 Restrições

É garantido funcionamento apenas para versões atuais do Google Chrome e Mozilla Firefox.

4 Implementação

Este capítulo aborda a implementação da aplicação (utilizando a base do SIPML5), juntamente com o nosso servidor VOIP, o Asterisk. Nele será abordado as decisões do projetos, uso das principais bibliotecas utilizadas no desenvolvimento, em destaque o Socket.io, quais foram as dificuldades em todo o processo, quais decisões foram tomadas para resolver os problemas em questão.

4.1 Decisões do projeto

No início deste projeto, havia poucos softwares que implementavam bem toda a pilha do protocolo SIP em Javascript. O único que se apresentou grande no cenário até então foi o SIPML5. Este disponibilizava apenas o serviço de ligação áudio/vídeo de uma ponta a outra, sendo necessário primeiramente preencher alguns campos para o registro num servidor SIP. E foi encima desta plataforma que toda a aplicação foi desenvolvida, e testada.

Atualmente existem outras aplicações quem desempenho o mesmo papél que o SIPML5. É o caso SIP.js e do jsSIP. Feito teste de ambos em seus respectivos sites, SIP.js mostrou que não funciona tão bem no Google Chrome (41.0.2272.76). Funcionou normalmente apenas no Mozilla Firefox. O jsSIP não mostrou funcionamento em seu próprio site, tanto no Firefox, quanto no Chrome. Não recebia ligação, e ao realizar, aparecia apenas uma mensagem de ”chamando”, mas não tocava no ramal destino.

4.2 Desenvolvimento das funcionalidades

Nos tópicos a seguir, será descrita o desenvolvimento de toda a aplicação, algumas das decisões tomadas durante o projeto, e como é o relacionamento entre as funcionalidades.

4.2.1 Sistema de acesso

Explicar como é o processo de login do sipml5. explicar que ele monta a pilha sip e tal.

4.2.2 Estabelecimento de chamadas

Para o estabelecimento chamadas, foi utilizada a base do sipML5, que por sua vez foi desenvolvida utilizando a tecnologia WebRTC. Conforme mostrado na figura 2.9, toda a regra de negócio foi contruida de maneira simples, para que o próprio desenvolvedor web, possa trabalhar e desenvolver outras aplicações. Em poucas linhas foram definidas a pilha SIP, informando todos os dados do usuário no servidor VOIP, tipo de comunicação (linha 10) e qual o ramal que deverá receber a ligação (linha 15).

Nas figuras XX1 e XX2, mostra os dois estados do ramal. A figura XX1 mostra quando o usuário está ativo no sistema, sem realizar nenhuma ação. A figura XX2 mostra quando o usuário está em ligação com algum outro ramal conectado no servidor Asterisk. Note que no canto direito além do usuário ser informado sobre todo o estado da ligação, o estado dos ramais são alterados visualmente. Todas essas informações mostradas aos usuário, são coletadas através do Asterisk Manager API em conjunto com Socket.io. No anexo X, é mostrada partes do código responsáveis por esse processo.

4.2.3 Lista de contatos e estado de presença

Como o AMI libera acesso via TCP com o Asterisk, é possivel enviar comandos (*actions*) ao Asterisk, que retornará com a resposta da requisição, ou seja, ao abrir a página da aplicação, é enviado uma *action* para o Asterisk, solicitando os ramais que foram declarados no sip.conf do Asterisk. Sobre a resposta da requisição, é feita a manipulação dos dados, mostrando de maneira simples, a lista de ramais que estão registrados no servidor e o estado de cada um destes. Os códigos do anexo X, explica como é feito todo processo, desde o envio da *action*, até a manipulação dos dados.

O fluxo explicado anteriormente, pode ser vista na Figura XX

4.2.4 Mensagens Instântaneas

4.3 Dificuldade encontradas e suas soluções

Durante o desenvolvimento da aplicação, houveram algumas dúvidas e problemas que acabavam impactando no funcionamento do projeto. Foram tomadas medidas corretivas e mudanças em partes do projeto.

- falar sobre a escolha do php para coletar os usuário online, dpois foi decidido fazer por javascript.

- integrar o projeto sipml5 com o Sails.js - utilização do Socket.io para captura de ramais - implementação do chat - problemas com NAT/ configuração correta do Asterisk - bugs, quando aplicação origina ou recebe ligações na máquina .54;

5 Testes e resultados obtidos

-

5.1 Metodologia

5.2 Ambiente de testes

5.3 Avaliação de resultados

6 Conclusão

6.1 Resumo do trabalho desenvolvido

6.2 Dificuldades Encontradas

6.3 Trabalhos Futuros

Referências Bibliográficas

[1] BORGES, F. *Webrtc: Estudo e análise do projeto*. 2013. p. 56. Dissertação (Mestrado em Física) - Sistema de Telecomunicações, Instituto Federal de Santa Catarina SJ, Santa Catarina, 2013.

[2] VARANDA, J. H. O. *Ice: Uma solução geral para travessia de nat*. 2008. p. 92. Dissertação (Mestrado em Física) - Faculdade de Ciências da Computação, Universidade de Brasília, Distrito Federal, 2008.

[3] Voz sobre ip - voip, Nov. 2011. Internet: http://tele.sj.ifsc.edu.br/ msobral/rmu/slides/aula-22.pdf.

[4] AMARAL, V. M. F. *Framework e cliente webrtc*. 2013. Dissertação (Mestrado em Física) - Escola de Engenharia, UM, 2013.

[5] Javascript, June 2014. Internet: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript.

[6] Webrtc, June 2012. Internet: http://www.webrtc.org/reference/architecture.

[7] html5, Mar. 2014. Internet: https://developer.mozilla.org/pt-BR/docs/Web/HTML/Using_HTML5_audio_and_video.

[8] Getting started with webrtc, fev 2014. Internet: http://www.html5rocks.com/en/tutorials/webrtc/basics/.