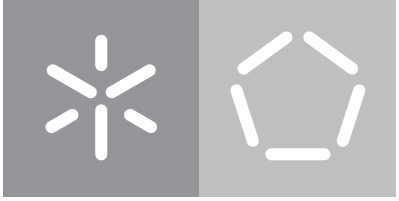
Universidade do Minho
Escola de Engenharia

Hugo Daniel da Costa Cunha Machado

Otimização de protocolo para distribuição sincronizada de áudio multicanal em redes sem fios

Março, 2023



Universidade do Minho
Escola de Engenharia

Hugo Daniel da Costa Cunha Machado

Otimização de protocolo para distribuição sincronizada de áudio multicanal em redes sem fios

Dissertação de Mestrado

Mestrado em Engenharia de Telecomunicações e Informática

Trabalho efetuado sob a orientação do(a)

Bruno Alexandre Fernandes Dias

Março, 2023

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

_____, _____

(Localização) (Data)

_____

(Hugo Daniel da Costa Cunha Machado)

Agradecimentos

Em primeiro lugar, gostaria de agradecer ao Professor Bruno Dias por me ter apoiado todo este tempo, pela disponibilidade, pela paciência e por todo o conhecimento partilhado.
Em segundo lugar, gostaria de agradecer a toda a minha família, em especial aos meus pais por todo o esforço e conforto que sempre me proporcionaram.
Em terceiro lugar, um enorme obrigado a todos os meus amigos que se mantiveram mais próximos neste últimos anos, obrigado por todo o apoio neste trabalho e obrigado pela a descompressão, lazer e sanidade que me ofereceram.

Resumo

As comunicações sem fios têm ganho popularidade porque oferecem soluções práticas possibilitando ritmos de transferência e níveis de fiabilidade que, não sendo equivalentes às das tecnologias com fios, são já suficientes para suportar grande parte das aplicações distribuídas, incluindo aplicações multimédia. Esta popularidade é facilmente observável no contexto mais restrito das soluções para implementação de sistemas de distribuição áudio. Nesta área interessa-nos a distribuição áudio entre um servidor central e os vários clientes musicais que reproduzem os canais de áudio. A sua implementação tem que lidar com a limitação do ritmo da informação e com problemas de dessincronização temporal entre os elementos. A maioria dos produtos comercialmente disponíveis integra tecnologias proprietárias, tornando as soluções não compatíveis entre si. Na Universidade do Minho este problema já foi abordado em dois trabalhos em 2014 e 2018, resultando na definição dum sistema aberto e sem requisitos especiais de *hardware*. Os resultados dos testes realizados aos protótipos foram encorajadores, mas ficou por analisar a eficácia dos algoritmos de contração/expansão do áudio quando há uma sincronização temporal elevada e ultrapassar a excessiva simplicidade do protocolo de distribuição de áudio, por suportar apenas dois canais e não incluir mecanismos de deteção e correção de erros. Neste projeto foram desenvolvidos três protocolos comunicação para efetuar o cálculo da diferença temporal, a distribuição de áudio multicanal e controlar a reprodução de áudio. Além destes foram desenvolvidos também algoritmos e mecanismos para o cálculo do mesmo instante temporal em diferentes dispositivos e para a contração/expansão temporal do áudio. Por fim, as aplicações cliente e servidor desenvolvidas integram estas soluções de forma harmoniosa, resultando num excelente nível de sincronização e desempenho.

Palavras-chave: Tecnologias sem fios, distribuição de áudio, áudio multi-canal, sincronização de diferentes dispositivos, código aberto.

Abstract

Wireless communications have grown in popularity because they provide practical solutions that enable transfer rates and levels of reliability that, while not identical to wired technologies, are sufficient to support the majority of distributed applications, including multimedia applications. This popularity is obviously visible in the more narrow context of audio distribution implementations solutions. We're interested in audio distribution between a central server and the various music clients that play the audio channels in this area. Its implementation must deal with the information rhythm's limitation as well as issues of temporal desynchronization between the elements. Since most commercially available products incorporate proprietary technologies, the solutions are incompatible with one another. This problem has already been addressed at the University of Minho in two previous works [2][7], resulting in the definition of an open-source system with no special hardware requirements. The results of the prototype tests were encouraging, but it remained to analyze the effectiveness of the audio contraction/expansion algorithms when there is high temporal synchronization and to overcome the audio distribution protocol's excessive simplicity, as it only supports two channels and lacks error detection and correction mechanisms. Three communication protocols were created in this project to calculate the temporal difference, distribute multichannel audio, and control audio reproduction. In addition, algorithms and mechanisms for calculating the same temporal instant in different devices, as well as for audio temporal contraction/expansion, were developed. Finally, the developed client and server applications integrate these solutions harmoniously, resulting in an excellent level of synchronization and performance.

Keywords: Wireless technologies, audio distribution, multi-channel audio, synchronization of different devices, open source.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Glossário

Buffering	É o termo utilizado quando um computador armazena informação temporariamente na sua memória, enquanto esta é lida.
Lip Synchronization	Sincronização entre o som e imagem.
Master Clock	Um relógio mestre é um relógio de precisão que fornece sinais de temporização para sincronizar relógios clientes como parte de uma rede de relógio.
Multicast	É a comunicação em grupo em que a transmissão de dados é enviada para um grupo de dispositivos simultaneamente.
Multi-Streaming	É transmitir conteúdo de media para mais que uma plataforma simultaneamente.
Offset	A diferença temporal entre dois relógios.
Payload	É a carga útil de uma unidade de transmissão de dados.
Time Skew	Diferença entre a frequência de relógio de dois dispositivos.

Siglas

ALSA	Advanced Linux Sound Architecture
API	Application Programming Interface
CODEC	Coder-Decoder
CPU	Central Processing Unit
DCCP	Datagram Congestion Control Protocol
DMTS	Delay Measurement Time Synchronization
DOS	Denial Of Service
FM	Frequency Modulation
FTSP	Flooding Time Synchronization Protocol
IP	Internet Protocol
LAN	Local Area Network
MAC	Media Access Control
NTP	Network Time Protocol
P2P	Peer-to-Peer
PTP	Precicion Time Protocol
RAOP	Remote Audio Output Protocol
RBS	Reference Broadcast Synchronization

RTCP Real-time Tranport Control Protocol
RTP Real-time Transport Protocol
RTSP Real Time Streaming Protocol
RTT Round Trip Time

SCTP Stream Control Transmission Protocol

TCP Transport Control Protocol
TPSN Timing-sync Protocol for Sensor Networks

UDP User Datagram Protocol

Wi-Fi Wireless Fidelity
WLAN Wireless Local Area Network

Capítulo 1

Introdução

Com o recente e exponencial avanço tecnológico, é possível verificar que as soluções sem fios demonstram ser cada vez mais proeminentes no mundo atual. Em relação às soluções com fios, a praticabilidade e comodidade das supra-mencionadas são fatores de destaque, visto que ambas permitem a obtenção de uma qualidade de utilização muito próxima.

Esta popularidade deriva da reprodução de áudio, tanto em auriculares sem fios como em sistemas de reprodução de áudio de alta-fidelidade sem fios. É importante realçar que, apesar da crescente inovação nesta área, as soluções existentes são exclusivas ao fabricante, significando que cada um possui a sua própria solução para distribuição e sincronização de áudio em redes sem fios, sendo esta uma característica distintiva e inerente ao *hardware* produzido.

Considerando estes aspetos, surge a necessidade de desenvolver um sistema capaz de reproduzir áudio multi-canal em redes sem fios independente do hardware utilizado e suportado na sua globalidade em *software* de código aberto. A implementação desta solução envolverá vários clientes musicais - idealmente um por cada canal de áudio - e será capaz de lidar com a dessincronização entre eles e distribuir eficientemente cada canal de áudio pelo respetivo cliente musical.

1.1 Objetivos

Este trabalho tem como principal objetivo desenvolver um sistema capaz de sincronizar a reprodução de áudio em diferentes dispositivos numa rede sem fios. De forma a atingir esta finalidade será necessário cumprir os seguintes parâmetros:

- Investigar e entender os limites da perceção auditiva humana;

- Aperfeiçoar um mecanismo para calcular, com precisão, a dessincronização temporal entre diferentes dispositivos em redes sem fios;
- Estudar os atuais protocolos de transporte;
- Definir e justificar quais os protocolos de comunicação e algoritmos mais apropriados para a solução;
- Estudar os atuais protocolos de distribuição de conteúdos multimédia;
- Desenvolver um protocolo de distribuição de áudio multi-canal;
- Distribuir eficientemente os dados de áudio pelos vários dispositivos;
- Calcular o instante inicial de reprodução nos diferentes dispositivos;
- Sincronizar ativamente a reprodução de áudio nos vários dispositivos;
- Desenvolver um algoritmo de manipulação da taxa de reprodução do áudio;
- Desenvolver um protocolo para controlar a reprodução de áudio, isto é, iniciar, pausar ou parar a reprodução nos vários dispositivos no mesmo instante;
- Desenvolver duas aplicações finais que combinem os aspetos previamente mencionados.

1.2 Estrutura do Documento

Este documento encontra-se divido em seis capítulos:

- **Introdução**
  Descrição, contextualização e motivação do projeto e objetivos a alcançar;
- **Estado de arte**
  Descrição dos conceitos fundamentais da perceção auditiva humana, da dessincronização temporal e das tecnologias relacionadas com a distribuição de áudio e semelhantes ao projeto;
- **Solução proposta**
  Exposição das etapas do desenvolvimento do projeto, nomeadamente decisões e as suas designadas justificações, bem como as especificações dos protocolos desenvolvidos;
- **Implementação**
  Explicação das estruturas e do funcionamento das aplicações desenvolvidas;

- **Testes e resultados**

  Descrição dos testes realizados nas aplicações desenvolvidas e os resultados obtidos no final do projeto;

- **Conclusão**

  Apresentação das considerações finais e possível continuidade do projeto de forma a progredir e melhorar o trabalho realizado.

Capítulo 2

Estado da Arte

Neste capítulo são estudados e descritos todos os princípios científicos e tecnologias essenciais ao desenvolvimento do projeto. O enquadramento teórico começa por abordar os limites da audição humana e o funcionamento dos osciladores de cristal. Posteriormente, são analisados e comparados os protocolos e algoritmos de sincronização temporal mais relevantes, e os protocolos de transporte. Por fim, são apresentadas algumas tecnologias semelhantes e trabalhos relacionados com o tema.

2.1 Audição Binaural referencia

A audição binaural [12] é a capacidade de interpretação de sons através de ambos os ouvidos. Esta capacidade é também responsável pela localização espacial, pelo efeito de precedência, pela fusão binaural dos sons e pela perceção da fala.

2.1.1 Fusão Binaural dos sons

A fusão binaural é um efeito percetivo que permite fundir sons de forma a que os ouvintes percebam apenas um evento auditivo. Este fenómeno evita que um pequeno eco, ou que duas fontes de áudio que transmitem os mesmos sons, causem múltiplas imagens sonoras ao ouvinte, permitindo uma perceção auditiva única e coerente. É importante notar que esta capacidade não é ilimitada. O artigo [17] refere uma experiência na qual múltiplos ouvintes foram submetidos, num ambiente controlado, a fornecer uma impressão subjetiva da quantidade de sons "clique"conseguiam detetar em diferentes condições temporais. Em intervalos curtos de aproximadamente 5 milissegundos, a maioria dos ouvintes relata ouvir apenas um som mas, à medida que esse intervalo aumentava, um maior número de sujeitos explica

ouvir dois sons. Ao alcançar o intervalo de 8 a 10 milissegundos, praticamente 100% dos ouvintes foi capaz de distinguir os dois sons. É também referido que, além da desfasagem, existem outras mudanças percetivas, e que estes resultados variam com o tipo de sinal, nível do sinal, direção das fontes e se os sons são apresentados em campo livre ou em auriculares.

A medida mais comummente utilizada para definir o limite temporal da perceção de um som fundido de dois sons é o limite do eco. Os limites do eco possuem uma grande variação (entre os 2 e os 50 milissegundos), uma vez que estão dependentes do tipo do estímulo. Em breves estímulos, os limites do eco demonstram ser mais curtos, enquanto que em estímulos mais duradouros, como o ruído ou a fala, os limites do eco são maiores.

2.2 Dessincronização temporal

Atualmente, todos os dispositivos com uma unidade central de processamento possuem um gerador temporal interno responsável pelo processamento organizado de múltiplas tarefas. Este gerador temporal origina impulsos elétricos com uma determinada frequência que é comercialmente referida como a 'frequência do *Clock*' do CPU [9]. Além deste gerador, é necessário garantir a existência de uma fonte de referência com uma precisão confiável e, para o efeito, a melhor opção é um oscilador de quartzo. Este oscilador é essencial para estabilizar a frequência do CPU. Apesar destes osciladores possuírem uma precisão confiável, esta não é absoluta, o que faz com que ao longo do tempo o valor temporal gerado pelo relógio se afaste do valor temporal real.

2.2.1 Frequência dos osciladores de cristal

A grande vantagem dos osciladores de cristal [13] é o facto de possuírem uma grande estabilidade na sua frequência, no entanto, esta não pode ser alterada com muita precisão, uma vez que são mecanismos extremamente frágeis. Devido à condição descrita, múltiplos fatores podem afetar a estabilidade da sua frequência tais como variações climáticas, como a temperatura, a humidade, e a pressão, e variações na fonte de alimentação.

Partindo destes fatores, existem dois tipos principais de erros temporais que podem ser usados para medir a variação das frequências: a diferença das frequências operacionais (ou o deslocamento de tempo) descrito pelo termo *Skew* ou *Clock Drift*, e a diferença temporal entre relógios descrita pelo termo *Offset*.

2.2.2 Oscilador de cristal e a reprodução de áudio

Atualmente, todos os sistemas de áudio possuem um oscilador de cristal responsável por garantir as frequências de amostragens corretas, tanto na reprodução como na gravação do áudio.

Quando se trata de sistemas de áudio mais complexos com vários subsistemas, estes devem estar referenciados na mesma fonte temporal, ou seja, no mesmo relógio ou oscilador de cristal, uma vez que diferentes relógios possuem diferentes frequências operacionais, como foi referido anteriormente. Estas diferenças causam o deslocamento temporal (*Skew*) que consequentemente conduz a um desalinhamento acústico ou, por outras palavras, dessincronização na reprodução do áudio.

2.3 Sincronização de conteúdo multimédia

Esta área de sincronização de conteúdos multimédia é um domínio de crescente popularidade nas últimas décadas, existindo um maior conhecimento e soluções face a esta problemática.

Segundo [6], a sincronização de conteúdos multimédia encontra-se dividida em três tipos:

- *Intra-stream synchronization* - Manutenção de cada fluxo de media temporalmente no recetor. O recetor precisa de garantir que durante a reprodução não existem interrupções e que a taxa de reprodução original é garantida;
- *Inter-stream synchronization* - Sincronização de fluxos de diferentes pertencentes à mesma aplicação, também conhecido por *lip-synchronization* (ex: sincronização de áudio com imagem);
- *Inter-destination synchronization* Sincronização de diferentes fluxos em diferentes dispositivos ao mesmo tempo.

2.4 Protocolos e algoritmos de sincronização temporal

Sabendo que este projeto visa implementar um protocolo que permita reproduzir áudio multi-canal em vários dispositivos, e tendo em conta os aspetos referidos anteriormente, percebe-se que é necessário sincronizar ou calcular a dessincronização com enorme precisão. De forma a alcançar este objetivo foram estudados múltiplos protocolos e algoritmos de sincronização temporal nos trabalhos anteriores [2] e [7]. Estes trabalhos foram a principal base de estudo para o trabalho aqui desenvolvido.

2.4.1 *Network Time Protocol*

O *Network Time Protocol* [11] é o protocolo mais comum e mais utilizado para sincronizar os relógios da grande parte dos computadores e outros dispositivos.

Este protocolo possui uma arquitetura do tipo de cliente-servidor para o funcionamento do algoritmo do protocolo e possui ainda uma outra arquitetura hierárquica chamada de *Stratum Levels*. Nesta hierarquia existe, na primeira camada, a *Stratum-0*, onde normalmente existe um dispositivo ligado a um relógio atómico que é utilizado como referência temporal. Na seguinte camada, *Stratum-1*, existe um servidor

ligado a este recetor. De seguida, na *Stratum-2*, existe um outro servidor ligado ao servidor da camada anterior, e assim sucessivamente para as seguintes camadas. Portanto sabe-se que à medida que a camada aumenta a precisão vai diminuindo.

O seu funcionamento permite que o cliente calcule o valor de dessincronização (ou *Offset*) em relação ao servidor, podendo assim adaptar o seu relógio local.

O algoritmo que este implementa para calcular o *Offset* consiste nos seguintes passos:

1. Pedido temporal por parte do cliente, o cliente guarda o valor do instante (t0);
2. O servidor na receção do pedido regista o instante (t1);
3. O servidor depois responde, e na resposta regista o instante de resposta (t2) e o instante da receção do pedido;
4. O cliente na receção da resposta regista o instante (t3);
5. O cliente calcula o *Offset* através da fórmula 1.

$$offset = ((t1 - t0) - (t2 - t3))/2 \tag{1}$$

Este protocolo oferece uma precisão na ordem de 1 milissegundo em redes pequenas ou locais, e na ordem das dezenas de milissegundos na *Internet*.

2.4.2 *Precision Time Protocol*

O *Precision Time Protocol* [8] é um protocolo de sincronização utilizado para sincronizar relógios em redes de área local (LAN). Por ser usado em LAN, a latência da rede é mais previsível, ajudando, desta forma, a melhorar a precisão. Este possui uma arquitetura do tipo *master-slave*, onde o *master* possui o *Master-Clock* que é utilizado como a referencia temporal absoluta.

O processo de seleção do *master* consiste na determinação do dispositivo com o relógio mais preciso. Este processo funciona através de uma eleição entre os sistemas onde são avaliados os seguintes atributos:

1. *Priority* 1 - Parâmetro definido pelo administrador que permite a escolha do sistema *master*;
2. Classe do relógio;
3. Precisão do relógio;
4. *Priority* 2 - Parâmetro definido pelo administrador que permite a escolha do sistema *master* quando os restantes atributos são equivalentes;

5. 'ID' único - Usado para desempatar quando os restantes atributos são equivalentes.

Este protocolo implementa um algoritmo semelhante ao descrito anteriormente, em 2.4.1, no entanto, diverge no facto do *master* ser quem inicia o processo de sincronização e quem calcula o *Offset*. Após o cálculo, o *Offset* é comunicado ao respetivo *slave*.

Este protocolo oferece uma precisão na ordem dos 100 nanossegundos até às dezenas de microssegundos.

2.4.3 *Reference Broadcast Synchronization*

O *Reference broadcast synchronization* (RBS) [18] é um protocolo de sincronização que utiliza a camada física para fazer anúncios com os dados de sincronização.

O funcionamento deste protocolo difere em relação aos mais comuns. De uma forma simples, imagina-se dois nós participantes que recebem um *Beacon* vindo de um nó mestre. Estes dois registam o instante em que recebem o *Beacon* e depois comparam os instantes entre eles, calculando o respetivo *Offset*. Manifestamente, se existir mais nós participantes para sincronizar haverá mais que um anúncio, sendo assim a quantidade de anúncios proporcionalmente direta à precisão de sincronização.

A grande vantagem deste protocolo é a eliminação da incerteza do remetente. Não obstante, este protocolo sofre quando o nó mestre falha levando à eleição de um novo nó mestre. Quando a rede é fisicamente grande e possui intermediários (*Router, Switch, Hub*), resultam tempos de propagação mais longos.

2.4.4 *Delay Measurement Time Synchronization*

O *Delay Measurement Time Synchronization* (DMTS) [14] é um protocolo de sincronização semelhante ao *Reference broadcast synchronization* 2.4.3, no entanto, este tenta minimizar todos os possíveis atrasos envolvidos e a troca de mensagens entre os nós vizinhos.

O seu funcionamento também se baseia em anúncios mas, para que este consiga minimizar os atrasos do lado do emissor, regista apenas o instante temporal quando o canal de transmissão está livre. Desta forma, os restantes atrasos envolvidos na transmissão estão do lado do recetor. O recetor regista o instante da chegada do anúncio à interface e, de seguida, após ler o anúncio, volta a registar o instante. Assim, é possível calcular todos os atrasos e sincronizar o relógio local.

O DMTS possui a mesmas vantagens e desvantagens que o RBS, e ainda uma menor complexidade computacional no cálculo dos atrasos.

2.4.5 *Timing-sync Protocol for Sensor Networks*

O *Timing-sync Protocol for Sensor Networks* (TPSN) [18] é um protocolo de sincronização concebido para redes de sensores. Este possui uma arquitetura servidor-cliente e organiza a rede numa topologia de árvore.

O funcionamento deste consiste em duas fases. A primeira fase corresponde à fase de descoberta, e a segunda de sincronização. Para começar a fase de descoberta é necessário existir o nó raiz, tipicamente o nó que possui o relógio mais preciso. Este nó raiz inicia a fase de descoberta enviando para os nós vizinhos o seu respetivo nível (nível 0). Com a receção desta informação, os vizinhos sabem que pertencem ao nível 1, e este processo continua até todos os nós envolvidos possuírem um nível. A fase de sincronização é semelhante, inicialmente começa entre o nível 0 e o nível 1, depois os nós de nível 1 com os nós de nível 2, e assim sucessivamente.

O algoritmo sincronização implementado é semelhante ao descrito em 2.4.1.

Este protocolo tem como grande vantagem a obtenção de uma sincronização com grande precisão nos vários sensores, e os registos dos instantes são efetuados na camada de acesso ao meio (MAC). Mas, por outro lado, não é aplicável em redes com frequentes alterações na sua topologia.

2.4.6 *Flooding Time Synchronization Protocol*

O *Flooding Time Synchronization Protocol* (FTSP) [18] é um protocolo semelhante ao TPSN, mas que procura corrigir e melhorar as suas desvantagens.

A principal diferença entre estes protocolos é o facto do FTSP não necessitar de uma topologia em árvore, dado que este possui um nó mestre que faz os anúncios para todos os nós que se encontram a seu alcance. O anúncio enviado possui a sua informação temporal e esta mensagem é apenas registada com o instante temporal no momento de transmissão. Os nós recetores registam o instante no qual recebem este anúncio e, a partir dos dois instantes calculam o respetivo *Offset*. Caso os nós recetores possuam nós vizinhos fora do alcance do nó mestre, estes replicam o processo para sincronizar todos os nós participantes. O nó mestre é periodicamente reeleito.

As grandes vantagens deste protocolo são a sua robustez e dinâmica. A robustez impede que a falha de uma ligação ou de um nó cause problemas para a sincronização dos restantes nós, e a sua dinâmica permite mudanças na topologia da rede e no seu escalonamento. Este protocolo oferece também uma precisão na ordem dos microssegundos.

2.4.7 *Cristian's Algorithm*

O *Cristian's algorithm* [19] é um método de sincronização de relógios utilizado em redes com baixa latência.

O seu funcionamento é simples e consiste na troca de duas mensagens entre um cliente e um servidor. O cliente emite um pedido ao servidor e regista o instante (*t0*) de envio. De seguida, o servidor regista o momento (*t_server*) da receção do pedido e envia-o como resposta ao cliente. O cliente, ao receber esta mensagem, regista o instante (*t1*). Através destes três instantes, o cliente consegue calcular o seu tempo sincronizado com servidor. Este cálculo é efetuado segundo a fórmula 2.

$$t\_client = t\_server + (t1 - t0)/2 \tag{2}$$

2.4.8 *Berkeley Algorithm*

O *Berkeley Algorithm* [19] é uma técnica de sincronização de relógios utilizado em sistemas distribuídos. Esta técnica assume que nenhuma máquina possui uma fonte temporal precisa.

O funcionamento desta técnica começa por selecionar o nó mestre, utilizando um algoritmo de eleição. Depois, periodicamente, o nó mestre calcula o tempo nos nós participantes através do *Cristian's Algorithm* (2.4.7). Após calcular os tempos em todos os nós participantes, o nó mestre calcula a média destes e calcula a diferença desta média com o seu tempo local. Por fim, esta diferença é somada ao seu tempo local e enviado para todos os nós participantes.

2.4.9 Sincronização em redes sem fios

Todos os protocolos anteriormente referidos podem ser aplicados em redes sem fios, mas nem todos têm em consideração os atrasos envolvidos, podendo resultar num desempenho e precisão diferentes. Partindo do artigo [10], os atrasos envolvidos em redes sem fios encontram-se avaliados na tabela 1.

Tabela 1: Avaliação dos atrasos em redes sem fios

Atraso	**Tipo de atraso**	**Descrição**
Tempo de envio	Não determinístico	Tempo de preparação da mensagem para as camadas inferiores
Tempo de acesso ao meio	Não determinístico	Tempo de espera que o canal de transmissão fique livre
Tempo de transmissão	Determinístico	Tempo de escrita da mensagem na 'interface' radio
Tempo de propagação	Determinístico (negligenciável em distâncias inferiores a 300 metros)	Tempo de transmissão da mensagem pelo ar
Tempo de receção	Determinístico	Tempo da leitura da mensagem na 'interface' radio
Tempo de receção	Não Determinístico	Tempo de chegada da mensagem à camada correspondente

Deste modo, a eliminação ou minimização dos atrasos não determinísticos faz com que a eficiência e precisão do protocolo aumente consideravelmente em redes sem fios.

2.5 Reprodução áudio em redes sem fios

A reprodução de áudio em redes sem fios é uma área em constante evolução [4]. A exploração desta área começou com a utilização de dispositivos rádio FM e com a consequente descoberta e utilização de dispositivos Bluetooth e Wi-Fi.

Com este avanço tecnológico surgiu também a necessidade de melhorar a qualidade do áudio nestes dispositivos e, para tal, foram desenvolvidos CODECs que funcionam com base na rentabilização da largura de banda limitada.

Nos últimos anos, os fabricantes deste tipo de dispositivos direcionar o seu interesse para o Wi-Fi, uma vez que esta tecnologia está implementada em todos os dispositivos, oferece uma largura de banda superior e possui melhores alcances. No entanto, ainda não existem soluções padronizadas para a distribuição de áudio como existe no Bluetooth, implicando que o desenvolvimento destas fique sobre a responsabilidade do fabricante, tornando-as, na maior parte dos casos, soluções proprietárias.

Os principais desafios na transmissão de áudio em redes sem fios passam principalmente pela sincronização dos dispositivos e pela qualidade da ligação, onde se destacam os seguintes aspetos:

- Desempenho da 'interface' rádio.
- Largura de banda.
- Latência e *jitter* da rede.
- Perda de pacotes.
- Interoperabilidade.

2.6 Protocolos de Transporte

2.6.1 *Transport Control Protocol*

O *Transport Control Protocol* (TPC) [15] é o padrão de transmissão de dados atuais da Internet, apesar de que nos últimos anos tenham surgido diversas tecnologias que vieram substituir algumas das aplicações deste protocolo.

Uma vez que este garante que todos os dados por ele enviados chegam ao destino de forma ordenada e sem erros, rapidamente se tornou no protocolo padrão da Internet. O TCP estabelece uma ligação entre

o cliente e servidor e lida com as perdas, corrupção ou duplicação de dados, ajustando-se às condições da rede (quando a rede está mais congestionada, adapta a taxa de envio). Estes mecanismos estão abstraídos, fornecendo assim uma utilização simples. Embora estes sejam ideais para uma transmissão fiável de dados, possuem um custo de complexidade e tempo de processamento.

2.6.2 *User Datagram Protocol*

O *User Datagram Protocol* (UDP) [16] é utilizado para serviços de tempo real como, por exemplo, jogos e videoconferências. Este protocolo é melhor no diz respeito a desempenho e latências, porque não possui mecanismos extras, sendo assim orientado ao datagrama. Como tal, os dados enviados podem ser perdidos, corrompidos ou chegar desordenados e, por esta razão, as implementações com UDP normalmente acabam por ter mecanismos adicionais. As principais razões que causam a falha na entrega de dados são a segmentação dos pacotes, interferência no meio de transmissão, a sobrecarga nos nós da rede e a memória disponível nas 'interfaces' de rede.

Este protocolo pode ser uma excelente opção dependendo das necessidades da aplicação, podendo fazer adaptações externas para melhorar a fiabilidade.

2.6.3 *Datagram Congestion Control Protocol*

O *Datagram Congestion Control Protocol* [5] é um protocolo não orientado à ligação, desenvolvido para propor soluções no âmbito do tráfego em tempo real e multimédia.

Este protocolo implementa mecanismos de controlo de congestionamento sem perder consideração pela entrega atempada dos dados, priorizando desta forma as mensagens mais recentes às mensagens mais antigas, e de fiabilidade na entrega dos dados. Este é um protocolo versátil, pois permite a definição do mecanismo mais apropriado, não tornando necessário a implementação destes pela parte da aplicação.

2.6.4 *Stream Control Transport Protocol*

O *Stream Control Transport Protocol* [21] é um protocolo orientado à ligação, sendo atualmente reconhecido como uma versão melhorada do TCP.

Este protocolo fornece uma transmissão de dados fiável, ordenada, com mecanismos de controlo de congestionamento e com melhor latência. Outras novidades que este protocolo traz são o *multi-homing*, esta é uma característica particularmente interessante para redes sem fios porque permite que o dispositivo mude de rede sem perder a ligação, o *multi-streaming*, transmissão de vários fluxos de dados independentes e proteção contra ataques DoS.

2.6.5 *Real-time Transport Protocol*

O *Real-time Transport Protocol* (RTP) [1] é um protocolo aplicacional para o transporte de dados multimédia em redes IP. Normalmente utiliza o UDP, mas pode ser configurado para utilizar TCP.

É o protocolo '*standard*' para transporte de áudio e vídeo em tempo real na Internet. Fornece propriedades de transferência de dados continua em tempo real, com suporte de carimbos temporais para a reprodução atempada dos dados, deteção de perdas e desordenação, segurança, identificação dos conteúdos, grupos *Multicast* e suporta vários *codec's* de multimédia. Por outro lado, não garante poupança na largura de banda nem qualidade do serviço.

O cabeçalho dos pacotes deste protocolo possuem os seguintes parâmetros:

- *Version*: campo com tamanho de 2 bits. Indica a versão do protocolo.
- *P(Padding)*: campo com tamanho de 1 bit. Indica se o pacote possui *padding*.
- *X(Extension)*: campo com tamanho de 1 bit. Indica a presença de um cabeçalho extra especifico à aplicação, entre o cabeçalho e os *payload*.
- *CC(Contributor Source Count)*: campo com 4 bits. Indica o número de fontes de contribuição.
- *M(Marker)*: campo com tamanho de 1 bit. Indica um sinal especifico à aplicação, por exemplo, os dados em questão possuem uma relevância especial.
- *PT(Payload Type)*: campo com tamanho de 7 bits. Indica o formato do *payload*.
- *Sequence Number*: campo com tamanho de 16 bits. Indica o número de sequência do respetivo pacote.
- *Timestamp*: campo com tamanho de 32 bits. Indica o instante em que os dados devem ser reproduzidos. Caso o pacote chegue mais tarde que o respetivo instante, este deve ser ignorado.
- *SSRC(Synchronization source identifier)*: campo com tamanho de 32 bits. Identifica unicamente a fonte de um fluxo.
- *CSRC(Contributing Source)*: campo com um tamanho de 32 bits. Identifica a fontes de contribuição do respetivo fluxo de dados.

O RTP funciona a par com o *Real-time Tranport Control Protocol* que, fornece dados para a gestão da sessão e controlo da qualidade do serviço.

2.7 Protocolos de controlo de reprodução de conteúdos multimédia

2.7.1 *Real Time Streaming Protocol*

O *Real Time Streaming Protocol* (RTSP) [1] é um protocolo aplicacional, utilizado para o controlar a transmissão de dados em tempo real, como se fosse um controlo remoto.

É um protocolo sem noção de ligação, possuindo apenas um identificador de sessão, apesar deste poder ser implementado sobre TCP. Isto significa que o cliente tanto pode abrir e fechar as ligações com o servidor sempre que envia um pedido ou pode simplesmente ser utilizado com UDP. Este protocolo foi concebido para ser utilizado com o RTP, para fornecer um serviço multimédia completo na Internet.

Atualmente os comandos disponíveis no RTSP são os seguintes:

- *OPTIONS* - Uma das entidades informa a outra das opções que pode aceitar.
- *DESCRIBE* - O cliente pede a descrição dos dados multimédia da sessão.
- *ANNOUNCE* - Quando é enviado do cliente para o servidor, anuncia a descrição dos dados multimédia da sessão. Quando é enviado do servidor para o cliente, é uma atualização da descrição da sessão em tempo real.
- *SETUP* - O cliente pede ao servidor para alocar recursos para uma transmissão, sendo assim iniciada a sessão.
- *PLAY* - O cliente pede ao servidor para iniciar a transmissão de dados.
- *PAUSE* - O cliente pede ao servidor para pausar a transmissão de dados.
- *TEARDOWN* - O cliente pede ao servidor para terminar a transmissão de dados e libertar os recursos associados.
- *GET_PARAMETER* - Pedir o valor de um parâmetro.
- *SET_PARAMETER* - Definir o valor de um parâmetro.
- *REDIRECT* - O servidor informa o cliente que tem de se ligar a outro servidor.
- *RECORD* - O cliente inicia uma gravação multimédia, conforme a descrição da sessão.

2.8 Soluções alternativas

Neste subcapítulo foram estudadas soluções semelhantes à que este projeto visa implementar. Todas estas são soluções são *market ready*, de fácil montagem e funcionamento. É de notar que todas estas soluções possuem mecanismos proprietários, tornando os problemas que este projeto envolve mais difícil de resolver. Algumas destas também possuem *hardware* próprio tornando este tipo soluções únicas a cada manufator. Com isto, surge a necessidade de desenvolver uma solução aberta, livre da necessidade de *hardware* e cem por cento apoiado no software.

2.8.1 *Wireless Speaker Audio Association*

A *Wireless Speaker Audio Association* (WiSA)[22] é uma associação que atualmente produz vários tipos de produtos, mas destaca-se neste contexto uma vez que desenvolveu um protocolo para distribuição de áudio multicanal sincronizado em redes sem fios, utilizando tecnologia própria.

Atualmente possui duas tecnologias de distribuição de áudio: uma que suporta quatro canais de áudio não compresso amostrado a 16-bit 48 KHz sobre redes Wi-Fi comuns, possuindo uma latência média de 40 milissegundos, uma sincronização por canal média de 20 microssegundos e não existem perdas de dados devido a algoritmos de correção de erros. A outra tecnologia suporta oito canais de áudio não compresso amostrado a 24-bit 48/96 KHz, sobre uma rede '*Wireless*' dedicada. Esta rede é a U-NII 5 GHz Spectrum, através desta é possível obter uma latência media de 5,2 milissegundos e uma sincronização media de 1 microssegundo por canal. Por usar uma rede 'Wireless' dedicada conseguem alterar facilmente o canal áudio se for detetada alguma interferência e manter desta forma uma comunicação livre de congestionamentos.

2.8.2 SONOS

A SONOS[20] é uma empresa focada no desenvolvimento de produtos de áudio, e foi também das primeiras a desenvolver tecnologias para distribuição de fluxos de áudio em redes sem fios.

Atualmente possuem produtos de distribuição áudio multicanal, mas os detalhes técnicos deste não estão publicamente divulgados como a WiSA.

2.8.3 Sony

A Sony é uma das maiores empresas de fabrico de produtos eletrónicos atualmente, e enquadra-se neste contexto por um produto recentemente lançado, o HT-A9.

Este produto é constituído por quatro dispositivos reprodutores e um dispositivo que os controla. A comunicação entre estes é feita por Wi-Fi e Bluetooth. Além dos vários formatos de áudio suportados, não são referidas mais informações técnicas.

2.8.4 Apple

A Apple [3] além de ser conhecida por todos os seus produtos também é conhecida pelo seu 'software', e mais em específico neste contexto pelo protocolo AirPlay.

Este protocolo iniciou-se como um protocolo de distribuição de fluxos de áudio em redes sem fios, agora possui muitas mais funcionalidades. Sabe-se que para a distribuição de áudio utiliza um protocolo proprietário conhecido por *Remote Audio Output Protocol* (RAOP) baseado nos protocolos RTP/RTSP. Este funciona em redes Wi-Fi. Também suporta áudio não compresso amostrado a 16-bit 44.1 KHz. Atualmente as funcionalidades suportadas pelo protocolo relacionadas com este projeto são, a capacidade de reproduzir o mesmo áudio sincronizado em diferentes dispositivos e a capacidade suportar dois canais de áudio, isto é, cada canal reproduzido num dispositivo diferente.

2.9 Outros casos de estudo

No âmbito da distribuição e reprodução de áudio multicanal sincronizado em redes sem fios, foram também estudados outros trabalhos semelhantes, dos quais se destacam os [2] e [7].

Começando pelo trabalho [2], foi comprovado que é possível sincronizar o início da reprodução de áudio nos vários dispositivos, em redes sem fios, utilizando apenas protocolos livres, nomeadamente o NTP e o PTP. Neste foi detetado também, que a constante sincronização dos relógios aumenta a instabilidade dos relógios, tornando esta o atividade desaconselhada para este objetivo, e que as condições atmosféricas também influenciam a estabilidade do mesmo. Com estes aspetos em consideração, neste foi decidido utilizar o PTP para monitorização contínua da dessincronização e ao invés de tentar atualizar os relógios, utilizar as diferenças calculadas para calcular a deriva temporal (*Time Skew*). E utilizar esta deriva para manter a sincronização do áudio através de adição e remoção de amostras de áudio. Neste trabalho foi concluído que a melhor solução para o problema seria desenvolver um simples algoritmo que calcula-se esta deriva e que fosse capaz de remover e adicionar, de uma forma proporcional à deriva calculada.

No trabalho [7], foi alcançado o objetivo da reprodução de sincronizada de áudio multicanal em redes sem fios. Este começou por estudar a forma de como o tempo flui nos diferentes relógios dos dispositivos e a influência dessa diferença na reprodução. E com base do trabalho anterior [2] e nas suas conclusões, foi desenvolvido um algoritmo para calcular as diferenças temporais e as respetivas derivas ao longo do tempo numa monitorização continua. Este algoritmo utiliza a distribuição normal(ou Gaussiana) para a filtragem das várias diferenças temporais calculadas. Depois calcula um valor médio através do intervalo de amostras obtidas. Por fim, para calcular a deriva utiliza a regressão linear com valores médios obtidos para estimar a deriva temporal. A solução desenvolvida enquadra-se no problema e fornece valores estáveis, não sofre de fracas estimações de diferenças temporais. Foi utilizado o mesmo mecanismo de adição e

remoção de amostras para manter a sincronização ao longo do tempo. Por fim neste trabalho foi também desenvolvida uma aplicação com as soluções propostas, para ser possível testar o funcionamento final. E foram desenvolvidos também mecanismos de controlo da reprodução e de distribuição.

Capítulo 3

Solução Proposta

Partindo dos conhecimentos adquiridos no capítulo anterior 2, é possível saber quais as tecnologias e protocolos farão sentido implementar, e utilizar para delinear um conjunto de objetivos e requisitos a cumprir. Neste capitulo serão justificadas todas as escolhas feitas relacionadas com a sincronização temporal, por outras palavras cálculo da dessincronização, a estratégia utilizada para a adaptação da taxa de reprodução do áudio, a técnica de distribuição de áudio multicanal e o protocolo de controlo da reprodução de áudio.

A solução proposta consiste primariamente em três aspetos, o primeiro é o cálculo da dessincronização instantânea entre o dispositivo servidor e cada dispositivo reprodutor, e a partir dos valores calculados, calcular a dessincronização ao longo do tempo. O segundo aspeto é a adaptação da taxa de reprodução do áudio, isto traduz-se em esticar o comprimir o áudio temporalmente, para que os dispositivos reproduzam o áudio de forma síncrona ao longo do tempo. Por fim, o terceiro aspeto trata-se da distribuição do áudio multicanal sobre redes sem fios.

O principal objetivo deste projeto é construir uma solução puramente em *Software*, que funcione em qualquer tipo de *Hardware* independente das suas características. Portanto, esta solução será construída com algoritmos de baixa complexidade, e protocolos de comunicação simples, utilizando recursos já existentes. Esta solução será implementada em Linux e utilizará a API do ALSA para a reprodução do áudio. Cumprindo estes requisitos esta solução será facilmente implementada em qualquer sistema Linux e em qualquer rede IP, suportando redes sem fios e redes com fios, pois esta é independente da camada física.

3.1 Escolha do protocolo de transporte

Com o conhecimento adquirido pelo estudo dos vários protocolos de transporte no subcapítulo 2.6, é possível determinar o que vai ser usado para cada protocolo e quais as suas as razões.

Iniciando pelo protocolo de sincronização ou de cálculo da dessincronização temporal, sabe-se que este necessita de possuir a menor latência e complexidade possível, evitando assim a existência atrasos externos à própria comunicação. Sabe-se também que este não precisa de ser sensível a falhas ou perdas, pois no caso de uma ocorrência desta natureza o cálculo da dessincronização não será afetado. Com estes requisitos definidos o protocolo de transporte utilizado será o UDP.

Depois para o protocolo de distribuição de áudio multicanal sobre redes sem fios, existe um paradigma diferente com um conjunto de requisitos cumprir. Em primeiro lugar, é necessário garantir que a taxa de reprodução do áudio é superior à taxa de transmissão dos dados, portanto a garantia de entrega ou a retransmissão de pacotes está limitada. Apesar de a retransmissão estar limitada, é necessário que esta exista, porque as redes sem fios são mais subjetiveis a falhas que as redes com fios. Tal como o protocolo anterior, é necessário garantir uma baixa latência e complexidade. Partindo destes requisitos e analisando em pormenor o protocolo mais utilizado atualmente para a distribuição de conteúdo multimédia, o RTP 2.6.5, entende-se que este possui mais mecanismos que os que necessários para este protocolo que não foi desenvolvido para transferir canais de áudio independentes para diferentes dispositivos, mas é algo implementável com este. Por estas razões será adotado o protocolo de transporte UDP, pois permite desenvolver um mecanismo mais específico para esta solução.

Por fim, para o protocolo de controlo da reprodução do áudio, não serão necessários requisitos muito específicos. O principal requisito será a garantia de entrega dos pacotes e baixa latência. Para este propósito o protocolo de transporte TCP funcionaria perfeitamente, mas para garantir uma menor latência é mais viável utilizar o protocolo de transporte UDP. Para garantir a entrega dos pacotes serão implementados mecanismos ao nível da aplicação para esse efeito.

3.2 Arquitetura

A arquitetura proposta é constituída por um sistema servidor, que será o controlador ou gestor, e por um conjunto de sistemas cliente, que serão os reprodutores do áudio. Esta arquitetura encontra-se representada na figura 1.

Respetivamente ao cálculo da dessincronização temporal, este papel será da responsabilidade do sistema servidor que deverá comunicar as respetivas diferenças temporais aos sistemas clientes respetivos. No que lhe concerne, os sistemas clientes deverão utilizar estas diferenças para calcular a dessincronização total ao longo do tempo.

Depois a distribuição do áudio e controlo do áudio será responsabilidade do sistema servidor, este

irá enviar mensagem de controlo da reprodução, incluindo comunicar o instante e a sequência em que o sistema cliente deverá iniciar a reprodução do áudio. Este deverá também gerir a distribuição atempada do áudio por cada sistema cliente. No que lhe concerne, o sistema cliente irá respeitar as ordens do servidor e reproduzir o áudio.

Por fim, para gerir a sincronização da reprodução do áudio ao longo do tempo, os sistemas clientes deverão utilizar os valores calculados da dessincronização total ao longo do tempo para gerir a taxa de reprodução do áudio.

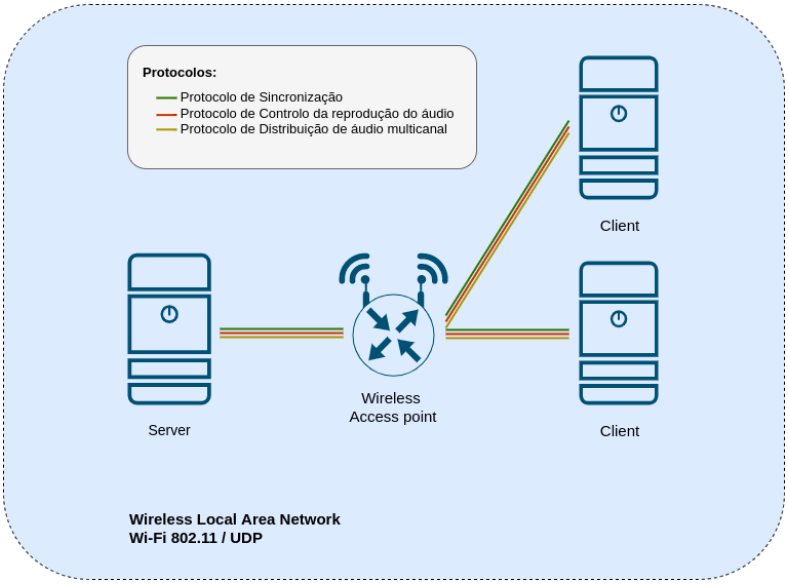


Figura 1: Arquitetura proposta

3.3 Cálculo da dessincronização

Nas dissertações anteriores [2] e [7], foi elaborado um algoritmo para calcular a dessincronização ou diferença temporal (também vulgarmente conhecido como *Offset*) entre dois dispositivos, neste caso sendo o dispositivo controlador e o dispositivo reprodutor de áudio. Este consiste na diferença de duas marcas temporais, correspondentes a cada dispositivo, do mesmo instante. Como estes dispositivos não se encontram conectados fisicamente, houve a necessidade de usar outras medidas para o cálculo da dessincronização.

Sabendo que as comunicações nas redes sem fios não são instantâneas, foi necessário ter em conta todos os atrasos existentes. Isto é, quando o dispositivo reprodutor recebe uma mensagem do dispositivo controlador, sabe-se que existem vários atrasos até a mensagem chegar, portanto, não é possível garantir que o instante em que a mensagem chega é o mesmo de quando a mensagem sai. Por outro lado, se o dispositivo reprodutor responder, no instante em que recebe a mensagem, quando esta chega ao dispositivo controlador, este consegue calcular a duração total da troca de mensagens, fenómeno conhecido

como RTT (*Round Trip Time*). Portanto, é possível calcular com exatidão o atraso total.

Este algoritmo foi baseado neste principio e executado da seguinte forma:

1. O dispositivo controlador inicia este processo enviando um pacote e registando o respetivo instante (interpretado por *t1*);
2. Quando o dispositivo reprodutor recebe esta mensagem regista o instante (*t2*) e envia-o para o dispositivo controlador;
3. Por fim, na chegada desta mensagem o dispositivo controlador regista o instante (*t3*) e guarda o instante (*t2*) recebido na mensagem.

Este processo está representado através de um diagrama de sequência na figura 2 e a seguir na formula 3.

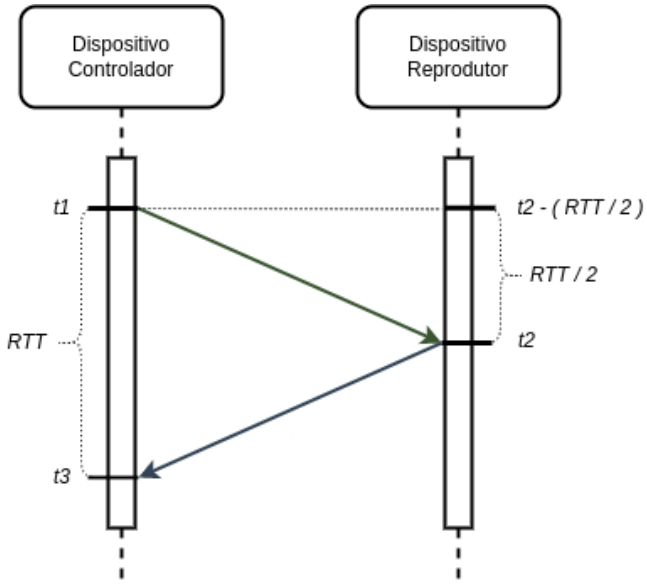


Figura 2: Diagrama de sequência do algoritmo de cálculo da dessincronização

$$offset = t1 - (t2 - ((t3 - t1)/2)) \tag{3}$$

Compreendendo este processo, sabe-se que metade do RTT não poderá fornecer valores de dessincronização exatos, pois pode existir uma maior atraso na primeira ou na segunda mensagem. Portanto, os seguintes passos deste algoritmo basearam na utilização de técnicas estatísticas para minimizar o erro no cálculo dos valores de dessincronização.

3.3.1 Análise do algoritmo implementado previamente

Na dissertação anterior [7], o primeiro passo para minimização do erro derivado do RTT começou por analisar os valores de RTT obtidos numa pequena aplicação de teste.

Com os resultados deste pequeno teste foi possível entender que os valores de RTT seguem uma distribuição Gaussiana ou normal.

A distribuição Gaussiana representa uma curva em forma de sino, esta é parametrizada pela média (μ) e desvio padrão (σ). Este parâmetro determinam o centro e a forma da curva como se pode ser visualizado a figura 3.

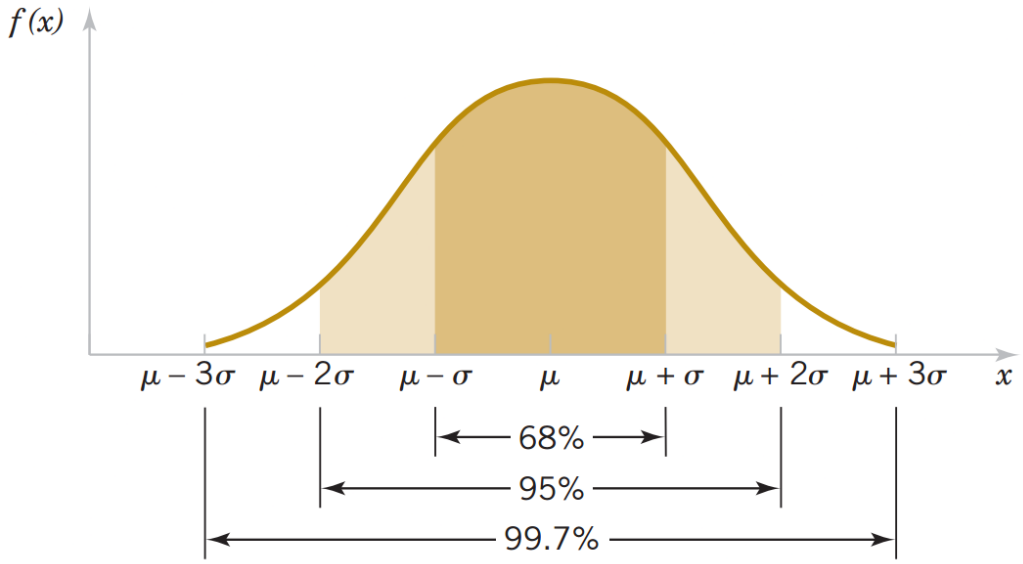


Figura 3: Distribuição Gaussiana ou normal

Com base nestes resultados, foi decidido que os valores de RTT deveriam ser filtrados por um intervalo de confiança, para que os valores de RTT anormais fossem descartados antes de o serem utilizados para o cálculo do *offset*.

O intervalo de confiança escolhido foi de 68% correspondendo à fórmula 4.

$$\mu - \sigma < \text{Intervalo de confiança} < \mu + \sigma \tag{4}$$

$$\mu = \frac{\sum_{i=1}^{n} x_i}{n} \tag{5}$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \mu)^2}{n-1}} \tag{6}$$

Após a filtragem dos valores de RTT, o passo seguinte foi uma simples média dos valores de *Offset*, excluindo o maior e menor valor de *Offset* calculado. Foi escolhido um período de 10 segundos para

a recolha dos valores, isto é, durante 10 segundos são recolhidos os valores e no fim desse período é calculado o *Offset*.

Sabendo o total funcionamento deste algoritmo, nas seguintes figuras é possível observar o seu resultado.

Na figura 4 é apresentado um gráfico a comparar o 5 é apresentado um gráfico com o erro do *Offset* calculado relativamente ao *Offset* real.

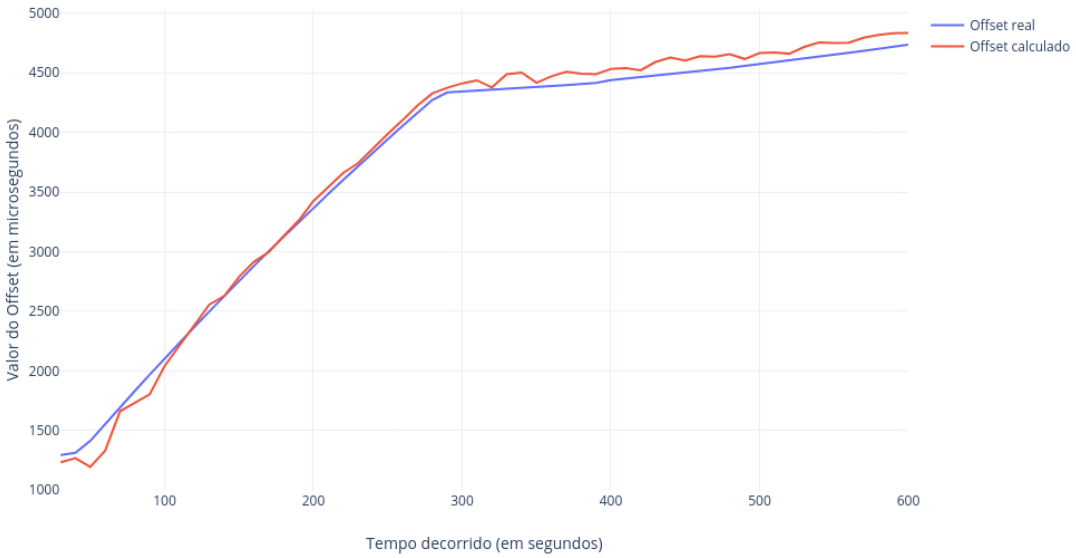


Figura 4: Gráfico de comparação entre *Offset* real e calculado com o antigo algoritmo

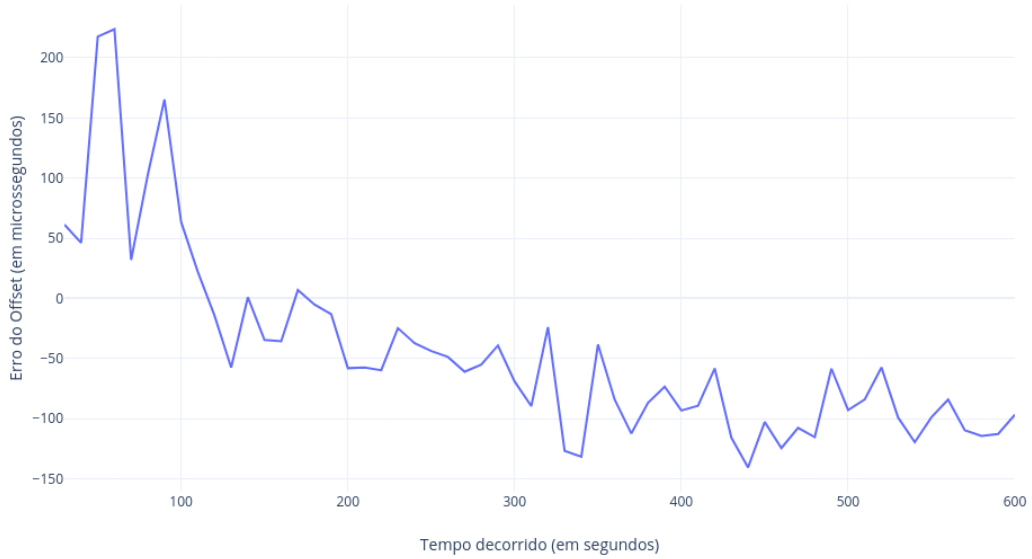


Figura 5: Gráfico com o erro do *Offset* calculado relativamente ao real com o antigo algoritmo

À primeira vista e observando apenas o gráfico da figura 4, pode parecer que este algoritmo funciona bastante bem, pois o *Offset* calculado parece acompanhar bastante bem o *Offset* real. Mas se o gráfico

da figura 5 for observado com detalhe é possível notar que o maior erro calculado foi superior a 200 microssegundos e que depois o erro começou a tender para o intervalo entre -50 e -150 microssegundos, quando deveria andar à volta dos 0 microssegundos. Apesar destes valores não parecerem prejudiciais para a sincronização do áudio, podem implicar num erro acumulativo, que ao fim de várias horas poderá ser notado.

Portanto, é necessário melhorar este algoritmo para um que tenha oscilações num intervalo menor e que este se situe à volta do zero.

3.3.2 Estudo de um novo algoritmo

Após a análise dos resultados anteriores e entendendo a metodologia seguida, foi decidido verificar graficamente o valores de *Offset* obtidos sem filtros em comparação com o *Offset* real. O resultado encontra-se no gráfico da figura 6.

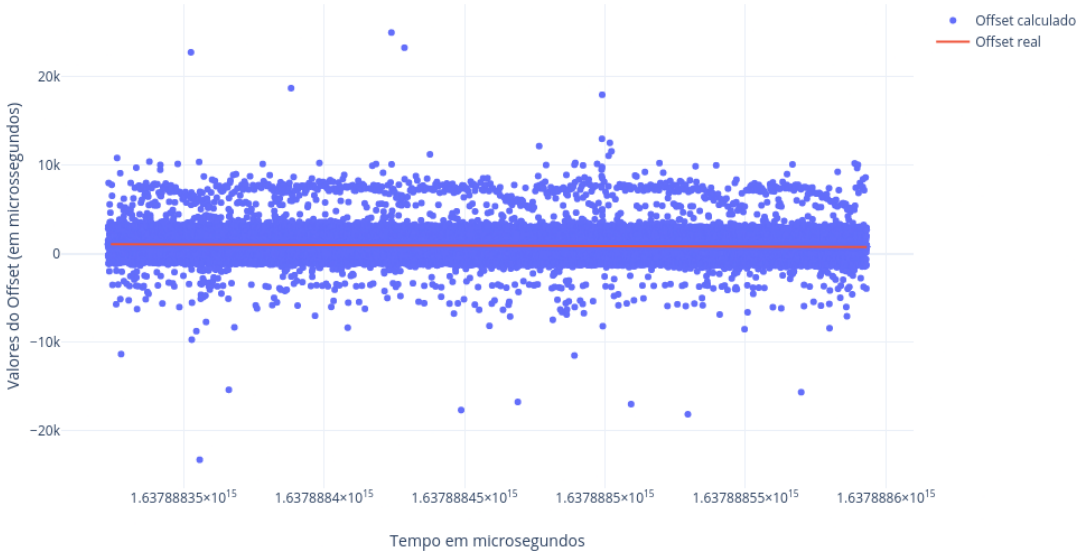


Figura 6: Gráfico com todos os valores de *Offset* calculados

Partindo do gráfico presente na figura 6 é possível concluir que a maior concentração de valores encontra-se à volta da linha do *Offset* real. Para fazer uma análise mais minuciosa segue-se na figura 7 o mesmo gráfico, mas ampliado na zona onde há uma maior concentração.

Com o gráfico da figura 7 é possível notar com maior precisão que a concentração de valores de *Offset* calculados encontra-se num intervalo de cerca de 500 microssegundos e dentro desse intervalo encontra-se a linha do *Offset* real.

Reunindo esta primeira conclusão é possível desenvolver uma primeira versão de um possível algoritmo final. Este poderá calcular os valores de *Offset* durante um dado intervalo de tempo e guardá-los

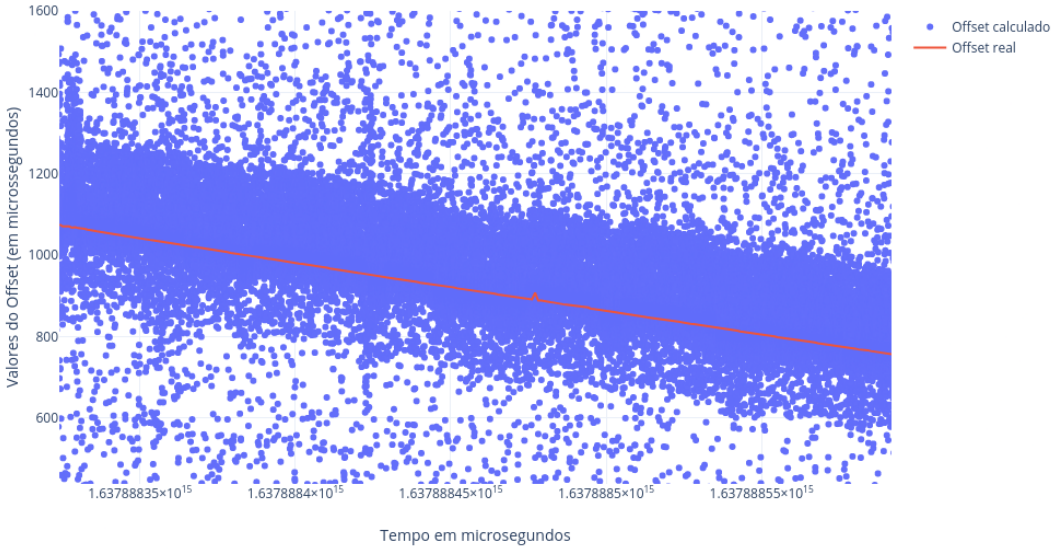


Figura 7: Gráfico 6 ampliado

numa estrutura de dados de forma ordenada. Depois pode verificar, de forma iterativa, se as extremidades da estrutura possuem uma diferença superior que 500 microssegundos, caso não se confirmar passa para as próximas extremidades. Chegando ao fim deste ciclo é feita a média dos restantes valores contidos na estrutura. Este algoritmo encontra-se descrito em 1.

Algoritmo 1 Algoritmo de cálculo de *Offset* V1

```
Data: offset_list[n] ∨ n > 0 ∨ offset_list is sorted
start_index = 0
end_index = 0
while offset_list[end_index] - offset_list[start_index]) > 500 do
    start_index = start_index + 1
    end_index = end_index - 1
end while
sum = 0
len = end_index - start_index
for index = [start_index, end_index] do
    sum = sum + offset_list[index]
end for
offset = sum / len
```

Tendo em conta o algoritmo 1 e para o comparar de forma justa, vão ser utilizados o mesmo tipo de gráficos usados com no subcapítulo anterior 3.3.1. Estes gráficos encontram-se nas figuras 8 e 9.

Começando pelo gráfico da figura 9 é possível notar uma melhoria notória relativamente às oscilações, isto é, o erro máximo encontra-se ligeiramente superior 60 microssegundos. Mas, por outro lado,

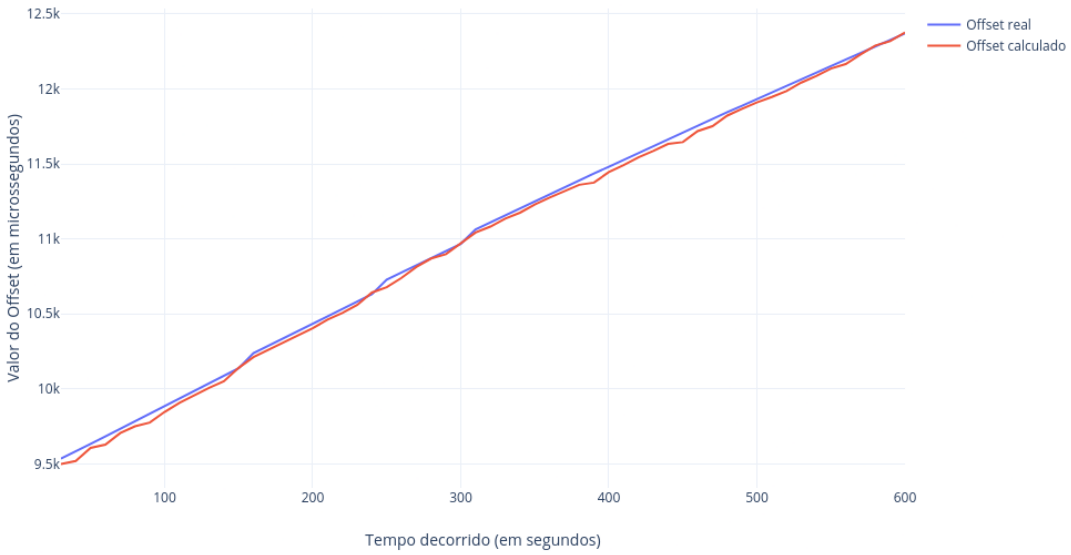


Figura 8: Gráfico de comparação entre *Offset* real e calculado com a primeira versão do algoritmo

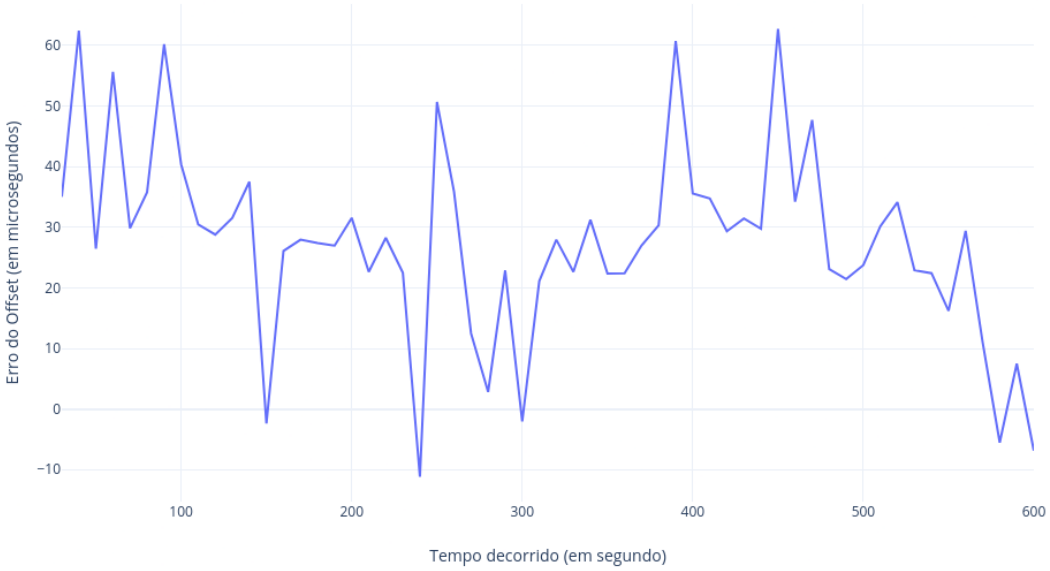


Figura 9: Gráfico com o erro do *Offset* calculado relativamente ao real com a primeira versão do algoritmo

as oscilações não se encontram à volta dos 0 microssegundos, portanto é necessário melhorar este algoritmo.

De volta aos gráficos das figuras 6 e 7 e analisando o algoritmo 1, nota-se que existe a necessidade de determinar um centro na concentração, porque não é possível assumir que existem tantas amostras acima como abaixo da concentração.

Para determinar este centro foram escolhidas duas alternativas, a média e a moda dos *Offset's*. Para obter a moda dos *Offset's* foi necessário agrupar estes em intervalos, pois há uma enorme variedade de valores, para o efeito foram utilizados intervalos de 5 microssegundos. E depois foi utilizada uma nova estrutura de dados para guardar a frequência de cada intervalo. Este algoritmo encontra-se descrito em 2.

Algoritmo 2 Algoritmo de calculo da moda dos *Offset's*

```
Data: offset_list[n] ∨ n > 0
aux = 0
max = 0
for index = [0, n] do
    aux = offset_list[index] / 5
    offset_freq[aux] = offset_freq[aux] + 1
    if offset_freq[aux] > offset_freq[max] then
        max = aux
    end if
end for
mode = max × 5
```

Após obter o valor central, o algoritmo 1 foi adaptado para calcular o intervalo de 500 microssegundos de volta do valor central. Isto é, enquanto a diferença entre o valor central e a extremidade for superior a 250 microssegundos, a extremidade é decrementada, a mesma coisa acontece para a extremidade inferior. Quando ambas extremidades se encontrarem num intervalo de 500 microssegundos é calculada a média com os valores restantes. Este algoritmo encontra-se descrito em 3.

Visto que existem duas alternativas para calcular o valor central, media e moda, e para continuar a desenvolver o algoritmo é necessário escolher uma. Para o efeito estes serão comparados graficamente e matematicamente. Para a comparação matemática será utilizado uma fórmula de erro absoluto médio 7, isto é, será feita a média do valor absoluto das diferenças entre os respetivos *Offset* calculado e *Offset* real.

$$\text{erro absoluto médio} = \frac{\sum_{i=1}^{n} |\text{Valor\_real}_i - \text{Valor\_calculado}_i|}{n} \tag{7}$$

Os gráficos encontram-se nas figuras 10 e 11. Para o erro absoluto médio foi utilizada uma tabela para efetuar a comparação. Tendo em consideração que a aplicação que corre o algoritmo desenvolvido, calcula

Algoritmo 3 Algoritmo de cálculo de *Offset* V2

```
Data: offset_list[n] ∨ n > 0 ∨ offset_center ∨ offset_list is sorted
start_index = 0
while offset - offset_list[start_index]) > 250 do
    start_index = start_index + 1
end while
end_index = 0
while offset_list[end_index] - offset_center) > 250 do
    end_index = end_index - 1
end while
sum = 0
len = end_index - start_index
for index = [start_index, end_index] do
    sum = sum + offset_list[index]
end for
offset = sum / len
```

os valores de *Offset* a cada 10 segundos durante 10 minutos e o erro absoluto médio é respetivo aos 10 minutos de execução. Foi necessário executar o programa varias vezes, para comparar os resultados obtidos pelas duas alternativas, da forma mais justa.

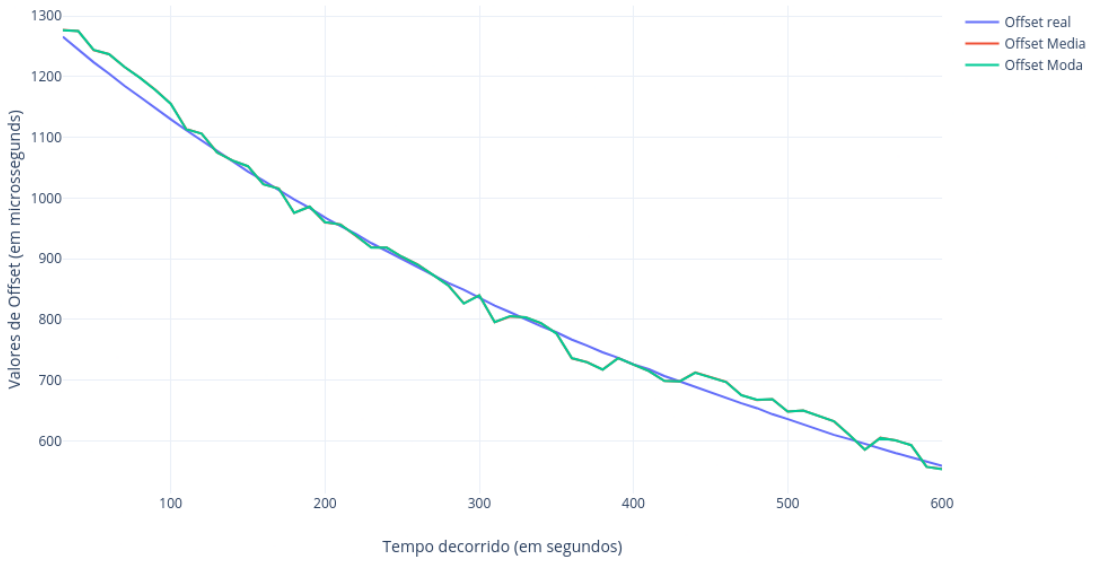


Figura 10: Gráfico de comparação entre *Offset* real e os calculados com a segunda versão do algoritmo

Partindo do gráfico da figura 10, não há muito a concluir, parece não existir uma melhoria notável relativamente ao algoritmo 1. Por outro lado, no gráfico da figura 11, já se encontram melhorias. A primeira é o do erro máximo que se encontra nos 30 microssegundos, metade do algoritmo anterior. E a segunda melhoria é o intervalo das oscilações que se encontra à volta dos 0 microssegundos. É possível dizer com estes resultados, que este algoritmo se encontra num bom caminho.

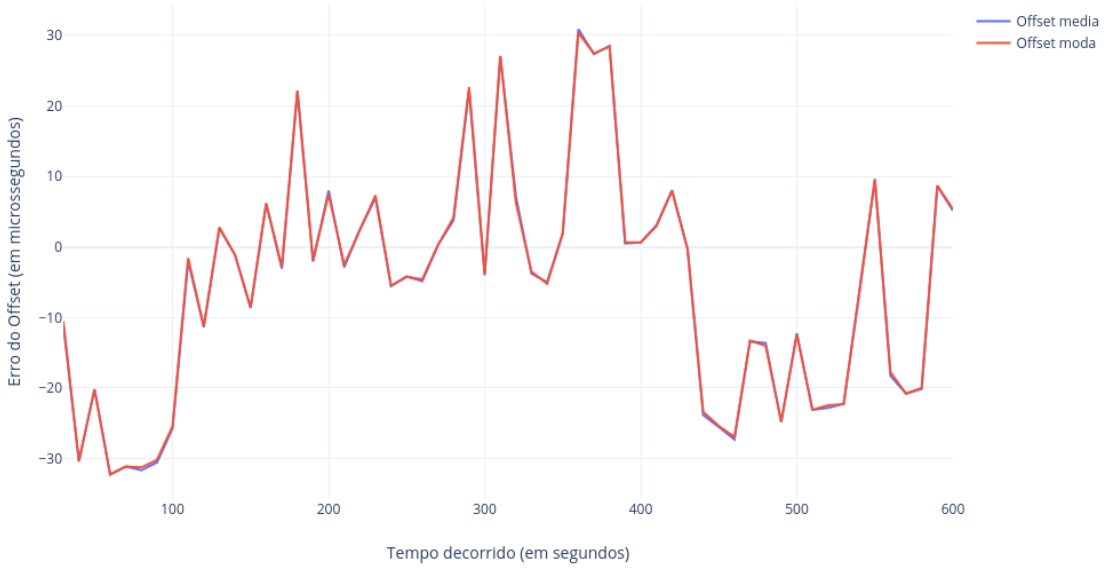


Figura 11: Gráfico com os erros dos *Offset* calculados relativamente ao real com a segundo versão do algoritmo

Tabela 2: Resultados dos erros médios produzidos pelas duas alternativas

Iteração	Erro com média	Erro com moda
1	13,43	13,35
2	21,67	21,54
3	22,49	22,64
4	15,84	15,95
5	10,90	10,85
6	10,76	10,55
7	9,16	9,09
8	15,17	15,13
9	12,50	12,53

Analisando os resultados presentes na tabela 2, é aparente que não existe grande diferença entre as duas alternativas, ambos resultados de cada iteração possuem uma diferença muito pequena e não estabelecem nenhum padrão. Portanto, com as próximas etapas do protocolo estas alternativas serão novamente comparadas, para determinar a melhor.

3.4 Cálculo da deriva da dessincronização

No capítulo anterior 3.3 foi abordado o algoritmo de cálculo da dessincronização entre dispositivos diferentes em redes sem fios. Mas para bom funcionamento deste protocolo, apenas o algoritmo anterior não é suficiente. É necessário calcular a deriva da dessincronização ao longo do tempo (também vulgarmente conhecido como *Time Skew*). Esta medida irá determinar a dessincronização num determinado intervalo

de tempo. No que lhe concerne, esta será a medida que vai ser utilizada para garantir a sincronização da reprodução áudio.

O *Time Skew* de uma forma simples pode ser calculado através da diferença entre dois *Offset's* a dividir pelo tempo decorrido entre eles, descrito na fórmula 8.

$$\textit{Time Skew} = \frac{O_i - O_{i+1}}{T_{i+1} - T_i} \tag{8}$$

Na dissertação anterior foram definidas duas formulas de cálculo do *Time Skew* para suavizar os valores calculados. A fórmula 9 é respetiva à média dos *Time Skew* calculados e a fórmula 10 é respetiva à fórmula da regressão linear. Estas técnicas foram utilizadas para nivelar possíveis oscilações derivadas dos valores calculados.

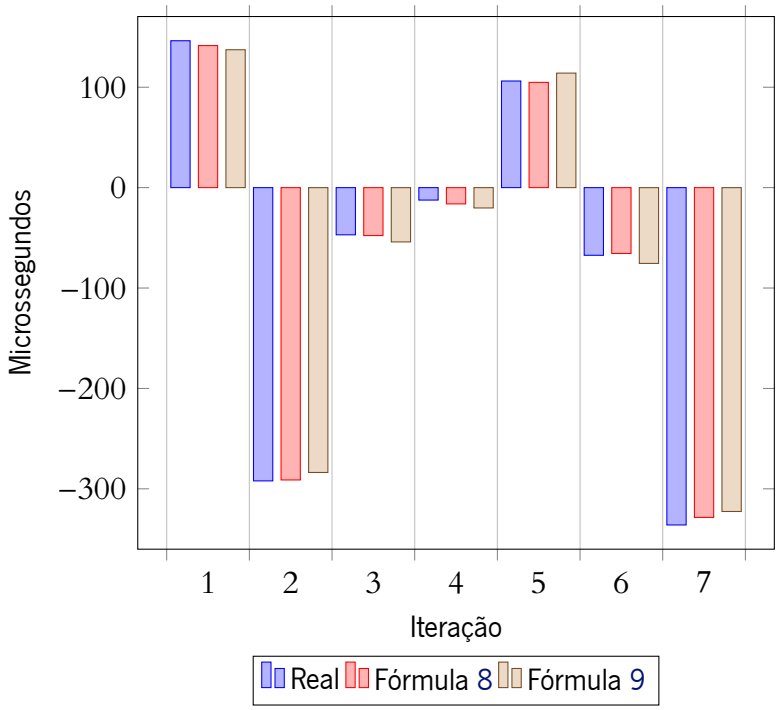
$$\textit{Time Skew} = \frac{1}{n}\sum_{i=1}^{n}\frac{O_i - O_{i+1}}{T_{i+1} - T_i} \tag{9}$$

$$\textit{Time Skew} = \frac{\overline{OT} - \bar{O}\bar{T}}{\bar{O^2} - (\bar{O})^2} \tag{10}$$

Mas, por outro lado, se os valores de *Time Skew* calculados forem acumulados, é possível obter uma maior precisão e controlo. Isto porque as possíveis oscilações nas estimações correspondem a erros nos cálculos do *Offset's*, e estas podem estar localizadas acima ou abaixo do valor real. Seguindo este raciocínio se estas forem acumuladas são ao mesmo tempo niveladas.

Para comprovar que a fórmula 8 é uma técnica superior para o cálculo do *Time Skew's*, serão comparados os resultados das somas de cada uma das técnicas, inclusive a soma dos valores reais para termo de comparação. Cada iteração corresponde à recolha dos valores durante um período de 10 minutos, e será utilizado um dos algoritmos discutidos no capítulo anterior, o algoritmo escolhido não é relevante para esta comparação. Infelizmente não foi possível utilizar a fórmula 10, pois esta é utilizada para determinar o declive da reta desenhada pelos *Offset's* ao longo do tempo. Este método não é ideal para utilização porque por um lado requer demasiada complexidade computacional e caso a direção da reta mude vai fornecer dados errados. Por outro lado, este método seria útil caso fosse necessário prever o rumo da dessincronização.

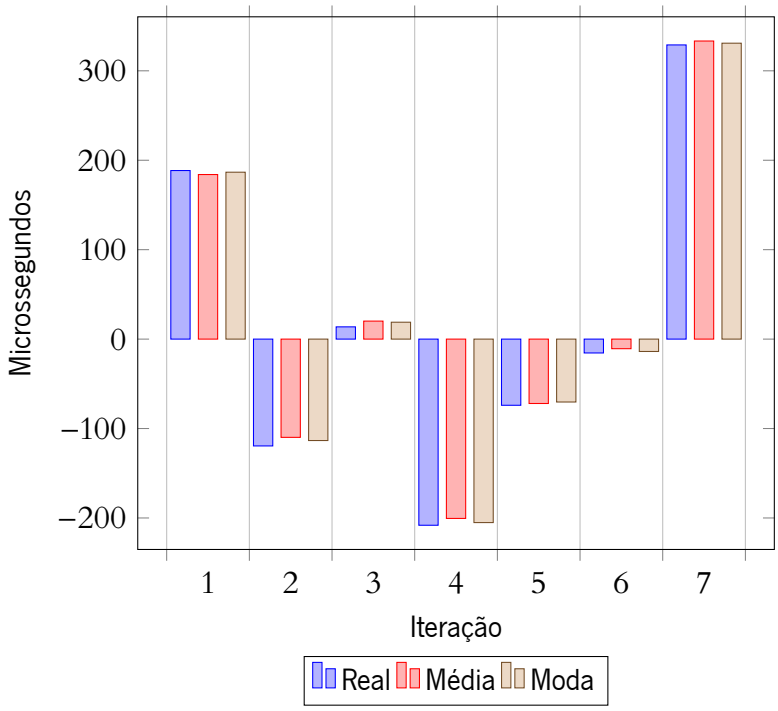
Resultados das somas dos valores de *Time Skew* calculados pelas várias técnicas



Analisando os resultados obtidos no gráfico anterior, é óbvio que a soma dos valores calculados pela fórmula 8 oferece melhores resultados do que pela fórmula 9. Os resultados não parecem ser muito distintos, isto dá-se porque os testes feitos tinham uma duração de 10 minutos apesar de não ser por valores muito distintos, caso fossem teste de 1 hora as diferenças seriam potencialmente superiores.

Por fim, para determinar qual a melhor alternativa das comparadas anteriormente na tabela 2, estas alternativas serão novamente comparadas, mas através da técnica discutida neste capitulo. Os resultados obtidos foram submetidos às mesmas condições que os resultados no gráfico seguinte.

Resultados das somas dos valores de *Time Skew* calculados com as diferentes alternativas



Com os resultados obtidos no gráfico anterior, é possível verificar que os resultados usando a moda para calcular o valor centro da distribuição de *Offset's*, é ligeiramente melhor que a média. Sabendo que os resultados obtidos estão em microssegundos, é óbvio que a diferença entre estes é ínfima e muito pouco relevante. Por estas razões a alternativa escolhida foi a moda.

3.5 Sincronização ativa do áudio

Com o mecanismo discutido anteriormente é possível calcular a diferença temporal dos dispositivos com enorme precisão, portanto fica a faltar um mecanismo que corrija as diferença temporais calculadas. Para o efeito, foi delineado, na dissertação anterior [7], uma estratégia de remoção e a adição de amostras para que a reprodução de seja mais rápida ou mais lenta, com objetivo de se sincronizar com os restantes dispositivos envolvidos na reprodução de áudio.

Para calcular o número de amostras a manipular, foi elaborado um mecanismo que verifica quantas amostras cabem, temporalmente, na atual dessincronização (no *Time Skew* acumulado). Desta forma é possível alterar o áudio com a maior precisão e manter o valor de *Time Skew* acumulado, preciso e atualizado. O tempo da amostra é calculado através da frequência de amostragem do áudio alvo. É também subentendido que o valor acumulado do *Time Skew* é atualizado em tempo real.

Este mecanismo encontra-se descrito no algoritmo 4.

Algoritmo 4 Algoritmo do número de amostras a manipular

```
Data: time_skew_sum ∨ sample_rate
frame_period = 1.000.000 / sample_rate
n_frames = time_skew_sum / frame_period
if Integer(n_frames) != 0 then
    time_skew_sum -= Integer(n_frames) * frame_period
end if
```

3.6 Reamostragem do áudio

No trabalho anterior [7] foi descrito, que para sincronizar a reprodução de áudio entre os vários dispositivos adicionava-se e removia-se amostras de áudio, para estender e comprimir o áudio temporalmente. Esta técnica por si só sem manipular o resto do sinal de áudio vai causar irregularidades, sendo estas audíveis ou não. Para resolver este problema foi decidido elaborar um algoritmo de reamostragem para que a forma do sinal de áudio fosse preservado, embora existam perdas de informação.

Para desenvolver esta estratégia, foi necessário garantir que a remoção ou adição de amostras seja distribuída de forma uniforme pelo segmento de áudio. Dito isto, foi feita uma divisão da escala temporal, para que todas as amostras fossem adaptadas ao novo tamanho do segmento. E a seguir, remove-se a segunda amostra incluída no mesmo intervalo temporal inteiro. Na adição a estratégia é a mesma, mas adiciona-se uma amostra quando a segunda há um salto de uma unidade, e a amostra introduzida é apenas uma média da amostra anterior e seguinte.

Na tabela 3 segue-se um exemplo da remoção de amostras, onde foram removidas 3 amostras de um segmento de 12 amostras, isto significa que a escala temporal passou de 1 para 0,75 = ((12 - 3) / 12).

Tabela 3: Exemplificação da remoção de amostras

Original	Divisão Temporal	Remoção	Divisão Temporal
0	0.00	0	0.00
1	0.75	1	0.75
5	1.50	5	1.50
8	2.25	8	2.25
10	3.00	10	3.00
6	3.75	6	3.75
2	4.50	2	4.50
-1	5.25	-1	5.25
-5	6.00	-5	6.00
-8	6.75	-8	6.75
-10	7.50	-10	7.50
-6	8.25	-6	8.25

Na tabela 4 segue-se um exemplo da adição de amostras, onde foram adicionadas 3 amostras ao

mesmo segmento de 12 amostras, isto significa que a escala temporal passou de 1 para 1,25 = ((12 + 3) / 12).

Tabela 4: Exemplificação da adição de amostras

Original	Divisão Temporal	Adição	Divisão Temporal
0	0.00	0	0.00
1	1.25	1	1.25
5	2.50	5	2.50
8	3.75	8	3.75
10	5.00	10	5.00
6	6.25	6	6.25
2	7.50	2	7.50
-1	8.75	-1	8.75
-5	10.00	-5	10.00
-8	11.25	-8	11.25
-10	13.50	-10	13.50
-6	14.00	-6	14.00

Após a conceção destas estratégias, estas foram postas à prova. Para verificar a qualidade dos resultados foi utilizado um segmento áudio de uma onda sinusoidal. Nas figuras 12 e 13 encontra-se a onda original, com a adaptação da escala temporal para possuir o mesmo comprimento que a onda alterada.

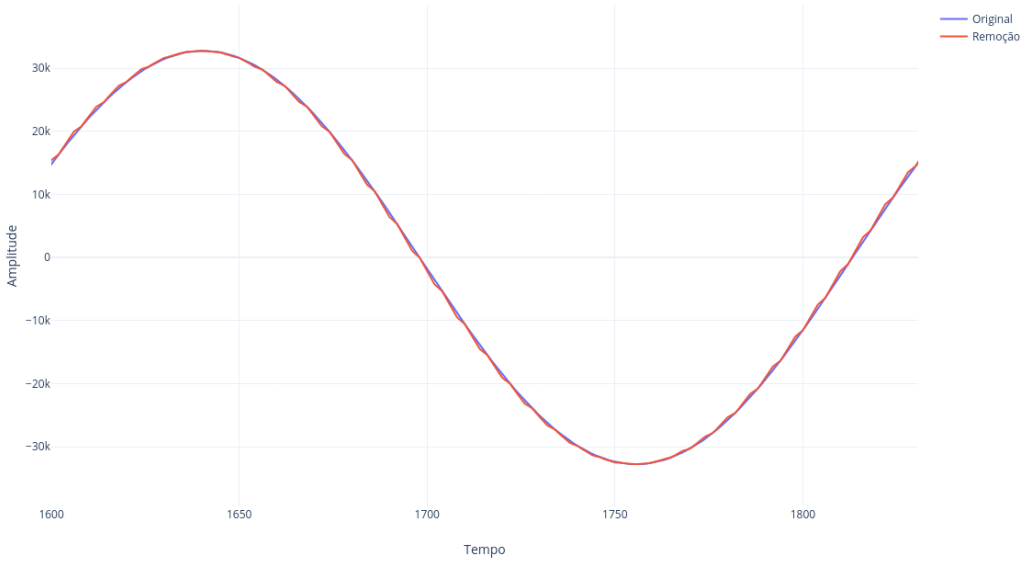


Figura 12: Gráfico de comparação da onda original com a onda produzida com a estratégia de remoção de amostras

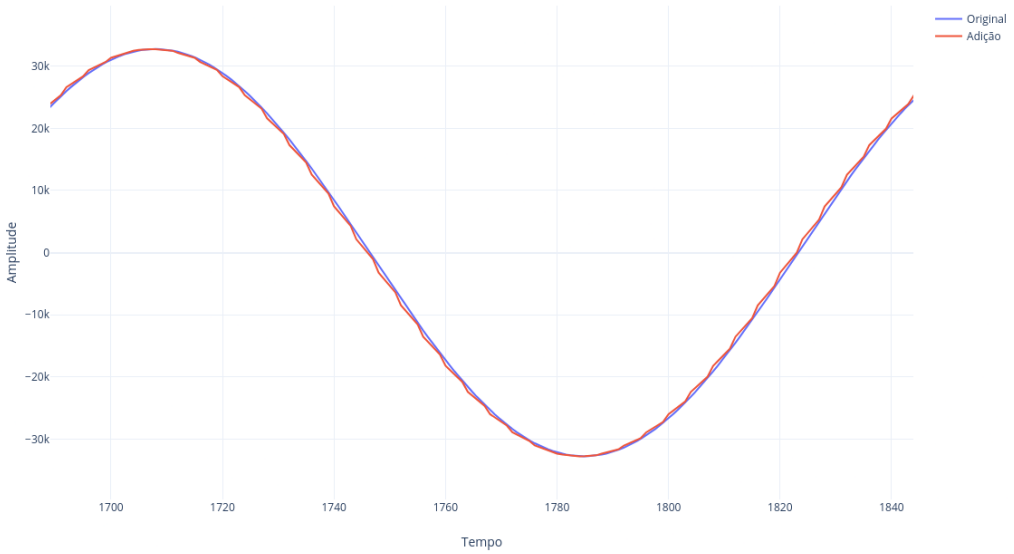


Figura 13: Gráfico de comparação da onda original com a onda produzida com a estratégia de adição de amostras

Partindo destas figuras é possível observar múltiplas distorções ao longo da onda, neste caso a distorção produzida não parece resultar numa clara distorção sonora, mas para outros segmentos poderá, além da clara perda de qualidade de áudio. Por estas razões foi decidido melhorar a estratégia utilizada sem recorrer à necessidade adicionar filtros digitais.

A seguinte estratégia foi um aprofundamento das técnicas anteriores. Ou seja, se a escala temporal for dividida, esta deixa de ser inteira e passa a ser decimal, mas se forem calculados os valores inteiros a partir dos decimais, é possível obter as amplitudes intermediarias e desta forma manter a forma mais próxima da original da onda.

Após a delineação inicial da estratégia, foi necessário encontrar um método simples para calcular um ponto intermédio entre dois pontos existentes. O método escolhido foi a equação da reta.

A equação da reta é conhecida pela fórmula 11. E esta é definida a após o cálculo do declive da reta, representado na fórmula 12.

$$y = mx + b \tag{11}$$

$$m = \frac{y_1 - y_2}{x_1 - x_2} \tag{12}$$

Partindo destas escolhas foram desenvolvidos os algoritmos 5 e 6, para a compressão e extensão, respetivamente, do áudio.

Nas figuras 14, 16, 15 e 17 encontram-se os resultados obtidos através dos algoritmos anteriores.

Partindo dos resultados obtidos, é possível dizer que esta nova estratégia e muito melhor que a simples remoção ou adição de amostras. Esta solução possui resultados muito semelhantes aos resultados obtidos

Algoritmo 5 Algoritmo de remoção de amostras

```
Data: input[input_len] ∨ len > 0 ∨ n_frames > 0
output_len = len - n_frames
decimal_step = output / input_len
output[0] = input[0]
for index = [1, len] do
    current_x += decimal_step
    if (integer) current_x != (integer) previous_x then
        output[current_x]=calculate(current_x, previous_x, input[index-1], input[index], decimal_step)
    end if
    previous_x = current_x
end for
```

Algoritmo 6 Algoritmo de adição de amostras

```
Data: input[input_len] ∨ len > 0 ∨ n_frames > 0
output_len = len - n_frames
decimal_step = output / input_len
output[0] = input[0]
for index = [1, len] do
    current_x += decimal_step
    if (integer) current_x == (integer) previous_x + 2 then
        output[current_x-1]=calculate(current_x-1, previous_x, input[index-1], input[index], decimal_step)
    end if
    output[current_x]=calculate(current_x, previous_x, input[index-1], input[index], decimal_step)
    previous_x = current_x
end for
```

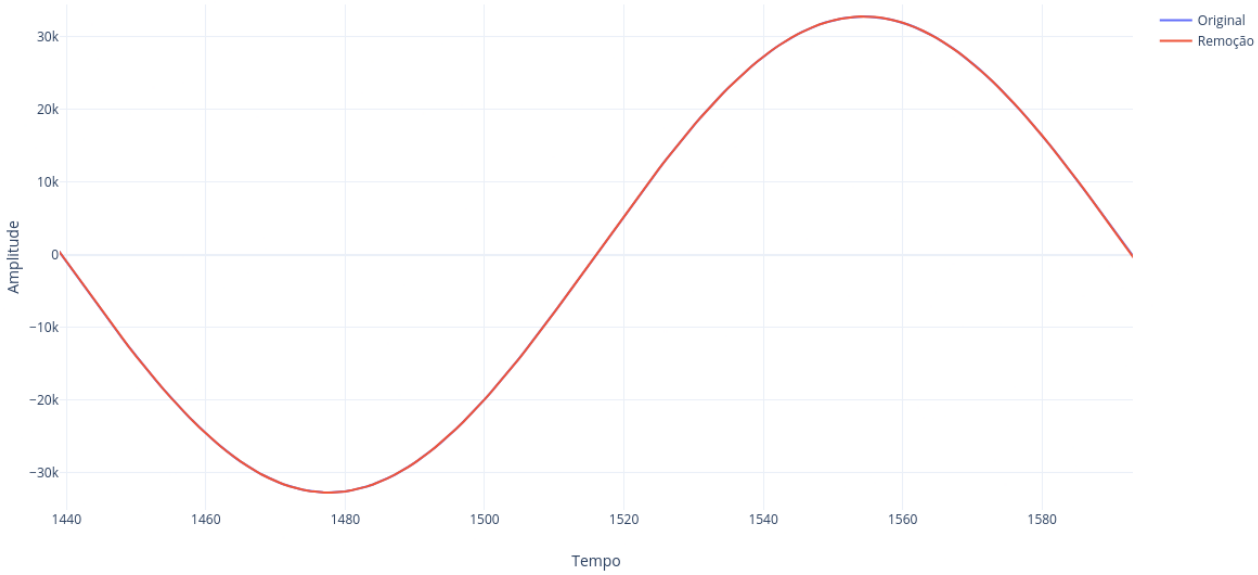


Figura 14: Gráfico de comparação da onda original com a onda produzida com a estratégia final de remoção de amostras

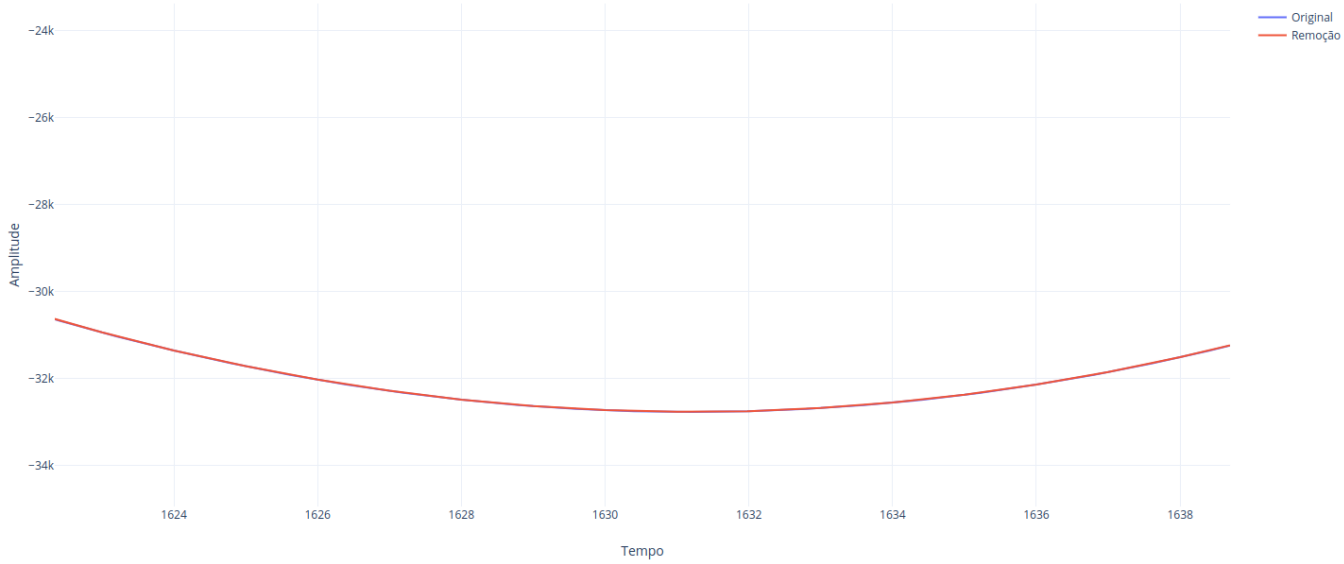


Figura 15: Gráfico da figura 14 ampliado

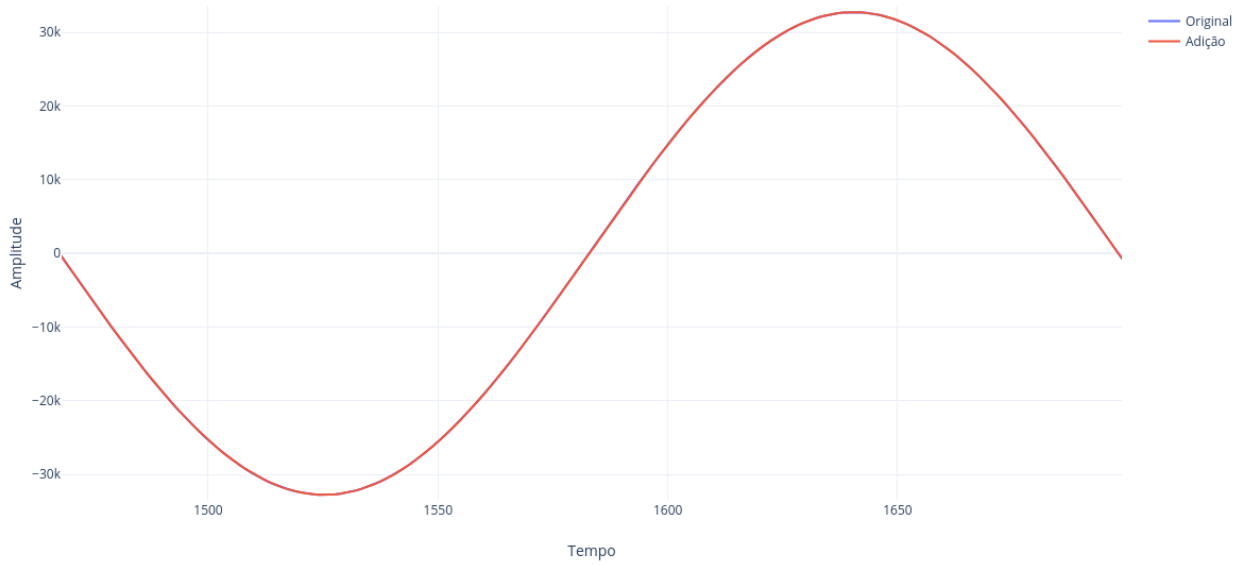


Figura 16: Gráfico de comparação da onda original com a onda produzida com a estratégia final de adição de amostras

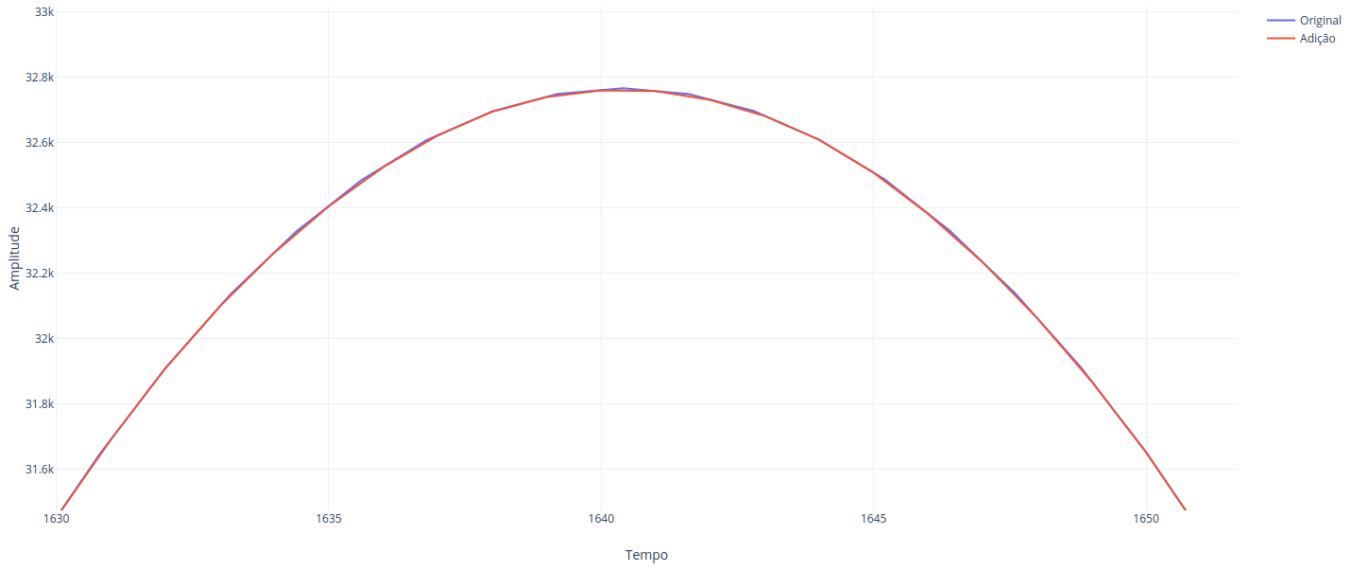


Figura 17: Gráfico da figura 16 ampliado

com bibliotecas de reamostragem que utilizam filtros, mas ao contrário dessas a complexidade desta é muito inferior.

3.7 Cálculo do inicio da reprodução de áudio

Com os algoritmos de cálculo da dessincronização desenvolvidos, a próxima etapa é a sincronização inicial, para que a reprodução de áudio inicie com o menor valor de dessincronização possível.

Para o efeito, serão utilizados os valores de *Offset* calculados com os valores de RTT médios obtidos. Partindo dos valores de *Offset* é possível observar três cenários possíveis:

1. Os dispositivos clientes encontram-se temporalmente mais avançados que o servidor. Neste caso é necessário encontrar o dispositivo mais avançado para calcular a mesma referência temporal entre os dispositivos clientes e após obter este é calculada a diferença entre este e cada dispositivo;

2. Os dispositivos clientes encontram-se temporalmente mais atrasados que o servidor. Neste caso a referência temporal comum será o cálculo da soma dos próprios atrasos.

3. Caso os dois cenários anteriores se verifiquem, é necessário combinar as duas estratégias para calcular a referência temporal comum a todos.

Uma vez que os dispositivos possuam a mesma referência temporal, é necessário obter o RTT médio de cada dispositivo e selecionar o que possui o maior valor. Após obter o maior valor de RTT, adicionasse este ao momento de cada dispositivo cliente. Este valor é necessário para garantir que qualquer atraso que exista na rede não interfira com o início da reprodução.

Garantindo estes parâmetros, é possível garantir que o início sincronizado da reprodução, mas dependendo da aplicação em questão, poderá ser necessário envolver um atraso extra. Seja este para garantir o *buffering* dos dados ou para sincronizar com algo externo, por exemplo, imagem ou vídeo.

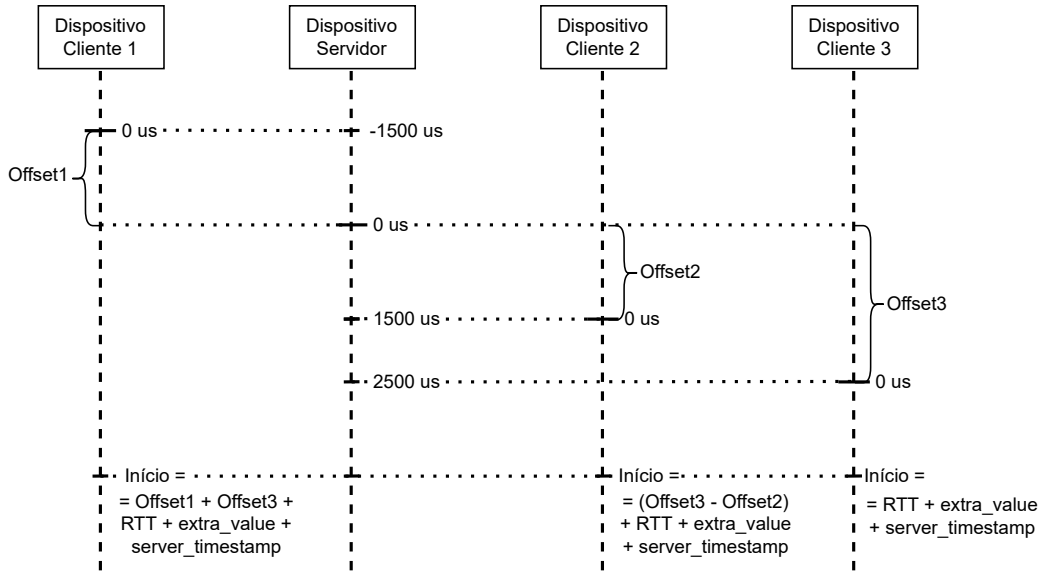


Figura 18: Demonstração gráfica do cálculo do início sincronizado

Na figura 18 é possível observar um exemplo constituído por quatro dispositivos, três clientes e um servidor, e as suas linhas temporais. Neste exemplo encontram-se dois dispositivos clientes mais avançados que dispositivo servidor e um mais atrasado que o dispositivo servidor. Ou seja, este exemplo retrata o terceiro cenário descrito anteriormente e o cálculos efetuados para cada dispositivo para calcular a designada referencia temporal comum para cada dispositivo.

3.8 Técnica para distribuição de áudio multicanal

Partindo das soluções estudadas no capítulo anterior 2, sabe-se que apenas o RTP foi desenhado exclusivamente para a distribuição de dados multimédia em tempo real. Mas possui funcionalidades que não são necessárias, para o presente caso e pode ser melhorado noutros. O SCTP possui também funcionalidades interessantes para este fim, mas também implementa outras funcionalidades que poderão atrasar a distribuição de áudio em tempo real. Outro aspeto a ter em consideração é que nenhum destes protocolos foi pensado exclusivamente para redes sem fios.

Com estes aspetos em conta foi decido desenvolver um novo protocolo focado nos seguintes objetivos:

- Transmissão de dados em tempo real;
- Fiabilidade em redes sem fios;

- Ordenação dos dados;
- Suporte múltiplas transmissões em simultâneo (para suportar, neste caso, vários canais de áudio);
- Controlo de congestão da rede;

Partindo destes objetivos, decidiu-se que a partir do início da transmissão, esta não deverá parar para a retransmissão, ou seja, esta deverá reproduzir sequencias de áudio nulas (ou a zero). Supondo que a taxa de transmissão de dados é superior que a taxa da reprodução de dados, a diferença entra duas disponibiliza a possibilidade da retransmissão. Ou seja, caso hajam pacotes de dados perdidos, é possível retransmiti-los enquanto estes não forem reproduzidos. Depois, sempre que as condições de rede permitirem, é possível dobrar a quantidade de dados enviados, aumentando assim a taxa de transmissão de dados. No caso de perdas ou erros, a quantidade de dados pode ser diminuída evitando mais perdas ou erros, e desta forma controlando o congestionamento da rede.

Partindo das condições em que este projeto foi realizado, estas são dois Raspberry Pi's numa rede local Wi-Fi de 2,4 GHz com um *Wireless Access Point*, os valores médios de RTT obtidos foram 2 milissegundos e o valor máximo médio de RTT foi de 25 milissegundos.

Partindo destes valores, estabeleceu-se para este exemplo utilizar uma janela temporal inicial de 100 milissegundos, isto significa que a janela de dados equivale a 100 milissegundos de áudio. Por exemplo, no caso de áudio amostrado a 16 bits e 44,1 KHz seriam transmitidos 8820 bytes de dados. Foi decido este valor para ter em consideração a necessidade de retransmissão, as possíveis retransmissões deverão pertencer à janela temporal.

Relativamente às retransmissões, decidiu-se utilizar 4 confirmações por janela temporal. Possibilitando assim a retransmissão enquanto a janela de dados não foi transmitida na sua totalidade. Desta forma evita-se fazer uma confirmação muito grande no final ou esperar pelo fim da transmissão da janela para fazer as retransmissões. O número de confirmações é fixo, para reduzindo assim confirmações desnecessárias. Por outro lado, para garantir que todas as retransmissões sejam confirmadas, estas serão confirmadas individualmente. Esta decisão possibilita também a confirmação de pacotes fora da sua janela temporal, isto é possível na eventualidade da reprodução de áudio estiver mais atrasada que a transmissão de dados.

Por fim, para aproveitar da melhor forma a largura de banda disponível, o tamanho das janelas serão multiplicadas por dois, sempre que a transmissão não apresentar retransmissões, no caso de existirem retransmissões o tamanho desta deverá passar para metade, nunca sendo menor que o intervalo mínimo referido anteriormente. E caso a janela seja enviada na sua totalidade com sucesso antes do intervalo temporal acabar, o emissor deverá passar para a janela seguinte.

Na figura 19 está representado um diagrama temporal em que representa a solução descrita. Neste exemplo pode-se supor que o emissor transmite 100 milissegundos de uma áudio amostrado a 16 bits

e 44,1 KHz, ou seja, 8820 bytes de dados. Esta quantidade de dados está distribuída por 8 pacotes de dados. Neste exemplo é apenas demonstrado uma janela de dados.

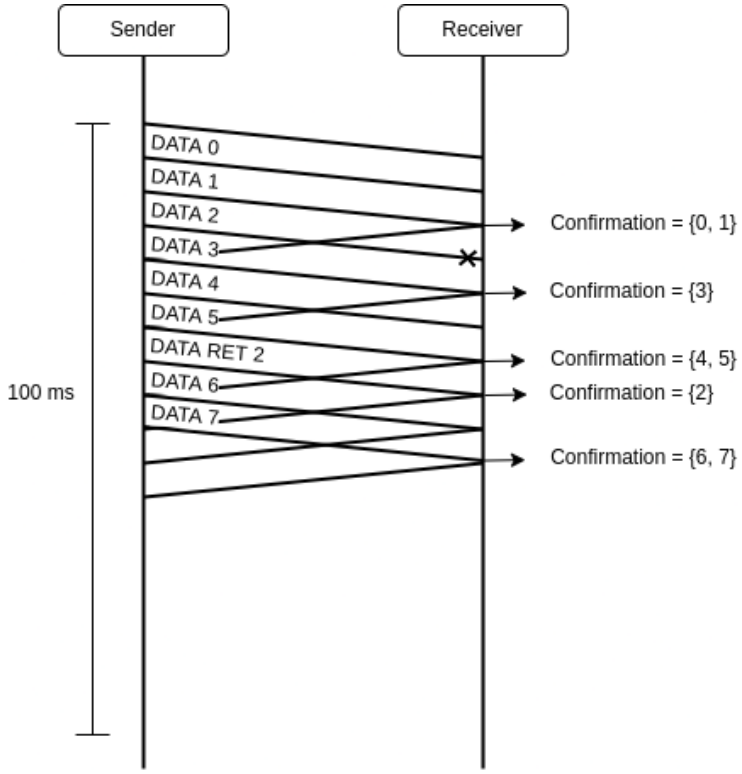


Figura 19: Diagrama de temporal da solução proposta para a distribuição de áudio multicanal

3.9 Arquitetura Protocolar

A arquitetura dos protocolos desenvolvidos foi feita para que cada um fosse independente, possibilitando assim a utilização dos mesmos para objetivos diferentes e em tecnologias diferentes, por exemplo, usar *Bluetooth* para a distribuição de áudio.

O primeiro protocolo desenvolvido é responsável pela monitorização da diferença temporal entre relógios, ao qual foi atribuído o acrónimo TiMP, que significa *Time Monitoring Protocol*. O segundo foi o protocolo de distribuição de áudio multicanal, ao qual foi atribuído o acrónimo de MASP, que significa *Multi-Channel Audio Streaming Protocol*. Por fim, o último foi o protocolo de controlo da reprodução do áudio em múltiplos dispositivos, ao qual foi atribuído o acrónimo de MDPCP, que significa *Multi Device Playback Control Protocol*.

Na figura 20 pode-se encontrar o enquadramento deste protocolos.

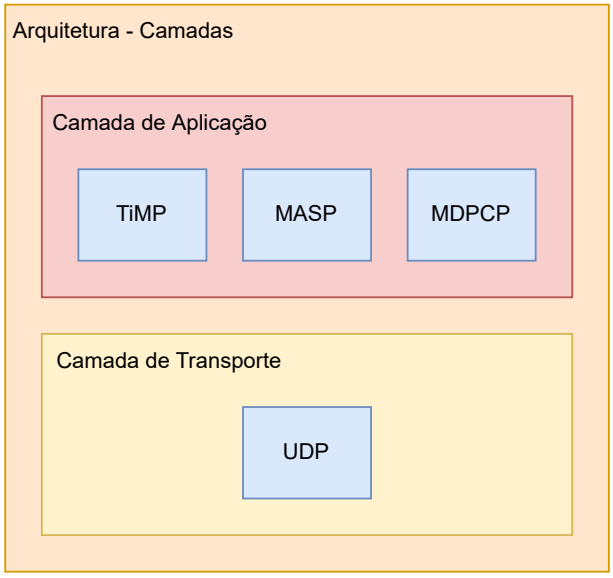


Figura 20: Arquitetura das aplicações

3.9.1 TiMP

O protocolo TiMP funciona numa arquitetura cliente-servidor e possui quatro tipos de mensagens diferentes, todas elas com o mesmo formato. O formato das mensagens está especificado na imagem 21.

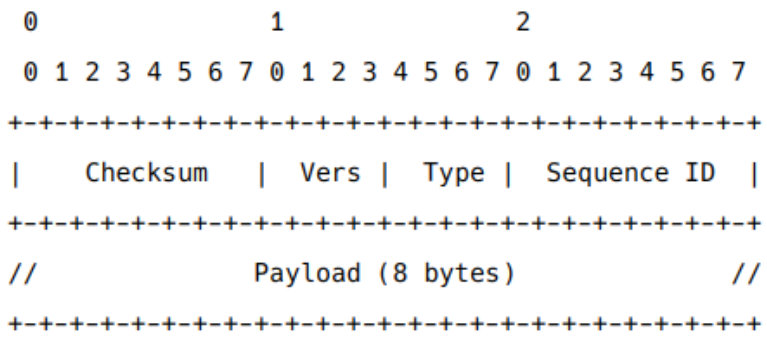


Figura 21: Formato das mensagens do TiMP

Descrição dos parâmetros:

- *Checksum* - Parâmetro de 1 byte, utilizado para a deteção de erros;
- *Vers* - Parâmetro de 4 bits, dedicado à versão do protocolo;
- *Type* - Parâmetro de 4 bits, utilizado para a identificação do tipo de mensagem;
  - *Timestamp Request 0000*
  - *Timestamp Response 0001*
  - *Offset Inform 0010*
  - *Offset Acknowledge 0011*
- *Sequence ID* - Parâmetro de 1 byte, usado para identificar a respetiva resposta ao respetivo pedido;
- *Payload* - Parâmetro de 8 bytes, para o transporte dos *timestamp's* ou das estimações calculadas.

3.9.2 MASP

O protocolo MASP funciona numa arquitetura de *Peer-to-Peer*, ou seja, qualquer das entidades pode ser transmissor ou recetor e possui um tipo de mensagem. O formato da mensagem deste protocolo está especificado na imagem 22.

Descrição dos parâmetros:

- *Checksum* - Parâmetro de 2 bytes, utilizado para a deteção de erros;

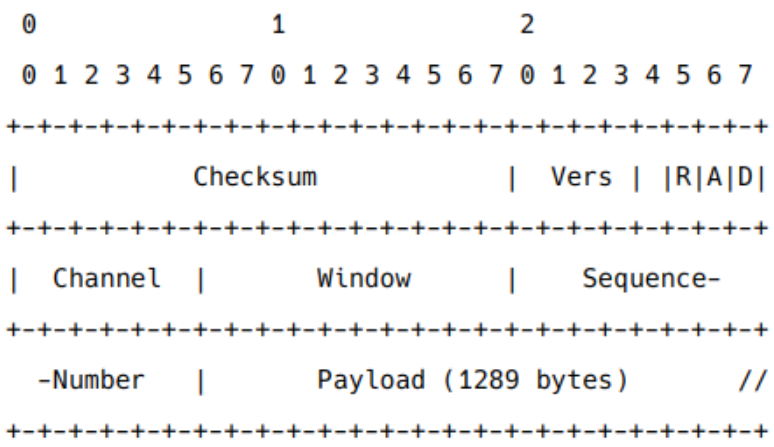


Figura 22: Formato das mensagens do MASP

- *Vers* - Parâmetro de 4 bits, dedicado à versão do protocolo;
- *R - Retransmission* - Parâmetro de 1 bit, utilizado para a indicar que a mensagem é uma retransmissão;
- *A - Acknowlodge* - Parâmetro de 1 bit, utilizado para indicar que a mensagem é uma confirmação;
- *D - Data* - Parâmetro de 1 bit, utilizado para indicar que a mensagem é de dados;
- *Channel* - Parâmetro de 6 bits, utilizado para indicar o canal de áudio a que o seu conteúdo pertence (possibilitando a existência de 64 canais diferentes);
- *Window* - Parâmetro de 10 bits, utilizado para indicar o tamanho da janela atual;
- *Sequence Number* - Parâmetro de 2 bytes, usado para identificar a sequência de dados atual;
- *Payload* - Parâmetro de 1289 bytes, utilizado para transportar o dados de áudio.

Para o campo *Payload* foi desenhada uma cápsula para levar os dados de áudio acompanhados dos seus parâmetros, sendo estes a frequência de amostragem e bits por amostra, especificados por um código, e a quantidade de dados. O formato desta capsula esta especificado na figura 23.

Descrição dos parâmetros:

- *Audio Type* - Parâmetro de 1 byte, utilizado para indicar o tipo de áudio. Por exemplo:
  1. FA: 44100 Hz e BPS: 16 bits;
  2. FA: 96000 Hz e BPS: 24 bits;
- *Audio Data Length* - Parâmetro de 2 bytes, utilizado para indicar o tamanho dos dados em amostras;
- *Audio Data* - Parâmetro de 1286 bytes, utilizado para transportar os dados de áudio;

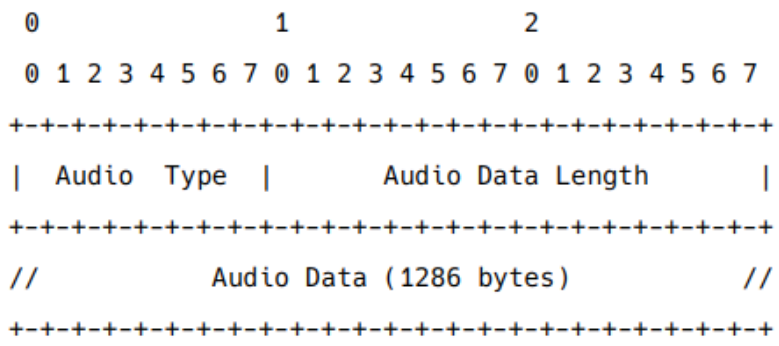


Figura 23: Formato do *Audio Payload*

3.9.3 MDPCP

O protocolo MDPCP funciona numa arquitetura cliente-servidor. O formato das mensagens está especificado na imagem 21.

O formato das mensagens deste protocolo está especificado na imagem 24.

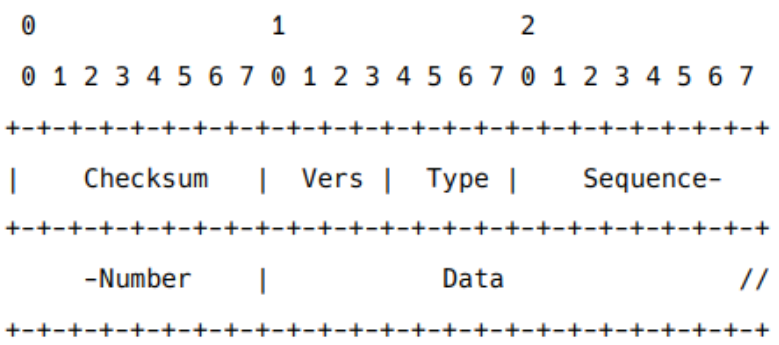


Figura 24: Formato das mensagens do MDPCP

Descrição dos parâmetros:

- *Checksum* - Parâmetro de 1 bytes, utilizado para a deteção de erros;
- *Vers* - Parâmetro de 4 bits, dedicado à versão do protocolo;
- *Type* - Parâmetro de 4 bit, utilizado para indicar que a mensagem é uma confirmação;
    1. *New Device Hello* - Mensagem de introdução do cliente ao servidor;
    2. *New Device Hello ACK* - Resposta à introdução do cliente;
    3. *Get Device Specs* - Mensagem de pedido das especificações de 'Hardware' do cliente;
    4. *Get Device Specs ACK* - Resposta com as especificações de 'Hardware';

5. *Play* - Comando de início de reprodução;
6. *ACK* - Confirmação de receção do comando *Play*;
7. *Pause* - Comando para pausar a reprodução;
8. *Pause ACK* - Confirmação de receção do comando *Pause*;
9. *Stop* - Comando de paragem da reprodução;
10. *Stop ACK* - Confirmação de receção do comando *Stop*;
11. *Custom Command* - Comando costumizável;
12. *Custom Command ACK* - Confirmação de receção do comando *Custom Command*;

- *Sequence Number* - Parâmetro de 2 bytes, usado para identificar a sequência de dados atual;
- *Data* - Parâmetro de tamanho flexível, utilizado para transportar os dados respetivos a cada tipo de comando.

Capítulo 4

Implementação

Neste capitulo descrevem-se o funcionamento e estruturas desenvolvidas a partir das estratégias concebidas no capítulo anterior 3. Através destas foi possível desenvolver três protocolos de comunicação independentes, que por duas aplicações, servidor e cliente, alcançam a reprodução sincronizada de áudio multicanal em redes sem fios. Estas implementações foram desenvolvidas na linguagem C++ em ambiente Linux, sendo utilizado o módulo ALSA para a reprodução de áudio no dispositivo.

Neste trabalho a aplicação de servidor faz a gestão da reprodução de áudio, do calculo das diferenças temporais, a distribuição do áudio, isto pelos vários clientes a ele conectados. No outro lado, a aplicação cliente no que lhe concerne executa os comandos de controlo de áudio, reproduz a áudio, e calcula e faz a manipulação necessária ao áudio, para que este esteja sincronizado relativamente aos restantes clientes.

4.1 Time Monitoring Protocol (TiMP)

4.1.1 Estrutura

O protocolo de monitorização da diferença temporal (TiMP) foi implementado numa classe. Esta possui um processo paralelo que trata da troca de pacotes de dados e calcula as estimações das diferenças temporais (*Offset*) autonomamente. Desta forma é possível fazer uma interação simples com esta e abstrair todo o processo de quem a implementa. A classe implementada possui duas faces, uma de servidor e outra de cliente.

Através do objeto construido, pela classe servidor, é possível usar este a partir dos seguintes métodos:

- Construir o objeto;

- Definir as suas configurações - Configurar o tempo limite de cada resposta, numero máximo de pacotes perdidos, período de recolha de cada estimação;
- Inicializar o seu funcionamento;
- Parar o seu funcionamento;
- Receber os valores estimados da diferença temporal;
- Verificar o estado da ligação.

Ainda na classe servidor, a rotina principal desta é baseada no algoritmo discutido no subcapítulo 3.3. E, depois, foi implementada outra classe no seu interior para abstrair o cálculo da estimação da diferença temporal, o algoritmo deste é descrito em 3.3.2. Esta classe é a *Offset-Calculator*, e possui os seguintes métodos:

- Construir o objeto;
- Definir as suas configurações - Permitindo selecionar os intervalos da moda e media utilizados;
- Calcular a estimação da diferença temporal;
- Calcular o RTT médio;
- Adicionar nova amostra de *Offset*;
- Limpar dados;

Por fim, no lado servidor, para não existam problemas de concorrência entre processo exterior e interior à classe, foram implementados os devidos mecanismos de controlo de concorrência.

Na classe do lado cliente, existe também um processo paralelo, que apenas responde ao servido e recebe as estimações por ele calculadas. A sua classe possui os mesmo métodos que a classe servidor.

4.1.2 Funcionamento

Este protocolo possui um funcionamento simples e síncrono, baseado num sistema de pergunta e resposta. Este após a sua ativação, o lado servidor começa por mandar uma mensagem, ou pacote, a requisitar o tempo no cliente e, o cliente responde com o tempo correspondente ao momento de receção da mensagem do servidor. No momento em que o servidor recebe a mensagem do cliente, este guarda o tempo em que esta foi recebida. Após obter estes três momentos o servidor consegue calcular a diferença temporal(*Offset*) e o RTT correspondentes a esta troca, a seguir alimenta a classe responsável pela estimação da diferença temporal. Esta rotina funcionará durante um determinado período temporal,

o qual é configurável, justamente com este é possível também configurar o tempo de *timeout*, o número de máximo de pacotes falhados, para definir ligação perdida, e também pode ser definido o tempo entre pares de mensagens, sendo este o par pergunta e resposta.

Mal o tempo definido para as trocas de mensagens seja ultrapassado, é calculada a estimação da diferença temporal e verificada se é valida. A validade desta é verificada pela quantidade valores utilizados para a estimação. Este parâmetro é também configurável com o intervalo da moda e media. Caso o valor não seja valido, o processo de recolha de amostras temporais é retomado. Caso a estimação seja valida, o valor estimado é enviado para o cliente. Por sua vez é necessária uma confirmação por parte do cliente para prosseguir para processo de recolha de amostras temporais.

Este protocolo serve outro propósito nas aplicações finais, serve para informar o estado da ligação. Ou seja, se enquanto o cliente estiver a funcionar normalmente significa que a ligação encontra-se ativa, uma vez que o cliente deixe de responder, sabe-se que houve um problema com o cliente e é possível terminar os restantes protocolo de forma segura.

Foi feita uma segunda implementação deste protocolo visando tentar otimizar o seu funcionamento, através de uma implementação assíncrona. Nesta vertente a rotina de recolha de amostras era contínua, ou seja, não existia paragem para calcular e enviar a estimação da diferença temporal. Este tinha um processo paralelo que efetuava o cálculo da estimação, sempre que o número de amostras recolhidas alcançava o valor definido, este tratava de calcular o *Offset* estimado e, de seguida, enviar o valor de estimação. Mas após comparar os resultados obtidos pelas duas versões do protocolo, concluiu-se que não eram extraordinariamente superiores. Por esta razão decidiu-se manter o método síncrono. No capítulo seguinte vão ser comparados os resultados obtidos.

4.2 Multi-Channel Audio Streaming Protocol (MASP)

4.2.1 Estrutura

O protocolo de distribuição de áudio multicanal (MASP) segue a mesma estrutura que o protocolo descrito anteriormente, isto é, este foi implementado numa classe e utiliza dois processos paralelos no seu interior para efetuar a troca de pacotes dados. Abstraindo desta forma a complexidade do seu funcionamento e permitindo uma interação simples. Este protocolo segue uma arquitetura P2P(*Peer-to-Peer*), ou seja, não possui uma clara distinção de quem possui o papel de emissor e recetor, permitindo que ambos possuam qualquer papel.

Através do objeto construido é possível usar este a partir dos seguintes métodos:

- Construir o objeto;

- Definir os seus parâmetros - Definição do tamanho temporal de cada janela de dados e definir o número de confirmações por janela;
- Iniciar o seu funcionamento;
- Parar o seu funcionamento;
- Limpar os dados;
- Enviar dados;
- Receber dados.

Para conseguir alcançar o algoritmo descrito em 3.8, foi necessário implementar outros componentes. Estes foram implementados em classes próprias que são as seguintes:

- *Buffer* de dados a enviar;
- *Buffer* de dados recebidos;
- Gestor da janela de envio;
- Gestor de confirmações por janela;
- *Audio Payload*;

Na classe *Buffer* de dados a enviar é feita a gestão dos dados a enviar. Nesta os dados são processados e geridos entre pacotes de dados a enviar e a confirmar. Através desta é possível saber qual o próximo pacote de dados a enviar, quando esperar pelas confirmações e quando avançar para a janela de dados seguinte. Esta possui os seguintes métodos:

- Construir o objeto;
- Adicionar dados;
- Verificar se todos os dados já foram enviados;
- Verificar se os dados estão dentro da validade;
- Definir o tamanho da janela atual de dados;
- Pedir o próximo pacote de dados;
- Verificar se existe próximo pacote para enviar;
- Confirmar pacote de dados recebido;

- Limpar dados;

Na classe *Buffer* de dados recebidos é feita a gestão dos dados recebidos. Nesta os dados são ordenados pelo seu número de sequência, permitindo a esta formar as confirmações com a lista de números de sequência ordenada dos pacotes de dados recebidos. Esta possui os seguintes métodos:

- Construir o objeto;
- Adicionar pacote de dados;
- Pedir lista de pacotes de dados confirmados;
- Pedir dados;
- Limpar dados;

Na classe gestora da janela de envio, com o nome indica, é feita a gestão do tamanho da janela de envio. Nesta é possível definir o tamanho mínimo da janela, pedir o tamanho atual da janela ou notificar que houve uma perda/falha no envio de um pacote de dados. Esta possui um funcionamento muito simples, se o objeto gestor foi nutrificado de alguma falha, quando for pedido o tamanho da janela, o tamanho devolvido será metade da janela anterior. Caso não tenha sido feita alguma notificação de falha, quando for pedido o tamanho da janela, o tamanho devolvido será o dobro.

Na classe gestora de confirmações por janela é feita uma gestão de quando deverão ser enviadas as confirmações, através do tamanho da janela e das sequências recebidas. Nesta é possível definir o número de confirmações por janela, definir o tamanho da janela atual e verificar através do número de sequência se é para enviar uma confirmação. Esta através do tamanho da janela e do número de confirmações por janela, consegue determinar em que números de sequência deverão ser enviadas confirmações.

A classe *Audio Payload* tem como finalidade serializar e desserializar os dados de áudio contidos no pacote de dados. Esta é utilizada como uma extensão na estrutura do pacote de dados. Permitindo transferir o formato do áudio e o áudio em si.

4.2.2 Funcionamento

Este protocolo possui como principal finalidade a transmissão de áudio multicanal com máxima fiabilidade possível. Esta fiabilidade é determinada pelo tempo de reprodução dos dados enviados, ou seja, enquanto o determinado segmento de áudio não for reproduzido, a sua retransmissão irá ocorrer.

No lado emissor deste protocolo, a rotina de envio de dados através do tamanho do *Payload* de dados, da frequência de amostragem e do número de bytes por amostra, calcula em primeiro lugar a duração de cada pacote de dados e depois com o tamanho mínimo temporal da janela, calcula o tamanho de cada janela. Após determinar as durações pretendidas inicia transmissão dos pacotes de dados, até enviar o

tamanho a determinada janela. Se durante o processo de transmissão forem reportados pacotes de dados falhados, estes possuem prioridade sobre os restantes que ainda não foram enviados. Chegando ao fim da transmissão da janela de dados, este espera até que todas as confirmações sejam recebidas ou até que a validade da janela acabe. Após terminar o envio da janela, este verifica se ainda existem dados a serem enviados no *Buffer* de dados a enviar. Se sim este calcula o tamanho da próxima janela de dados, este calculo depende do sucesso da janela anterior, e calcula a validade temporal da janela, e repete o processo descrito. Ainda no lado emissor deste protocolo, existe uma rotina responsável pela receção das confirmações, presente processo paralelo. Esta simplesmente informa o *Buffer* de dados a enviar de todos os pacotes confirmados e informa o gestor da janela de envio se ocorreram perdas ou falhas.

No lado recetor deste protocolo existe apenas uma rotina responsável pela receção de dados. Está em cada pacote de dados recebido, através do tamanho da janela calcula em que sequencia deverá enviar as confirmações, adiciona o pacote de dados ao *Buffer* de dados recebidos. Caso seja recebida uma retransmissão, a confirmação deste é efetuada no momento. Caso não seja retransmissão é verificado através do número de sequência se é para enviar as confirmações ou não.

4.3 Multi Device Audio Playback Control Protocol (MDPCP)

4.3.1 Estrutura

O protocolo de Controlo de Reprodução de Áudio em Vários Dispositivos (MDPCP), foi implementado em três classes diferentes. Uma dedicada a receber as ligações dos novos dispositivos, outra responsável por enviar os comandos de controlo (classe servidor) e a última responsável por receber os comandos de controlo (classe cliente). Este possui uma arquitetura Cliente-Servidor.

Na classe responsável por receber novas ligações existem os seguintes métodos:

- Construir o objeto;
- Esperar pela ligação do cliente;

Na classe servidor, existem os seguintes métodos:

- Construir o objeto;
- Pedir as especificações de 'Hardware';
- Começar a reprodução de áudio;
- Pausar a reprodução de áudio;
- Parar a reprodução de áudio;

- Método para outros comandos personalizáveis.

Os métodos descritos, correspondem aos comandos enviados ao cliente para que este execute a respetivas ações. O método existente para comandos personalizáveis visa ser adaptado a futuras implementações deste protocolo, tornando este mais flexível. Pode ser utilizado, por exemplo, para controlar o volume.

Na classe cliente, existem os seguintes métodos:

- Construir o objeto;
- Conectar ao servidor;
- Iniciar o seu funcionamento;
- Esperar pelo próximo comando.

Esta classe, como as anteriores, implementa uma estrutura muito simples. Trata da troca das mensagens respetivas ao protocolo e fornece os comandos necessários para a gestão da reprodução do áudio.

4.3.2 Funcionamento

O funcionamento deste protocolo começa pela primeira classe descrita, que trata do estabelecimento da ligação entre o cliente e servidor. Nesta troca de pacotes, o cliente começa por enviar uma mensagem denominada por *"New Device Hello"*, e recebe a seguir a resposta, *"New Device Hello ACK"*, indicando as três portas, respetivas a cada protocolo, que o cliente deve utilizar.

No lado do servidor o funcionamento depende do que lhe seja comandado. No comando de início de reprodução, é necessário de enviar o exato momento temporal em que a reprodução deverá iniciar e o número da sequência em que este deve começar. Isto para garantir que todos os dispositivos iniciam a reprodução no mesmo momento e na mesma sequência de áudio. O número de sequência é principalmente usado para garantir que todos os clientes iniciem a reprodução após uma pausa na mesma sequência de áudio. No comando de pausar, não é enviada informação extra ao cliente, mas o cliente na sua resposta envia o número de sequência da última sequência de áudio que reproduziu. No comando de parar a reprodução de áudio, não é enviado nem recebido informação extra do cliente.

4.4 Aplicação Servidor

4.4.1 Estrutura

A aplicação servidor é composta por uma classe gestora do dispositivo e um conjunto de funções que gere esta classe. Tanto a classe como a restante aplicação implementa processos paralelos. É na classe

gestora onde se encontram os três protocolos desenvolvidos a trabalhar em conjunto.

Através desta classe é possível aceder aos seguintes métodos:

- Construir o objeto;
- Inicializar o seu funcionamento;
- Parar o seu funcionamento;
- Definir as configurações de cada protocolo;
- Pedir as estimações temporais atuais;
- Pedir o número de sequência de áudio atual;
- Definir o número de sequência de áudio;
- Pedir o canal de áudio;
- Iniciar/Continuar a reprodução de áudio;
- Verificar se a reprodução está parada;
- Verificar se a reprodução está pausada;
- Enviar comando;

As restantes funcionalidades desta aplicação estão distribuídas em quatro funciones principais:

1. Função gestora de novos dispositivos - Esta é gerida por um processo independente, que recebe as ligações de novos dispositivos e coloca os numa estrutura de dados partilhada;
2. Função de início da reprodução de áudio - É nesta função que o algoritmo descrito em 3.7 é aplicado;
3. Função de pausa da reprodução de áudio;
4. Função de paragem da reprodução de áudio;

4.4.2 Funcionamento

O funcionamento desta aplicação está principalmente espalhada pelas quatro funções descritas anteriormente. Quando a aplicação servidor é iniciada são iniciados dois processos paralelos. O primeiro tem a função de receber novos dispositivos e iniciá-los. O segundo é responsável pela interação humana, isto é, receber os comandos do utilizador da aplicação e enviar estes para os restantes dispositivos através das últimas três funções descritas anteriormente.

Quando um novo dispositivo é recebido é iniciada a classe gestora do dispositivo. No que lhe concerne, o primeiro processo contido nesta, vai inicializar as classes responsáveis pelos protocolos implementados. Após os protocolos serem corretamente inicializados, este processo vai esperar até que lhe seja pedido enviar um comando. O segundo processo, vai esperar até que lhe sejam passados dados de áudio para enviar.

Quando a função de início da reprodução de áudio é chamada, esta começa por verificar o estado da reprodução nos vários dispositivos, isto é, verifica se estes estão pausados ou parados. Caso algum não esteja, esta função é interrompida. Caso estes estejam em conformidade, são usadas as estimações temporais de cada dispositivo para calcular o momento do início da reprodução em cada dispositivo, através o algoritmo estudado em 3.7. Além do cálculo do momento, é também escolhido o número de sequência de áudio onde os dispositivos devem começar. Depois, caso os dispositivos se encontrem pausados, é enviado o momento e número de sequência a cada respetivo dispositivo através do método iniciar/continuar a reprodução de áudio. Caso os dispositivos se encontrem parados, é iniciado um novo processo que vai ler o ficheiro de áudio, passar os dados para as classes gestoras responsáveis pelo respetivo canal de áudio, e segue-se o mesmo funcionamento.

Por fim, as funções de pausar e parar a reprodução de áudio possuem um funcionamento muito semelhante. Por cada dispositivo é verificado se este já se encontra no respetivo estado, isto é pausado ou parado, se sim passa para o dispositivo seguinte, se não é enviado o respetivo comando para pausar ou parar a reprodução.

4.5 Aplicação Cliente

4.5.1 Estrutura

A aplicação cliente é composta por uma classe principal que, através da informação recebida através dos três protocolos desenvolvidos, faz a gestão da reprodução de áudio. Esta classe implementa processos paralelos.

Através desta classe é possível aceder aos seguintes métodos:

- Construir o objeto;

- Inicializar o seu funcionamento;
- Parar o seu funcionamento;
- Definir as configurações de cada protocolo;

Para gerir todas as funcionalidades que esta classe implementa, foi necessário desenvolver outros componentes para auxiliar esta. Os componentes implementados foram os seguintes:

- Uma classe que implementa o algoritmo descrito em 3.5, chamada *Frames-Calculator*;
- Uma classe que abstrai e simplifica as funcionalidades do ALSA, chamada *ALSA-Playback*;
- Uma classe para auxiliar na comunicação entre processos para comandar a reprodução, chamada de *Playback-State*;
- Um conjunto de funções que implementam o algoritmo de reamostragem descrito em 3.6;

Na classe *Frames-Calculator*, são calculadas as amostras a remover/adicionar do áudio. Estas são calculadas através do *Time Skew*, que, é calculado a partir das diferenças temporais(*Offset's*). Através desta classe é possível aceder aos seguintes métodos:

- Construir o objeto;
- Adicionar valor de diferença temporal(*Offset*);
- Calcular número de amostras;
- Limpar valores das estimações;

A classe *ALSA-Playback* é uma simples extensão das principais funcionalidades do ALSA abstraídas numa classe. Através desta classe é possível aceder aos seguintes métodos:

- Construir o objeto;
- Iniciar Dispositivo com os parâmetros do áudio;
- Reproduzir sequencia de dados de áudio;
- Desligar dispositivo;

A classe *Playback-State* é utilizada para comunicar os comandos de início, pausa e parar para o processo responsável pela reprodução do áudio. Através desta classe é possível aceder aos seguintes métodos:

- Construir o objeto;
- Definir o momento de início da reprodução;
- Esperar pelo momento de início da reprodução;
- Pausar a reprodução;
- Verificar se está em reprodução;

4.5.2 Funcionamento

O funcionamento desta aplicação começa pela inicialização da classe cliente. Esta, começa por estabelecer a ligação com o servidor, iniciar os outros dois protocolos desenvolvidos e um processo para se responsabilizar pela reprodução dos dados de áudio.

O processo principal, na classe cliente, é responsável por receber e executar os comandos vindos do servidor. Este quando recebe o comando de iniciar/continuar a reprodução de áudio, define qual é o número de sequência onde vai começar e, através da classe *Playback-State*, define o momento em que o áudio deve ser iniciar. O mesmo acontece para os comandos de pausar/parar, através da classe *Playback-State*, pausa-se a reprodução e no caso de paragem os dados de áudio recebidos são eliminados.

O processo responsável pela reprodução de áudio, começa por configurar os parâmetros de áudio na classe do ALSA e fica à espera do momento de início da reprodução de áudio, na classe *Playback-State*. Quando chega o momento de início, este começa por pedir a sequência de dados de áudio selecionada, depois adiciona o *Offset* calculado à classe *Frames-Calculator* e, através deste, verifica se a deriva temporal(*Time Skew*) nesse momento é suficiente para manipular o áudio. Se sim são calculados o número de amostras a manipular e através do algoritmo desenvolvido de reamostragem estas são adicionadas/-removidas, e depois a sequência manipulada é reproduzida através da classe do ALSA. Caso a deriva temporal(*Time Skew*) não seja suficiente para a manipulação do áudio, este é reproduzido.

Capítulo 5

Testes e Resultados

Concluindo o capitulo anterior 4, é necessário testar a implementação feita e demonstrar os resultados obtidos. Para tal, baseando na arquitetura descrita em 3.2, foram utilizados os seguinte dispositivos com as respetivas características:

- Servidor - ThinkPad t490s, com um processador i7-8565U e 16 GB de RAM;
- Cliente 1 - Raspberry Pi 3 Modelo B, com um processador Broadcom BCM2837 e 1 GB de RAM;
- Cliente 2 - Raspberry Pi 3 Modelo A+, com um processador Broadcom BCM2837B0 e 512 MB de RAM;
- Ponto de acesso sem fios - AC750 WI-FI de 2.4GHz

Neste capitulo os valores escolhidos para as varias comparações serão a soma das derivas das diferenças, soma dos *Time Skew's*, e a soma de amostras musicais amostradas a 44.1 KHz (ou seja cada 1/44100 = 22.67us). Foram estes os valores escolhidos pois estão no formato utilizado pelas aplicações desenvolvidas. É importante notar que todos estes testes foram executados numa rede domestica e não num ambiente isolado.

5.1 Resultados do TiMP

Neste testes, o dispositivo retratado como cliente 1 acima, foi utilizado como servidor e o cliente 2 foi utilizado como cliente. Esta foi a estratégia adotada para que fosse possível utilizar os pinos dos Raspberry Pi's para comunicar os instantes representativos do tempo real do dado dispositivo. É importante também

notar que cada gráfico ou teste é independente, ou seja, os valores obtidos não são reutilizados em gráficos diferentes.

5.1.1 Comparação entre método síncrono e assíncrono

Como foi descrito no sub-capitulo 4.1.2, foram desenvolvidas duas versões do protocolo TiMP. Uma que utilizava um método síncrono e outra que utilizava um método assíncrono.

Nas imagens seguintes 25 e 26, são demonstrados os valores da soma de *Time Skew's* reais e calculados, através das duas versões do protocolo ao longo de, aproximadamente, 4 minutos.

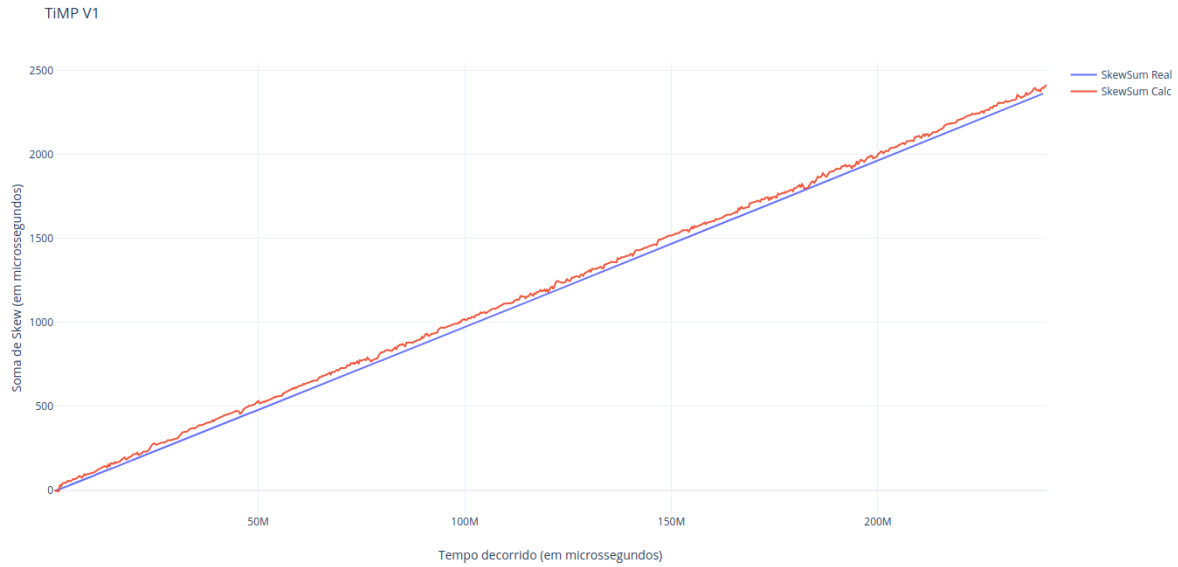


Figura 25: Comparação das somas de *Time Skew* calculadas e reais da versão síncrona

Partindo de uma análise visual, é possível observar que ambos resultado são bastantes bons. É também possível afirmar que a versão 2 do protocolo TiMP, a versão assíncrona apresenta resultados mais próximos dos reais.

Analisando de uma forma estatística, através do erro médio absoluto os resultados, pode-se afirmar, com certeza, que os resultados obtidos pela versao 2 do protocolo TiMP foram superiores. Estes resultados encontram-se na tabela 5.

V1 (Síncrono)	**V2 (Assíncrono)**
38.5015	28.2703

Tabela 5: Resultados do erro médio absoluto (em microssegundos) da soma de *Time Skew*

Nas imagens seguintes 27 e 28, são demonstrados os valores da soma de amostras musicais reais e calculadas, através das duas versões do protocolo ao longo de, aproximadamente, 4 minutos.

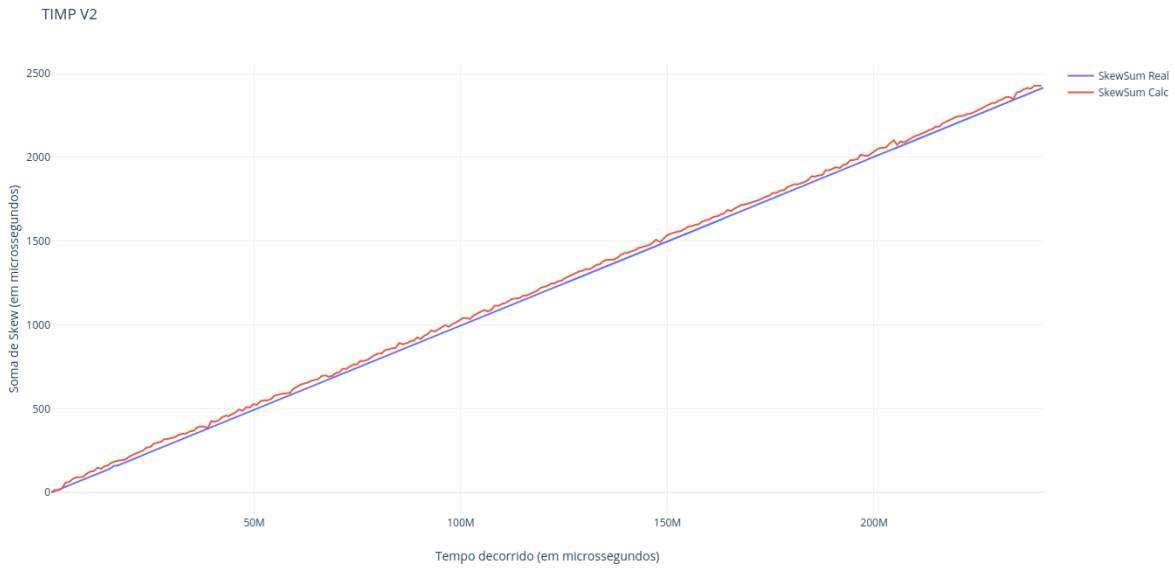


Figura 26: Comparação das somas de *Time Skew* calculadas e reais da versão assíncrona

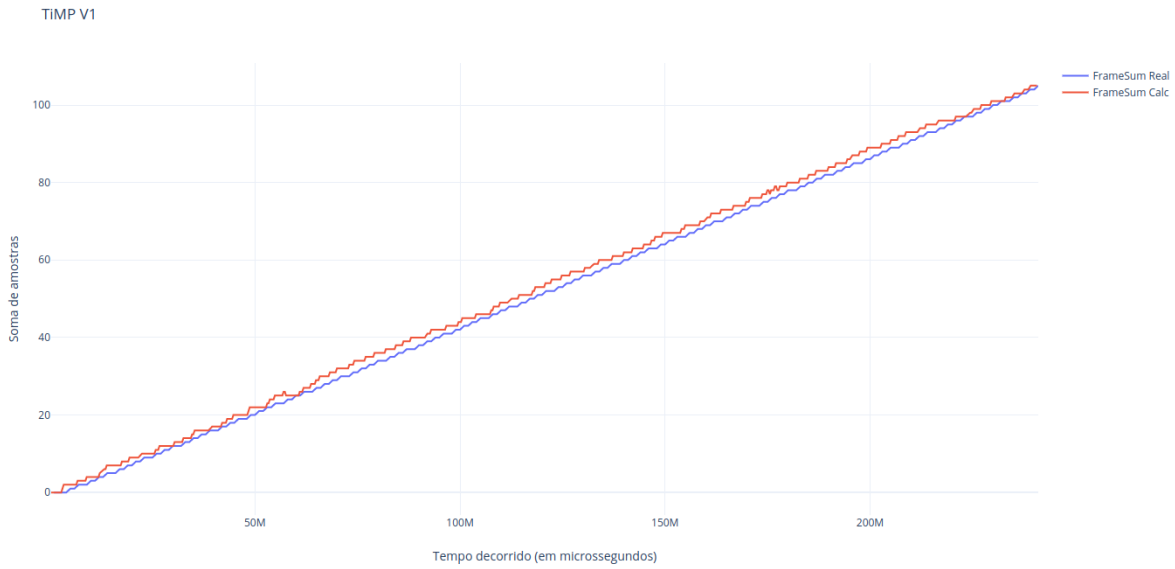


Figura 27: Comparação das somas de amostras musicais calculadas e reais da versão síncrona

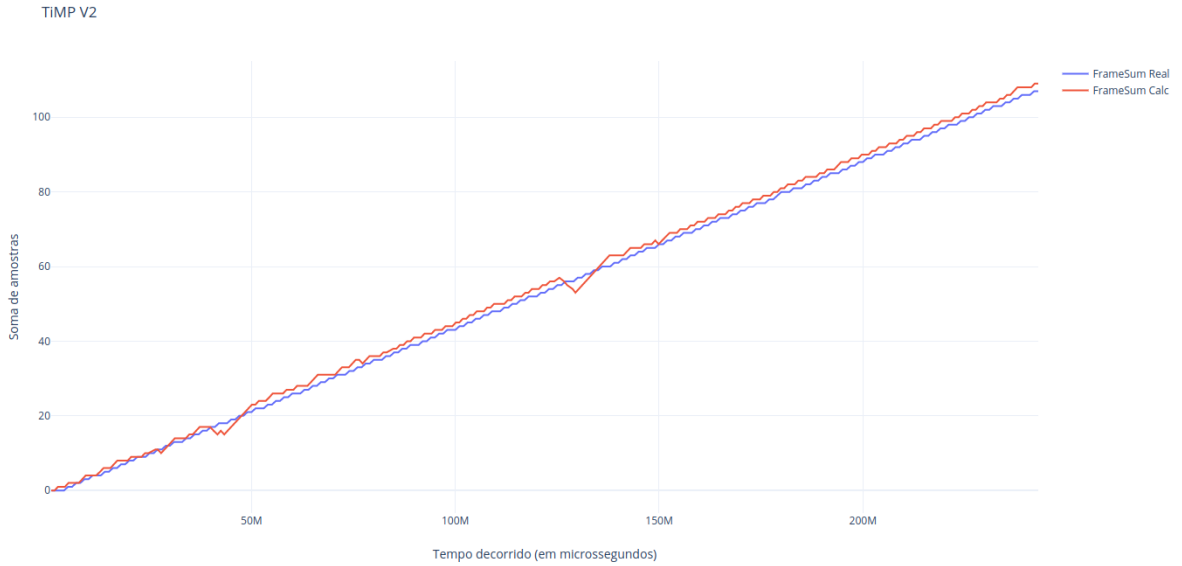


Figura 28: Comparação das somas de amostras musicais calculadas e reais da versão assíncrona

Novamente através de uma análise visual, mas desta vez analisando em numero de amostras, é possível observar que os resultados encontram-se bastante equivalentes.

Em termos de erro médio absoluto, da soma de amostras, pode se dizer que a diferença foi menor, mas a versão 2 do protocolo TiMP continua a com um erro inferior. Estes resultados encontram-se na tabela 6.

V1 (Síncrono)	**V2 (Assíncrono)**
1.7095	1.4526

Tabela 6: Resultados do erro médio absoluto da soma de amostras musicais

Com base nos resultados obtidos, sabe-se que nos testes feitos a versão 2 obteve melhores resultados. Mas como esta versão não foi Tao bem explorada como a versão 1, decidiu-se utilizar a versão 1. De qualquer das formas, para futuros trabalhos, esta versão deve ser melhor estudada e otimizada pois oferece melhores perspetivas.

5.1.2 Variação da carga da rede

Os seguintes testes foram feitos para analisar o comportamento e impacto que a ocupação da largura de banda da rede tem no calculo da deriva da diferença temporal *Time Skew*.

Estes testes foram feitos com o auxílio da ferramenta IPerf3, onde foi definida uma largura de banda entre dois dispositivos diferentes dos envolventes pelo protocolo TiMP.

5.1.2.1 10 Mbits

Neste primeiro teste foram apenas ocupados 10Mbits da rede. Os resultados obtidos podem ser observados nas figuras 29 e 30.

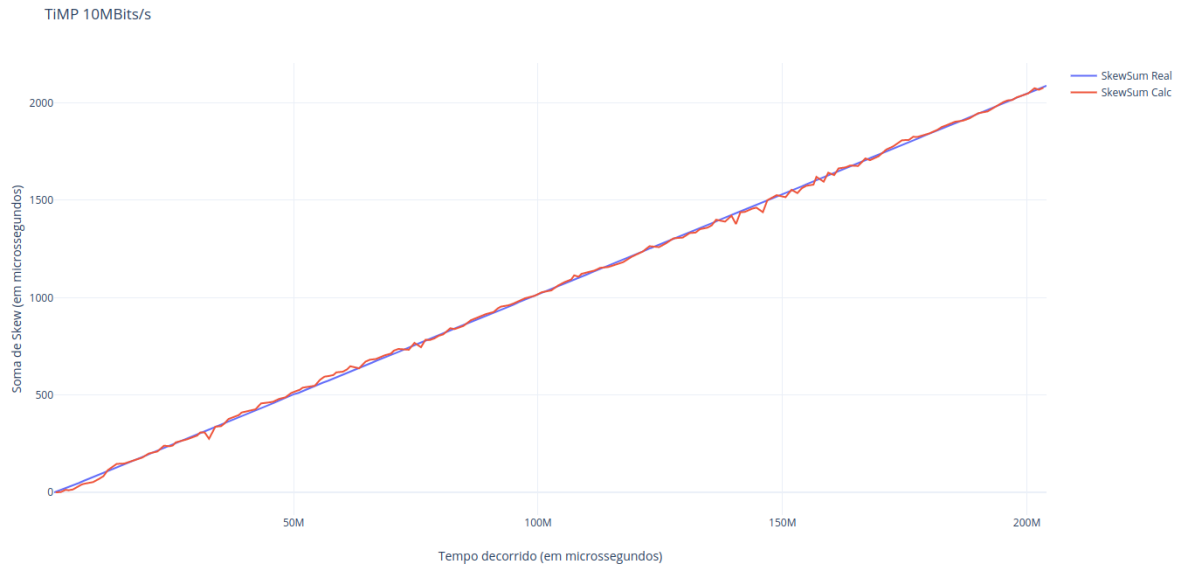


Figura 29: Comparação das somas de *Time Skew* calculadas e reais pelo protocolo TiMP com 10Mbits/S da rede ocupados

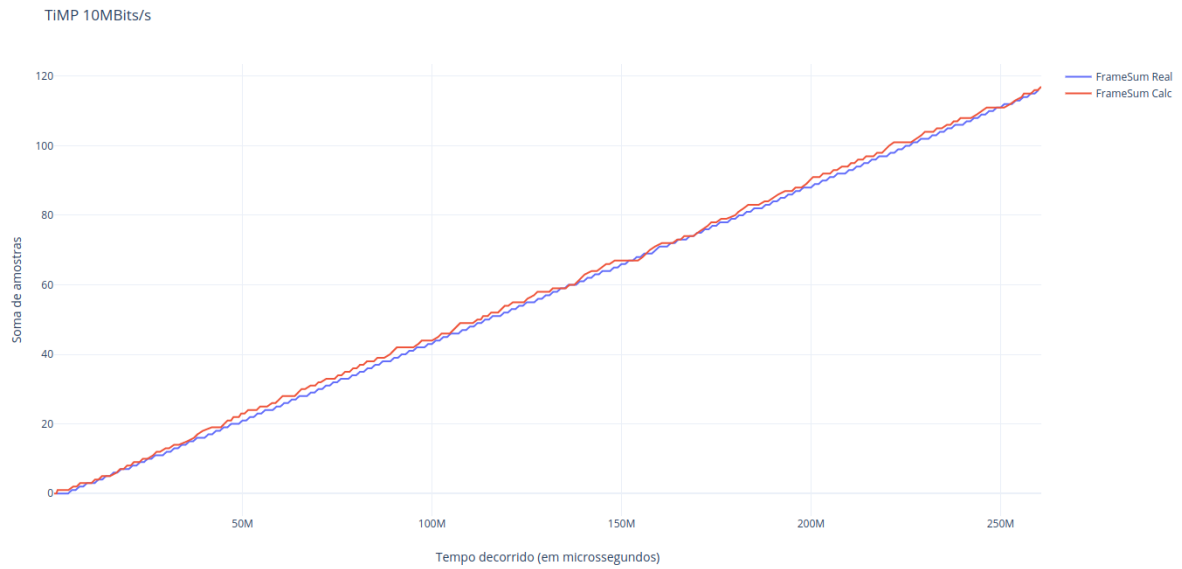


Figura 30: Comparação das somas de amostras musicais calculadas e reais pelo protocolo TiMP com 10Mbits/S da rede ocupados

Observando os gráficos obtidos, é possível verificar que o protocolo TiMP não sofre nenhum impacto. Calculando os erros médios absolutos observa-se que o seu funcionamento mantém-se bastante bom. Os erros médios absolutos encontram-se na tabela 7.

Soma de *Time Skew*	**Soma de amostras musicais**
10.0312	1.1923

Tabela 7: Resultados do erro médio absoluto da soma de *Time Skew* e amostras musicais

5.1.2.2 50 Mbits

No segundo teste foram ocupados 50Mbits da rede. Os resultados obtidos podem ser observados nas figuras 31 e 32.

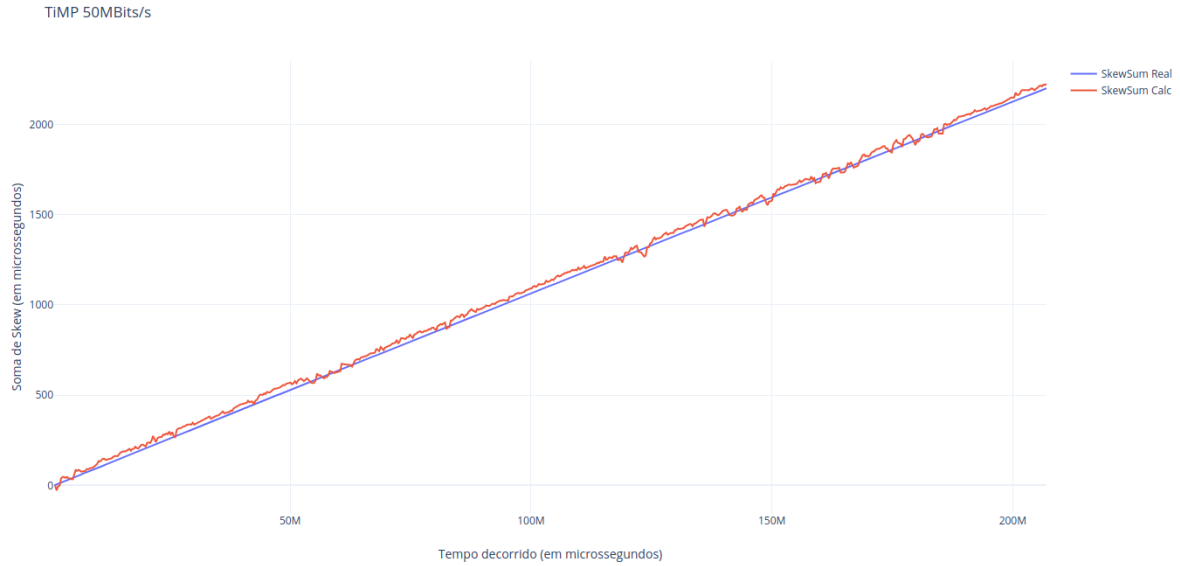


Figura 31: Comparação das somas de *Time Skew* calculadas e reais pelo protocolo TiMP com 10Mbits/S da rede ocupados

Novamente observando os gráficos obtidos, é possível afirmar que a precisão diminui ligeiramente em relação ao teste anterior mas continua sem um impacto notório. Calculando os erros médios absolutos observa-se que o seu funcionamento mantém-se bastante bom. Os erros médios absolutos encontram-se na tabela 8.

Soma de *Time Skew*	**Soma de amostras musicais**
23.8993	1.6201

Tabela 8: Resultados do erro médio absoluto da soma de *Time Skew* e amostras musicais

5.1.2.3 100 Mbits

Por fim, no terceiro e último teste de ocupação da largura, foram ocupados 100 Mbits da rede. Os resultados obtidos podem ser observados nas figuras 33 e 34.

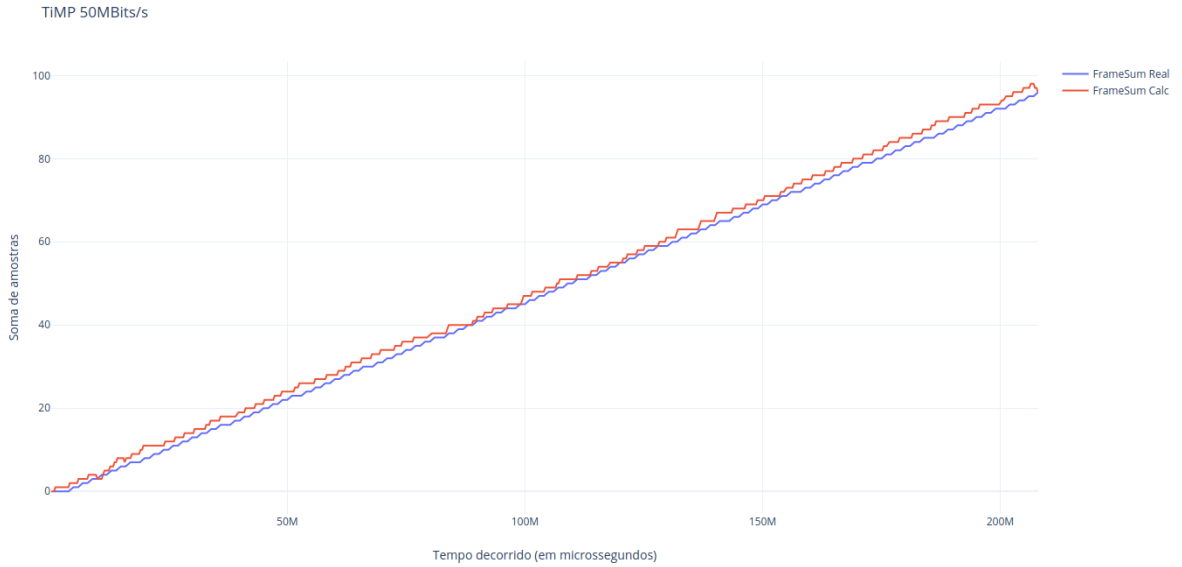


Figura 32: Comparação das somas de amostras musicais calculadas e reais pelo protocolo TiMP com 50Mbits/S da rede ocupados

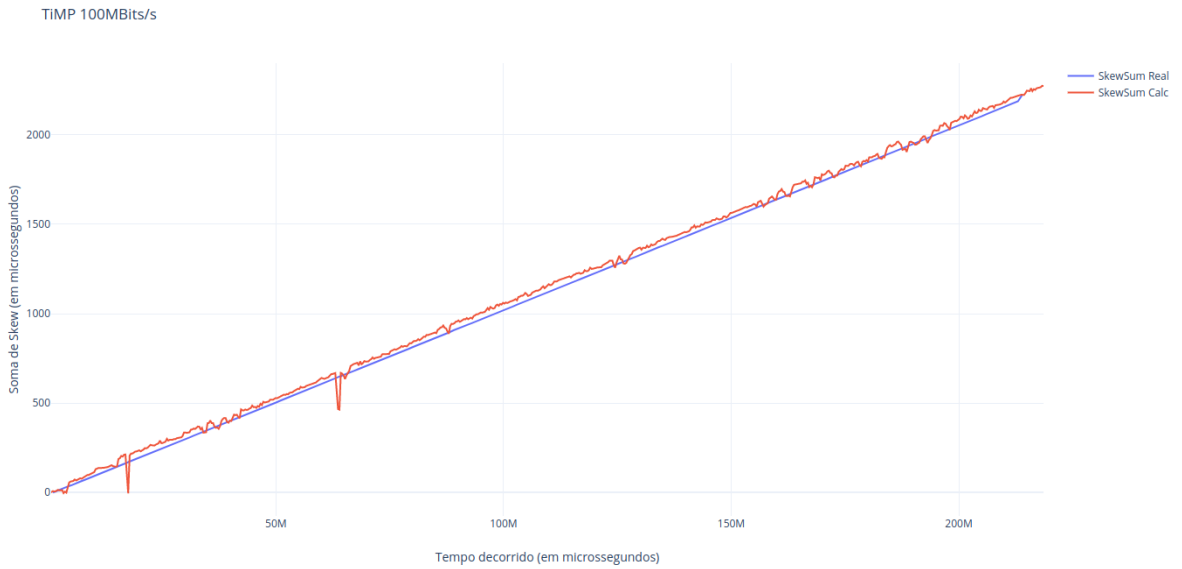


Figura 33: Comparação das somas de *Time Skew* calculadas e reais pelo protocolo TiMP com 100Mbits/S da rede ocupados

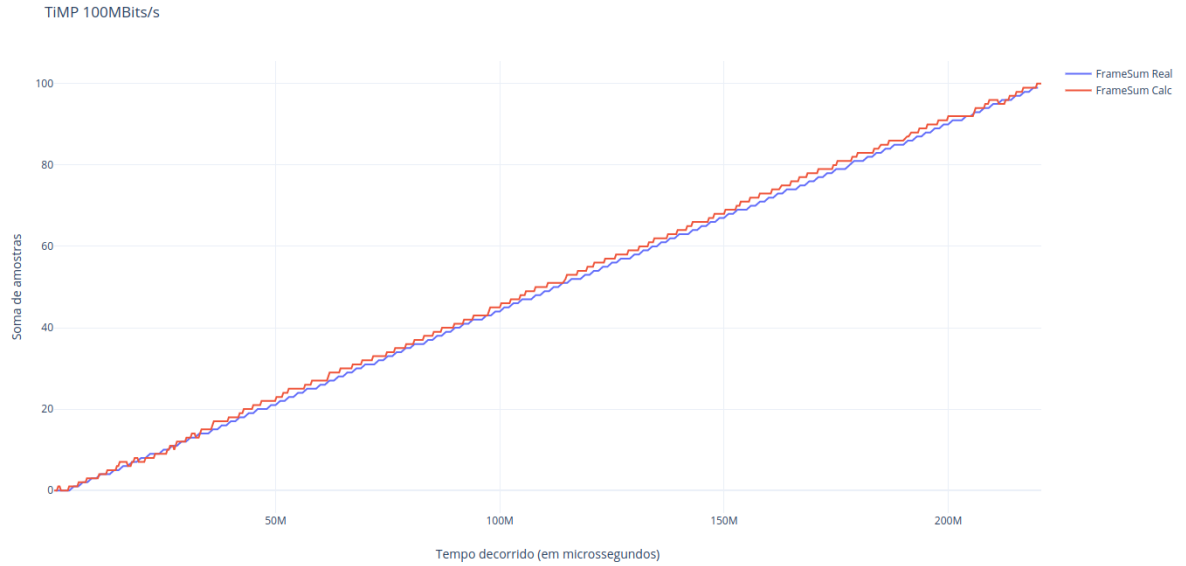


Figura 34: Comparação das somas de amostras musicais calculadas e reais pelo protocolo TiMP com 100Mbits/S da rede ocupados

Observando os gráficos obtidos, é possível afirmar que o comportamento mantém-se constante independentemente da largura de banda utilizada. Calculando os erros médios absolutos observa-se que o seu funcionamento mantém-se bastante bom. Os erros médios absolutos encontram-se na tabela 9.

Soma de *Time Skew*	**Soma de amostras musicais**
27.3746	1.1545

Tabela 9: Resultados do erro médio absoluto da soma de *Time Skew* e amostras musicais

5.1.3 Variação da temperatura nos dispositivos

No seguinte teste pretende-se verificar o impacto da temperatura no funcionamento do protocolo. Para este propósito foi usado um secador no dispositivo cliente e manteve-se o dispositivo servidor à temperatura ambiente. As temperaturas foram medidas durante a execução do através do comando `vcgencmd measure_temp`. No dispositivo cliente a temperatura manteve-se entre os 66°C e os 71°C, no dispositivo cliente a temperatura manteve-se entre os 42°C e 47°C. Os resultados obtidos podem ser observados nas figuras 35 e 35.

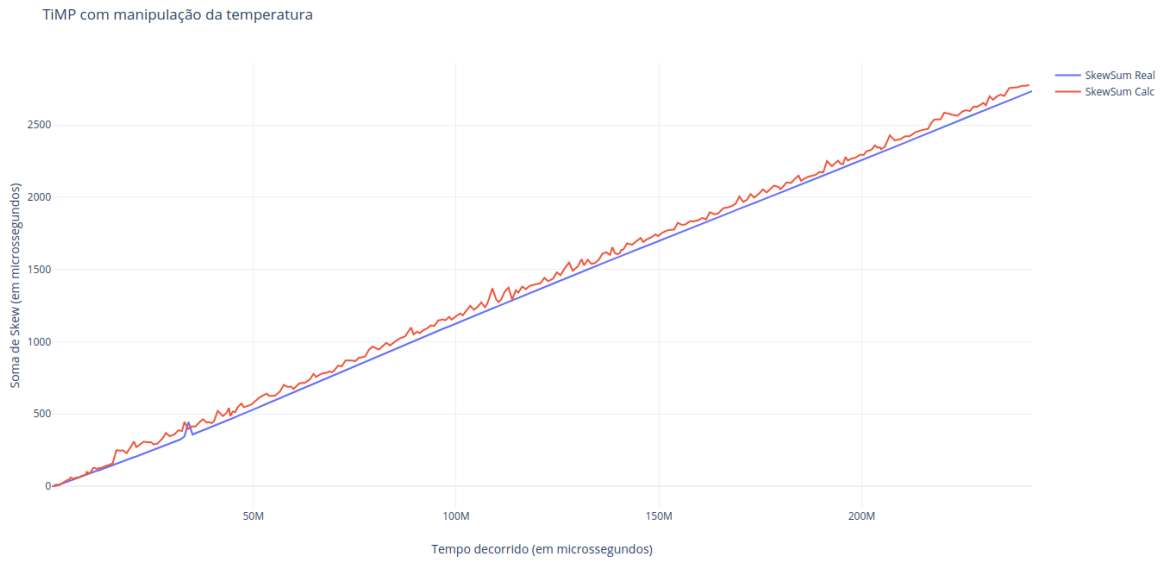


Figura 35: Comparação das somas de *Time Skew* calculadas e reais pelo protocolo TiMP com manipulação da temperatura

Observando os gráficos obtidos, é possível afirmar que já se nota algum impacto no funcionamento do protocolo. Pelos valores dos erros médios absolutos obtidos pode-se dizer o mesmo, a precisão baixou ligeiramente mas continua boa, isto analisando o erro médio absoluto da soma de *Time Skew*. Analisando o erro médio absoluto da soma de amostras musicais o impacto não é tão grande. Aqui vê-se também uma grande vantagem desta técnica que é dar uma margem de tolerância. Os erros médios absolutos encontram-se na tabela 9.

Soma de *Time Skew*	**Soma de amostras musicais**
49.4607	1.7551

Tabela 10: Resultados do erro médio absoluto da soma de *Time Skew* e amostras musicais

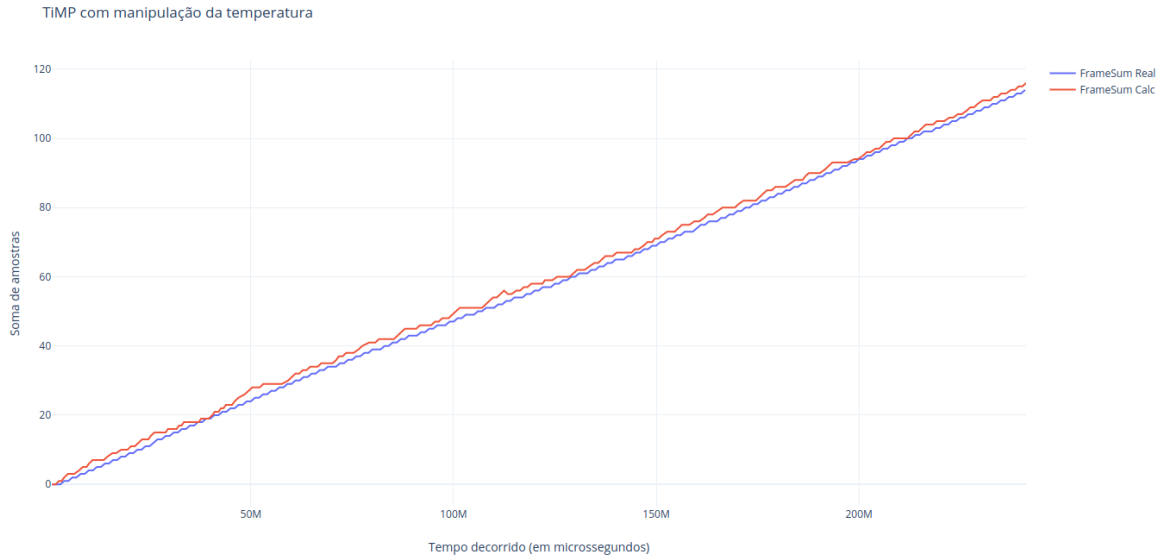


Figura 36: Comparação das somas de amostras musicais calculadas e reais pelo protocolo TiMP com manipulação da temperatura

5.2 Sincronização Inicial

No seguinte teste, foi testada a precisão no calculo do momento inicial de reprodução. Para este efeito os pinos do Raspberry Pi's foram ligados a um osciloscópio.

Um detalhe considerado foi a inicialização da interface e a configuração da mesma. Nos trabalhos anteriores [2] e [7], esta configuração era realizada após o momento inicial de reprodução e, visto que este processo possui um tempo indeterminado acabava por causar discrepâncias na reprodução inicial. Para evitar esta ocorrência, a configuração da interface é feita antes do momento inicial de reprodução, ou seja, aquando da receção do primeiro pacote de dados de áudio pelo cliente. Neste encontram-se os parâmetros do áudio que o cliente utiliza para configurar a interface de som. Desta forma, os atrasos causados por esta configuração ficam contidos no tempo extra (por exemplo, o tempo extra para *Buffering*), que pode ser definido inicialmente. Por outro lado este funcionamento requer que os parâmetros do áudio sejam sempre os mesmos, não podendo alternar estes durante a execução.

Na figura 37, estão presentes 4 testes de inicio da reprodução, onde se pode verificar uma gama de possíveis resultados. Nesta imagem é possível observar um atraso mínimo em que, com a dada divisão temporal, não se consegue estimar o seu valor. Por outro lado, noutro teste, consegue-se observar um atraso de aproximadamente 300 microssegundos. Obviamente foram feitos mais testes que estes 4. Estes foram os escolhidos porque possuem os resultados mais populares.

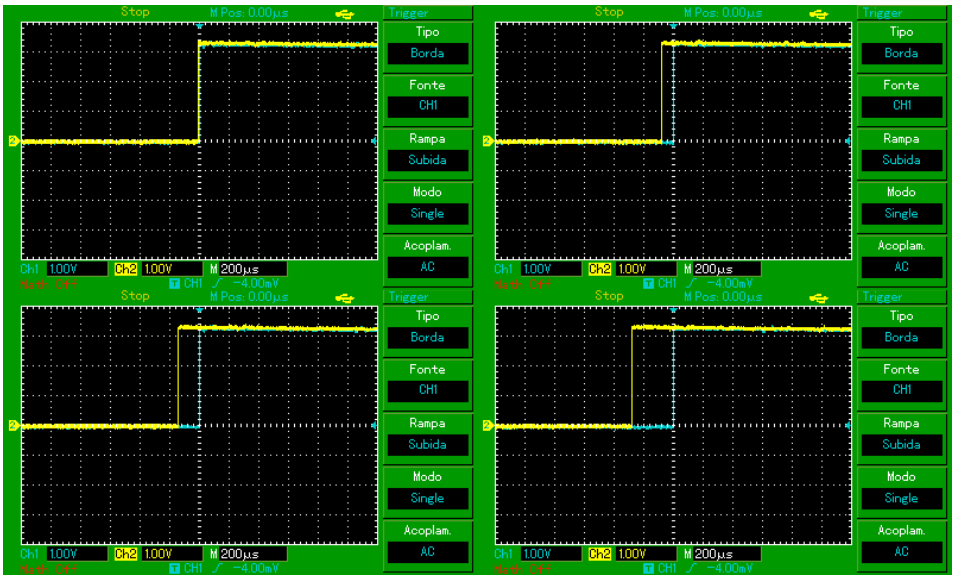


Figura 37: Resultados do inicio da reprodução medidos no osciloscópio

5.3 Funcionamento geral

Por fim foi testado o funcionamento geral do projeto final. Para este efeito as saídas de áudio dos Raspberry Pi's foram ligadas ao osciloscópio e foi usado ficheiro de áudio para reproduzir uma onda sinusoidal de 1KHz. Foram executados dois testes independentes para demonstrar o seu funcionamento.

No teste apresentado na figura 38, pode-se observar três capturas do ecrã do osciloscópio, correspondentes a três momentos seguidos. Na primeira captura observam-se as ondas sincronizadas. Na seguinte, nota-se uma dessincronização naturalmente devido aos diferentes ritmos dos seus relógios. Na ultima captura é possível reparar que a dessincronização é ligeiramente inferior, podendo observar aqui uma correção da dessincronização. Devido ao tempo que o osciloscópio demora efetuar as capturas não foi possível obter melhores capturas.

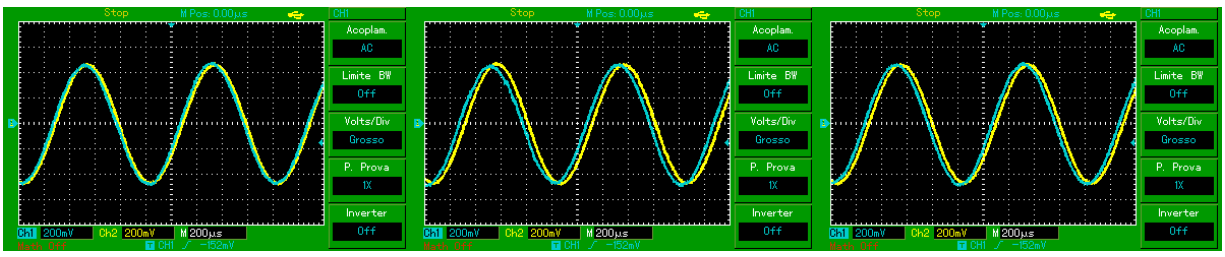


Figura 38: Teste funcionamento geral

O teste seguinte teve como objetivo observar a dessincronização resultante ao fim de 1 hora de funcionamento. Este foi feito nas mesmas condições que o teste anterior.

Na figura 39 é possível observar uma captura de ecrã do osciloscópio no inicio de funcionamento. Nesta figura é nota-se uma dessincronização inicial de aproximadamente 80 microssegundos. Na figura

40, que possui a captura de ecrã após 1 hora de funcionamento, observa-se que a dessincronização resultante diminui para, aproximadamente, 30 microssegundos.

Analisando os resultados obtidos, pode-se assumir que houve uma dessincronização de 50 microssegundos no espaço de 1 hora. O que comprova que os algoritmos definidos podem ser mais refinados ainda, mas não deixa de ser um resultado excelente.

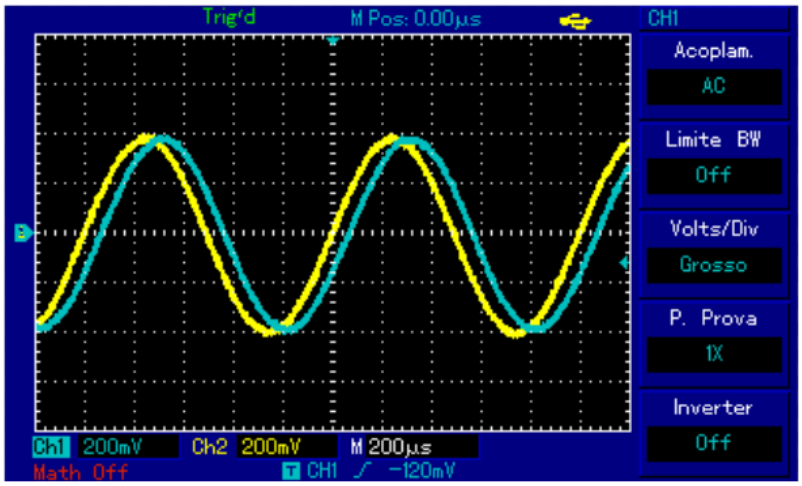


Figura 39: Captura de ecrã dos osciloscópio no inicio do funcionamento

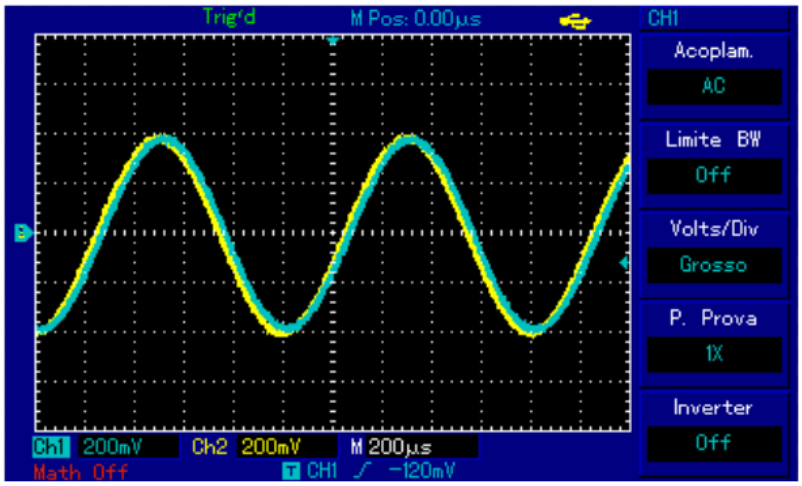


Figura 40: Captura de ecrã dos osciloscópio no fim de 1 hora de funcionamento

Capítulo 6

Conclusão

Como principal objetivo no trabalho integrado nos trabalhos desta dissertação foi proposto desenvolver um sistema estruturado, totalmente apoiada em *software* e sem recurso a tecnologias proprietárias, que conseguisse efetuar a distribuição de áudio multicanal em redes sem fios. Foi previsto que a solução desenvolvida conseguisse calcular a dessincronização com precisão suficiente para permitir um mecanismo eficiente de sincronização e distribuição de áudio em diferentes dispositivos terminais. Além disso, a arquitetura deveria incluir um protocolo simples, mas capaz de efetuar a distribuição de áudio para vários dispositivos com mecanismos de deteção e correção de erros, comprimir e expandir o áudio sem comprometer a qualidade do mesmo e calcular o momento inicial de reprodução de cada dispositivo. Após a análise critica dos resultados obtidos de todos estes e experimentações efetuadas, incluindo testes funcionais do prototipo final, pode concluir-se que os objetivos foram atingidos com um elevado grau de sucesso.

Este projeto iniciou-se com a análise do algoritmo de cálculo da dessincronização temporal e da deriva temporal entre diferentes dispositivos desenvolvido no trabalho [7], seguido por uma otimização do mesmo, alcançou-se resultados aproximadamente 40% superiores ao algoritmo original. Para o cálculo da deriva temporal seguiu-se uma abordagem diferente, em vez de se calcular uma aproximação faz-se a soma das diferenças das estimações das diferenças temporais. Visto que os resultados obtidos em 3.3.2 variavam em torno do zero, conclui-se que esta nova solução era mais apropriada foi a solução mais apropriada, garantido assim que a soma tende para um valor central, corrigindo assim os erros positivos e negativos das estimações. A partir das estimações das diferenças temporais foi desenvolvida uma técnica para calcular o mesmo instante em diferentes dispositivos, possibilitando a sincronização inicial nos múltiplos dispositivos.

Na etapa seguinte desenvolveu-se o mecanismo de expansão e compressão do áudio que produz

resultados muito próximos de uma ferramenta de reamostragem. Observando os resultados obtidos em 3.6, nota-se a clara diferença entre os resultados de uma simples interpolação linear entre amostras e o mecanismo desenvolvido. Para calcular a taxa de compressão/expansão do áudio foi desenvolvido que calcula o número de amostras com base no valor acumulado da deriva temporal e na taxa de amostragem do respetivo áudio.

Adicionalmente, desenvolveu-se também uma estratégia de distribuição de áudio multicanal visando efetuar uma transmissão fiável de múltiplos fluxos de dados em tempo real em redes sem fios. Esta estratégia em vez de usar *timestamp's* para a reprodução, como no RTP, possui uma validade de transmissão por cada pacote de dados. O que significa que cada pacote pode ser retransmitido num certo intervalo de tempo, definido pela taxa de reprodução no cliente. Para a retransmissão de pacotes são utilizadas técnicas de deteção de erros (*Checksum*), de ordenação de pacotes e confirmações. E para aumentar a eficiência da transmissão é utilizado um mecanismo de controlo de fluxo semelhante aos implementados pelo TCP e SCTP.

Partindo das estratégias previamente mencionadas, foram desenvolvidos três protocolos de comunicação devidamente estruturados e integrados:

- *Time Monitoring Protocol* (TiMP), que é responsável por calcular as estimações das diferenças temporais e, por estar sempre em comunicação, possui a funcionalidade de reportar o estado da ligação;
- *Multi-Channel Audio Streaming Protocol* (MASP), que é responsável pela distribuição de áudio multicanal;
- *Multi Device Audio Playback Control Protocol* (MDPCP), que é responsável pelo envio dos comandos de controlo de reprodução do áudio.

Por fim, todas a soluções e protocolos desenvolvidos foram integrados em duas aplicações protótipos para ser possível validar os resultados obtidos. Comprovou-se que é possível desenvolver uma solução para distribuição de áudio multicanal sincronizado em redes sem fios normalizadas definindo e implementando apenas mecanismos em *Software*, sem a necessidade de *Hardware* ou sistemas operativos especiais. Analisando os resultados descritos no capítulo 5, verifica-se um funcionamento integrado correto e promissor. Também foram feitos alguns testes auditivos à aplicação, embora estes não tenham sido referidos, pois a análise através do osciloscópio é muito mais precisa. Nestes testes, apesar de não se notar dessincronia na reprodução, notam-se volumes diferentes apesar do ganho ser o mesmo e as colunas do mesmo modelo nos diferentes dispositivos, um fenómeno que ainda não foi possível entender e resolver, mas parece claro tratar-se de um problema de implementação do código do protótipo.

6.1 Trabalhos futuros

Após alcançar os objetivos propostos para esta dissertação, pode-se refletir, por onde pode evoluir e melhorar.

O primeiro aspeto a rever seria no cálculo da diferença temporal entre dispositivos, apesar de terem sido alcançados valores de erro absoluto ótimos, este necessita de uma maior consistência nos valores que calcula. Porque, no longo prazo, é mais importante a consistência ou estabilidade dos valores estimados do que a precisão dos mesmos.

Adicionalmente, pode destacar-se a possibilidade de desenvolvimento de uma aplicação de reprodução de áudio que conseguisse gerir uma biblioteca de músicas, que integrasse os serviços de *Streaming* mais populares e que suportasse vários formatos de áudio. Atualmente, a grande parte de serviços de *Streaming* já possuem API's para serem utilizadas por terceiros. Para o suporte a vários formatos de áudio a utilização da *Framework* 'FFmpeg' seria suficiente, possuindo a vantagem de ser uma *Framework* de código aberto. A reprodução de vídeo é outra possibilidade que pode também ser integrada nesta aplicação.

Outra evolução deste projeto, mais ambiciosa, seria a integração da aplicação como um serviço no próprio sistema operativo, conseguindo assim intercetar todo o áudio que se direciona a placa de som. Esta teria uma integração semelhante à que o *Bluetooth* possui nos sistemas operativos atualmente.

Bibliografia

[1] R. A. Durresi. “RTP, RTCP, and RTSP - Internet Protocols for Real-Time Multimedia Communication..” Em: *The Industrial Information Technology Handbook. CRC Press LLC.* (2005).

[2] P. R. F. Alves. “Sincronização áudio em sistemas de alta fidelidade sem fios, Escola de Engenharia da Universidade do Minho.” Em: (2014).

[3] “Apple Airplay.” Em: (). url: https://developer.apple.com/airplay/.

[4] E. Bar. “Wi-Fi audio: Capabilities and challenges, Texas Instruments”. Em: (2015). url: https://www.ti.com/lit/wp/swry018/swry018.pdf.

[5] S. E. Kohler M. Handley. “RFC4340: Datagram Congestion Control Protocol (DCCP).” Em: (2006).

[6] M. F. Boronat J. Lloret. “Multimedia group and inter-stream synchronization techniques: A comparative study. Information Systems”. Em: (2008). doi: 10.1016/J.IS.2008.05.001.

[7] O. Gylytskyy. “Distribuição e Sincronização de Áudio Digital em Redes Sem Fios, Escola de Engenharia da Universidade do Minho.” Em: (2018).

[8] E. John. “IEEE 5888 Standard for a precision Clock Synchronization Protocol for Networked Measurement and Control Systems, a Tutorial. National Institute of Standards and Technology (NIST)”. Em: (2005).

[9] P. Krzyzanowski. “Clock Synchronization.” Em: (). url: https://people.cs.rutgers.edu/~pxk/417/notes/clocks.html.

[10] T. M. Sarvghadi. “Overview of Time Synchronization Protocols in Wireless Sensor Networks”. Em: *2nd Internacional Conference on Electronic Design, ICED 2014.* (2014). doi: 10.1109/ICED.2014.7015799.

[11] D. Mills. “RFC958: Network Time Protocol (NTP).” Em: (1985). doi: 10.17487/RFC0958.

[12] B. P. Avan F. Giraudet. “Importance of Binaural Hearing.” Em: (2015). doi: 0.1159/000380741.

[13] I. Pandzic. “Clock Synchronization and Monotonic Clocks.” Em: (). url: https://inelpandzic.com/articles/clock-synchronization-and-monotonic-clocks/.

[14] S. Ping. “Delay Measurement Time Synchronization For Wireless Sensor Networks.” Em: (2003).

[15] J. Postel. “RFC793: Transport Control Protocol (TCP).” Em: (). url: https://www.ietf.org/rfc/rfc793.txt.

[16] J. Postel. “RFC768: User Datagram Protocol (UDP).” Em: (1980). url: https://www.ietf.org/rfc/rfc768.txt.

[17] W. S. R. Y. Litovskya H. S. Colburn. “The precedence effect”. Em: *The Journal of the Acoustical Society of America* (1999). doi: 10.1121/1.427914.

[18] M. Roche. “Time Synchronization in Wireless Networks.” Em: (). url: https://www.cse.wustl.edu/~jain/cse574-06/ftp/time_sync/index.html.

[19] R. Sehgal. “Simulation Engine for Analysis and Comparison between Cristian’s and Berkeley clock synchronization algorithms”. Em: (2007).

[20] “Sonos | Wireless Speakers and Home Sound Systems.” Em: (). url: https://www.sonos.com/.

[21] R. Stewart. “RFC4960: Stream Control Transmission Protocol.” Em: (2007).

[22] “Wireless Spatial Sound | WiSA”. Em: (). url: https://www.wisatechnologies.com/.