UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

VICTOR SANT'ANA LEMOS

Audio Markings: Uma biblioteca para comunicação via áudio entre aplicações Web

VITÓRIA – ES
Fevereiro - 2017

VICTOR SANT'ANA LEMOS

Audio Markings: Uma biblioteca para comunicação via áudio entre aplicações Web

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Centro Técnológico da Universidade Federal do Espírito Santo, como parte dos requisitos para a obtenção do título de Mestre em Informática.

Orientador: Prof. Dr. Celso A. Saibel Santos
Coorientador: Prof. Dr. Leandro L. Costalonga

VITÓRIA - ES
2017

Dados Internacionais de Catalogação-na-publicação (CIP)
(Biblioteca Setorial Tecnológica,
Universidade Federal do Espírito Santo, ES, Brasil)

Lemos, Victor Sant'Ana, 1989-

L557a *Audio Markings* : uma biblioteca para comunicação via áudio entre aplicações Web / Victor Sant'Ana Lemos. – 2017.

79 f. : il.

Orientador: Celso Alberto Saibel Santos.

Coorientador: Leandro Lesqueves Costalonga.

Dissertação (Mestrado em Informática) – Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Som. 2. Sincronização. 3. Aplicações Web. 4. Audio. I. Celso Alberto Saibel Santos. II. Costalonga, Leandro Lesqueves. III. Universidade Federal do Espírito Santo. Centro Tecnológico. IV. Título.

CDU: 004

VICTOR SANT'ANA LEMOS

AUDIO MARKINGS: UMA BIBLIOTECA PARA COMUNICAÇÃO VIA ÁUDIO ENTRE APLICAÇÕES WEB

Dissertação submetida ao programa de Pós-Graduação em Informática do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para a obtenção do Grau de Mestre em Informática.

Aprovada em 20 de fevereiro de 2017.

COMISSÃO EXAMINADORA

Prof. Dr. Leandro L. Costalonga
Universidade Federal do Espírito Santo
Coorientador

Prof. Dr. Guido Lemos de Souza Filho
Universidade Federal da Paraíba
Avaliador Externo

Prof. Dr. Rodrigo Laiola Guimarães
Universidade Federal do Espírito Santo
Avaliador Interno

Agradecimentos

Agradeço à minha família, aos meus orientadores e aos meus colegas, dentro e fora do Laboratório de Pesquisas em Redes e Multimídia (LPRM).

Agradeço também à FAPES, pelo apoio financeiro durante a realização deste trabalho (Processo 68040610, T.O. 999).

RESUMO

Os aparelhos eletrônicos que carregamos dia-a-dia, como *smartphones*, *tablets* e *notebooks*, estão integrados de tal forma às nossas vidas que talvez seja difícil notar o nível de fluidez com que operam hoje. Apesar dos avanços na questão da qualidade de experiência do usuário, a integração desses dispositivos ainda traz uma série de problemas a serem resolvidos. Em primeiro lugar, aplicações desenvolvidas para *notebooks* e *desktops* precisam ser refeitas, ou ao menos adaptadas, para rodar em um aparelho móvel. Além disso, a comunicação entre essas aplicações ainda é, na maior parte, dependente de uma conexão com a Internet, até mesmo para atividades simples como sincronização local, ou transmissão de mensagens pequenas. O desenvolvimento de aplicações Web pode ser uma solução ao primeiro problema, dado que uma mesma aplicação desenvolvida para essa plataforma poderia, a princípio, ser executada por qualquer navegador sendo executado no dispositivo. Por outro lado, isso não resolve a questão da comunicação entre estes dispositivos. Soluções como Bluetooth e NFC, em parte preenchem essa lacuna da comunicação local, mas esbarram na dependência de hardware especializado, ou em problemas de compatibilidade. Se formos considerar o cenário de desenvolvimento Web, o problema é ainda maior, uma vez que ainda não existe uma padronização para essas tecnologias em navegadores. Esse trabalho busca atuar nesse problema, empregando o som na comunicação entre aplicações Web. Dispositivos para captura e reprodução de áudio são quase onipresentes nesses aparelhos e, melhor ainda, o seu acesso através de navegadores Web já é hoje uma realidade. Nesse trabalho apresentamos as soluções já existentes na área de comunicação digital por áudio, além de descrever e avaliar nossa própria solução.

Palavras-chave: Áudio, comunicação, Web, sincronização, HTML5.

ABSTRACT

The electronic devices that we carry in our daily lives, such as smartphones, tablets and notebooks, are so integrated in our lives that it's hard to notice how fluid they work nowadays. Despite advances on the field of user experience, the integration of these devices still brings a number of problems to be solved. Firstly, applications developed for notebooks and desktops need to be rewritten, or at least adapted, to work on mobile devices. Besides that, communication between applications is still, on the most part, dependent on an Internet connection, even on simple tasks, such as local synchronization, or the transmission of small messages. The development of Web applications is a feasible solution to the first problem since the same application developed for this platform could, in principle, be executed by any browser running on the device. On the other hand, the issue of communication between these devices remains unsolved. Solutions such as Bluetooth and NFC may fill this gap about local communication, but stumble with specialized hardware dependency, or compatibility problems. If we consider Web development scenario, the problem only gets bigger, as there is still no standard for these technologies in browsers. This work seeks to address this problem, by employing the sound in the communication between Web applications. Peripherals for recording and playback of audio content are almost omnipresent on these devices and, better yet, access to them over the browser is already a reality today. In this work we present some existing solutions on this field of digital communication over audio, and we also describe and evaluate our own solution.

Keywords: Audio, communication, Web, synchronization, HTML5.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo

1

Introdução

Os dispositivos eletrônicos com os quais interagimos frequentemente, chegaram a um alto nível de sofisticação, formas de interação e multifuncionalidade. Os aplicativos desenvolvidos para *smartphones*, *tablets* e *notebooks* estão presentes em diversas atividades do nosso dia-a-dia, sendo aplicados para funcionalidades simples, como despertadores, até controles complexos para dispositivos eletroeletrônicos em casas inteligentes.

A ubiquidade desses dispositivos em nosso cotidiano impulsionou o surgimento de soluções para permitir que trabalhem cada vez mais em conjunto, através de formas de comunicação transparentes ao usuário. Quando pensamos em comunicação transparente entre dispositivos, normalmente lembramos-nos de tecnologias como *Wi-Fi*, *Bluetooth* e *NFC* (*Near-Field Communication*), formas de estabelecer comunicação sem a necessidade de fios, onde os aparelhos não precisam mais estar fisicamente conectados. Apesar de representarem um salto considerável em relação aos seus predecessores, essas tecnologias ainda possuem limitações, como a dependência de *hotspots*, problemas de compatibilidade, ou necessidade de *hardwares* especializados.

Considerando tais limitações, soluções alternativas para a transmissão rápida de pequenas quantidades de informação começaram a se popularizar. Tecnologias como *QR Code*, *audio watermark* e *audio fingerprinting* permitem que aparelhos possam interagir entre si, sem a necessidade de estarem conectados fisicamente, ou que sequer estejam na mesma rede, fazendo uso de *hardwares* para captura e reprodução de formas de mídia.

1.1 Motivação

Humanos utilizam os sentidos para estabelecerem comunicação, dessa forma, é de se esperar que os aparelhos eletrônicos a nossa volta também possuam alguma forma de interagir com esses sentidos, para que consigam interagir conosco. Assim, *hardwares* para a captura e reprodução de formas de mídia, em especial áudio e vídeo, estão presentes em grande parte dos aparelhos que nos cercam, fazendo deles ótimos candidatos para utilização como meios de comunicação, não só com o usuário, mas também entre aparelhos e aplicações.

Dentre as formas de mídia que podem ser exploradas para uso em comunicação entre dispositivos e aplicações, o áudio é um candidato interessante, em parte pela forma como o som se propaga pelo espaço, evitando que o usuário tenha que posicionar seus aparelhos de forma especial para capturar o sinal, mas também pela forma como a audição humana não consegue captar sinais em determinadas faixas do espectro sonoro[1] [1,2]. Essas características permitem que a comunicação por áudio entre dispositivos seja praticamente imperceptível ao usuário.

Vale ressaltar ainda que, apesar de projetados para trabalharem dentro dos limites da audição humana, os microfones e saídas de áudio encontrados em aparelhos como *tablets*, *smartphones* e *notebooks*, em geral são capazes de capturar e reproduzir sinais em frequências consideradas como inaudíveis ou semi-inaudíveis ao ser humano.

1.2 Objetivos

Este trabalho tem como **objetivos principais** (1) apresentar e avaliar estratégias para a comunicação entre aplicações em dispositivos móveis por meio de mensagens de áudio, codificadas em frequências próximas ao limite superior de audição humana e (2) propor e avaliar um modelo e uma biblioteca para o desenvolvimento de soluções Web, que usem o áudio como principal estratégia de comunicação entre dispositivos.

Para concretizar tais objetivos, temos os seguintes **objetivos específicos**:

- Obter os conhecimentos necessários sobre a sintetização, a captura e a análise de sinais sonoros e aplicá-los na comunicação entre aplicações de diferentes dispositivos.

[1] No texto da dissertação, iremos considerar que o espectro sonoro é composto pelo conjunto de todas as componentes de frequência que compõem os sons audíveis (20Hz a 20kHz) e não audíveis pelo ser humano (fora da faixa anterior).

- Traçar o retrato atual do desenvolvimento de aplicações Web, assim como as tecnologias envolvidas (navegadores Web, HTML5 e os padrões da W3C).
- Propor, especificar e modelar uma biblioteca para o desenvolvimento de aplicações Web, que façam uso do áudio como forma de comunicação.
- Aplicar a biblioteca no desenvolvimento de aplicações do tipo "prova de conceito" (PoC), para ilustrar o seu uso em casos reais, onde a comunicação por áudio é desejada.
- Testar e comparar a solução com outras semelhantes, em termos de funcionalidades, vantagens e limitações.

1.3 Metodologia

O foco inicial do trabalho foi a sincronização entre aplicações em aparelhos *mobile*, em especial para a aplicação em uma orquestra distribuída, utilizando os sons produzidos por diferentes *smartphones* e *tablets*. A partir deste foco, o trabalho começou como uma pesquisa sobre técnicas de sincronização que permitissem a execução de uma mesma música de forma sincronizada em todos os dispositivos envolvidos, a partir das plataformas de comunicação, captura e reprodução de áudio disponíveis em tais aparelhos.

Durante a pesquisa, foi detectado que o uso de técnicas para captura e processamento de áudio para comunicação estava se tornando alvo de trabalhos recentes, mas com um foco maior em aplicações desktop ou para comunicação submarina [3,4,5,6]. Notando o crescente interesse na área e a quase ausência de soluções para o desenvolvimento Web que façam proveito dessa estratégia de comunicação, decidiu-se por focar o trabalho no desenvolvimento de uma biblioteca para facilitar a produção de soluções nesta plataforma.

Dessa forma, foi realizada inicialmente uma pesquisa sobre técnicas para comunicação entre dispositivos a partir da captura e da reprodução de áudio. Identificadas as principais técnicas e aplicações, foi realizada uma pesquisa sobre a viabilidade de tal estratégia para o uso em navegadores Web. Uma vez confirmando a viabilidade, foi conduzido um estudo sobre o desenvolvimento de aplicações Web, assim como o desenvolvimento de bibliotecas para o uso na criação desse tipo de aplicação.

A fase final do trabalho identificou quais componentes deveriam estar presentes em uma biblioteca que permitisse o rápido desenvolvimento de soluções Web utilizando a mesma estratégia de comunicação. Ao mesmo tempo, foram priorizadas as tecnologias para

simplificar a implementação dessa biblioteca, com suporte ao maior número de dispositivos e cenários possíveis.

1.4 Estrutura da Dissertação

Esta dissertação está organizada em 5 capítulos. O primeiro capítulo aborda a introdução onde é realizada uma breve descrição do problema tratado, a motivação para o seu desenvolvimento, os principais objetivos do trabalho e a metodologia utilizada. No segundo capítulo são apresentados alguns conceitos de base, necessários para o entendimento do trabalho, em conjunto com os trabalhos relacionados identificados, com uma análise comparativa sobre os mesmos. O terceiro capítulo apresenta detalhes mais técnicos, referentes a implementação, arquitetura, plataforma e outros aspectos sobre a biblioteca *Audio Markings*, uma das principais contribuições deste trabalho. No quarto capítulo são apresentados os resultados obtidos em testes sobre a biblioteca, assim como casos de uso. Por fim, no quinto capítulo são apresentadas as contribuições e as considerações finais sobre o trabalho e o futuro da pesquisa.

Capítulo

2

Conceitos de Base e Trabalhos Relacionados

Nesse capítulo serão apresentados alguns dos conceitos de base necessários para o entendimento da proposta do trabalho. Estes conceitos estão associados a algumas propriedades físicas da onda sonora e também às técnicas envolvidas na digitalização, análise e sintetização de sinais acústicos.

Após a introdução dos conceitos de base, são apresentados alguns dos principais trabalhos relacionados à proposta de comunicação, entre dispositivos e aplicações, por meio de sinais de áudio. Esses trabalhos estão organizados de acordo com a estratégia de modulação adotada. Em um primeiro momento são apresentados os trabalhos que empregam a modulação por chaveamento de frequência, seguidos pelos que fazem uso do chaveamento de fase. A parte final do capítulo traz uma análise comparativa das estratégias e decisões adotadas pelos desenvolvedores nestes trabalhos.

2.1 Fundamentos de Áudio

Durante todo este texto, o termo som será utilizado quando quisermos nos referir puramente à manifestação física do movimento de ondas no espaço, enquanto o termo áudio será reservado para quando esse fenômeno tiver como origem ou destino algum dispositivo eletrônico, como alto-falantes, amplificadores, microfones ou outros periféricos.

O entendimento sobre as propriedades físicas do som, assim como as técnicas utilizadas para a sua captura, tratamento, sintetização e reprodução são fundamentais para a compreensão do resto do trabalho. Dessa forma, essa seção busca detalhar tais conceitos.

2.1.1 Propriedades Físicas do som

O som pode ser interpretado como o movimento de ondas, conduzidas através de um meio elástico, seja ele gasoso, líquido ou sólido, como ar, água, metal ou concreto [1]. Esse movimento possui características, como frequência, intensidade e velocidade, que determinam como o ouvido humano, ou outro meio de detecção, irá reagir ao movimento das ondas [2].

O som depende de um meio elástico para se propagar e, dessa forma, também é caracterizado por esse mesmo meio [1,2]. Como exemplo, se duas pessoas estiverem sobre os trilhos de um trem e uma delas joga uma pedra em um dos trilhos, o outro indivíduo irá escutar dois sons decorrentes da colisão, um conduzido pela vibração produzida no trilho e outro pelo ar. O som vindo do trilho chegará primeiro, pois o som é conduzido mais rapidamente pelo metal sólido do que pelo ar. As frequências produzidas e as intensidades também são diferentes, pois alguns meios são mais elásticos, ou conservam energia de formas diferentes.

A perturbação causada no meio é chamada de onda. Como podemos observar na Figura 1, uma onda sonora se propaga através do movimento de partículas encontradas no meio, criando zonas de maior e menor pressão. A fonte do som provoca a vibração e colisão dessas partículas, provocando o surgimento de tais áreas de maior e menor densidade (maior e menor pressão), que constantemente se juntam e dissipam em bandas [7].

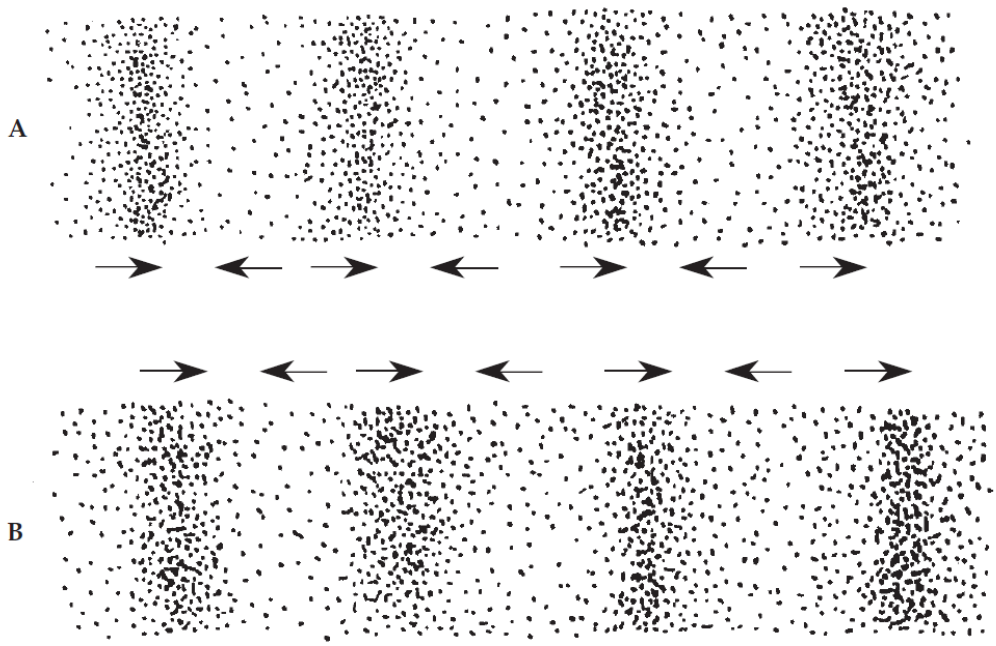

Figura 1. Ondas sonoras se deslocando pelo meio, o fenômeno causa (A) o surgimento de áreas de maior e menor densidade de partículas. Em (B) podemos ver que um instante depois a onda se moveu um pouco para a direita. Fonte: [1].

A onda sonora pode ser classificada como uma onda longitudinal, ou onda de pressão, devido à forma como se desloca no espaço, movendo-se na mesma direção da oscilação dos corpos que estejam em seu caminho [8].

Em termos matemáticos, o som pode ser representado como uma função de níveis de pressão em relação ao tempo [7], conforme o gráfico da Figura 2. É possível notar que a representação da onda na Figura 2 é análoga àquela da Figura 1, com os picos representando as áreas de maior densidade de partículas (maior pressão), enquanto os vales representam áreas de menor densidade (menor pressão).

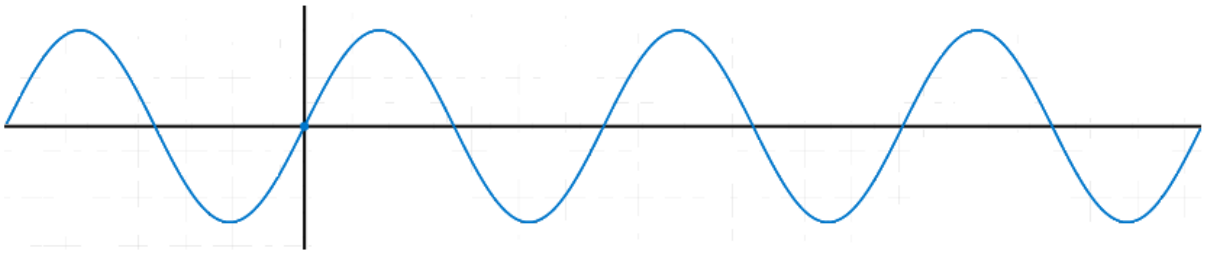

Figura 2. Representação gráfica da função do som. O gráfico apresenta os níveis de pressão (eixo vertical) em função do tempo (eixo horizontal). Fonte: Produção do próprio autor.

2.1.2 Áudio digital

Para que seja possível trabalhar com a análise ou manipulação de áudio, é preciso primeiro de formas para a captura e reprodução de sinais sonoros. Ferramentas para esse fim começaram a surgir no início do século 20, como o microfone, capaz de capturar a informação encontrada em ondas de pressão no ambiente (como ondas sonoras) e converter essa informação em sinais elétricos. Para o processo inverso, alto-falantes recebem sinais elétricos e, de acordo com a informação contida nesses sinais, produzem ondas sonoras [7].

Uma vez que a informação sonora é capturada e convertida, é possível trabalhar com o áudio de formas mais minuciosas, seja aplicando filtros no sinal, para remoção de ruídos, seja na análise do sinal, para obtenção de informações que o ouvido humano, em princípio, não seria capaz de identificar facilmente.

A conversão do sinal acústico para o meio digital é chamado de *Analog-to-Digital Conversion* (ou ADC) [9]. A Figura 3 ilustra os quatro principais processos envolvidos na conversão para o meio digital, que são: Captura do sinal acústico, aplicação de filtros, amostragem e armazenamento.

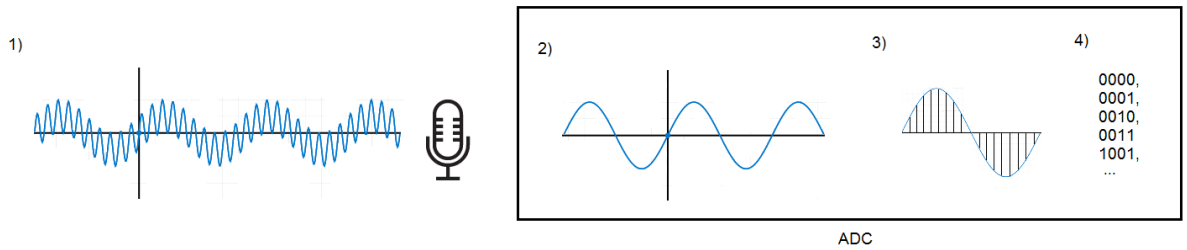

Figura 3. Principais etapas do processo de conversão do sinal analógico para digital (ADC): 1) Captura do sinal acústico, 2) aplicação de filtros, 3) amostragem e 4) armazenamento. Fonte: Produção do próprio autor.

Uma vez capturada a informação sonora através de um microfone, aplicam-se filtros sobre o sinal. A aplicação de filtros pode variar, desde um simples *low-pass filter* para a remoção de frequências muito altas (que saiam da faixa de audição humana), até filtros mais sofisticados, como o *echo cancellation*, presente na maioria dos aparelhos de comunicação usados hoje.

Com o sinal devidamente filtrado, é realizado o processo de amostragem (ou *sampling*), onde são coletadas amostras do sinal à uma determinada taxa de amostras por segundo. Uma taxa de amostragem utilizada em boa parte das aplicações de áudio é a de 44.1 kHz. Esta taxa permitiria, de acordo com o teorema de Nyquist, a representação de sinais com frequências de até 22.05 kHz [7,9].

Durante a etapa de amostragem também é realizado um processo importante, chamado de quantização. Cada amostra coletada contém a magnitude do sinal naquele instante, esse valor é armazenado de acordo com a precisão desejada para a aplicação. Grande parte das aplicações envolvendo o áudio hoje utilizam 16 ou 24 bits para o armazenamento desses valores de intensidade [7]. A quantização é essa redução da magnitude real do valor detectado em cada amostra para a escala definida para a aplicação.

A Figura 4 demonstra em maiores detalhes os processos de amostragem e quantização. O valor armazenado não será exatamente o mesmo valor do sinal analógico, é possível observar a magnitude dessa diferença na Figura 4 (em azul). A resolução do sinal pode ser melhorada de três formas: Aumentando a precisão (definindo uma quantidade maior de bits para o armazenamento dos valores encontrados em cada amostra), aumentando a taxa de amostragem, ou aumentando ambos [9].

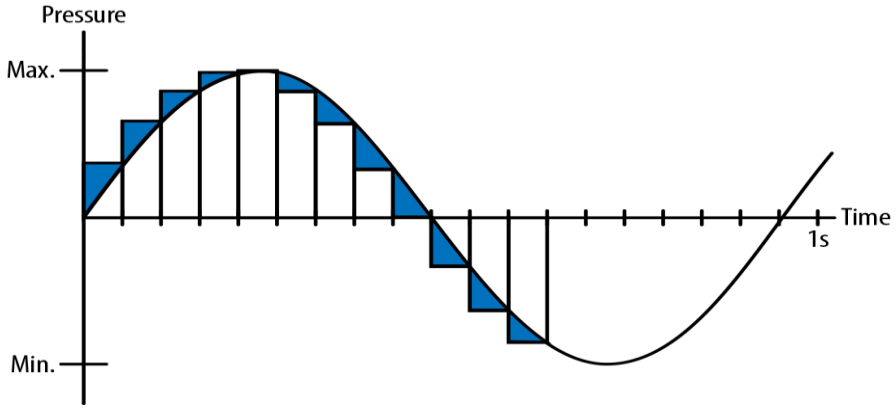

Figura 4. Demonstração dos processos de amostragem e quantização. Fonte: [7].

Uma vez que as informações sobre o sinal acústico tenham sido finalmente convertidas para o meio digital, é possível utilizar a capacidade de processamento de dados dos computadores para realizar análises sobre esses sinais. Um dos processos chave para a extração de informações em áudio digital é a análise espectral. Esse processo permite a representação de um sinal antes representado no domínio do tempo $f(t)$ para o domínio da frequência $F(k)$. A técnica usada para essa conversão é chamada de Análise de Fourier.

A Análise de Fourier define que ao analisarmos uma onda periódica, independente do seu nível de complexidade, é possível decompor essa onda em um conjunto de senoides cujas frequências são harmônicos de uma frequência fundamental, em fases e amplitudes particulares [9]. Harmônicos nesse caso representando qualquer sinal cujas frequências sejam múltiplos inteiros da frequência fundamental.

A Figura 5 ilustra as diferentes representações de um mesmo sinal acústico, visto tanto no domínio do tempo $f(t)$, quanto no domínio das frequências $F(k)$. A análise espectral nos permite analisar com maior clareza a concentração de energia nas frequências presentes em um sinal acústico.

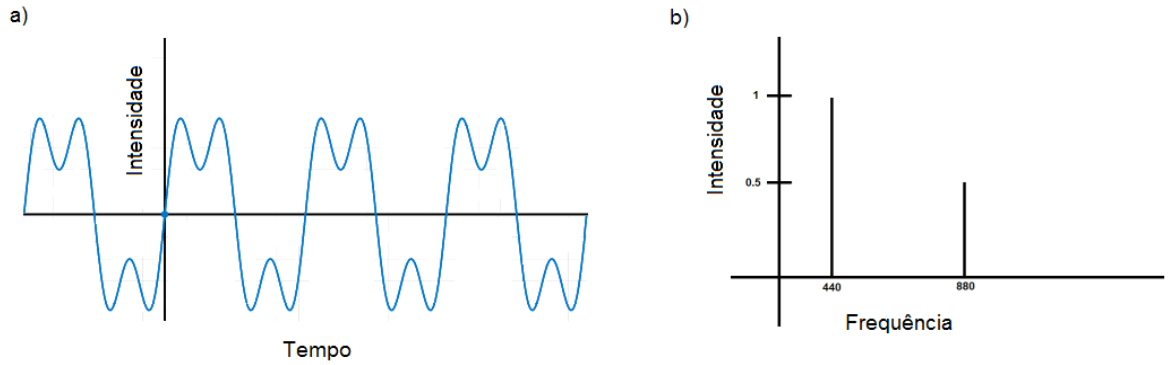

Figura 5. Representações de um mesmo sinal acústico. Enquanto a) ilustra o sinal no domínio do tempo, b) apresenta o mesmo sinal representado no domínio das frequências. Fonte: Produção do próprio autor.

Da mesma forma que é possível analisar um sinal através da decomposição do mesmo em senoides, é possível também a criação de sinais através de um processo similar. A Síntese de Fourier (ou Síntese Espectral) é o processo de construção de um sinal complexo, a partir da especificação das intensidades presentes nos harmônicos de uma frequência fundamental [9].

Como exemplo da Síntese de Fourier, o sinal presente na Figura 5 poderia ser gerado usando a frequência 440 Hz como frequência fundamental. Como o primeiro harmônico de uma frequência fundamental é a própria fundamental, o valor de intensidade para o primeiro harmônico nesse caso seria 1. O segundo harmônico, representado pela frequência de valor 880 Hz, teria como valor de intensidade 0.5. Como as únicas frequências presentes nesse sinal são 440 e 880 Hz, os demais harmônicos teriam intensidade igual a 0.

Após a produção do sinal, seja através da Síntese de Fourier, seja através de manipulações feitas sobre um sinal capturado por um microfone, é interessante que essa informação seja reproduzida como som em um ambiente. Através de um processo conhecido como *Digital-to-Analog Conversion* (DAC) [9], é possível converter a informação digital novamente em ondas sonoras.

No passado as operações sobre o sinal de áudio eram realizadas puramente por meio de ferramentas analógicas, mas hoje grande parte dessas ferramentas possuem equivalentes digitais. Graças a esse avanço, é possível aplicar técnicas como a modulação de sinais no meio acústico em uma variedade de plataformas e dispositivos diferentes.

2.2 Modulação por Chaveamento de Frequência

A modulação por chaveamento de frequência, também conhecida como FSK (*Frequency Shift Keying*), divide o espectro de frequências encontrado na onda portadora entre os valores que podem ser enviados durante uma transmissão [10]. Na sua forma mais simples, conhecida como *Binary FSK*, o espectro da onda portadora é divido em 2 frequências, uma representando o valor 0 e outra o 1, conforme observado na Figura 6.

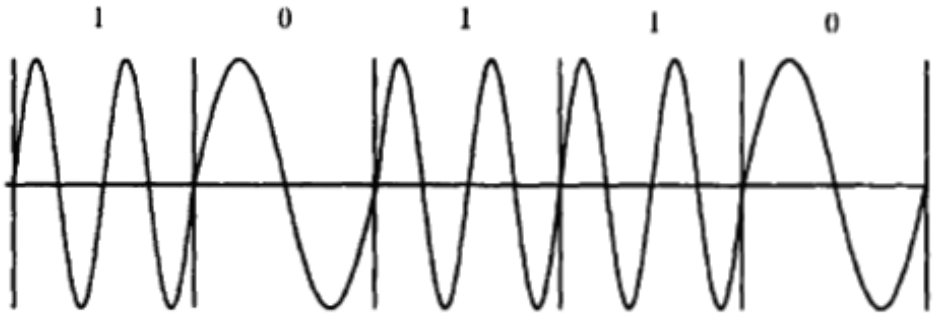

Figura 6. Exemplo de modulação em *Binary FSK*, extraído de [10]. Sinal contendo a informação '10110', enviado de forma serial.

Entre os principais trabalhos relacionados ao tema de comunicação por áudio entre dispositivos, que adotam o FSK como estratégia de modulação, estão o `minimodem`, Chirp, Google Tone e a `sonicnet.js`, apresentados em maiores detalhes nas seções abaixo.

2.2.1 Minimodem

Talvez o mais conhecido dentre os trabalhos listados, o `minimodem` [11] é um programa para a transmissão em áudio de dados entre computadores. As informações são enviadas de forma serial em binário, a própria aplicação gera os sinais de áudio través de osciladores, mas também é possível utilizar arquivos de áudio contendo sinais pré-gerados.

O `minimodem` foi desenvolvido em C, para sistemas GNU/Linux e é controlado através de comandos em um console, conforme o exemplo da Figura 7. A solução trabalha primariamente com sinais em frequências dentro do espectro audível, entre 1 a 3 KHz, mas pode ser configurada para trabalhar no espectro sonoro inaudível.

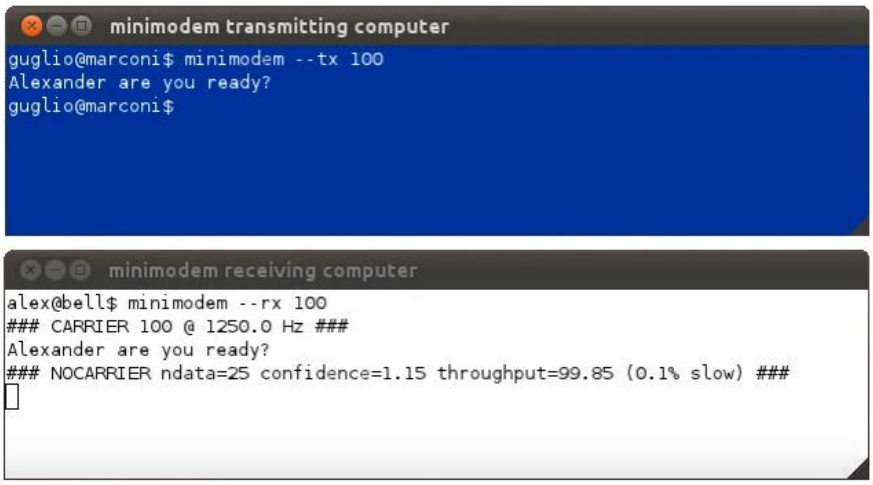

Figura 7. Exemplo do `minimodem`. O primeiro computador, representado pelo console azul, transmite a mensagem "*Alexander are you ready?* ", que é capturada e apresentada na segunda máquina. Fonte: [11].

Desenvolvido primariamente por Kamal Mostafa, o sistema possui código aberto, disponível no GitHub [12], através da licença GNU GPL, versão 3 [13]. Essa licença autoriza a utilização do código fonte, desde que qualquer trabalho dele derivado esteja igualmente aberto e disponível sob os mesmos termos.

Em [14] os autores Hanspach e Goetz empregam o `minimodem` para estabelecer formas de comunicação secretas, ou *covert channels*. A intenção em uma *covert channel* é realizar a comunicação entre dois aparelhos, sem que essa seja detectada por meios de monitoramento de redes padrão. Para impedir que esses sinais transmitidos sejam percebidos por humanos, foram usadas frequências próximas a 18.6 KHz.

Outro trabalho que faz uso do `minimodem` é o apresentado em [15]. Nessa pesquisa, os autores modelaram um sistema de monitoramento de peixes, em uma piscicultura. Com um módulo preso ao corpo do animal, contendo sensores e um microfone, é utilizada a comunicação acústica para a transmissão de dados entre o sensor e um módulo imerso no tanque (*Sink*), conforme ilustrado na Figura 8. A escolha do som como meio para comunicação se deve ao fato de que em ambientes submarinos, a partir de determinadas profundidades, ondas de rádio sofrem um alto nível de atenuação [3,4].

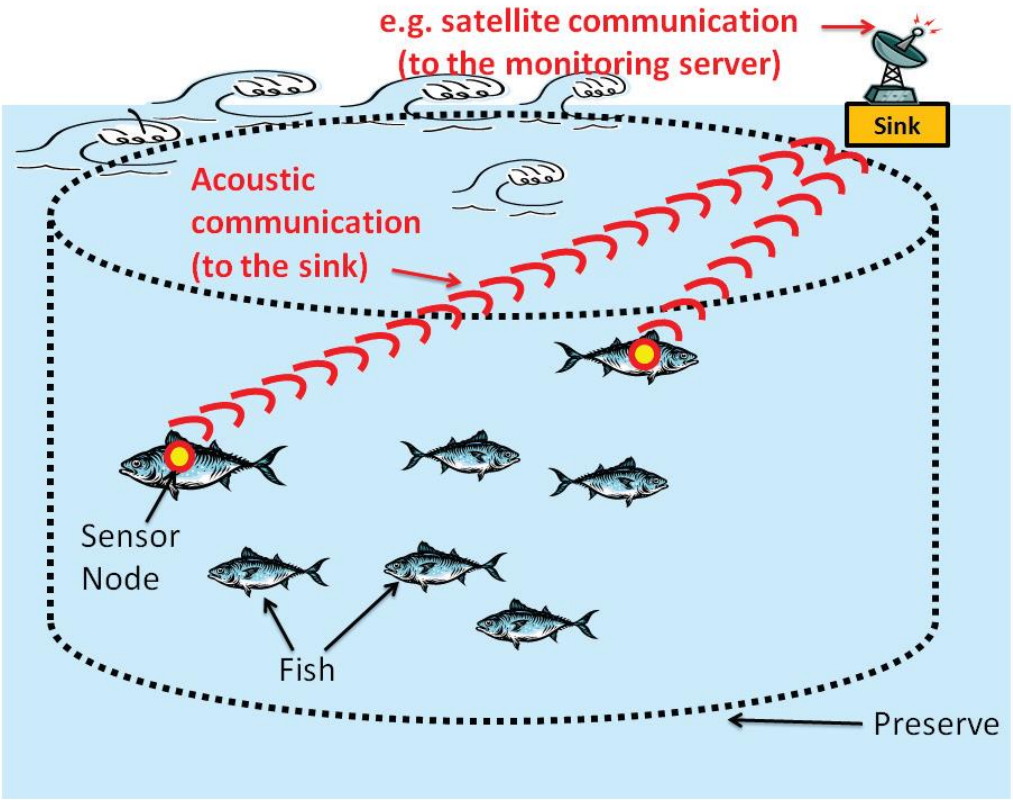

Figura 8. Cenário apresentado em [15], para monitoramento de peixes em um tanque. Fonte: [15].

2.2.2 Chirp

Desenvolvido pela companhia inglesa Asio (Asio Limited), o Chirp [16] consiste em um conjunto de tecnologias para a transmissão e recepção de informação, através do som. A unidade de informação enviada, chamada pela empresa de "*chirp*", é um sinal acústico no espectro audível, que contém um *array* de caracteres, definidos no alfabeto da aplicação. Assim como o `minimodem`, a própria aplicação se encarrega de gerar os sinais de áudio, mas é possível utilizar sinais armazenados em arquivo.

Apesar de modulado por chaveamento de frequências, o Chirp não utiliza o padrão *Binary FSK*, optando por uma alternativa chamada MFSK (*Multiple Frequency Shift Keying*) [17], onde o espectro da onda portadora é dividido em mais do que duas frequências, permitindo o envio (não simultâneo) de mais do que 2 caracteres diferentes. No caso do Chirp, é empregado um alfabeto contendo 32 caracteres diferentes.

A empresa oferece um conjunto de funcionalidades, na forma de SDKs, para o desenvolvimento da parte de comunicação por áudio em aplicações. O Chirp é uma ferramenta comercial, seu uso é controlado por meio de chaves de acesso para uso dos SDKs. A ferramentas é disponibilizada para múltiplas plataformas, possuindo implementações diferentes para cada uma delas. As plataformas disponíveis e suas particularidades podem ser observadas na Tabela 1.

Tabela 1. Conjunto de plataformas que o Chirp atende e as funcionalidades presentes em cada uma.

Plataforma/ Funcionalidades	Transmissão	Recepção	Modo *Offline*
iOS	✓	✓	✓
Android	✓	✓	✓
Javascript	✓	✘	✘
Python	✓	✘	✘
Arduino / Spark	✓	✘	✘

Conforme a tabela apresenta, é possível notar que nem todas as plataformas possuem as mesmas funcionalidades. Como apresentado pela empresa, a tecnologia envolvida na

decodificação do sinal de áudio é de origem proprietária, portanto linguagens e plataformas mais abertas, como *JavaScript* e Arduino, se restringem apenas ao envio de dados. A opção de trabalhar em modo *offline* também é restrita a apenas IOS e Android, uma vez que nessas plataformas é possível controlar o uso da ferramenta mais facilmente, enquanto no caso das plataformas mais abertas, a empresa controla o acesso através de autenticadores *online*.

As mensagens, ou *chirps*, são enviadas de forma serial, o aparelho que realiza a transmissão envia um caractere da mensagem por vez. Cada mensagem possui 20 caracteres, os dois primeiros são um identificador, seguidos pelos 10 caracteres da mensagem, com os 8 caracteres restantes reservados para correção de erros.

Apesar de por padrão a ferramenta trabalhar dentro do espectro de frequências audíveis, entre 1.7 e 10 KHz, a mesma possui um modo ultrassônico. Quando utilizada em modo ultrassônico, a ferramenta apresenta um alfabeto mais limitado, com 16 caracteres diferentes, além de diminuir o tamanho da mensagem, perdendo 2 dos 10 bits reservados para o dado a ser enviado.

O Chirp é utilizado por empresas conhecidas, como Blizzard, Activision e Kickstarter. A Activision em particular emprega a ferramenta em um de seus jogos, o Skylander [18], para a transmissão de dados entre o jogo e uma aplicação para *smartphones*. No jogo é possível criar personagens customizados, que podem ser transferidos para o *smartphone* do usuário através de mensagens em áudio, conforme observado na Figura 9.

Figura 9. Instruções no jogo Skylander, sobre como importar personagens do jogo, chamados de Imaginators, para um aplicativo de *smartphone*, através de comunicação em áudio. Fonte: [18].

2.2.3 Google Tone

Desenvolvido por Alex Kauffmann e Boris Smus, o Google Tone [19] é uma extensão para o navegador Chrome [20], que permite a transmissão de URLs entre aparelhos, através do som. A extensão está disponível sem custo e pode ser instalada através da loja de extensões para o Chrome (Chrome Web Store).

Para utilizar a extensão na transmissão de URLs, os aparelhos devem estar próximos e com o navegador Chrome aberto, ambos conectados à uma conta Google e com a extensão habilitada, como ilustra a Figura 10. Uma vez que o usuário decida compartilhar a sua URL atual, basta clicar no ícone da extensão no navegador, e a URL será compartilhada entre os demais aparelhos presentes.

Figura 10. Cenário de Utilização do Google Tone. Uma vez acionado, a URL da página atual é transmitida. Fonte: [19].

O Google Tone utiliza uma forma especial de modulação por chaveamento de frequências, o DTMF (*Dual Tone Multi-Frequency*) [21], a qual é empregada em grande parte por sistemas de telecomunicações, como telefones e correios de voz. Similar ao MSFK, utilizado pelo Chirp, o DTMF separa o espectro da onda portadora em várias frequências diferentes, mas nesse caso as frequências são enviadas em pares.

Por adotar essa forma de modulação, o Google Tone acaba por emitir sinais em frequências do espectro audível, entre 1 a 3 KHz. Dessa forma, em áreas com som ambiente, como próximo a televisões ou conversa alta, a ferramenta apresenta uma queda na qualidade da transmissão, conforme observado em [22].

Por ser uma extensão exclusiva para o navegador Chrome, essa ferramenta não pode ser utilizada em outros navegadores. Também não foi encontrada nenhuma versão para *smartphones* ou outros aparelhos móveis, tendo o uso restrito à *desktops* e *notebooks*. Não foi possível obter acesso ao código fonte da ferramenta, ou maiores detalhes sobre sua

arquitetura. As informações encontradas foram aquelas divulgadas em [19] ou observadas em experimentos conduzidos em [22].

2.2.4 Sonicnet.js

Similar à solução apresentada nessa dissertação, a `sonicnet.js` [23] é uma biblioteca desenvolvida para facilitar a utilização de comunicação via áudio entre aplicações Web. A biblioteca foi produzida por Boris Smus, também um dos desenvolvedores responsáveis pelo Google Tone.

Da mesma forma que o Chirp, a `sonicnet.js` trabalha com sinais modulados em MFSK transmitidos de forma serial. A solução possui um alfabeto de 42 caracteres distintos, mais dois caracteres especiais para início e final de transmissão. Diferente do Chirp, uma mensagem não tem tamanho definido, desde que a aplicação identifique os caracteres inicial e final, o que for encontrado no meio será interpretado como uma mensagem.

Por padrão, a biblioteca trabalha com frequências entre 18.5 a 19.5 KHz, enviando um caractere da mensagem a cada 0.3 segundos. Pela escolha das frequências, é possível notar que a `sonicnet.js` também trabalha fora da audição humana. Mas caso o desenvolvedor não se importe de gerar sinais no espectro audível, a biblioteca pode ser configurada para utilizar qualquer frequência que o aparelho seja capaz de emitir.

Além de identificar apenas mensagens que apresentem os caracteres inicial e final, a biblioteca apresenta outras formas de prevenção de erros. Caso sejam identificadas frequências com intensidades muito distantes do esperado (esses valores foram pré-definidos pela biblioteca), a `sonicnet.js` reinicia as suas operações. Caso o mesmo caractere seja detectado mais do que 2 vezes em sequência (parâmetro configurável), o mesmo é ignorado.

Desenvolvida em *JavaScript*, a biblioteca também faz uso das mesmas APIs de áudio para navegadores Web que a solução desenvolvida nesse trabalho, a *MediaStream* [24] e a *Web Audio* [25]. O código fonte está disponível online, sob a licença Apache 2.0 [26]. Através dessa licença é possível utilizar e modificar a solução desenvolvida, desde que uma cópia da licença esteja disponível e que qualquer modificação realizada ao código fonte seja devidamente documentada.

Uma das aplicações desenvolvidas pelo próprio autor, afim de exemplificar o uso da sua biblioteca, está ilustrada na Figura 11. Nessa demonstração, um usuário seleciona uma

entre 6 imagens em um aparelho, essa mesma imagem deve então ser enviada para qualquer aparelho que esteja nas proximidades e que também esteja utilizando a mesma aplicação. Essa aplicação é bem similar à uma das provas de conceito desenvolvidas nesse trabalho, apresentada em maiores detalhes no capítulo 4.

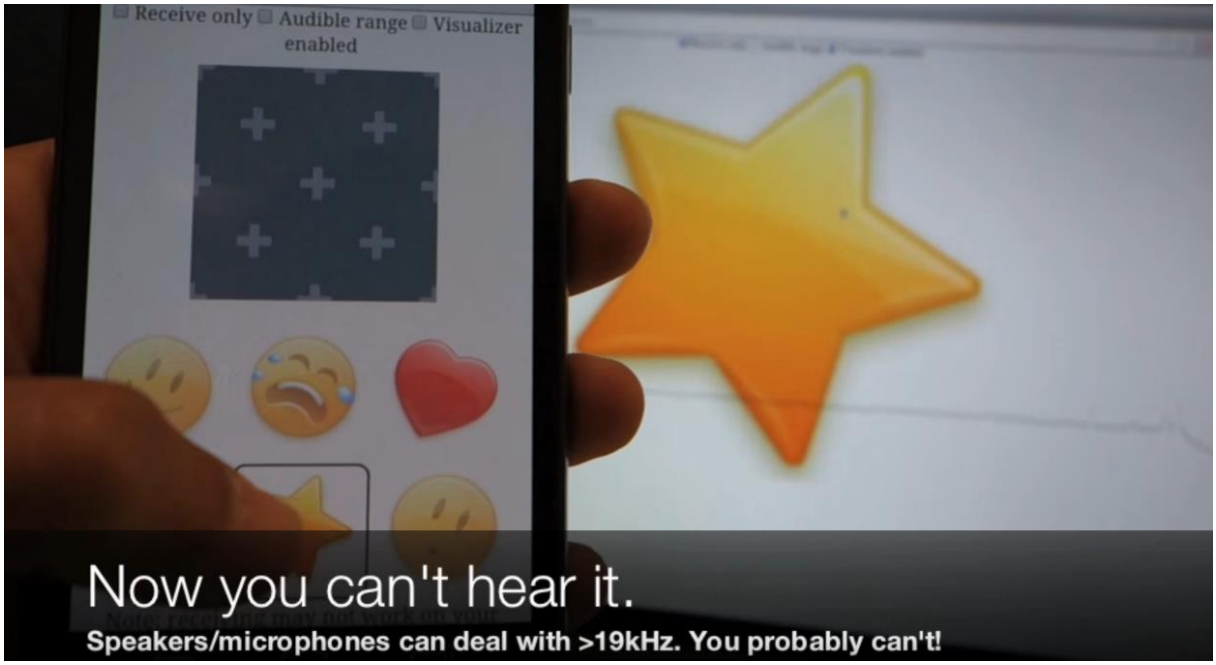

Figura 11. Demonstração de uso da biblioteca `sonicnet.js`. A mesma imagem selecionada no *smartphone* é transmitida através do áudio para um *notebook*, ao fundo. Fonte: [27].

Em [28] os autores utilizam a biblioteca em seus experimentos. O trabalho consiste em avaliar se no cenário de comunicação entre *Smart TVs* e aplicações de segunda tela, a comunicação via áudio se mostra uma opção viável. Os resultados foram no geral positivos, destacando também a capacidade dos aparelhos televisores de enviar sinais acústicos em frequências fora da audição humana, abrindo as portas para mais essa plataforma como candidata para uso na comunicação via áudio.

2.3 Modulação por Chaveamento de Fase

Na modulação por chaveamento de fase, ou PSK (*Phase Shift Keying*), a diferença entre as fases de uma onda sonora é o recurso explorado na criação dos diferentes valores que podem ser transmitidos [10]. Assim como a FSK, nessa estratégia a forma mais simples de modulação consiste em associar os valores 0 e 1 a algum recurso presente na onda, no caso do PSK esse recurso é fase do sinal. Também conhecida como *Binary PSK*, nessa estratégia um valor é associado à fase 90° e outro à 270°, conforme ilustrado na Figura 12.

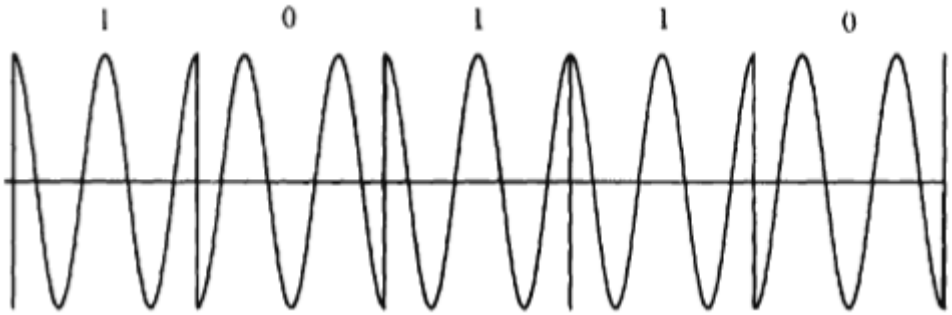

Figura 12. Exemplo de modulação em *Binary PSK*, extraído de [10]. Sinal contendo a informação ‘10110’, enviado de forma serial.

Entre os principais trabalhos relacionados ao tema de comunicação por áudio entre dispositivos, que adotam o PSK como estratégia de modulação, estão o GUWMANET e o AudioModem, apresentados em maiores detalhes nas seções abaixo.

2.3.1 GUWMANET

O GUWMANET (*Gossiping in Underwater Acoustic Mobile Ad-hoc Network*) é um protocolo de redes, projetado para comunicação submarina através de mensagens em áudio [4]. Projetado pela Fraunhofer em conjunto com a Bundeswehr (forças armadas da Alemanha), o protocolo é uma proposta para atender aos cenários definidos no projeto RACUN (*Robust Acoustic Comunication in Underwater Networks*) [6], que busca estabelecer um padrão para a comunicação em ambientes submarinos, de forma similar aos padrões de rede estabelecidos para ambientes terrestres.

O padrão define mensagens de 128 bits, separando 96 bits para o dado a ser transmitido, os demais bits são destinados a informações como destinatário, origem e prioridade. Por padrão, as frequências utilizadas vão de 3.5 a 7.5 KHz. O GUWMANET não define uma forma de detecção ou correção de erros, mas em [4] os responsáveis pelo padrão sugerem o uso de mecanismos como *Forward Error Correction* e *checksum*.

Diferente dos outros trabalhos apresentados, no GUWMANET a informação não é transmitida de forma serial. Nesse caso ocorre um processo de multiplexação, onde os diferentes bits da mensagem são enviados todos de uma vez, cada um em uma frequência diferente da onda portadora. Cada sinal enviado é modularizado com uma forma especial de modulação por chaveamento de fases, conhecido como QPSK (*Quadrature Phase Shift Keying*) [10], onde são utilizadas 4 fases diferentes do sinal para representar os caracteres enviados.

Em [5] o padrão foi testado em ambiente marinho. Os testes foram realizados na costa italiana e consistiam em dois cenários diferentes de detecção de ameaças, onde módulos imersos, contendo sensores e comunicadores, deveriam detectar e reportar a descoberta de submarinos e minas. Nos dois cenários, os módulos utilizando o GUWMANET como protocolo obtiveram uma média de 90% de acerto no reconhecimento de pacotes durante as transmissões.

2.3.2 AudioModem

Desenvolvido pela empresa francesa Applidium, o AudioModem [29] é um *software* para transmissão de mensagens, utilizando os dispositivos de entrada e saída de áudio encontrado em aparelhos *mobile*. Projetado para a plataforma iOS, o programa foi desenvolvido utilizando a linguagem Objective-C.

O AudioModem utiliza um processo de modulação similar ao *Binary PSK*, associando os valores 0 e 1 às fases da onda portadora. No entanto, durante a transmissão do sinal, cada bit enviado é na verdade o resultado de uma operação XOR entre o bit atual da mensagem e o bit anterior, conforme ilustrado na Figura 13. Essa estratégia é conhecida como DBPSK (*Diferential Binary Phase Shift Keying*) [10] e no processo de demodulação explora a diferença das fases entre os sinais capturados, no lugar de definir um padrão, para identificar os valores enviados em uma mensagem.

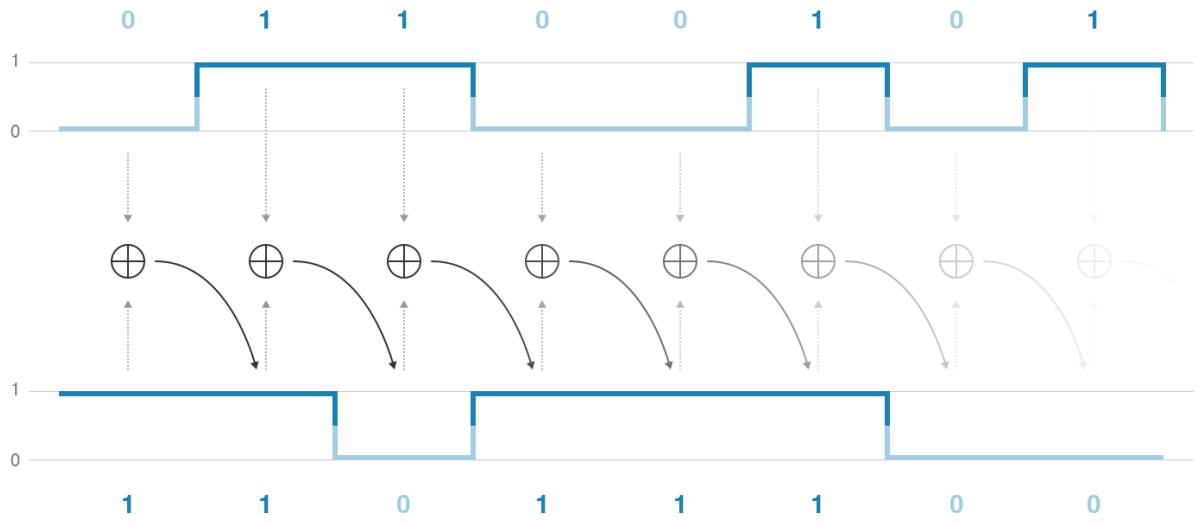

Figura 13. Exemplo do processo de modulação de sinais em DBPSK. A informação original a ser transmitida, ‘01100101’, foi convertida para ‘11011100’. Fonte: [29].

A aplicação trabalha com sinais entre as frequências de 18.4 e 20.8 KHz, saindo da faixa audível. A solução faz uso de duas APIs da plataforma iOS, o vDSP, responsável pelas operações de análise e sintetização de sinais, assim como a Audio Queue Service, para adquirir acesso ao microfone dos aparelhos.

O código fonte do AudioModem está disponível online, sob uma licença elaborada pela própria Applidium. Essa licença permite a utilização, modificação e redistribuição do código, desde que uma cópia da mesma esteja presente junto ao código fonte.

2.4 Considerações e Comparações Entre os Trabalhos

Os trabalhos apresentados nesse capítulo abordaram de formas diversas o cenário atual de comunicação via áudio entre aplicações. Enquanto alguns apresentaram soluções alternativas para casos já conhecidos, como envio de mensagens de texto, transmissão de dados ou sincronização de conteúdo, outros focaram no fornecimento de bibliotecas e ferramentas, para abstrair as particularidades envolvidas no processamento de áudio. Dessa forma, a análise das suas propostas, decisões e estratégias adotadas contribuíram para o desenvolvimento e aprimoramento da solução apresentada nessa pesquisa.

Apesar de buscarem resolver problemas diferentes, é possível realizar uma comparação entre as estratégias adotadas pelos desenvolvedores, avaliando os resultados obtidos e dessa forma analisando como essas decisões podem contribuir para o desenvolvimento de novas soluções na área.

2.4.1 Estratégia de Modulação

A estratégia de modulação que cada solução adota afeta não só o processo de geração dos sinais acústicos, mas também que tipo de impurezas devem ser tratadas durante a interpretação desses sinais. As limitações das plataformas adotadas também são um fator importante na escolha da estratégia de modulação.

A modulação por chaveamento de frequências, ou FSK, foi a estratégia de modulação digital utilizada nos primeiros sistemas de comunicação [10]. Essa estratégia é também uma das mais simples, os recursos necessários para a sua utilização podem ser encontrados em uma variedade de dispositivos. Essa simplicidade permite que essa solução não só seja consideravelmente rápida, mas também que consuma poucos recursos [30].

A simplicidade do FSK tem um custo, essa forma de modulação é altamente vulnerável a ruídos [10], seja proveniente do ambiente ou gerado pelos próprios dispositivos de áudio. Como essa estratégia analisa o nível de intensidade em frequências diferentes, e algumas de suas variantes procura por áreas de maior intensidade dentro do espectro, ruídos encontrados nessas frequências podem facilmente introduzir erros no processo de detecção de mensagens.

Dos trabalhos apresentados, o `minimodem` é o único que faz uso da forma mais simples do FSK, separando o espectro em apenas duas frequências. Essa estratégia sacrifica a quantidade de informação enviada em cada sinal, mas o espectro necessário para a mesma é consideravelmente menor que as outras soluções. Dessa forma, o `minimodem` poderia ser mais facilmente adaptado para trabalhar em ambientes com um espectro disponível mais limitado.

As demais soluções apresentadas que fazem uso do FSK utilizam versões derivadas dessa forma de modulação, empregando uma quantidade maior de frequências diferentes, o que permite o envio de uma maior quantidade de informação por sinal. O Google Tone em especial faz uso de duas frequências diferentes por sinal, o que aumenta ainda mais a quantidade de informação enviada por vez, mas ao custo de gerar sons no espectro audível.

A modulação por chaveamento de fase é provavelmente a estratégia de modulação mais utilizada na indústria de comunicação, com uma alta gama de versões diferentes derivadas dessa estratégia [10]. Mais complicada que a FSK, a modulação por chaveamento de fase demanda maior precisão na análise do sinal, especialmente em casos onde mais de duas fases diferentes são utilizadas, aumentando consideravelmente as exigências em termos de poder computacional [3]. Por não depender tanto do nível de intensidade detectado nos sinais, o PSK é menos vulnerável à ruídos. O PSK também demanda menos em questão de largura de espectro, mesmo em comparação com casos como o do `minimodem`.

O AudioModem busca resolver o problema de precisão através de uma variação do PSK, calculando a diferença entre as fases dos sinais capturados. Essa estratégia também remove a necessidade de que os dispositivos saibam de antemão quais valores cada fase representa.

Diferente de todos os outros trabalhos apresentados, o GUWMANET é o único que não envia a mensagem de forma serial, cada sinal pode conter uma mensagem completa [3,4].

Essa estratégia permite uma transmissão mais rápida de conteúdo, ao preço de ser mais vulnerável à erros de transmissão, até mesmo a ruídos. A velocidade é um fator importante no caso do GUWMANET, uma vez que a transmissão de sinais já é naturalmente mais lenta em ambientes marinhos [4].

2.4.2 Espectro de Frequências Adotado

Considerando o espectro de frequências audíveis ao ser humano, assim como a forma como os dados a serem transmitidos podem ser codificados em sinais de áudio, gerando ruídos e em sua maioria desagradáveis, é visível a preocupação dos desenvolvedores em aplicar estratégias para esconder, ou ao menos controlar, o contato dos usuários com os sinais gerados na transmissão.

Trabalhos como o AudioModem e a biblioteca `sonicnet.js`, buscam tornar esses sinais imperceptíveis ao ouvido humano, adotando frequências inaudíveis ou semi-inaudíveis, acima dos 18 KHz [14,27,28]. Outros casos, como o Chirp e o `minimodem`, adotam primariamente frequências no espectro audível, mas oferecem a opção para utilização do espectro inaudível. Apesar de oferecerem essa alternativa, é possível notar que essas soluções não foram desenvolvidas com essa estratégia em mente, ou que essa opção fornece algumas limitações, como o tamanho reduzido de mensagens no caso do Chirp.

Uma alternativa adotada por outros trabalhos, foi a de tornar os sons gerados menos desagradáveis aos ouvidos dos usuários. É o caso do Chirp e o Google Tone, onde toda ou parte dos sinais gerados estão concentrados entre as frequências de 1 a 10 KHz. Segundo o Chirp, os sons emitidos são similares ao canto de um pássaro. No caso do Google Tone, é possível notar que as frequências utilizadas são as frequências fundamentais de algumas notas musicais. Em ambos os casos, como a informação é transmitida de forma serial, esse problema é amenizado também pelo fato de que cada sinal tem uma duração baixa, não expondo o ouvinte ao mesmo ruído por longos períodos.

O único trabalho a não adotar nenhuma dessas estratégias foi o GUWMANET. Por se tratar de um trabalho para uso no ambiente submarino, é de se entender que o espectro audível ao ser humano não foi uma das preocupações durante o desenvolvimento. De fato, esse assunto não é abordado nos cenários que o projeto RACUN [6] apresenta, que foram o alvo durante o desenvolvimento do GUWMANET.

2.4.3 Disponibilidade de Plataformas

As plataformas que uma solução deve atender são um fator importante no desenvolvimento, podendo afetar não só a disponibilidade da solução, mas limitar quais estratégias e recursos os desenvolvedores terão à sua disposição. No caso da comunicação por áudio, a única exigência geral é que essa plataforma tenha alguma forma de acesso aos dispositivos de áudio encontrados nos aparelhos, o que reflete na gama de plataformas diferentes encontradas ao analisar os trabalhos na área.

Dos trabalhos apresentados nesse capítulo, o `minimodem` e o AudioModem são os mais limitados em questão de plataformas em que atendem. O `minimodem` atende apenas a sistemas GNU/Linux, diminuindo consideravelmente quantidade de aparelhos que podem fazer uso do mesmo, excluindo completamente a solução do mundo *mobile*. No caso do AudioModem, apenas sistemas iOS são capazes de executar o programa, restringindo não só o sistema operacional, mas também o *hardware* que pode ser usado.

O Google Tone já atende a uma variedade maior de aparelhos, sendo uma extensão para o navegador Chrome, o mesmo pode ser usado em qualquer sistema operacional, desde que o mesmo tenha instalado uma versão compatível do navegador. A dependência de um navegador específico é um problema, mesmo que o Chrome seja talvez um dos navegadores mais utilizados, alguns aparelhos simplesmente não possuem uma versão do mesmo, ou ao menos uma versão compatível com a extensão, como é o caso de aparelhos *mobile*, onde não existe a possibilidade de se utilizar extensões no navegador.

Soluções como o Chirp e a `sonicnet.js` atendem a uma variedade considerável de aparelhos e sistemas operacionais diferentes. As duas soluções fazem isso de formas diferentes, o Chirp possui implementações em várias plataformas, enquanto a `sonicnet.js` possui uma única implementação, mas em uma plataforma que atende a vários dispositivos. O Chirp possui a limitação de que nem todas as suas implementações são iguais, apresentando versões com recursos a menos. A `sonicnet.js` é um caso similar ao Google Tone, podendo ser usado apenas em aparelhos que possuam um navegador Web instalado, com a diferença de que nesse caso a solução não depende de um navegador específico, como o Chrome.

Capítulo

3

A Biblioteca *Audio Markings*

A *Audio Markings* é uma biblioteca *opensource* disponível *online*[2], para o desenvolvimento de aplicações Web, que busca facilitar o uso do áudio como meio de comunicação entre aplicações.

Tanto a solução desenvolvida nesse trabalho quanto os seus casos de uso estão disponíveis em repositório aberto, sob a licença MIT [31]. A licença autoriza o uso da biblioteca e o código fonte sem custos, de qualquer natureza, permitindo a edição de seu conteúdo, venda ou distribuição de softwares e produtos que façam o uso de parte ou toda a solução, com a única restrição de que uma cópia da licença esteja presente.

3.1 Processo de Desenvolvimento

O desenvolvimento da biblioteca baseou-se no processo denominado *Evolving Frameworks*, proposto por Roberts e Johnson em 1996 [32]. Nesse trabalho, os autores defendem que antes mesmo do planejamento de um *framework*, os desenvolvedores envolvidos devem primeiro produzir aplicações ou protótipos que poderiam se beneficiar do uso dessa solução. Apesar de focada no desenvolvimento de *frameworks*, a estratégia pode ser aplicada no desenvolvimento de outras soluções reutilizáveis para a produção de *softwares*, como bibliotecas.

Ainda segundo Roberts e Johnson [32], o conhecimento sobre as abstrações que uma biblioteca deve contemplar para ser reutilizada em várias aplicações, só pode ser adquirido com a criação de diferentes aplicações. Em geral, nem os especialistas de domínio possuem a visão necessária para deduzir quais abstrações a biblioteca deve apresentar, nem os

[2] Endereço de acesso: https://github.com/jonkoala/AudioMarkings.js

programadores entendem o domínio o suficiente para decidir quais funcionalidades a solução deve ter, antes de efetivamente desenvolver uma aplicação que poderia fazer uso da mesma.

Dessa forma, para o desenvolvimento desse trabalho primeiramente foram desenvolvidos três casos de uso, que empregavam o áudio como forma de comunicação, a fim de extrair os principais requisitos para a biblioteca, além de permitir uma avaliação inicial da sua viabilidade. Os três casos de uso corresponderam a aplicações Web, utilizando apenas as APIs *Web Audio* e *MediaStream* para a sintetização, captura e análise de áudio. Em uma etapa posterior, os mesmos casos de uso foram novamente desenvolvidos, dessa vez utilizando a biblioteca gerada. Estes experimentos serão descritos em maiores detalhes no capítulo 4.

Ao final do desenvolvimento das aplicações iniciais, foi realizado um levantamento sobre qual seria o formato ideal para a codificação das mensagens a serem transmitidas, como essas mensagens poderiam ser convertidas em áudio e quais funcionalidades que uma biblioteca para comunicação por áudio, para aplicações similares, deveria disponibilizar.

Ficou definido que as mensagens deveriam ser codificadas em binário, a conversão dessas mensagens para sinal acústico seria feita através osciladores, empregando a Síntese de Fourier, distribuindo os bits da mensagem como intensidades entre os harmônicos de uma frequência fundamental. Estes pontos serão detalhados na seção 3.2.

Nesse ponto, as funcionalidades levantadas como importantes para a biblioteca foram:

- Codificação e decodificação de mensagens em binário para decimal;
- Abstração do acesso da aplicação aos dispositivos de entrada e saída de áudio presentes no aparelho, além do tratamento de erros durante esse acesso;
- Abstração completa das APIs que lidam com a parte de áudio, uma vez que as mesmas fazem uso de conceitos muito específicos sobre a área;
- Conversão de mensagens em binário para o sinal de áudio equivalente, através da Síntese de Fourier;
- Recuperação de informação em áudio para o seu equivalente em binário, através da Análise de Fourier;
- Tratamento de inconsistências na transmissão, como ruídos e erros de leitura do sinal acústico;

- Rotina própria para checagem regular de mensagens a cada *frame* do navegador (normalmente a uma taxa de 60 *fps*, dependendo do navegador);
- Permitir o reaproveitamento de *AudioContexts* (recurso utilizado para trabalhar com áudio em navegadores), caso a aplicação já faça uso de um, por motivos de otimização [7,25].

Além do levantamento de funcionalidades, também foi realizado um estudo sobre quais características se espera de uma biblioteca em *JavaScript*, para desenvolvimento de aplicações Web. Nesse estudo foram analisadas duas bibliotecas conhecidas, a *jQuery* [33] e a *Socket.IO* [34]. Ambas são empregadas no desenvolvimento Web e em aplicações que lidam com constante interação com o usuário.

Nesse ponto, as características identificadas como importante foram:

- Trabalhar seguindo o paradigma orientado a eventos;
- Tratar de forma assíncrona as rotinas internas da biblioteca (checagem de novas mensagens e disparo de eventos);
- Utilizar dois módulos separados, sendo um para a transmissão de mensagens (*Emitter*) e outro para recepção de mensagens (*Receiver*).

A avaliação das partes da biblioteca foi realizada tanto durante o seu desenvolvimento, com testes unitários, quanto ao final do processo, com o desenvolvimento de aplicações que façam o uso das funcionalidades providas por ela. As aplicações desenvolvidas ao final do processo de avaliação foram exatamente as mesmas associadas aos casos de uso propostos no início do trabalho, mas dessa vez utilizando a biblioteca para cuidar da comunicação. Além dessas aplicações, também foram conduzidos testes para identificar as limitações da estratégia de comunicação por áudio entre aplicações, como taxa de erro na transmissão e distâncias em que a solução funciona sem maiores problemas. Os resultados destes testes serão detalhados no capítulo 4 desse trabalho.

3.2 Estratégia de Comunicação

Para facilitar a compreensão de como a biblioteca transmite e recebe mensagens, estabelecemos os seguintes termos no contexto desta seção:

- Sinal: É o sinal de áudio que contém a informação codificada a ser intercambiada entre os aparelhos.

- Mensagem: É a informação codificada e enviada por meio de um sinal e que é decodificada pelo receptor.

A *Audio Markings* é dividida em duas classes principais, *Emitter* e *Receiver*, cada uma desempenhando um papel diferente na comunicação entre aplicações. O *Emitter* fica encarregado de codificar e emitir as mensagens, enquanto e *Receiver* capta o sinal pelo microfone do aparelho, analisa o mesmo e o decodifica em uma mensagem.

A Figura 14 ilustra os passos da conversão de uma mensagem até a sua transmissão, executados pelo *Emitter*. Primeiramente o *Emitter* recebe uma mensagem a ser transmitida, essa mensagem é um número em decimal, que é então codificado para o seu equivalente em binário. O valor em binário é utilizado como base na geração do sinal acústico (os detalhes dessa conversão serão explicados na seção 3.2.2), que é transmitido pela saída de áudio do aparelho em que a aplicação esteja sendo executada.

Figura 14. Estágios de uma mensagem a ser transmitida pelo *Emitter*: a) Mensagem em decimal; b) Mensagem codificada para binário; c) Mensagem convertida para um sinal acústico; d) Transmissão do sinal acústico. Fonte: Produção do próprio autor.

É possível observar na Figura 15 o processo inverso, de captura do sinal e conversão para o seu valor original, executados pelo *Receiver*. Primeiro o *Receiver* captura o sinal de áudio pelo microfone presente no aparelho, em seguida é realizada uma análise sobre o sinal, extraindo a mensagem em binário, que por fim é decodificada para seu equivalente em decimal. Com a mensagem identificada, o *Receiver* dispara qualquer evento associado a ela.

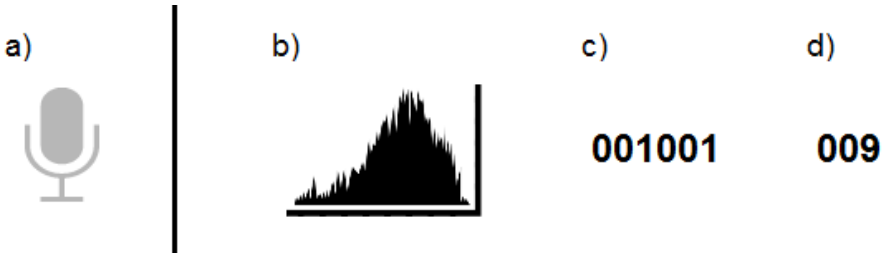

Figura 15. Estágios de uma mensagem capturada pelo *Receiver*. a) Captura do sinal de áudio; b) Sinal capturado pelo microfone do aparelho; c) Mensagem codificada em binário; d) mensagem em decimal. Fonte: Produção do próprio autor.

Como já mencionado, a solução foi desenvolvida para trabalhar de forma orientada a eventos, tanto no *Emitter* quanto no *Receiver*. Em outras palavras, a mudança de estados em ambos os módulos ocorre de forma assíncrona, assim que um evento esperado ocorre no ambiente de execução.

A Figura 16 ilustra um cenário genérico de utilização dos dois recursos em aplicações e aparelhos diferentes. Uma aplicação que faça uso do *Emitter* aguarda até que algum evento interno demande a transmissão de uma mensagem. No caso do *Receiver*, a aplicação aguarda até que alguma mensagem diferente seja detectada para que uma ação relacionada àquela mensagem seja realizada.

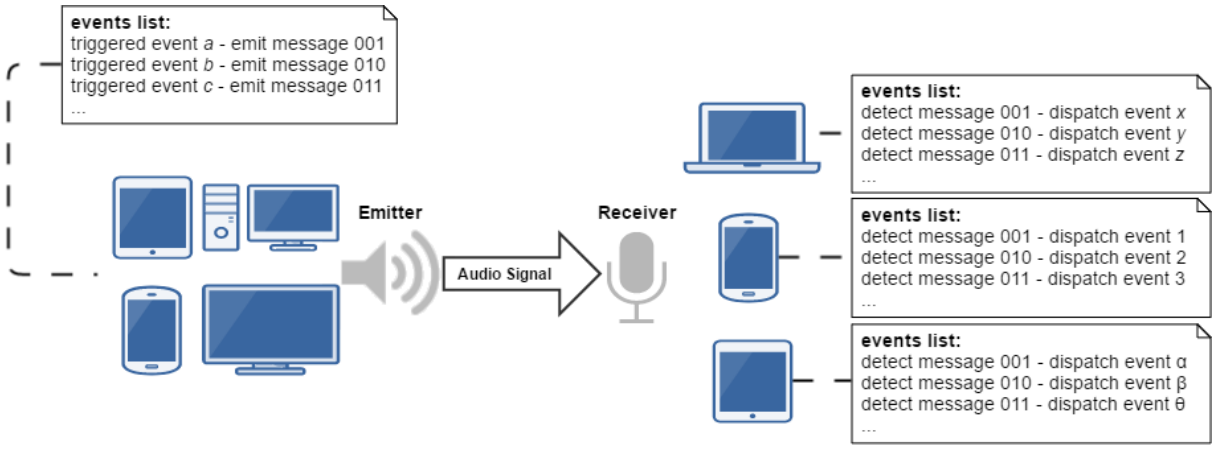

Figura 16. Cenário de utilização dos recursos da *Audio Markings*. Fonte: Produção do próprio autor.

Para facilitar a compreensão do cenário apresentado na Figura 16, vamos supor que um desenvolvedor pretende aplicar a biblioteca *Audio Markings* no desenvolvimento de uma aplicação para um jogo de bingo. Um aparelho irá executar uma aplicação que controla o sorteio dos números, enquanto os demais executam aplicações com cartelas que devem ser preenchidas de acordo com o sorteio. A fim de automatizar parte do processo, sempre que um novo número for sorteado, o mesmo deve ser marcado automaticamente nas cartelas que possuem esse número, independente da posição que o mesmo aparecer na cartela.

Uma forma de aplicar a biblioteca nesse cenário seria utilizar o *Emitter* na aplicação que controla os sorteios e o *Receiver* nas aplicações das cartelas. Sempre que um número novo é sorteado, um evento interno da aplicação de sorteio informa ao *Emitter* que uma mensagem equivalente àquele número deve ser emitida. As aplicações das cartelas aguardam até que uma mensagem contendo o novo número sorteado seja detectada pelo *Receiver*, assim que isso ocorre, um evento interno é disparado informando como a aplicação deve interpretar aquele número. Como um mesmo número pode aparecer em posições diferentes em cada

cartela, uma mesma mensagem pode disparar eventos diferentes nas aplicações. Uma mensagem contendo o número sorteado “22” pode disparar o evento “marque o número na primeira posição da cartela” em um aparelho, ou “marque o número na quinta posição da cartela” em outro, ou até mesmo “exiba uma mensagem de consolação” caso a cartela do aparelho não possua o número sorteado.

3.2.1 Transmissão e Recepção de Mensagens

A estratégia utilizada na *Audio Markings*, para transmissão e captura de mensagens em áudio, possui algumas similaridades com os trabalhos apresentados no capítulo 2. A estratégia de modulação pode ser comparada com a MFSK, adotada pelo Chirp e a biblioteca `sonicnet.js`, ou o DTMF do Google Tone. Nestes trabalhos, primeiro é definido um alfabeto contendo os caracteres usados para a transmissão dos dados, sendo que cada caractere será enviado em uma mensagem diferente. Também é definido um espectro de frequências, que é dividido em regiões, cada uma correspondendo a um caractere. Como exemplo, Calixto et al. [28] em seus experimentos com a `sonicnet.js` utilizam um espectro entre 18.5 e 19.5 KHz, com um alfabeto de dois caracteres, ‘h’ e ‘s’, conforme ilustrado na Figura 17.

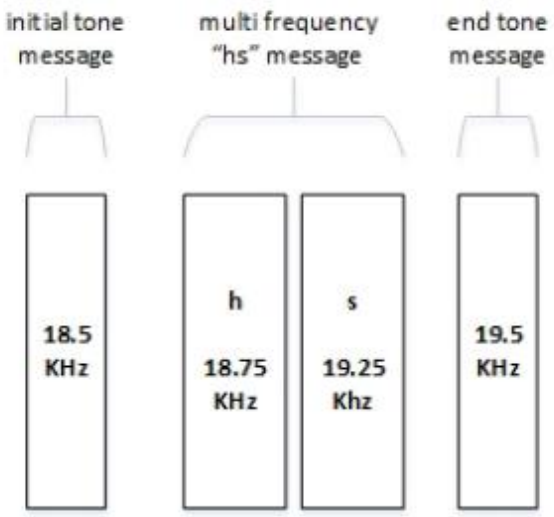

Figura 17. Exemplo usado nos experimentos de Calixto et al. [28]. Foi adotado um alfabeto de 2 letras, ‘h’ e ‘s’.

Para identificar qual caractere está sendo enviado em uma mensagem, o aparelho que capta o áudio deve realizar uma Transformada de Fourier no sinal e analisar o espectro definido. A partir daí, identifica-se qual das regiões possui o maior nível de intensidade, a fim de se descobrir qual dos caracteres foi enviado [28]. Note que com essa abordagem, o número

de mensagens diferentes é igual a *k*, onde *k* é o número de caracteres definidos pelo alfabeto da aplicação. Com essa estratégia, não é possível enviar dois caracteres ou mais em uma mesma mensagem, independentemente do tamanho do alfabeto ou do espectro definidos.

Buscando evitar tal limitação, a *Audio Markings* não analisa o espectro completo procurando pela região de maior intensidade. Nesse ponto a biblioteca utiliza uma estratégia similar à de multiplexação adotada pelo GUWMANET. A estratégia proposta neste trabalho também define um espectro e o divide em regiões, mas cada região é analisada independentemente, permitindo que mais de um caractere seja enviado em uma mensagem. Dessa forma, é possível aumentar o número de mensagens diferentes de *k* para 2^k, onde *k* é o número de caracteres definidos no alfabeto.

Separando o espectro e analisando cada região de forma independente, observamos que não é difícil codificar as informações em binário. Cada caractere pode representar um bit, a ordem dos bits no alfabeto representa a sua grandeza e a presença ou ausência de um caractere na mensagem cria o nosso conjunto de 0's e 1's. Como exemplo, em um alfabeto com 3 caracteres, 'a', 'b' e 'c', uma mensagem 'ac' apresenta apenas o primeiro e o terceiro caracteres do alfabeto, podendo ser representado como a mensagem 101 em código binário. A *Audio Markings* trabalha com mensagens codificadas utilizando essa lógica.

Por fim, para definir se um bit da mensagem possui valor 0 ou 1, junto com as regiões do espectro definidas para os bits da mensagem, também são definidas mais duas regiões extras. Essas regiões adicionais são utilizadas como referências, uma delas está constantemente com intensidade alta e a outra com intensidade constantemente baixa durante a transmissão da mensagem. Cada região do espectro usada para carregar os bits da mensagem é comparada com esses dois sinais extras. Se a região possui intensidade semelhante à do sinal com baixa intensidade, o mesmo representa um bit com valor 0; se for semelhante à do sinal com intensidade alta, o valor do bit é 1. A Figura 18 ilustra um exemplo da estratégia para uma mensagem de tamanho 8 bits.

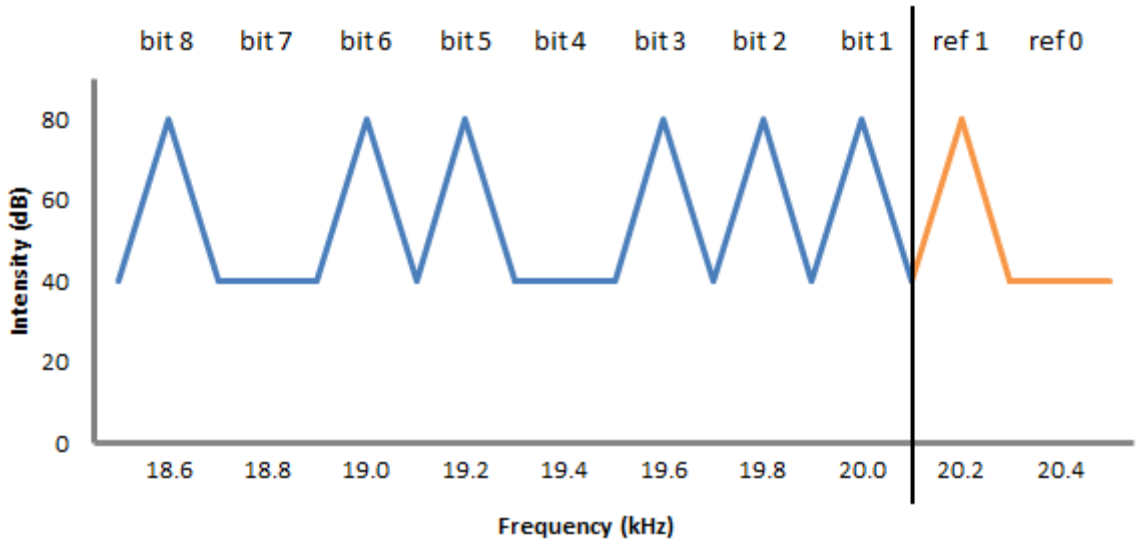

Figura 18. Exemplo de uma mensagem com o valor '10110111'. As frequências de 18.6 a 20 KHz carregam os bits que representam a mensagem, enquanto as frequências de 20.2 e 20.4 KHz são as referências para valor 1 e 0, respectivamente. Fonte: Produção do próprio autor.

Em termos mais técnicos, o sinal é gerado através da modulação por chaveamento de amplitude, também conhecida como ASK (*Amplitude Shift Keying*). Para enviar mais de uma informação por sinal (os múltiplos bits encontrados na mensagem) é realizada a multiplexação do mesmo, dividindo a informação entre as frequências da onda portadora. A estratégia de multiplexação entre as frequências é conhecida como FDM (*Frequency Division Multiplexing*) [10] e é similar à adotada pelo GUWMANET.

Diferente das outras formas de modulação apresentadas, a ASK faz uso de diferentes valores de amplitude para definir os valores transmitidos. Nesse trabalho é utilizada a forma mais simples da modulação por chaveamento de amplitude, conhecida como OOK (*On-Off Keying*), onde são utilizados apenas dois valores de amplitude para representar a informação a ser enviada [10].

3.2.2 Conversão e Análise dos Sinais de Áudio

Tanto a conversão de uma mensagem em binário para o seu equivalente em sinal acústico, quanto a análise de um sinal capturado para recuperar a informação transmitida, foram realizados com o uso das funcionalidades disponíveis na *Web Audio* API. O detalhamento de como essas funcionalidades foram empregadas e dos conceitos utilizados na conversão serão tratados na sequência do texto.

A geração de um sinal acústico é feita através do uso de um oscilador, produzindo um sinal em uma frequência definida, a sua frequência fundamental. Para que seja possível enviar um sinal em mais de uma frequência, é preciso aplicar a Síntese de Fourier, especificando quais harmônicos da frequência fundamental também devem ser emitidos.

Como exemplo da geração de um sinal em mais de uma frequência, a Figura 19 ilustra um caso onde se deseja enviar a informação ‘10110111’, utilizando um espectro de 100 a 800Hz. A frequência fundamental do oscilador é definida como 100Hz, o seu primeiro harmônico (que é a própria fundamental) será enviado, então é atribuído o valor 1 como intensidade, o segundo harmônico (200Hz) não será enviado, então possui valor 0, e assim os bits da mensagem vão sendo distribuídos entre o espectro do sinal. Observe que a *Web Audio* API trabalha com valores normalizados, entre 0 e 1, para a distribuição de intensidades entre os harmônicos de um sinal.

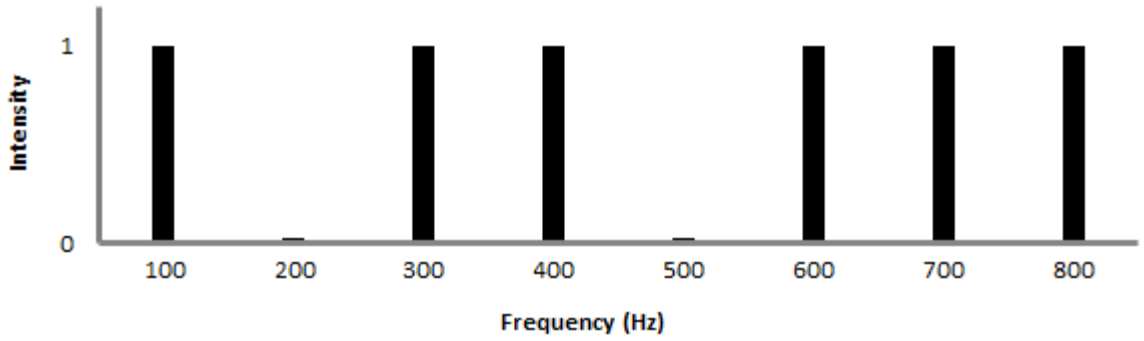

Figura 19. Exemplo da distribuição de intensidades entre os harmônicos de um sinal produzido por um oscilador. A mensagem gerada contém a informação ‘10110111’. Fonte: Produção do próprio autor.

No processo inverso, ou seja, na análise de um sinal capturado, é realizada a Análise de Fourier, onde é possível extrair a concentração de energia (as intensidades) nas frequências de um sinal. Para que seja possível identificar uma mensagem, é preciso que a aplicação que esteja analisando o sinal tenha a informação sobre qual espectro a mensagem se encontra.

Assumindo o mesmo cenário do exemplo de envio, onde a mensagem é ‘10110111’ e o espectro usado é de 100 a 800 Hz, a Figura 20 ilustra o sinal capturado. É possível observar que mesmo em frequências onde a intensidade foi definida como 0 durante a geração do sinal, a intensidade observada durante a análise do mesmo não é 0. Isso acontece pela presença de ruídos, que podem vir do ambiente, ou gerados nos próprios circuitos internos do microfone responsável pela captura do sinal sonoro.

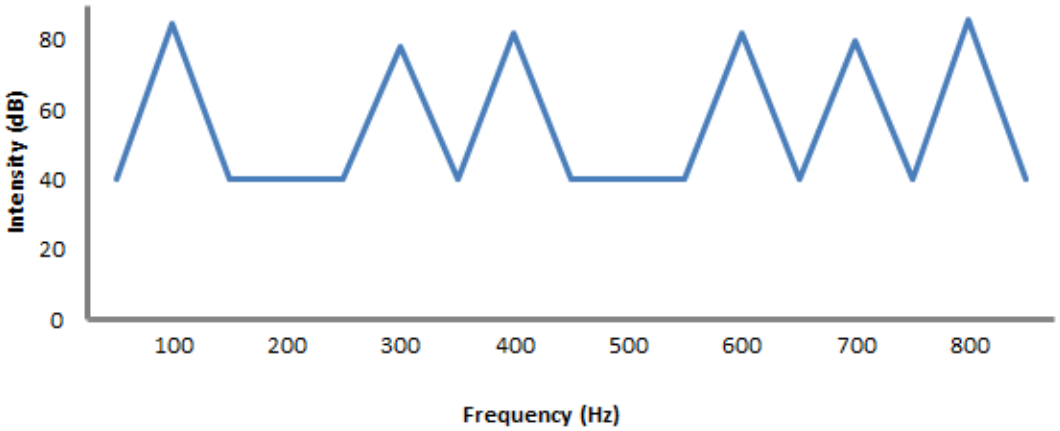

Figura 20. Exemplo do resultado da Análise Espectral de um sinal acústico. A mensagem detectada nesse sinal contém a informação ‘10110111’. Fonte: Produção do próprio autor.

A presença de ruídos não é o único fator que pode provocar problemas na transmissão e captura de sinais. Fatores como inconsistências no dispositivo de saída de áudio podem causar eventuais quedas no envio do sinal, fazendo com que sinal algum seja transmitido por alguns instantes. Outro problema é o fato de que os dispositivos para captura de áudio são, em geral, otimizados para trabalhar na faixa de frequências usadas na fala humana, em torno de 2 a 5.5 kHz, atenuando sinais fora dessa faixa. Buscando diminuir o impacto desses fatores, algumas estratégias foram aplicadas para melhorar a comunicação entre as aplicações.

A própria utilização de frequências de referência, para definir o que é um sinal equivalente a um bit 1 ou a um bit 0, foi a estratégia encontrada para amenizar tanto o problema da presença de ruídos, quando a atenuação de frequências sonoras fora da fala humana. Como diferentes microfones adicionam níveis diferentes de ruído e atenuação ao sinal capturado, assim como não é possível prever o ruído encontrado no ambiente, a designação de duas frequências de referência, que serão detectadas e analisadas da mesma forma que o resto do sinal, ajuda a solução a tomar a decisão sobre qual o valor que foi transmitido em cada frequência.

As inconsistências presentes tanto nos dispositivos de saída quando entrada de áudio geram flutuações nos sinais capturados, que podem provocar detecções erradas das mensagens. Essas flutuações podem ser causadas pela qualidade do microfone encontrado no aparelho que está recebendo o sinal, ou durante a própria transmissão.

Para contornar o problema anterior, uma lógica de *debouncing* foi implementada. Nesse caso, uma nova mensagem (isto é, um novo evento) precisa ser detectada (percebido) um número mínimo de vezes pelo receptor, antes de que seja efetuada uma mudança de estado na aplicação correspondente à mensagem (ao evento) recebida (percebido).

3.3 Plataforma Adotada

Segundo o W3C [35], uma aplicação Web pode ser definida como uma página Web, ou coleção de páginas, transmitidas via protocolo HTTP (*Hypertext Transfer Protocol*) [36] para um navegador Web, que consiga promover ao usuário uma experiência semelhante a de uma aplicação nativa [37].

De fato, uma tendência emergente no desenvolvimento Web são os *progressive Web apps* [38,39,40], aplicações que tentam emular de forma mais fiel ainda o funcionamento de aplicações tradicionais, mas mantendo o navegador como plataforma. Esses aplicativos são capazes de funcionar em modo *offline* e até mesmo esconder a interface do navegador onde estão sendo executados, tornando a experiência quase indistinguível de uma aplicação tradicional. Movimentos como esse levantam o seguinte questionamento:

> *Porque então criar uma aplicação Web em primeiro lugar, uma vez que um dos desafios dessa área é a tentativa de emular uma aplicação nativa?*

A resposta seria que quando uma aplicação Web é desenvolvida, em tese, essa única aplicação pode ser executada de forma semelhante em *desktops* ou dispositivos móveis de diferentes modelos e/ou fabricantes. Outra vantagem é a forma transparente com que esta aplicação é executada. O usuário pode desfrutar da solução sem ser questionado sobre instalações ou atualizações, uma vez que o processamento da aplicação é realizado pelo próprio navegador ou o servidor Web que presta o serviço.

Visando apresentar uma alternativa para comunicação entre aplicações desenvolvidas para navegadores Web, a biblioteca desenvolvida no contexto deste trabalho faz uso das linguagens e APIs nativas dessa plataforma. Desta forma, o entendimento sobre navegadores, assim como os recursos de desenvolvimento encontrados nos mesmos, se faz necessário para a compreensão de como a *Audio Markings* funciona.

3.3.1 Navegadores Web

Um navegador é um software que busca e apresenta conteúdo disponível na Web, comunicando-se com servidores usando o protocolo HTTP [41]. Um dos conteúdos comumente transmitidos dessa forma são documentos escritos em HTML [42], os quais são interpretados pelo navegador e apresentados como uma página ao usuário. Como já explicado, uma vez que essas páginas apresentem alguma forma de interação com o usuário e possuam funcionalidades semelhantes a uma aplicação *desktop*, nós podemos considerar que se trata de uma aplicação Web e não apenas de uma página Web.

Essa interação com o usuário é possível graças a APIs presentes nos navegadores. Como exemplo podemos citar a DOM [43], uma API para manipulação de documentos HTML. através da DOM é possível adicionar, remover e alterar elementos encontrados no documento HTML de uma página, alterando a visualização apresentada ao usuário. Alguns navegadores também disponibilizam funcionalidades e APIs exclusivas, como o `chrome.bookmarks` [44], presente no navegador da Google [20] e permite a manipulação da lista de favoritos do usuário.

Uma aplicação que é executada no navegador do usuário, e interage com as APIs presentes no mesmo, é conhecida como aplicação *client-side*. Em teoria essas aplicações podem ser desenvolvidas em qualquer linguagem, mas como a grande maioria dos navegadores apresentam interpretadores e APIs apenas para a linguagem *JavaScript* [45], a maior parte das bibliotecas (incluindo a *Audio Markings*) e aplicações *client-side* são desenvolvidas em *JavaScript*.

Enquanto a presença de APIs exclusivas em cada navegador pode ser interessante para atrair desenvolvedores e usuários para um fabricante em especial, é importante que as principais funcionalidades de um navegador trabalhem de forma semelhante, para evitar que seja necessário o desenvolvimento de várias versões de uma mesma aplicação Web. Atualmente, a organização responsável por tentar manter essa padronização é o W3C. O W3C não só tem preocupações com padrões de APIs importantes, como a própria DOM, mas com quase todos os aspectos envolvidos na Web em geral, desde URLs [46] à formatação de documentos HTML [42].

3.3.2 O Padrão HTML5

O HTML5 [42] é, primariamente, a quinta revisão publicada pela W3C de especificações para a linguagem HTML. Mas talvez mais interessante do que a adição de novas *tags* para documentos HTML é a quantidade de novas APIs, funcionalidades e padronizações que foram (e ainda estão sendo) incluídas com a nova revisão. Com a gradual adoção do padrão pelos fabricantes de navegadores Web, a tendência é que o uso de *plug-ins*, como *Flash* [47] e *SilverLight* [48], seja completamente eliminado com o tempo, uma vez que o novo padrão provê quase todas as funcionalidades que essas extensões agregavam.

Alguns recursos importantes presentes no HTML5 são o *Service Workers* [49, 50] e o *Web App Manifest* [51], ambos utilizados nas provas de conceito apresentadas no capítulo 4 deste trabalho, são fundamentais para a criação dos já mencionados *progressive Web apps*, permitindo que esses aplicativos trabalhem em modo *offline* e escondam a interface do navegador.

No contexto da biblioteca desenvolvida nesse trabalho, o HTML5 foi importante principalmente por padronizar o acesso do navegador aos dispositivos de entrada e saída de áudio encontrados nos aparelhos, através das APIs *MediaStream* [24] e *Web Audio* [25], respectivamente.

A Figura 21 ilustra como a *Audio Markings* acessa os dispositivos de áudio do aparelho através dessas APIs. A *MediaStream* cuida do acesso ao microfone do aparelho e de parte do processo de conversão do sinal acústico para o meio digital (ADC), se encarregando de realizar operações como amostragem e quantização do sinal. A *Web Audio* API é a responsável pelo acesso aos alto-falantes do aparelho, realizando também parte do processo de conversão do sinal digital para o seu equivalente acústico (DAC).

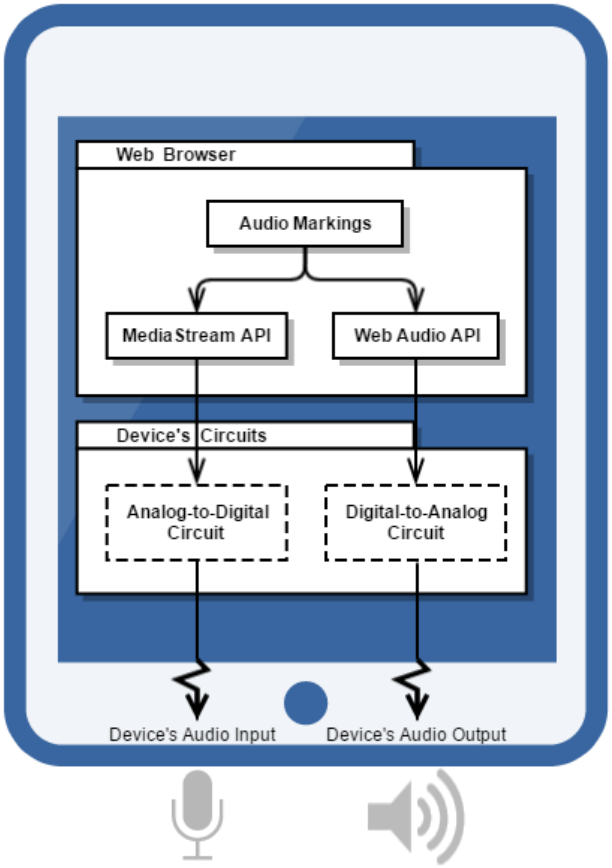

Figura 21. Acesso da *Audio Markings* aos dispositivos de áudio presentes em um aparelho, através das APIs padronizadas pela W3C. Fonte: Produção do próprio autor.

O HTML5 também trouxe funcionalidades para sintetização e análise de sinais acústicos para os navegadores, essas operações são disponibilizadas através da *Web Audio* API, que abstrai processos mais complexos e computacionalmente pesados, como a Análise e a Síntese de Fourier.

Anterior ao surgimento dessas APIs para acesso aos dispositivos de áudio dos aparelhos, a única maneira de emitir ou capturar sons em aplicações Web era através de *plug-ins* e extensões para navegadores. Dentre esses *plug-ins*, o mais famoso foi o *Flash*, da Adobe, presente em aplicações como o YouTube. O *Flash* chegou ser tão amplamente utilizado, ao ponto de vir instalado por padrão em navegadores como o Internet Explorer da Microsoft, responsável também pelo desenvolvimento do *plug-in* concorrente do *Flash*, o *SilverLight*.

Com a popularização desses *plug-ins*, ficou clara a demanda por recursos para permitir o acesso de aplicações Web aos dispositivos de entrada de saída de áudio dos aparelhos. Dessa forma, a padronização de formas nativas para esse acesso em navegadores, além da disponibilização de ferramentas para análise e sintetização de áudio, foram precupações da W3C ao definir os recursos que seriam disponibilizados com a vinda do HTML5.

3.4 Arquitetura da *Audio Markings*

A *Audio Markings* se apoia sobre duas APIs para trabalhar com áudio em navegadores, a *MediaStream* [24] e a *Web Audio* [25]. A Figura 22 ilustra a arquitetura da *Audio Markings*, assim como uma versão simplificada da arquitetura dessas APIs (apenas as classes que a biblioteca faz uso). Apesar de fazer uso desses recursos, a biblioteca é independente o suficiente para que caso essas APIs sejam descontinuadas, modificadas (como já ocorreu durante o desenvolvimento) ou que surja uma alternativa melhor, será possível a alteração da mesma sem que a interface com os desenvolvedores precise ser modificada.

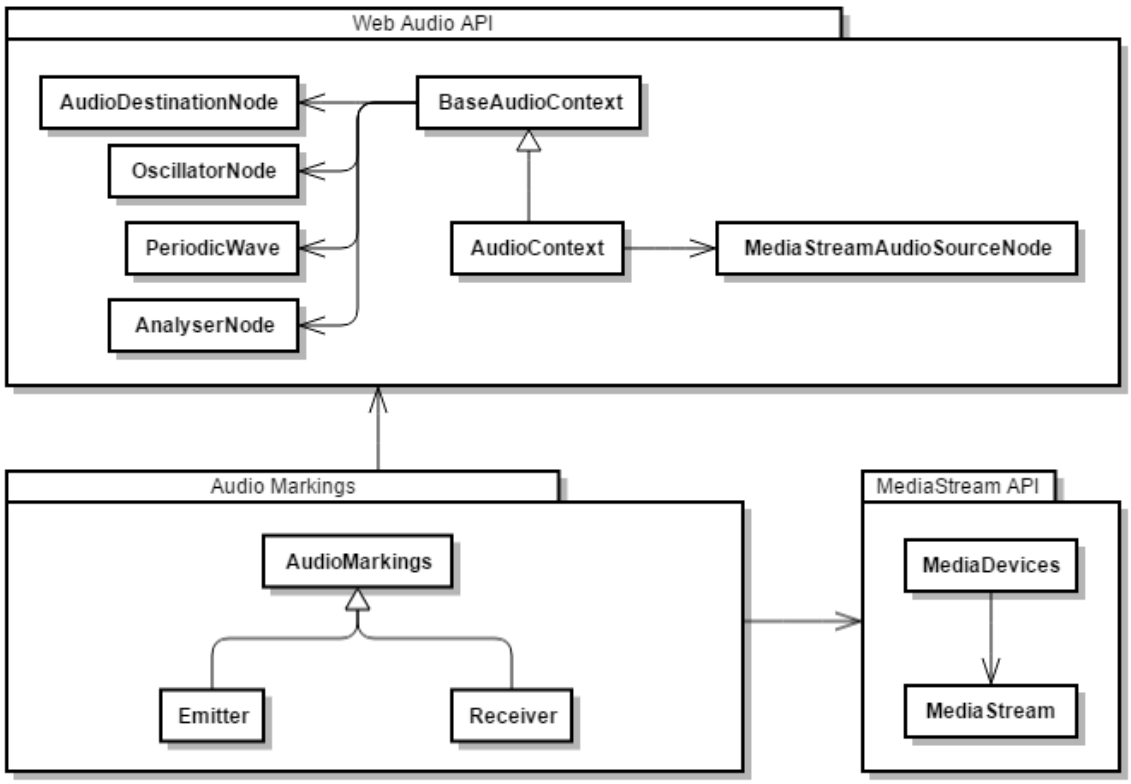

Figura 22. Arquitetura geral da *Audio Markings* e as APIs utilizadas. Fonte: Produção do próprio autor.

Conforme discutido, a *Audio Markings* é dividida em *Emitter* e *Receiver*, cada um desses componentes é uma classe que pode ser instanciada em um objeto. Como ambos fazem uso de alguns recursos em comum, existe uma terceira classe, a *AudioMarkings*, da qual as duas herdam. A *AudioMarkings* faz uso apenas da classe *AudioContext*, da *Web Audio* API. A *AudioContext* é utilizada na criação e administração de quase todos os recursos usados pelo resto da biblioteca, por isso a classe *AudioMarkings* se encarrega da inicialização ou reaproveitamento da mesma.

Assim como a *AudioMarkings*, o *Emitter* também faz uso apenas da *Web Audio* API. A classe *OscillatorNode* é responsável pela criação e manipulação de osciladores, utilizados para a geração dos sinais de áudio. Quando é necessário gerar sinais mais complexos, com informações em múltiplas frequências, é utilizada a classe *PeriodicWave* em conjunto com os osciladores. Por fim, a classe *AudioDestinationNode* é a interface entre a biblioteca e as saídas de áudio presentes nos aparelhos.

O *Receiver* utiliza as duas APIs, a *MediaStream* para a captura de sinais acústicos através do microfone e a *Web Audio* na análise desses sinais. Durante a captura do áudio, a classe *MediaDevices* serve como a interface entre a biblioteca e o microfone presente no aparelho, gerando um *stream* de áudio através da classe *MediaStream*. Como esse *stream* serve tanto para sinais de áudio quanto para vídeo, a classe *MediaStreamAudioSourceNode* é utilizada para extrair apenas o sinal de áudio. Uma vez extraído esse sinal, a análise espectral do mesmo é realizada através da classe *AnalyserNode*.

A seguir as classes da biblioteca serão detalhadas de forma mais específica, com informações sobre como as mesmas podem ser instanciadas e como utilizar os métodos disponíveis.

3.4.1 Classe *AudioMarkings*

Durante a sua inicialização, a *AudioMarkings* pode receber como parâmetro um *AudioContext*. Caso nenhum seja fornecido, a própria classe se encarrega da criação de um durante a sua inicialização.

Como já citado, o *AudioContext* é um recurso da *Web Audio* API e é empregado em quase todas as etapas no trabalho com áudio em navegadores. Por se tratar de um recurso pesado, é recomendável que apenas uma instância do mesmo seja usada por toda a aplicação,

mesmo que para finalidades diferentes [7,25]. Por essa razão a biblioteca permite a reutilização do *AudioContext*.

No geral, a classe *AudioMarkings* serve apenas para a inicialização dos recursos que o *Emitter* e o *Receiver* fazem uso. Mas caso uma mesma aplicação faça uso tanto do *Emitter* quanto do *Receiver*, essa classe pode ser diretamente instanciada, como forma de centralizar o uso do *AudioContext*.

3.4.2 Classe *Emitter*

O *Emitter* possui funcionalidades para alterar, iniciar e finalizar a transmissão de mensagens. Uma aplicação que faça uso do *Emitter* aguarda até que algum evento interno demande a transmissão de uma mensagem. Uma vez iniciada a transmissão, o *Emitter* enviará a mesma mensagem constantemente, até que essa seja alterada ou finalizada. Utilizando as configurações iniciais, é possível enviar 2^8 mensagens diferentes.

Assim como a *AudioMarkings*, a classe *Emitter* pode ser inicializada recebendo um *AudioContext* como parâmetro, caso nenhum seja fornecido a própria classe cria um durante a sua inicialização.

As funcionalidades da classe *Emitter* são:

- ***Initialize***: Permite mudar as frequências usadas pela aplicação. Os parâmetros são a frequência inicial (o começo do espectro utilizado), o intervalo entre as frequências usadas e a quantidade de bits que se deseja ter em uma mensagem. Por padrão, esses valores são 18600, 200 e 8, respectivamente, resultando em uma faixa de 18.6 a 20KHz, conforme o visto na Figura 18.
- ***SetMessage***: Especifica a mensagem a ser emitida. Como a *Audio Markings* trabalha com mensagens em números, o parâmetro de entrada é o número da mensagem que se deseja enviar. Em um exemplo onde são utilizados 8 bits para a mensagem, as mensagens vão de 00000000 (0) a 11111111 (255).
- ***Start***: Inicia a transmissão de mensagens; se nenhuma mensagem for definida, o valor 00000000 (0) é transmitido por padrão.
- ***Stop***: Interrompe a transmissão de mensagens. Não é necessário interromper a transmissão para trocar a mensagem a ser enviada; o *SetMessage* pode ser usado múltiplas vezes durante uma mesma transmissão.

3.4.3 Classe *Receiver*

O *Receiver* cuida da rotina de checagem dos sinais de áudio detectados pelo microfone do aparelho. Ao ser inicializado, é possível adicionar e remover eventos para mensagens específicas, ou ser notificado cada vez que o aparelho detectar uma mensagem nova. O *Receiver* possui uma rotina interna de checagem de mensagens, notificando a aplicação através da chamada de eventos.

Cada *Receiver* deve ser inicializado com um método para *callback* em caso de sucesso e outro para o caso de falha na aquisição do microfone do aparelho, uma vez que a requisição do microfone é uma operação assíncrona. Sempre que uma aplicação Web requer o uso do microfone, o usuário deve ser questionado se permite ou não que a aplicação tenha acesso [24]. No caso de algum erro durante a aquisição do microfone ou se o usuário negar o acesso, o método de erro será chamado, informando a fonte do problema. Assim como o *Emitter*, também é opcional fornecer um *AudioContext* durante a criação do *Receiver*.

As funcionalidades da classe *Receiver* são:

- ***Initialize***: Utilizado caso seja necessário trocar as frequências usadas pelo *Receiver*. Os parâmetros de entrada são um *array* contendo as frequências que se deve observar para a mensagem e as frequências de referência para sinal enviado e não enviado. Por padrão, os valores são 18600, 18800, 19000, 19200, 19400, 19600, 19800 e 20000 para a mensagem e 20200 e 20400 para as referências de sinal enviado e não enviado, respectivamente.
- ***AddMessageEvent***: Utilizado para adicionar um evento para um valor de mensagem específico. Sempre que o *Receiver* detecta uma mudança de mensagem, todos os eventos associados a essa mensagem são invocados de forma assíncrona. Os parâmetros são a mensagem (numérica) e a função a ser invocada.
- ***RemoveMessageEvent***: Utilizado para remover eventos associados a mensagens específicas. Os parâmetros são a mensagem associada e a função que seria invocada.
- ***OnChangeMessage***: Chamado sempre que o *Receiver* detecta uma mensagem diferente. Pode ser associado a uma função da aplicação para que a mesma seja sempre notificada sobre qualquer mensagem detectada.

- ***CheckMessage***: Utilizado durante a rotina de verificação que o próprio *Receiver* realiza para disparar os eventos associados a mensagens; ele pode ser utilizado caso o desenvolvedor prefira criar a sua própria rotina de checagem de mensagens. Quando chamado, verifica imediatamente qual a mensagem que está sendo transmitida, retornando seu número.
- ***GetIntensityValues***: Recebe um conjunto de valores de frequência e devolve a intensidade (em dB) detectada em cada um deles; também é utilizado nas rotinas internas do *Receiver* e pode ser útil para detectar níveis de ruído, proximidade e qualidade do sinal, ou até mesmo para detectar mensagens em outros padrões além dos definidos pela *Audio Markings*.

3.5 Detalhes de Implementação

A *Audio Markings* está configurada inicialmente para utilizar um espectro de 18.6 a 20.4 KHz para a comunicação, reservando as frequências de 18.6 a 20 KHz para a mensagem e 20.2 e 20.4 KHz como as referências para valor de bit, conforme ilustrado Figura 18. Tais valores foram definidos por serem considerados em outros trabalhos como faixas inaudíveis ou semi-inaudíveis ao ser humano [14,27,28]. Entretanto, esses valores podem ser configurados conforme necessidade da aplicação.

A biblioteca foi desenvolvida usando a linguagem *JavaScript* [45], para ser usada em navegadores que suportem os recursos do HTML5, em especial as APIs *MediaStream* [24] e *Web Audio* [25]. Devido a restrições de segurança da *MediaStream* API, sempre que um *Receiver* for inicializado o navegador irá questionar o usuário se o mesmo permite ou não o acesso ao microfone do aparelho, além disso, apenas aplicações hospedadas em servidores que usem o protocolo de transmissão seguro (HTTPS) podem requisitar o uso do microfone através do navegador [24].

Durante o desenvolvimento, a maior parte dos testes foram conduzidos usando o navegador Chrome [20], da Google, por ter sido um dos primeiros navegadores a implementar as APIs de áudio necessárias para esse trabalho, além de ser o mais presente em aparelhos *mobile*. Os testes que não necessitavam o uso dessas APIs, como controle de eventos e cálculos internos, foram conduzidos utilizando apenas o interpretador *JavaScript* do Node.js [52], por ser uma forma mais rápida de observar os resultados.

Para a realização dos testes com o *Receiver*, que necessitavam do uso do microfone do aparelho, foi desenvolvido um mini servidor HTTPS em Node.js, utilizado em uma rede local. Em casos onde o uso da rede local não era possível, as aplicações foram hospedadas em servidores externos, que ofereciam o serviço de HTTPS, como GitHub, DropBox e Amazon Web Services.

3.5.1 Ambiente de Testes Controlados

Por ser uma biblioteca que faz uso dos dispositivos de entrada e saída de áudio dos aparelhos, a *Audio Markings* e seus casos de uso se tornaram soluções difíceis de serem testadas, especialmente se comparados a outras aplicações Web. A dificuldade aqui reside no fato de que são necessários no mínimo dois aparelhos, para desempenharem os papeis de emissor ou receptor de mensagens, em vários testes. Para contornar o problema, foi desenvolvida uma estratégia para simular a presença de um *Emitter* ou um *Receiver* em uma mesma aplicação.

Pela própria natureza da *Web Audio* API, todas as suas classes que possuem o sufixo *Node* podem ser conectadas entre si, conforme observado na Figura 23. É dessa maneira que a biblioteca emite e recebe o áudio através dos dispositivos de entrada de saída dos aparelhos, através das classes *AudioDestinationNode* e *MediaStreamAudioSourceNode*, conectadas a osciladores e analisadores de áudio.

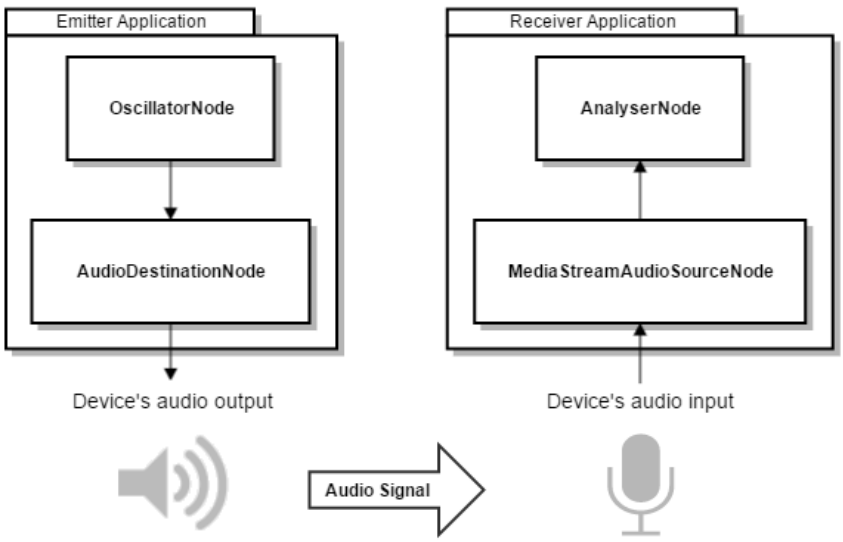

Figura 23. Comunicação por áudio entre uma aplicação emissora e uma receptora. Em um ambiente real. Fonte: Produção do próprio autor.

Através dessa arquitetura é possível realizar uma ligação direta entre um *Emitter* e um *Receiver*, conectando a classe *OscillatorNode*, presente no *Emitter*, ao *AnalyserNode* presente no *Receiver*, conforme ilustrado na Figura 24. Dessa forma, não é necessária a utilização ou presença de dispositivos de áudio para realizar testes, uma vez que o *Emitter* estaria se comunicando diretamente com o *Receiver*.

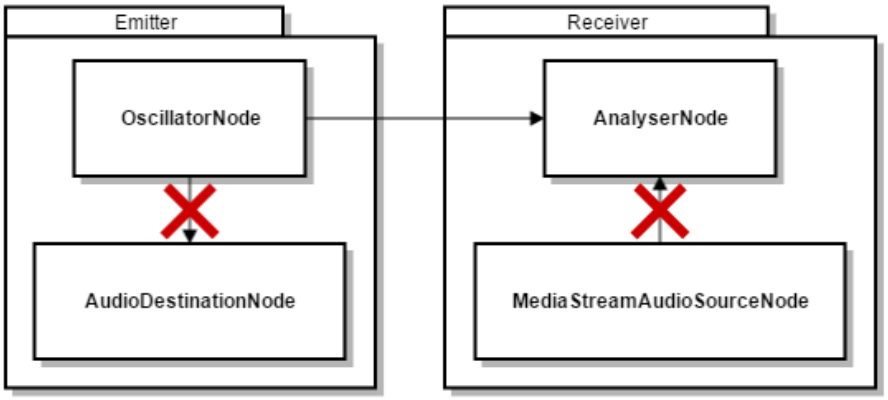

Figura 24. Comunicação por sinais de áudio entre um objeto da classe *Emitter* e um *Receiver*. Em um ambiente controlado. Fonte: Produção do próprio autor.

Utilizando a estratégia de conectar esses dois componentes é possível agilizar consideravelmente o desenvolvimento e validação de uma aplicação. No entanto, a estratégia traz algumas restrições. Ela cria um ambiente altamente controlado, onde um sinal gerado é diretamente analisado, sem sofrer os efeitos encontrado em ambientes reais, como atenuação de faixas de frequências muito altas, presença de ruído vindo do ambiente ou gerado pelos circuitos internos dos aparelhos. Dessa forma, é imperativo que durante o desenvolvimento de aplicações sejam realizados testes em ambientes reais, para validar se a aplicação irá de fato se comportar da forma esperada.

A estratégia apresentada é ideal para a realização de testes simples, como verificar se uma aplicação está respondendo adequadamente a eventos disparados pela *Audio Markings*, ou se a estratégia de comunicação adotada atende ou não às necessidades da aplicação. Os testes unitários aplicados durante todo o desenvolvimento da biblioteca e dos casos de uso foram todos realizados dessa forma. Mas, como já mencionado, essa estratégia não apresenta um substituto para os testes em ambientes reais e, dessa forma, durante todo o desenvolvimento da biblioteca e seus casos de uso, foram realizados também testes utilizando aparelhos e ambientes diversos, afim de garantir o funcionamento da *Audio Markings*.

Capítulo

4

Avaliações e Resultados

Nesse capítulo são apresentados os resultados dos testes e experimentos realizados sobre a *Audio Markings*, envolvendo a comunicação entre aplicações através do áudio. Primeiro serão apresentados os testes realizados sobre o desempenho da biblioteca, avaliando a sua taxa de erro na transmissão de mensagens e o impacto da distância entre os aparelhos na identificação correta dessas mensagens. Em seguida serão apresentadas três implementações do tipo "Prova de Conceito" (*Proof of Concept*, ou PoC) que ilustram como a biblioteca pode ser empregada no desenvolvimento de aplicações Web, em casos reais onde a sincronização local é desejada.

Neste trabalho, consideramos que o termo sincronização local se refere especificamente à sincronização temporal dos conteúdos apresentados por um conjunto distribuído de dispositivos fisicamente próximos. Em outras palavras, não há nenhum tratamento relacionado ao tempo de transmissão destes conteúdos. Um controle dos atrasos de apresentação dos conteúdos em cada dispositivo permitirá a sincronização temporal da apresentação distribuída como um todo.

4.1 Testes de Precisão

Afim de analisar a precisão durante a transmissão de mensagens entre soluções que façam uso da *Audio Markings*, foram desenvolvidas duas aplicações para a condução de testes. Uma aplicação é responsável por transmitir mensagens, utilizando sinais de áudio, enquanto uma segunda aplicação deve capturar esses sinais e recuperar corretamente as mensagens enviadas.

As duas aplicações foram desenvolvidas utilizando a biblioteca *Audio Markings*. Buscando avaliar também o impacto da estratégia de *debounce*, empregada no

desenvolvimento da biblioteca para melhorar a robustez na detecção de mensagens, todos os testes foram realizados uma vez com o recurso ligado e outra com o mesmo desabilitado.

A aplicação responsável por emitir as mensagens foi desenvolvida utilizando apenas o *Emitter* da *Audio Markings*, para criar e emitir mensagens em áudio. Através dessa aplicação é possível iniciar e interromper o envio de mensagens, assim como trocar a mensagem a ser enviada. Por ser uma aplicação simples, essa solução foi desenvolvida sem interface com o usuário, sendo controlada através do *console* presente nos navegadores.

A aplicação encarregada de detectar as mensagens foi desenvolvida empregando uma versão modificada da classe *Receiver*, para permitir que o recurso de *debounce* pudesse ser desligado ou religado, conforme a necessidade do teste. Quando acionada, a aplicação começa a identificar qualquer mensagem que esteja sendo transmitida nas proximidades, durante um período de tempo determinado. Ao final da sua execução, são exibidas todas as mensagens detectadas e durante quanto tempo cada uma foi identificada.

Sabendo de antemão quais mensagens serão transmitidas e analisando os resultados obtidos através das aplicações de detecção, é possível identificar o nível de acerto alcançado pela estratégia de comunicação implementada.

4.1.1 Condução dos Experimentos

A aplicação para transmissão das mensagens foi executada em um *notebook*, através de um navegador Chrome versão 50.0.2661. As mensagens foram emitidas utilizando uma caixa de som JBL Flip, conectada ao *notebook*.

A aplicação para detecção das mensagens foi executada em dois aparelhos móveis, um *smartphone* Nexus 4 e um *tablet* Nexus 7, ambos através de um navegador Chrome versão 50.0.2661. As mensagens foram detectadas utilizando os próprios microfones presentes nos aparelhos.

Os testes foram aplicados com os aparelhos *mobile* posicionados a uma distância inicial de 1 metro da fonte de áudio, essa distância foi aumentada gradualmente, até um total de 5 metros de distância. Todos os testes foram executados utilizando as mensagens 00000000 (0) e 11111111 (255), a motivação para a escolha desses valores será explicada na seção seguinte. As aplicações escutam os sinais transmitidos durante um período de 10

minutos, ao final da detecção é calculada a porcentagem desse tempo em que a mensagem esperada foi detectada pela solução.

4.1.2 Resultados dos Testes

Os resultados desse experimento podem ser observados nas tabelas 2 e 3. Como os aparelhos apresentaram análises consideravelmente diferentes, os resultados dos mesmos foram divididos entre duas tabelas, uma para o smartphone Nexus 4 e uma para o tablet Nexus 7. As tabelas são divididas entre resultados com a utilização ou não da estratégia de debounce, comparando o nível de acerto na detecção das mensagens em relação à distância entre os aparelhos e a fonte de áudio.

Tabela 2. Resultados dos testes de precisão (%) na detecção de mensagens a distâncias (m) variadas. Utilizando um *tablet* Nexus 7.

	Com *debounce*		Sem *debounce*	
Dist./Msg	0	255	0	255
1m	**100 %**	**96.33 %**	**100 %**	**93.68 %**
2m	**100 %**	**96.38 %**	**100 %**	**87.16 %**
3m	**100 %**	**93.45 %**	**100 %**	**94.57 %**
4m	**100 %**	**96.28 %**	**100 %**	**88.82 %**
5m	**100 %**	**94.2 %**	**100 %**	**80.83 %**

Tabela 3. Resultados dos testes de precisão (%) na detecção de mensagens a distâncias (m) variadas. Utilizando um *smartphone* Nexus 4.

	Com *debounce*		Sem *debounce*	
Dist./Msg	0	255	0	255
1m	**100 %**	**98.74 %**	**99.44 %**	**94.2 %**
2m	**100 %**	**97.66 %**	**100 %**	**73.26 %**
3m	**98.85 %**	**100 %**	**99.44 %**	**75.97 %**
4m	**98.61 %**	**97.9 %**	**98.22 %**	**86.95 %**
5m	**98.81 %**	**96.23 %**	**94.31 %**	**80.1 %**

Conforme observado nas tabelas, na presença da estratégia de *debounce*, o *tablet* Nexus 7 parece ter maior precisão para mensagens mais simples (0), mas tem um desempenho inferior para mensagens maiores (255) em comparação com o *smartphone* Nexus 4. Sem a lógica de *debounce*, o Nexus 7 parece ter um desempenho melhor no geral em comparação com o Nexus 4. Acreditamos que essa diferença observada no desempenho entre os aparelhos

se dê, em grande parte, pela qualidade dos microfones presentes nesses dispositivos, mesmo sendo de marcas iguais.

Note também que ocorre uma diminuição no nível de acerto quando o sinal usa muitos harmônicos no processo de sintetização. É possível notar esse efeito comparando os resultados obtidos entre as mensagens. A mensagem 0 faz uso de apenas 1 harmônico no sinal transmitido, enquanto para gerar o sinal da mensagem 255, a biblioteca precisa empregar 9 harmônicos no processo de geração do sinal.

A escolha das mensagens utilizadas no experimento, 0 e 255, se deu justamente para avaliar a relação entre a quantidade de harmônicos utilizados na geração do sinal, e o nível de acerto na detecção do mesmo. A mensagem 0 é consideravelmente mais simples de se detectar do que a mensagem 255, uma vez que um aumento no número de harmônicos utilizados na geração do sinal resulta na diminuição da amplitude geral do mesmo [9]. A consequência direta dessa diminuição da amplitude é o aumento da dificuldade na detecção correta da mensagem.

Não tão impactante quanto ao número de harmônicos no sinal, é possível notar que a distância entre os aparelhos e a fonte de áudio também influenciou os resultados. Isso já era esperado, uma vez que ondas acústicas são atenuadas com a distância [2,14].

Os resultados evidenciam também que a lógica de *debounce* contribui significativamente na precisão com que a solução detecta mensagens. De fato, é possível notar que ela contribui mais na detecção de sinais contendo um número maior de harmônicos do que em sinais mais simples.

Por fim, acreditamos que a estratégia de *debounce* seja importante para qualquer solução que busque realizar comunicação através de sinais de áudio, uma vez que em nenhum dos casos observados durante o experimento houve precisão menor que 90%, enquanto em alguns casos sem o recurso os resultados obtiveram taxas abaixo de 75%.

4.2 Aplicações Provas de Conceito

Esta seção discute as três aplicações Provas de Conceito (PoC) com o uso da *Audio Markings*, com o objetivo de verificar a sua viabilidade e avaliar se a mesma pode ser usada em situações reais envolvendo sincronização local em aplicações Web.

Para facilitar a compreensão dos papeis das aplicações nas provas de conceito, estabelecemos os seguintes termos no contexto desta seção:

- Aplicação principal ou emissora: Responsável por transmitir as mensagens, orquestrando a interação para as demais aplicações.
- Aplicação secundária ou receptora: Responsável por capturar as mensagens transmitidas pela aplicação emissora, apresentando algum conteúdo que esteja sincronizado ou de acordo com o que está sendo exibido na aplicação principal.

Os experimentos foram conduzidos em um ambiente isolado, com os aparelhos receptores a uma distância de menos de 1 metro do emissor. A infraestrutura para os experimentos consistiu de dois *notebooks*, um *smartphone* Nexus 4, um *tablet* Nexus 7, todos rodando as aplicações em um navegador Chrome, versão 50.0.2661; também utilizamos uma caixa de som JBL Flip e um microfone portátil Clone para os *notebooks*.

4.2.1 Cores Sincronizadas

Nessa PoC buscamos utilizar a percepção visual da mudança de cores para demonstrar o nível de sincronização que pode ser alcançado com a comunicação por áudio entre os dispositivos. Esse experimento consiste em duas aplicações HTML, cada uma implementando uma das classes principais da *Audio Markings*, *Emitter* e *Receiver*. A aplicação principal permite que o usuário escolha uma entre 127 cores disponíveis, a qual deve ser simultaneamente apresentada por todos os dispositivos receptores que estejam escutando. A Figura 25 ilustra o cenário das aplicações.

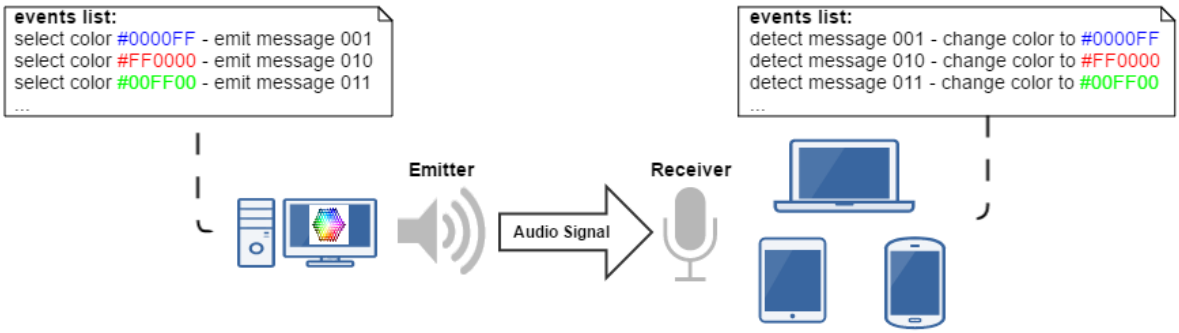

Figura 25. Esquema das aplicações que compõe a PoC “Cores Sincronizadas”. Fonte: Produção do próprio autor.

A aplicação emissora consiste em uma página HTML com um mosaico de quadrados de cores distintas. Essa aplicação implementa o *Emitter* para transmitir a cor selecionada. Assim que o usuário escolhe uma cor, a aplicação emite o sinal contendo a mensagem sobre qual cor as demais aplicações receptoras devem apresentar.

As aplicações receptoras implementam o *Receiver*. Elas ficam constantemente escutando os sinais emitidos pela aplicação principal, mudando de cor conforme as escolhas do usuário. Ao invés de utilizarem mensagens contendo a codificação RGB das cores a serem transmitidas, as duas aplicações contêm uma lista interna de todas as cores usadas, numeradas de 1 a 127, na qual apenas o índice da cor selecionada que é transmitido entre as aplicações.

As duas aplicações utilizam as frequências padrões definidas pela *Audio Markings*. A aplicação emissora também pode exibir em forma de QR Code a URL para a aplicação receptora, para facilitar o acesso ao endereço da mesma.

O código desenvolvido pode ser acessado através de um repositório aberto[3], que também contém links para as aplicações Web e um vídeo demonstrando a PoC em ação. A Figura 26 mostra uma imagem da execução da PoC, o notebook à esquerda está executando a aplicação emissora, enquanto os demais dispositivos estão executando as aplicações receptoras.

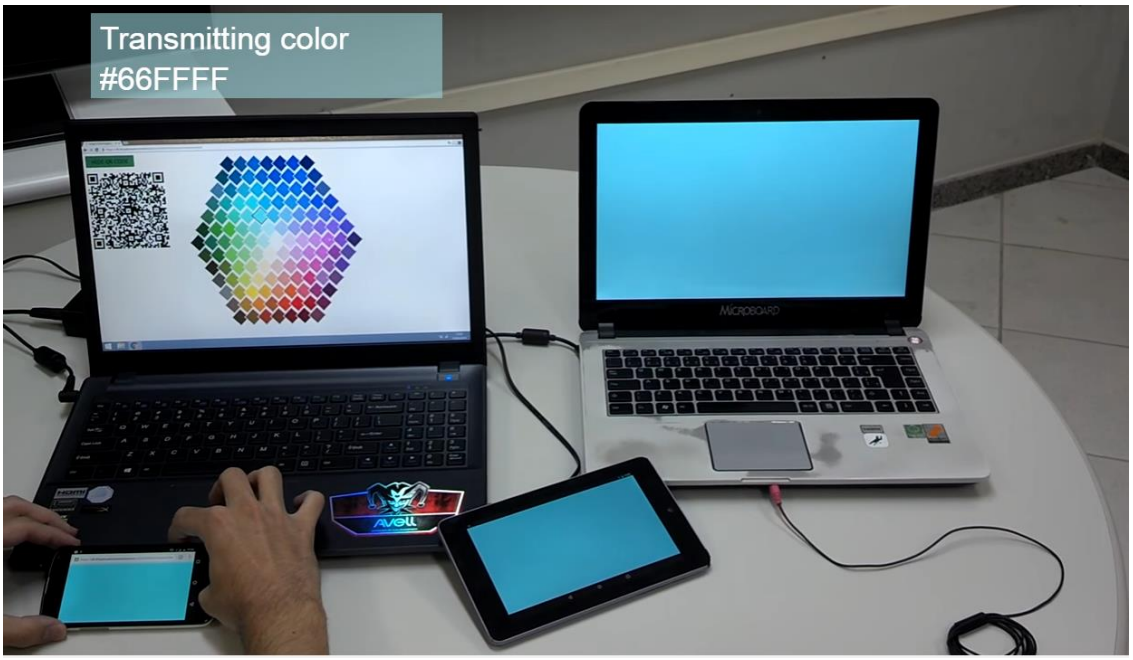

Figura 26. Prova de conceito Cores Sincronizadas. Fonte: Produção do próprio autor.

[3] Endereço de acesso: https://github.com/jonkoala/synchronized-colors

4.2.2 Performance Musical Distribuída

Essa PoC consiste no uso de um grupo de aparelhos para tocar uma música em conjunto, cada aparelho sendo responsável por tocar um dos instrumentos da composição. Assim, para que a música faça sentido aos ouvidos dos usuários, é necessário que os dispositivos reproduzam seus trechos e instrumentos de forma sincronizada.

Para permitir que cada aparelho toque um instrumento diferente da composição, a música original foi dividida em vários arquivos mp3, cada um contendo a parte correspondente a um instrumento da composição original. Além dos aparelhos que irão tocar cada um dos instrumentos, um aparelho foi designado como o ponto de sincronização.

A aplicação principal é o ponto de sincronização para as demais aplicações receptoras. Ela funciona como um maestro e informa a cada um dos outros aparelhos em qual instante da música a performance se encontra. Dessa forma é possível também que um usuário interagindo com o emissor possa alterar o instante da música a qualquer momento, e os demais aparelhos devem se ajustar para acompanhar o maestro.

A Figura 27 ilustra o cenário dessa aplicação. A cada 2 segundos o emissor transmite um sinal informando em qual instante a música se encontra. Os aparelhos receptores ficam constantemente escutando o sinal emitido. Ao detectar uma nova mensagem, cada aparelho verifica o quão adiantado ou atrasado ele se encontra com relação ao ponto de sincronização e se ajusta de acordo. Nesse experimento cada aparelho se ajusta caso esteja atrasado ou adiantado em mais de 0.5 segundos em relação ao emissor.

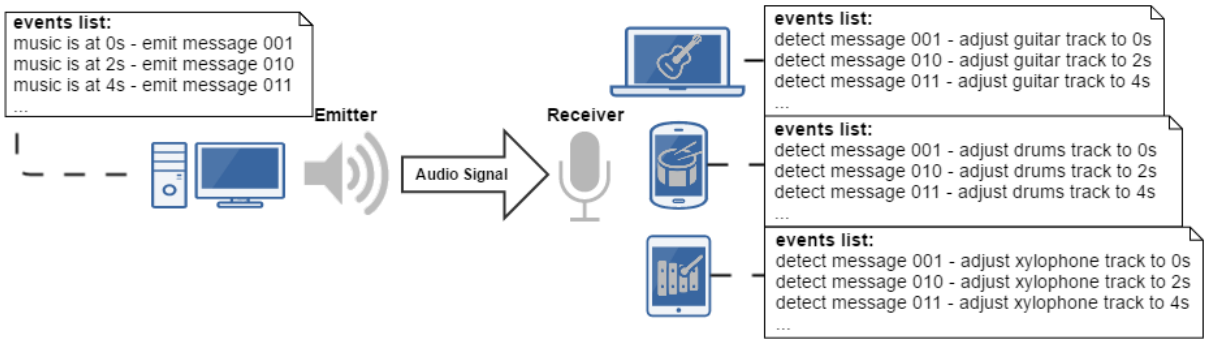

Figura 27. Esquema das aplicações que compõe a PoC “Performance Musical Distribuída”. Fonte: Produção do próprio autor.

A aplicação principal foi desenvolvida usando o *Emitter*, nas frequências padrões da *Audio Markings*. Junto com os sinais informando os instantes em que a música se encontra, mais dois sinais foram definidos: 11111111 (255) para pausa e 00000000 (0) como sinal neutro.

As aplicações receptoras implementam o *Receiver*, também usando as frequências padrões da biblioteca e definindo eventos para as mensagens de pausa e neutro. A informação de qual instrumento cada aparelho deve tocar é definida pela URL da aplicação. Para decodificar e reproduzir a música em mp3 no navegador foi desenvolvido um *player* usando a *Web Audio* API. Para facilitar o acesso às aplicações dos instrumentos, a aplicação emissora pode gerar as URLs específicas de cada instrumento, exibidas como um QR Code no corpo da página.

Essa PoC também demonstra que a *Audio Markings* consegue transmitir e receber mensagens sonoras mesmo em cenários onde outros sons estão sendo reproduzidos no mesmo ambiente. Isso é possível graças à estratégia de utilizar frequências fora da audição humana para a comunicação entre aplicações, uma vez que a maioria dos instrumentos são projetados para utilizar apenas as frequências audíveis, apresentando pouca ou nenhuma intensidade em frequências fora dessa faixa.

O código desenvolvido pode ser acessado através de um repositório aberto[4], que também contém links para as aplicações Web e um vídeo demonstrando essa PoC em ação. A Figura 28 demonstra a prova de conceito sendo executada, o notebook está atuando como o ponto de sincronização, enquanto os outros dispositivos efetivamente reproduzem a música, cada um desempenhando o papel de um instrumento diferente.

[4] Endereço de acesso: https://github.com/jonkoala/distributed-musical-performance

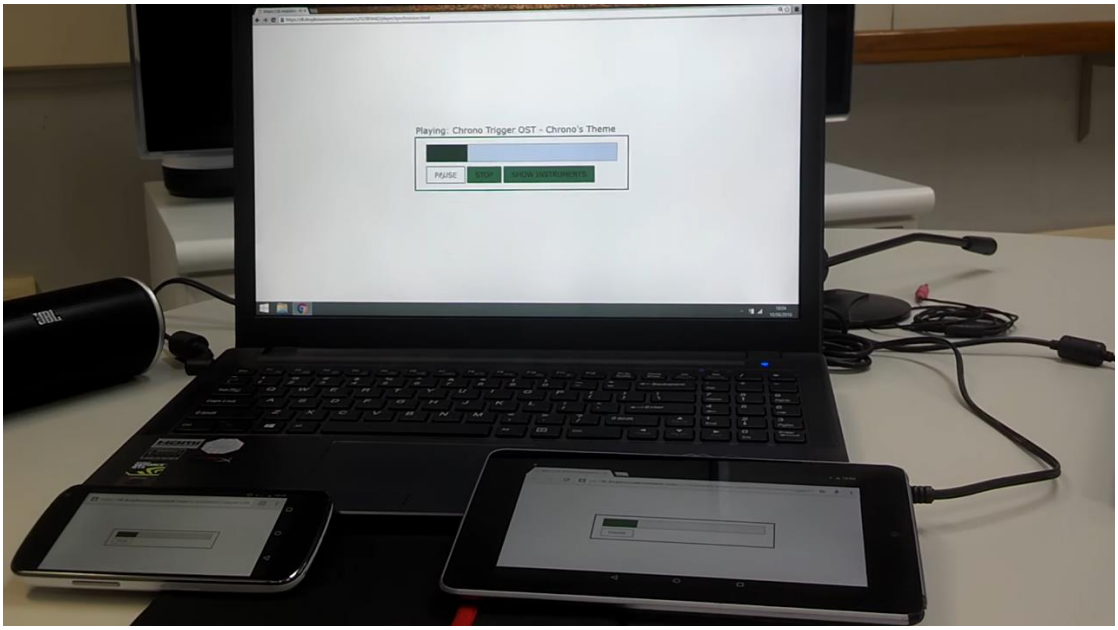

Figura 28. Prova de conceito Performance Musical Distribuída. Fonte: Produção do próprio autor.

4.2.3 Apresentação Sincronizada de Slides

Essa PoC tenta simular uma apresentação de slides, de forma simultânea entre vários aparelhos. Nesse cenário existe uma aplicação para a apresentação principal, contendo os slides, além de outras aplicações secundárias, com conteúdo complementar à apresentação, como imagens, diagramas ou informações extras.

A aplicação emissora orquestra a apresentação, emitindo o sinal referente ao slide que está sendo exibido. As aplicações secundárias captam o sinal que está sendo transmitido, decodificam a mensagem e exibem o conteúdo adicional sincronamente com o slide. O cenário dessa prova de conceito pode ser visto na Figura 29. Ambas as aplicações foram implementadas a partir de uma biblioteca já existente para apresentação de slides, a `reveal.js` [53].

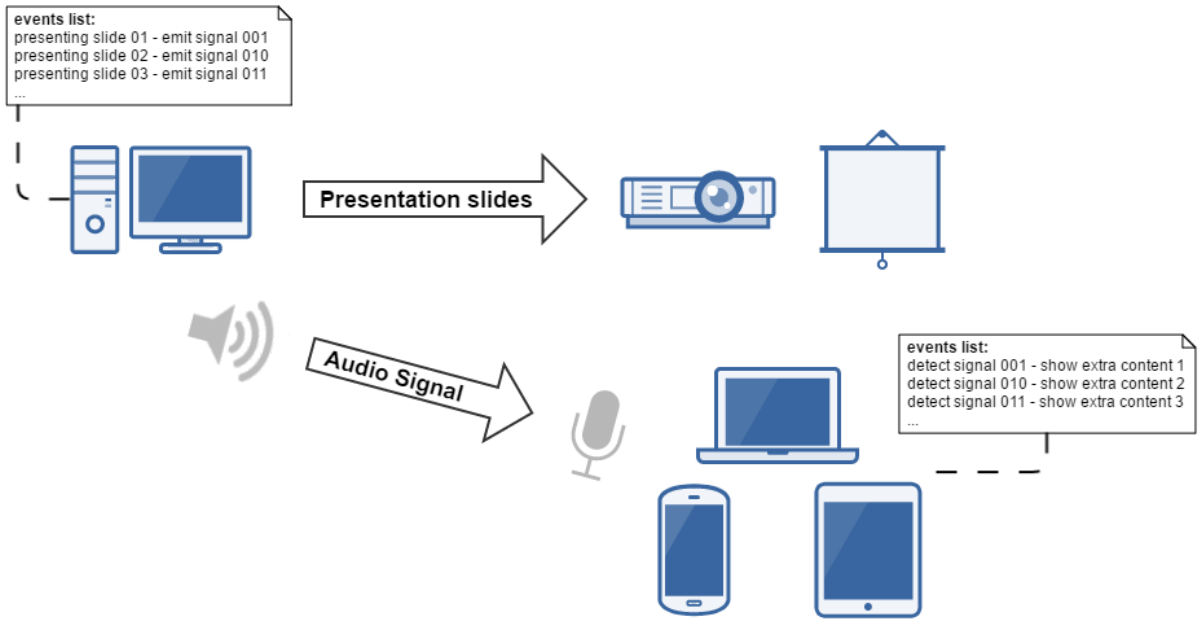

Figura 29. Esquema das aplicações que compõe a PoC "Apresentação Sincronizada de Slides". Fonte: Produção do próprio autor.

A aplicação de apresentação principal (a emissora) também foi desenvolvida usando um *Emitter* nas frequências padrões da biblioteca. Os sinais emitidos consistem na posição do slide em exibição, dessa forma, a mensagem 00000000 (0) representa o primeiro slide, 00000001 (1), o segundo, e assim por diante. O primeiro slide da apresentação contém a URL para as aplicações receptoras, na forma de QR Code, para facilitar o acesso.

As aplicações receptoras implementam o *Receiver*, também nas frequências padrões da *Audio Markings*. Essas aplicações são similares à emissora, contendo slides, no entanto a ordem em que esses slides são apresentados depende da informação transmitida pela aplicação principal. As aplicações receptoras atuam constantemente escutando os sinais transmitidos, trocando de slide sempre que uma mensagem nova é detectada.

O código desenvolvido na PoC pode ser acessado através de um repositório aberto[5], que também contém links para as aplicações Web e um vídeo demonstrando essa PoC em ação. É possível observar na Figura 30 a prova de conceito em execução, o *notebook* à esquerda está executando a aplicação emissora, enquanto os demais dispositivos estão executando as aplicações receptoras. Enquanto a aplicação emissora exibe o slide da apresentação, as aplicações receptoras exibem conteúdos extras, como imagens e diagramas.

[5] Endereço de acesso: https://github.com/jonkoala/synchronous-slideshow

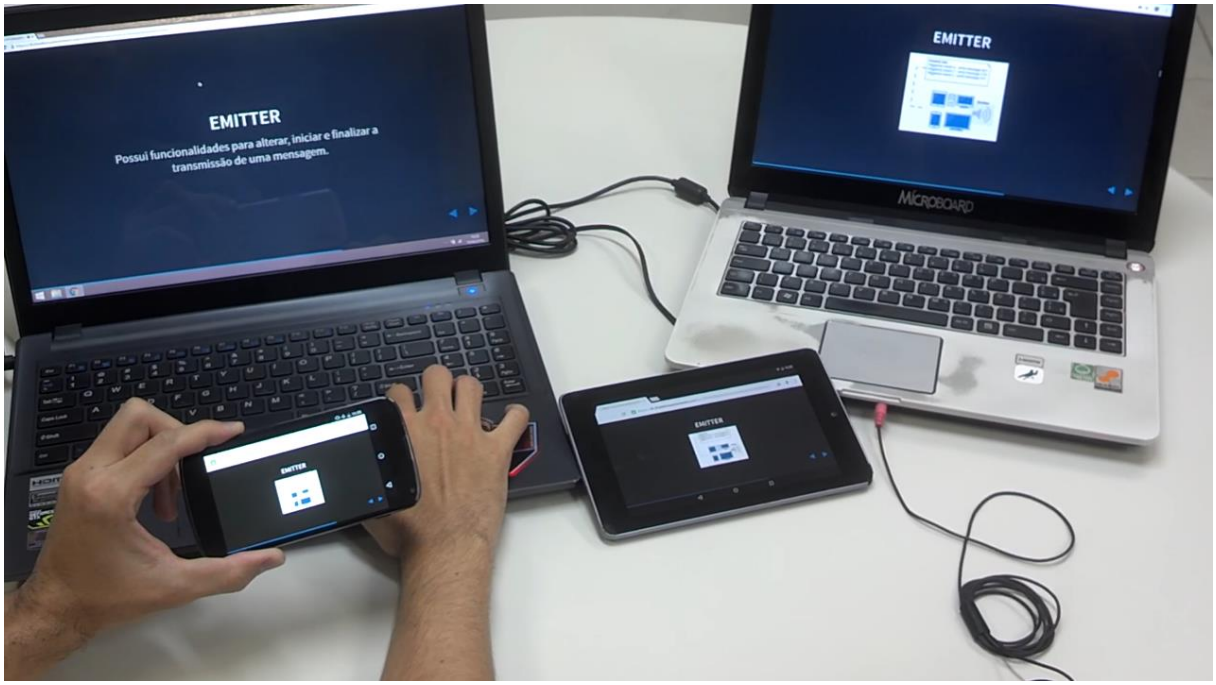

Figura 30. Prova de conceito Apresentação de Slides. Fonte: Produção do próprio autor.

4.3 Questões Práticas

Todas as PoCs foram hospedadas no servidor do GitHub, através da plataforma GitHub Pages [54]. Essa plataforma foi escolhida pela facilidade de atualização das soluções hospedadas, além de fornecer o serviço de HTTPS, necessário para a execução das aplicações que façam uso da classe *Receiver* da *Audio Markings*. Como mencionado no capítulo 3, todas PoCs também foram implementadas como *progressive Web apps*, ou seja, se o endereço da aplicação for acessado com sucesso ao menos uma vez, a mesma pode ser utilizada mesmo em modo *offline*.

Para a reprodução dos experimentos apresentados, é importante que alguns requisitos práticos sejam atendidos. Primeiramente, como já informado, para a execução de aplicações que façam uso da classe *Receiver* da biblioteca, é importante que a aplicação esteja hospedada em um servidor HTTPS. A escolha do navegador em que as aplicações serão executadas também é importante, atualmente alguns dos principais navegadores Web (Chrome, Firefox, Edge e Safari) são capazes de executar corretamente aplicações que façam uso do *Emitter*, no entanto, desses apenas o Chrome, Edge e algumas versões do Firefox são capazes de executar aplicações que empregam o *Receiver*.

Essa última restrição se da pelo fato de que nem todos os navegadores implementaram todas as funcionalidades da *MediaStream* API. No caso do Safari, as funções de acesso ao microfone ainda não foram implementadas, já o Firefox apresenta versões em que os filtros sobre o sinal capturado não são corretamente desabilitados. Por ser relativamente nova, as especificações da *MediaStream* ainda estão sendo implementadas e testadas pelos fabricantes de navegadores Web.

Capítulo

5

Considerações Finais

Nesse trabalho foi discutida a utilização de sinais de áudio como meio de comunicação entre aplicações, apresentando os conhecimentos necessários na construção de ferramentas para esse fim, além de demonstrar as estratégias e recursos que podem ser empregadas na sua construção. Algumas plataformas e situações onde a comunicação por áudio poderia ser aplicada foram discutidas, dando uma ênfase em especial para os navegadores Web, por ser uma plataforma que pode atender a uma gama considerável de aparelhos, além de carecer de ferramentas semelhantes.

Foram apresentados também trabalhos já existentes na área, assim como uma comparação sobre as estratégias adotadas. É interessante notar a diversidade de plataformas que esses trabalhos atendem, além dos problemas que cada um buscou resolver. Essa diversidade demonstra como essa forma de comunicação possui sim o seu papel, atendendo a problemas próprios, ou oferecendo uma solução alternativa a problemas já existentes.

Como forma de aplicar os conhecimentos apresentados, foi desenvolvida uma biblioteca, a *Audio Markings*, para auxiliar no desenvolvimento de aplicações Web que façam do áudio a sua forma de comunicação. As decisões tomadas durante a sua produção foram detalhadas no decorrer desse trabalho, o que pode ser utilizado não só para a produção de ferramentas similares, como também em soluções com propostas completamente diferentes, mas que possam fazer uso de algum conhecimento específico que foi envolvido na produção dessa biblioteca. O código fonte da *Audio Markings* está disponível online, através do GitHub, ao acesso de qualquer desenvolvedor que se demonstre interessado.

Três cenários de uso dessa estratégia de comunicação foram implementados. Todas utilizaram a *Audio Markings*, ilustrando diferentes situações onde a estratégia poderia ser empregada. As aplicações podem ser acessadas online, através de qualquer navegador que atenda aos padrões atuais da W3C, um vídeo demonstrando a utilização de cada aplicação também foi disponibilizado. Assim como a biblioteca, o código fonte de cada uma dessas soluções está disponível online, através do GitHub.

Por fim, apresentamos os resultados de testes realizados sobre o desempenho da biblioteca, com relação à precisão da comunicação. Os resultados obtidos demonstram o nível de precisão alcançado pela *Audio Markings*, mas também podem ser interessantes para se analisar o desempenho dos aparelhos utilizados, quando seus microfones e saídas de áudio precisam trabalhar em frequências fora espectro para o qual foram projetados.

Considerando a ubiquidade de navegadores Web e *hardwares* de áudio em aparelhos usados no dia-a-dia, somada ao crescente número de aplicações desenvolvidas com essa plataforma em mente, acredito que essa biblioteca represente uma contribuição importante para o ramo de desenvolvimento Web. Espero também que esse trabalho não se restrinja a apenas fornecer uma forma alternativa de comunicação, mas que instigue a comunidade de desenvolvedores sobre como aproveitar de forma criativa os recursos presentes nos aparelhos que temos a nossa volta.

5.1 Comparações com Trabalhos Relacionados

Nessa seção será feita uma comparação entre as decisões e estratégias adotadas no desenvolvimento da *Audio Markings*, com os trabalhos apresentados no capítulo 2. Como tópicos para comparação, serão adotados os mesmos itens abordados durante a comparação entre os trabalhos relacionados, a estratégia de modulação adotada, espectro de frequências em que atuam e a disponibilidade das soluções. Por fim, será apresentado um resumo das comparações, no formato de tabela.

5.1.1 Estratégia de Modulação

Como explicado no capítulo 3, a estratégia para geração de sinais adotada pela *Audio Markings* faz uso da modulação por chaveamento de amplitude e, similar ao GUWMANET, da multiplexação entre as frequências do sinal. As mensagens não são enviadas de forma

serial, cada sinal transmitido contém os 8 bits de informação, distribuídos entre as frequências definidas na onda portadora.

Essa estratégia não só envia mais dados por sinal, aproveitando melhor o espectro disponível para a aplicação, mas também permite que a *Audio Markings* possa usar frequências de referência para sinais em alta e baixa intensidade, o que ajudam a amenizar o problema de ruídos, comum na modulação por chaveamento de frequência e amplitude. Essa estratégia é flexível o suficiente para também permitir que a solução emule a modulação por chaveamento de frequências. Ou seja, se configurada para tal, a *Audio Markings* pode trabalhar igual a qualquer solução apresentada que faz usa da modulação FSK, sem precisar alterar o código fonte.

Uma opção seria adotar a modulação por chaveamento de fase, o que diminuiria consideravelmente o problema dos ruídos. Mas essa estratégia demanda uma quantidade consideravelmente maior de poder computacional, ainda mais em soluções como a *Audio Markings* ou o GUWMANET, onde o mesmo sinal contém informação em mais de uma frequência. Além da questão de performance, a plataforma adotada para o trabalho não possui hoje ferramentas para o controle preciso da fase de um sinal durante a sua geração [56]. É possível, no entanto, que essa realidade mude no futuro, uma vez que a manipulação da fase em sinais de áudio é uma das funcionalidades cotadas para a próxima versão da *Web Audio* API [56], responsável pela criação e manipulação de sinais de áudio em navegadores Web.

5.1.2 Espectro de Frequências Adotado

Por padrão, a *Audio Markings* trabalha no espectro de 18.6 a 20.4 KHz, o suficiente para que os sinais gerados sejam considerados como inaudíveis, ou semi-inaudíveis. Assim como outras soluções apresentadas, a biblioteca pode ser configurada para trabalhar com outros valores, inclusive gerando sinais no espectro audível.

Diferente do Chirp e do Google Tone, a *Audio Markings* não aplica nenhuma forma de tratamento para que os sons gerados no espectro audível sejam menos desagradáveis. Caso o desenvolvedor deseje por algum motivo gerar sons no espectro audível utilizando a biblioteca, o mesmo teria que realizar os seus próprios tratamentos para controlar a interação com o ouvinte. Uma solução simples, que não demanda alterações no código fonte, seria configurar a *Audio Markings* para utilizar frequências associadas a notas musicais, da mesma forma que o Google Tone.

5.1.3 Disponibilidade de Plataformas

A *Audio Markings* atende à mesma plataforma que a `sonicnet.js` e, dessa forma, possui as mesmas limitações associadas a ela. Para utilizar a biblioteca, o aparelho deve possuir dispositivos de entrada ou saída de áudio, além de ter um navegador Web compatível instalado. Por ser quase onipresente nos aparelhos *mobile* e *desktop* atuais, o navegador Web foi a plataforma escolhida.

Por se tratar de uma plataforma para soluções Web, o navegador também possuía a limitação de necessitar que o aparelho estivesse conectado à Internet para executar as aplicações. Mas com a vinda do padrão W3C para a utilização de *Service Workers* [49,50], esse limitador não existe mais, uma vez que aplicações Web podem agora trabalhar em modo *offline*, da mesma forma que uma aplicação nativa faria.

Casos como o Arduino e plataformas para sistemas embarcados não podem hoje fazer uso da biblioteca, uma vez que nenhum fabricante de navegador possui uma versão oficial para essas plataformas. No entanto, ferramentas como o *PIP* [57] e o *PhantomJS* [58] podem mudar esse cenário no futuro. Esses programas emulam o funcionamento de um navegador Web mais simplificado, até mesmo gerando apresentações visuais em consoles no caso do *PIP*. Nenhuma dessas ferramentas implementam hoje as APIs necessárias para o funcionamento da *Audio Markings*, mas podem representar uma alternativa viável no futuro.

5.1.4 Resumo da Comparação

Com o intuito de facilitar a visualização da comparação entre os trabalhos relacionados, e entre estes e o trabalho aqui apresentado, foi elaborada uma tabela sintetizando os tópicos abordados nessa seção.

A Tabela 4 apresenta de forma resumida as comparações realizadas entre os trabalhos relacionados e a biblioteca Audio Markings, utilizando os mesmos tópicos abordados na seção 5.1, a estratégia de modulação adotada, espectro de frequências em que atuam e, por fim, a disponibilidade das soluções.

Tabela 4. Resumo da comparação entre os trabalhos relacionados e a biblioteca Audio Markings.

	Estratégia de modulação	**Espectro adotado**	**Plataformas**
minimodem	Binary FSK	1 a 3 KHz (configurável)	GNU/Linux
Chirp	MFSK	1.7 a 10 KHz ou 18 a 20kHz	iOS, Android, Desktop, Arduino, Navegadores Web
Google Tone	DTMF	1 a 3 KHz	Navegador Web (Google Chrome)
sonicnet.js	MFSK	18.5 a 19.5 KHz (configurável)	Navegadores Web
GUWMANET	QPSK, com multiplexação	3.5 a 7.5 KHz	Sistema Embarcado
AudioModem	DBPSK	18.4 e 20.8 KHz	iOS
Audio Markings	ASK, com multiplexação FDM	18.6 a 20.4 KHz (configurável)	Navegadores Web

5.2 Contribuições do Trabalho

O desenvolvimento da *Audio Markings* foi uma das principais contribuições desse trabalho, disponibilizando uma biblioteca para facilitar o desenvolvimento de aplicações Web que utilizem o áudio como estratégia de comunicação. A biblioteca abstrai os conhecimentos sobre sintetização, modulação e análise de sinais de áudio, necessários para transmitir informações através de ondas sonoras, além de tratar problemas relacionados a ruídos e flutuações na captura dos sinais acústicos.

Apesar de abstraídos pela biblioteca, os conhecimentos necessário para o desenvolvimento da mesma também foram apresentados. Os principais procedimentos envolvidos na sintetização e análise de sinais de áudio foram apresentados e discutidos no capítulo 2. A aplicação desses conhecimentos no desenvolvimento de aplicações Web foi discutida em maiores detalhes no capítulo 3, apresentando as APIs nativas utilizadas e como as mesmas foram empregadas no desenvolvimento da *Audio Markings*.

Foram desenvolvidas 3 aplicações PoC que empregam o uso da comunicação por áudio, além de uma aplicação para avaliação da precisão alcançada através das estratégias aplicadas no desenvolvimento da *Audio Markings*. O código fonte das aplicações é aberto e está disponível em repositório online, assim como o código da própria biblioteca, permitindo que outros desenvolvedores possam utilizar como base no desenvolvimento de soluções similares.

Durante o desenvolvimento do trabalho também foi elaborado um artigo, apresentando a pesquisa e os resultados obtidos com a *Audio Markings* [55]. Esse artigo foi publicado no evento Web Media 2016, passando por uma banca examinadora, e apresentado durante uma das sessões técnicas do evento.

5.3 Limitações e Questões Práticas

A decisão de utilizar navegadores Web como plataforma para o desenvolvimento de aplicações, apesar de possuir as suas vantagens, pode limitar a quantidade de usuários com aparelhos habilitados a utilizar a *Audio Markings*. Isso porque nem todos os navegadores implementam as tecnologias usadas pela solução. Alguns dos recursos usados, em especial a criação de ondas periódicas [25] e o acesso ao microfone do aparelho de forma nativa (sem necessitar da instalação de *plug-ins*) [24], são relativamente novos e, portanto, é possível que navegadores mais antigos ainda não sejam capazes de utilizar a biblioteca.

O uso do microfone através do navegador exige também que o fabricante do aparelho permita o acesso da aplicação ao *hardware*. Além de, em alguns casos, não existir essa permissão, alguns modelos também aplicam, por padrão, filtros ao sinal capturado, tais como *echo cancellation* ou *low-pass filter*. Uma vez que o uso primário desses microfones é para a captação de voz humana, é de se esperar que os aparelhos sejam projetados para captar melhor o espectro da fala, mas quando é desejado usar a biblioteca para a comunicação em frequências inaudíveis, isso pode representar um problema. É possível requisitar que esses filtros sejam desabilitados [24] e, de fato, a *Audio Markings* faz essa requisição através da *MediaStream* API, mas por se tratar de uma tecnologia relativamente nova, alguns fabricantes podem não ter implementado ainda uma resposta adequada. Para contornar o problema, a biblioteca permite que frequências não filtradas sejam usadas, mas nesses casos, a maioria dos usuários seria capaz de escutar os sinais transmitidos.

A API usada para a geração e análise de sinais trabalha com uma taxa de amostragem de 44.1 KHz [25], o que permitiria, seguindo o teorema de Nyquist, a utilização de sinais com frequências de até 22.05 KHz, limitando o espectro que soluções Web podem trabalhar. Considerando que os sinais são gerados utilizando a síntese de Fourier, esse limite se torna ainda menor, uma vez que cada harmônico utilizado causa uma diminuição na amplitude geral do sinal [9]. Por padrão, a *Audio Markings* usa valores que não chegam a representar queda significativa na amplitude, mas se configurada para se comunicar em frequências muito próximas do limite superior da *Web Audio* API (22.05 KHz) ou muito acima de 20 KHz, é possível notar a diminuição drástica na amplitude do sinal, e por consequência uma diminuição na qualidade da transmissão.

5.4 Trabalhos Futuros

Além dos resultados obtidos, a área de pesquisa poderia se beneficiar através de diversas melhorias a partir do estado atual desse trabalho.

Como os navegadores Web estão sempre se atualizando, qualquer solução que use essa plataforma pode se beneficiar com o surgimento de novas ferramentas e padrões, e a *Audio Markings* não é exceção. Outras estratégias de modulação, captura de áudio ou tratamento de eventos podem ser aplicadas a biblioteca. Como já citado, hoje não é possível realizar a modulação por chaveamento de fases, mas com a vinda das ferramentas necessárias, a biblioteca pode se beneficiar de uma opção extra de modulação do sinal. Estratégias para detecção e correção de erros também poderiam ser aplicadas, reforçando o tratamento já realizado com a lógica de *debouncing*.

É possível notar no endereço onde a biblioteca foi publicada, que a mesma carece de documentação. É interessante que exista alguma documentação de fácil acesso a qualquer desenvolvedor que busque aplicar a *Audio Markings* a sua aplicação. Nesse trabalho já se encontra uma documentação sobre as classes e as funções que a biblioteca fornece, mas exemplos de como realizar tarefas simples com a biblioteca, como emitir um sinal ou adicionar um evento a uma mensagem, poderiam suavizar consideravelmente a curva de aprendizado na sua utilização.

For fim, a condução de mais testes seria de grande proveito para a pesquisa. Uma avaliação não só do desempenho da *Audio Markings*, mas também dos outros trabalhos apresentados, poderia evidenciar áreas de atuação para novas pesquisas, identificando possíveis melhorias, problemas a serem solucionados ou estratégias alternativas a serem aplicadas. A área de comunicação digital através do som ainda é relativamente pouco explorada, certamente ainda existem novos cenários a serem identificados, problemas a serem mapeados e, porque não, resolvidos em trabalhos futuros.

Referências

1. Everest, F.A., Pohlmann, K.C., 2009. The Master Handbook of Acoustics (Vol. 4). *New York: McGraw-Hill*.
2. Loy, G., 2006. Musimathics (Vol. 1). *Cambridge, MA: MIT Press*.
3. Berning, F., Radtke, T., Rautenberg, S., Motz, M., Nissen, I., 2014. A Realization of the Software Defined Radio Concept in an Underwater Communication Modem. In *Underwater Communications and Networking (UComms), 2014* (pp. 1-5). IEEE.
4. Goetz, M., Nissen, I., 2012. GUWMANET—Multicast routing in Underwater Acoustic Networks. In *Communications and Information Systems Conference (MCC), 2012 Military* (pp. 1-8). IEEE.
5. Goetz, M., Nissen, I., Otnes, R., Walree, P., 2015. Performance Analysis of Underwater Network Protocols within International Sea Trial. In *OCEANS 2015-Genova* (pp. 1-7). IEEE.
6. Kalwa, J., 2011. The RACUN-Project: Robust Acoustic Communications in Underwater Networks - an Overview. In *OCEANS 2011 IEEE-Spain* (pp. 1-6). IEEE.
7. Smus, B., 2013. *Web audio API*. O'Reilly Media, Inc.
8. Halliday, D., Resnick, R., Walker, J., 2009. Fundamentos de Física (Vol. 2): Gravitação, Ondas e Termodinâmica. *Rio de Janeiro: LTC*, 8.
9. Loy, G., 2007. Musimathics (Vol. 2). *Cambridge, MA: MIT Press*.
10. Xiong, F., 2006. Digital Modulation Techniques. *Artech House*.
11. Mostafa, K., 2013. minimodem. Disponível em <https://github.com/kamalmostafa/minimodem>. Acessado em Dezembro de 2016.
12. Wanstrath, C., 2008. GitHub. Disponível em <https://github.com/>. Acessado em Dezembro de 2016.
13. Free Software Foundation, 2007. GNU General Public License, Version 3. Disponível em <http://www.gnu.org/copyleft/gpl.html>. Acessado em Dezembro de 2016.
14. Hanspach, M., Goetz, M., 2014. On Covert Acoustical Mesh Networks in Air. *Journal of Communications*, 8(11).
15. Taniguchi, Y., 2015. Evaluation of Acoustic Data Communication for Fish Farm monitoring. In *Proceedings of DNCOCO 2015, Dec. 2015*, pp. 195–199.
16. Asio. Chirp. Disponível em <https://www.chirp.io/>. Acessado em Dezembro de 2016.

17. Hahn, P., 1962. Theoretical Diversity Improvement in Multiple Frequency Shift Keying. *IRE Transactions on Communications Systems, 10*(2), pp.177-184.
18. Activision, 2014. Skylanders: Trap Team. Disponível em <https://www.skylanders.com/>. Acessado em Dezembro de 2016.
19. Google, 2015. Google Tone. Disponível em <https://research.googleblog.com/2015/05/tone-experimental-chrome-extension-for.html>. Acessado em Dezembro de 2016.
20. Google. Chrome Web Browser. Disponível em <https://www.google.com/chrome/>. Acessado em Dezembro de 2016.
21. Luo, P., Ghassemlooy, Z., Le Minh, H., Khalighi, A., Zhang, X., Zhang, M., Yu, C., 2014. Experimental Demonstration of an Indoor Visible Light Communication Positioning System Using Dual-Tone Multi-Frequency Technique. In *Optical Wireless Communications (IWOW), 2014 3rd International Workshop in* (pp. 55-59). IEEE.
22. Jeon, K.M., Kim, H.K., Lee, M.J., 2016. Noncoherent Low-Frequency Ultrasonic Communication System with Optimum Symbol Length. In *International Journal of Distributed Sensor Networks, 2016*.
23. Smus, B., 2013. Sonicnet.js. Disponível em <https://github.com/borismus/sonicnet.js/>. Acessado em Dezembro de 2016.
24. W3C-World Wide Web Consortium, 2016. Media Capture and Streams. *W3C Candidate Recommendation*. Disponível em <https://www.w3.org/TR/2016/CR-mediacapture-streams-20160519/>.
25. W3C-World Wide Web Consortium, 2015. Web Audio API. *W3C Working Draft*. Disponível em <https://www.w3.org/TR/2015/WD-webaudio-20151208/>.
26. Apache Software Foundation, 2004. Apache License, Version 2.0. Disponível em <https://www.apache.org/licenses/LICENSE-2.0>. Acessado em Dezembro de 2016.
27. Smus, B., 2013. Ultrasonic networking on the Web. Disponível em <http://smus.com/ultrasonic-networking/>. Acessado em Dezembro de 2016.
28. Calixto, G.M., Angeluci, A.C., Kurashima, C.S., de Deus Lopes, R., Zuffo, M.K., 2014. Effectiveness analysis of audio watermark tags for IPTV second screen applications and synchronization. In *Telecommunications Symposium (ITS), 2014 International* (pp. 1-5). IEEE.
29. Applidium, 2014. AudioModem. Disponível em <https://github.com/applidium/AudioModem>. Acessado em Dezembro de 2016.

30. Ghovanloo, M., Najafi, K., 2004. A Wideband Frequency-Shift Keying Wireless Link for Inductively Powered Biomedical Implants. In *IEEE Transactions on Circuits and Systems I* 51(12), pp.2374-2383.
31. Open Source Initiative, 2006. The MIT license. Disponível em <https://opensource.org/licenses/MIT>. Acessado em Dezembro de 2016.
32. Roberts, D., Johnson, R., 1996. Evolving frameworks. *Pattern languages of program design*, 3.
33. jQuery Team. jQuery. Disponível em <https://jquery.com/>. Acessado em Dezembro de 2016.
34. Rauch, G., 2012. Socket.io. Disponível em <http://socket.io/>. Acessado em Dezembro de 2016.
35. World Wide Web Consortium (W3C). Disponível em <https://www.w3.org/>. Acessado em Dezembro de 2016.
36. Fielding, R., Reschke, J., 2014. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. Disponível em <https://tools.ietf.org/html/rfc7231>.
37. W3C-World Wide Web Consortium, 2010. Mobile web application best practices. *W3C Recommendation*. Disponível em <http://www.w3.org/TR/2010/REC-mwabp-20101214>.
38. Google, 2016. Progressive Web Apps. Disponível em <https://developers.google.com/web/progressive-web-apps/>. Acessado em Dezembro de 2016.
39. Microsoft Corporation, 2016. The Progress of Web Apps. Disponível em <https://blogs.windows.com/msedgedev/2016/07/08/the-progress-of-web-apps/>. Acessado em Dezembro de 2016.
40. Mozilla Foundation, 2016. Progressive Web Apps. Disponível em <https://developer.mozilla.org/en-US/Apps/Progressive>. Acessado em Dezembro de 2016.
41. Grosskurth, A., Godfrey, M.W., 2005. A Reference Architecture for Web Browsers. In *21st IEEE International Conference on Software Maintenance (ICSM'05)* (pp. 661-664). IEEE.
42. W3C-World Wide Web Consortium, 2014. HTML5 A vocabulary and associated APIs for HTML and XHTML. *W3C Recommendation*. Disponível em <https://www.w3.org/TR/2014/REC-html5-20141028/>.
43. W3C-World Wide Web Consortium, 2004. Document Object Model (DOM) Level 3 Core Specification. *W3C Recommendation*. Disponível em <https://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/>.

44. Google. chrome.bookmarks. Disponível em <https://developer.chrome.com/extensions/bookmarks>. Acessado em Dezembro de 2016.
45. ECMA International, 2016. Standard ECMA-262 ECMAScript Language Specification.
46. W3C-World Wide Web Consortium, 2014. URL. *W3C Working Draft*. Disponível em <https://www.w3.org/TR/2014/WD-url-1-20141209/>.
47. Adobe Systems. Flash player. Disponível em <http://www.adobe.com/products/flashplayer.html>. Acessado em Dezembro de 2016.
48. Microsoft Corporation. Silverlight. Disponível em <https://www.microsoft.com/silverlight/>. Acessado em Dezembro de 2016.
49. W3C-World Wide Web Consortium, 2015. Service Workers. *W3C Working Draft*. Disponível em <https://www.w3.org/TR/2015/WD-service-workers-20150625/>.
50. W3C-World Wide Web Consortium, 2015. Web Workers. *W3C Working Draft*. Disponível em <https://www.w3.org/TR/2015/WD-workers-20150924/>.
51. W3C-World Wide Web Consortium, 2016. Web App Manifest Living Document. *W3C Working Draft*. Disponível em <https://www.w3.org/TR/2016/WD-appmanifest-20161212/>.
52. Node.js Developers. Node.js. Disponível em <https://nodejs.org/>. Acessado em Dezembro de 2016.
53. El Hattab, H., 2015. reveal.js: The HTML Presentation Framework. Disponível em <https://github.com/hakimel/reveal.js/>. Acessado em Dezembro de 2016.
54. Wanstrath, C. GitHub Pages. Disponível em < https://pages.github.com/>. Acessado em Dezembro de 2016.
55. Lemos, V.S., Ferreira, R.F., Costa Segundo, R.M., Costalonga, L.L. and Santos, C.A., 2016. Local Synchronization of Web Applications with Audio Markings. In *Proceedings of the 22nd Brazilian Symposium on Multimedia and the Web* (pp. 159-166). ACM.
56. Harman, A., 2015. Phase-offset of oscillator nodes #541. *GitHub Issues*. Disponível em <https://github.com/WebAudio/web-audio-api/issues/541>. Acessado em Dezembro de 2016.
57. Anderson, C., 2014. PIP. Disponível em <https://github.com/zigwart/PIP-Arduino-Web-Browser>. Acessado em Dezembro de 2016.
58. Hidayat, A., 2011. PhantomJS. Disponível em <http://phantomjs.org/>. Acessado em Dezembro de 2016.