

分类号	<u>TP39</u>	密级	<u>公开</u>
UDC	<u>004</u>	学位论文编号	<u>D-10617-30852-(2019)-02065</u>

## 重庆邮电大学硕士学位论文

中文题目	<u>WebRTC 会议系统的回声消除算法研究</u> <u>与设计实现</u>
英文题目	<u>Research of Echo Cancellation</u> <u>Algorithm and Design Implementation</u> <u>for WebRTC Conference System</u>
学 号	<u>S160231068</u>
姓 名	<u>魏自强</u>
学位类别	<u>工程硕士</u>
学科专业	<u>计算机技术</u>
指导教师	<u>龙昭华 教授</u>
完成日期	<u>2019 年 3 月 8 日</u>

## 独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含他人已经发表或撰写过的研究成果，也不包含为获得重庆邮电大学或其他单位的学位或证书而使用过的材料。与我一同工作的人员对本文研究做出的贡献均已在论文中作了明确的说明并致以谢意。

作者签名：魏自强

日期：2019年6月1日

## 学位论文版权使用授权书

本人完全了解重庆邮电大学有权保留、使用学位论文纸质版和电子版的规定，即学校有权向国家有关部门或机构送交论文，允许论文被查阅和借阅等。本人授权重庆邮电大学可以公布本学位论文的全部或部分内容，可编入有关数据库或信息系统进行检索、分析或评价，可以采用影印、缩印、扫描或拷贝等复制手段保存、汇编本学位论文。

(注：保密的学位论文在解密后适用本授权书。)

作者签名：魏自强

日期：2019年6月1日

导师签名：

日期：2019年6月1日

## 摘要

随着互联网的不断创新，人们希望能够使用更方便简单的方式，通过音视频会议来加强彼此间的交流，而基于浏览器实现的音视频会议系统，具有实现跨平台，免安装，方便接入的优势，正好满足这一需求。同时，由于各式各样的技术层出不穷，尤其是 Google 开源的 WebRTC 项目，使得基于浏览器的多人会议系统的实现得以实施，所以 WebRTC 技术逐渐得到越来越多的关注和研究。自从 WebRTC 技术开源以来，一方面由 W3C 和 IETF 负责其标准工作的制定和发展，另一方面由 WebRTC 联盟主导其技术层次的升级更新。近年来，WebRTC 技术一直处于上升趋势，如国外的 YouTube 应用，Chrome、Mozilla、Opera、Edge、Safari 等浏览器厂商，国内的运营商、设备商、互联网等公司，都在积极对 WebRTC 技术进行研究和优化。无论是根据 WebRTC 框架内的模块展开核心算法研究，还是基于 WebRTC 实现音视频应用，近年来针对 WebRTC 展开的研究也在逐渐增多。

本文围绕 WebRTC 进行研究分析，主要工作内容如下：首先，本文根据 WebRTC 框架，介绍了其音频引擎、视频引擎、传输模块三大部分，其中音频引擎包含编解码器、NetEQ、回声消除和噪声抑制等，而回声是会议系统一直以来无法彻底解决的问题。通过对回声消除算法现状的研究，对 WebRTC 中的自适应回声消除算法进行分析，重点对其核心部分自适应滤波器算法展开研究，分别讨论 LMS 算法、NLMS 算法和块 LMS 算法的原理。通过研究归纳其回声消除算法原理，在固定步长的 NLMS 算法的基础上，结合文献提出改进的变步长 NLMS 算法，并通过仿真实验进行验证。其次，本文通过对 WebRTC 发展现状进行研究，根据 WebRTC 所提供的音视频会议系统技术展开讨论，分析与 WebRTC 相关的信令、NAT 穿越技术。根据项目需求，设计基于 socket.io 的会话管理机制，并实现基于 WebRTC 的多人音视频会议系统，在此基础上，实现即时消息通信，并通过扩展插件完成屏幕共享的功能。通过对系统进行测试和分析，表明在多人音视频通信过程中系统功能正常，音视频效果良好，基本满足参会人员的交流和使用需求。

**关键词：**WebRTC，自适应滤波，视频会议

## Abstract

With the continuous innovation of the Internet, people hope to use a more convenient and simple way to enhance the communication between each other through audio-video conferencing. And the audio-video conferencing system based on the browser has the advantage of realizing cross-platform, installation free and convenient access, which just meets this demand. At the same time, due to the endless emergence of various technologies, especially the open source WebRTC project of Google, the implementation of the browser-based multi-person meeting system maybe will be achieved, so its WebRTC technology has gradually received more and more attention and research. Since WebRTC technology has been open source, on the one hand, W3C and IETF were responsible for the formulation and development of its standard work, and on the other hand, the WebRTC Alliance leaded the upgrade and update of its technical level. In recent years, WebRTC technology has been on the rise. For example, foreign YouTube applications, Chrome, Mozilla, Opera, Edge, Safari browsers, and domestic operators, equipment vendors, the Internet companies are actively researching and optimizing WebRTC technology. For the WebRTC project, whether within its framework the core algorithm research is carried out, or the audio-video application is realized based on WebRTC, the research on WebRTC has been gradually increasing in recent years.

The thesis focuses on the research and analysis of WebRTC technology, and the main work contents are as follows: firstly, this thesis introduces the core parts according to the WebRTC framework: audio engine, video engines, transmission module. The audio engine includes codec, NetEQ, echo cancellation and noise suppression, and echo is a problem that the conference system has not been able to solve completely. Through the study of the current situation of echo cancellation, focusing on the analysis of adaptive echo cancellation algorithm in WebRTC, this thesis studied the core part adaptive filter algorithm, and discussed the principles of LMS algorithm, NLMS algorithm and block LMS algorithm respectively. By studied and summarized the principle of its echo cancellation algorithm, on the basis of the fixed step size NLMS algorithm and combining with the literature, an improved variable step size NLMS algorithm was proposed, and verified by simulation experiment. Secondly, through the research on the development status of WebRTC, this thesis studied the audio and video conferencing system technology provided by WebRTC,

and analyzed WebRTC related signaling, NAT through technology. According to project requirements, this thesis realized the WebRTC multi-person audio and video conference through the session management mechanism based on socket.io, and on the basis of this, realized instant message communication, and completed the screen sharing function through the extension plug-in. And then test and analyze according to the system shows that in the process of multi-person audio and video communication, the system runs normally, the audio and video quality is good, the system basically meets the communication and usage needs of participants.

**Keywords:** WebRTC, adaptive filter, video conference

## 目录

第 1 章 引言 .....	1
1.1 研究背景及意义 .....	1
1.2 研究现状 .....	2
1.2.1 WebRTC 技术研究现状 .....	2
1.2.2 回声消除研究现状 .....	4
1.3 论文主要工作和组织结构 .....	5
1.3.1 论文的主要工作 .....	5
1.3.2 论文的组织结构 .....	5
第 2 章 WebRTC 技术的研究 .....	7
2.1 WebRTC 框架及核心技术 .....	7
2.1.1 整体框架 .....	7
2.1.2 浏览器端核心接口 .....	9
2.1.3 协议分析 .....	11
2.2 WebRTC 会议系统相关技术 .....	12
2.2.1 信令技术 .....	12
2.2.2 NAT 穿越技术 .....	13
2.3 本章小结 .....	14
第 3 章 回声消除与自适应滤波器原理 .....	15
3.1 回声产生与回声消除原理 .....	15
3.2 自适应滤波器原理 .....	16
3.2.1 最速下降算法 .....	18
3.2.2 最小均方自适应滤波算法 .....	18
3.2.3 归一化最小均方自适应滤波算法 .....	20
3.2.4 块自适应滤波算法 .....	21

3.3 本章小结 .....	23
第 4 章 WebRTC 自适应回声消除算法分析及改进 .....	24
4.1 自适应回声消除算法原理与分析 .....	24
4.1.1 延迟估计方法 .....	25
4.1.2 线性自适应滤波器 .....	26
4.1.3 基于互相干性的非线性处理 .....	29
4.2 改进的变步长 NLMS 算法 .....	31
4.3 实验结果与分析 .....	32
4.4 本章小结 .....	36
第 5 章 基于 WebRTC 的会议系统的设计与实现 .....	37
5.1 系统简介与需求分析 .....	37
5.1.1 功能需求分析 .....	37
5.1.2 非功能需求分析 .....	38
5.2 系统整体架构设计 .....	38
5.2.1 整体模块设计 .....	39
5.2.2 会话管理信令消息设计 .....	40
5.2.3 客户端模块设计 .....	43
5.2.4 服务端架构设计 .....	44
5.3 系统关键技术实现 .....	45
5.3.1 会话管理机制 .....	46
5.3.2 WebRTC 音视频交互 .....	49
5.3.3 共享屏幕功能 .....	52
5.3.4 基于会议房间的即时消息 .....	55
5.4 系统测试与分析 .....	57
5.4.1 系统基本功能测试 .....	58
5.4.2 系统性能分析 .....	62
5.5 本章小结 .....	64

---

第 6 章 总结与展望 .....	66
6.1 总结 .....	66
6.2 展望 .....	67
参考文献 .....	68
致谢 .....	73
攻读硕士学位期间从事的科研工作及取得的成果 .....	74



## 第 1 章 引言

### 1.1 研究背景及意义

随着互联网的普及以及 HTML5 的发展,各种新业务、新技术和新标准不断涌现,Web 时代的到来已成为一种趋势,WebApp 已逐渐成为互联网时代的主要业务形态<sup>[1]</sup>。在 Web 时代,人们对 Web 实时多人通信提出了新的要求,能够在网页端实现音视频会议系统,方便用户沟通。另外,新的标签<video>、<audio>的定义,以及 WebRTC 的发展,为音视频在 Web 页面直接播放提供了支持,同时为会议系统在浏览器平台上的实现奠定了基础<sup>[2,3]</sup>。

根据发展历程,当前会议系统可分为基于设备的硬件会议系统、基于桌面应用的软件会议系统和基于浏览器的网页会议系统<sup>[4]</sup>。早期的会议系统,由于计算机性能不足,网络带宽资源匮乏,主要是依赖于嵌入式系统实现的硬件会议系统,其效果虽然较为稳定,但成本高,系统功能不易扩展。随着计算机和网络的发展,基于 PC 桌面的软件会议系统逐渐占据市场,利用计算机所具有的较高性能的 CPU 和稳定的网络带宽,软件会议系统在安装之后,操作方便,但占用资源较大,而且使用 C/S 架构,在此架构中桌面应用实现多媒体通信较为复杂,开发门槛高,跨平台实现也较困难,另外,各个厂商借助私有技术抢占市场,以至于开发成本过大,不利于软件维护和升级。随着互联网的普及,网页版会议系统应运而生,结合了主流的 HTML5 和 B/S 架构技术,用户可以不用安装软件,轻松打开浏览器便可进行在线会议,但如果使用传统的流媒体技术在浏览器之间实现音视频通信,则需要依赖私有技术,以 Adobe Flash 为例,这样的私有技术需要借助插件来在网页上播放音视频。在实际情况下,对于绝大多数用户来说,安装插件也是较为繁琐的过程。因此,在网页上实现跨平台,免安装,方便接入的会议系统显得尤为重要<sup>[5]</sup>,而 WebRTC 技术则满足该需求,也是 Web 通信未来发展的主要方向<sup>[6]</sup>。

实时多媒体业务通常需要先通过相应的硬件设备,分别采集到音视频数据,然后对数据进行处理,再经过网络传输给接收方,接收方收到数据后再同步播放,这个过程需要保证音频和视频的质量,而传统的会议系统发展至今,回声影响通话质量是一个长期困扰开发人员和用户的问题<sup>[7]</sup>,所以回声消除是保证音频质量必不可

少的一部分<sup>[8]</sup>，而回声消除也在会议系统领域有广泛的应用。随着 WebRTC 的发展和演变，基于现有的 WebRTC 自适应回声消除算法，进行优化和改进也是当前的研究热点<sup>[9]</sup>。

WebRTC 不仅可以作为一项开源技术框架，也可以作为一项 Web 开发标准，并且在对其理论研究和实际应用上，都是科研热点。WebRTC 的自适应回声消除算法集成了当前较新的理论和算法，来保证其音频的质量，是值得深入研究的。同时，随着 WebRTC 1.0 标准的制定以及各大浏览器厂商的支持，为实现基于 WebRTC 的会议系统奠定了基础。基于此，本文将着重研究 WebRTC 的自适应回声消除算法，根据 NLMS 算法进一步提出改进，同时设计与实现基于 WebRTC 的无插件、免安装的会议系统。

## 1.2 研究现状

### 1.2.1 WebRTC 技术研究现状

随着互联网的发展，网页实时通信便利的体验，已经培养了一大批目标用户，让用户认识到在浏览器端就可以进行实时通信，但必须安装浏览器的插件或扩展程序，而 WebRTC 的出现，解决了这一繁琐的问题。在 2010 年，互联网巨头谷歌收购了国际 VoIP 软件开发公司 Global IP Sound，并于 2011 年 Google 将其 GIPS 引擎开源并更名为 WebRTC，在此之前国内 QQ 厂商就是使用 GIPS 引擎提供的音频服务。随后 WebRTC 技术的发展状况，可以从两个方面来看<sup>[3, 10]</sup>：标准制定和浏览器支持。

在国际标准方面，WebRTC 标准是由 W3C 和 IETF 联合制定的。随着 WebRTC 的发展，2014 年 WebRTC 浏览器 API 标准由 W3C 宣布制定，2016 年 W3C 发布 WebRTC 的标准草案，2017 年初，W3C 公布了最新的 WebRTC 1.0 候选推荐标准，并持续对其更新，同时 W3C 的 HTML5 标准制定工作组也决定在 HTML5 标准的制定中与 WebRTC 共存，实现无缝沟通。在浏览器支持方面，虽然 WebRTC 是由谷歌、火狐和欧朋等浏览器厂商主导的开源项目，起初由于标准尚未成熟，各厂商在浏览器端的 JavaScript 实现各有不同。随着 WebRTC 1.0 标准的公布，各厂商也在相继更新、完善和统一在浏览器端的 JavaScript 功能接口。另外，近年来软件巨头微软

曾提出并设计出一个非标准的 ORTC 协议，最初 ORTC 是想和 WebRTC 抢占开发市场，但实际上 ORTC 的发展并不顺利，经过双方沟通，WebRTC 1.0 标准决定对其进行兼容，而微软的新浏览器 Edge 也同样开始支持 WebRTC。同时，随着 WebRTC 的发展趋势不断上升，2017 年苹果的 Safari 11 也开始支持 WebRTC。

在国内，浏览器厂商都不是很大，绝大多数厂商都是使用同样的开源代码，所以 WebRTC 的代码在各个厂商的浏览器上都是可以运行的。同时，国内许多互联网企业也相继投入到基于 WebRTC 的 PaaS(Platform as a Service)和 SaaS(Software as a Service)研发，如国内互联网巨头腾讯的 QQ 产品起初也是购买的 GIPS 的音视频服务，自 WebRTC 开源之后，转为自研路线，在此基础上形成自研的高效多媒体引擎。除此之外，国内的设备厂商和运营商也相继开始研究和应用 WebRTC 技术。运营商以电信为例，北京电信与北京邮电大学合作的天翼 RTC 平台项目，将 WebRTC 技术与现有的网络和通信进行融合，进一步丰富用户应用，从而发展更多的用户。设备商以华为为例，华为作为 IETF RTCWeb 工作组及 W3C 的首批成员之一，早已经开始服务 WebRTC 标准以及相关应用推广。基于 IMS(IP Multimedia Subsystem)业务，华为在业界第一个提出了基于 WebRTC 的 RCS(Rich Communication Suite)能力开发网关概念，并在相应的行业领域得到应用，此后基于 Web 的 RCS 能力开发概念逐渐成为业界共识。此外，WebRTC 技术与华为核心网 CaaS(Communications as a Service)能力开发解决方案具有相同的理念，以业务开发、互通操作兼容、开发使用简单为核心理念，该方案可将运营商 IMS 用户进一步扩展到 Web 领域。另外，2015 年国内举办了首届 WebRTC 大会，参会人员就 WebRTC 各方面进行讨论，如标准制定、技术应用，实际案例、功能演进等。至此，作为目前最热的新技术之一，WebRTC 技术已经成功引起国内外的广泛关注，也将成为 Web 实时通讯未来发展的方向。

WebRTC 提供了实现会议系统的关键技术<sup>[11]</sup>，基于 WebRTC 可以实现一套音视频实时通讯的方案，但由于通讯信令未纳入标准，各个厂商使用的信令协议有所不同<sup>[12]</sup>，以及 WebRTC 自身是基于 P2P 连接，实现过程中有网络穿越等问题，在实际的开发中，也并不是绝大多数人所想象的那么简单<sup>[13]</sup>。

1.2.2 回声消除研究现状

如今，虽然人们在不同场景下享受着会议系统中实时通信带来的便利，但是会议系统中回声问题也在一直困扰着人们。回声消除是为了消除通信过程中，由于直接或者间接声音的部分声音信号形成的声音信号反馈。通俗来讲，就是消除从对方传回来的自己的讲话声音。现有的回声消除技术主要有话筒阵列技术、移频技术、语音控制开关技术、子带中心削波技术、梳妆滤波器和自适应滤波器技术。但是前五种技术因各自的弊端一直难以推广<sup>[14]</sup>。其中自适应滤波器算法易于自适应实时处理，经过不断的研究，自适应滤波器理论得到很大的发展，成为数字信号处理领域的研究热点<sup>[15]</sup>。

自适应滤波算法主要分为两类算法：最小均方(Least Mean Square, LMS)自适应滤波算法和递归最小二乘(Recursive Least Square, RLS)自适应滤波算法<sup>[16]</sup>。这两类自适应滤波算法均是基于维纳滤波算法推导发展而来的，经过扩展和变形之后，逐渐形成 LMS 族和 RLS 族两大分支。

表 1.1 LMS 和 RLS 算法性能对比

	LMS	RLS
算法结构	简单	复杂
收敛速度	一般	较快
跟踪能力	较强	一般
数值鲁棒性	较强	一般
计算复杂性	低	高

如表 1.1 所示是 LMS 算法和 RLS 算法各项性能的对比，RLS 算法优势在于收敛速度较快，但计算复杂性较高，所需存储量极大，不利于实时实现，而 LMS 算法由于算法结构简单优良，计算复杂度低，鲁棒性好，有利于实时实现，因此更多的研究也集中在 LMS 算法上，在 LMS 算法的基础上，发展出归一化 LMS 算法、块 LMS 算法等。着眼于 WebRTC 的自适应回声消除算法，该算法在 LMS 算法的基础上，结合在频域上分段块的应用，使用归一化 LMS 算法<sup>[17]</sup>，实现了在运行过程中自适应处理回声消除。在理论研究中，各种改进的 LMS 算法也是基于归一化最小均方算法<sup>[18]</sup>，主要包括固定步长和变步长两大类，以变步长自适应算法为主。随着自适应理论的发展，对其分析和研究同样具有重要意义<sup>[19]</sup>。

## 1.3 论文主要工作和组织结构

### 1.3.1 论文的主要工作

本文主要结合会议系统，基于 WebRTC 开源框架，展开对 WebRTC 自适应回声消除算法和 WebRTC 会议系统设计与实现的研究。首先，根据当前主流的回声消除算法即自适应滤波器算法进行研究，对 WebRTC 自适应回声消除算法进行分析，在 NLMS 算法的基础上，提出变步长 NLMS 算法的改进，通过 MATLAB 仿真进行验证。其次，根据 WebRTC 在浏览器端的核心技术以及实现会议系统的相关技术进行分析，设计基于 socket.io 的会话管理机制，实现基于 WebRTC 的会议系统的音视频和即时消息通信，同时设计扩展插件实现屏幕共享的功能，通过实际应用测试多人参会效果。

### 1.3.2 论文的组织结构

在组织结构上，本文共分为六章，主要研究内容和章节编排如下：

第一章开篇介绍了本文的研究背景及意义，并叙述了 WebRTC 技术和回声消除算法的研究现状，以及本文的主要工作和组织结构。

第二章主要针对 WebRTC 整体框架进行分析，从而论述了实现 WebRTC 会议系统的核心技术和相关技术。为后续会议系统的设计与实现提供理论基础。

第三章在第一章回声消除研究现状的基础上，着重分析了会议系统通话过程中回声产生和回声消除的原理，研究了自适应滤波器的结构和工作原理，在最速下降算法的基础上，进一步详细地分析了 LMS、NLMS 和块 LMS 算法的原理。为对 WebRTC 回声消除算法的分析提供理论支持。

第四章在第三章的原理研究上，进一步对 WebRTC 的自适应回声消除算法进行了深度分析，是论文的主体部分，主要讲述了 WebRTC 在回声消除算法中的延迟估计方法、线性自适应滤波器、非线性处理三个方面。其中线性自适应滤波器是主要部分，以 NLMS 算法为核心。随后，本文基于 NLMS 算法提出改进，改善因使用固定步长所带来的问题，并通过实验仿真，验证算法在较小的稳态误差下，拥有较好的收敛速度和跟踪能力。

第五章是在第一章和第二章的基础上,结合项目需求,设计与实现基于 WebRTC 的会议系统,并对系统进行测试与分析,是论文的主体部分。本章设计基于 socket.io 的会话管理机制,进一步实现 WebRTC 音视频通信和即时消息通信,并根据实际需求,在本系统上实现了共享屏幕的功能,可以共享桌面、网页和应用窗口。最后,针对系统的模块功能和性能进行测试与分析。

第六章是总结与展望,通过对本文的研究工作进行总结,提出研究工作中的不足之处,为后续的研究工作做铺垫。

## 第 2 章 WebRTC 技术的研究

### 2.1 WebRTC 框架及核心技术

#### 2.1.1 整体框架

如图 2.1 所示，WebRTC 整体架构主要包括 Web 应用、Web API 和 Web 浏览器内部三层。Web 应用是 Web 开发者利用 Web API 实现无插件形式的实时音视频通信应用。Web API 是提供给 Web 开发者实现 Web 应用的 Javascript API，主要有 getUserMedia 接口、RTCPeerConnection 接口、RTCDataChannel 接口等<sup>[3]</sup>。浏览器层是 WebRTC 嵌入浏览器内核的部分，主要是为浏览器厂商提供的 WebRTC C++接口、会话管理、WebRTC 三大模块、底层音视频采集和网络输入输出等部分。

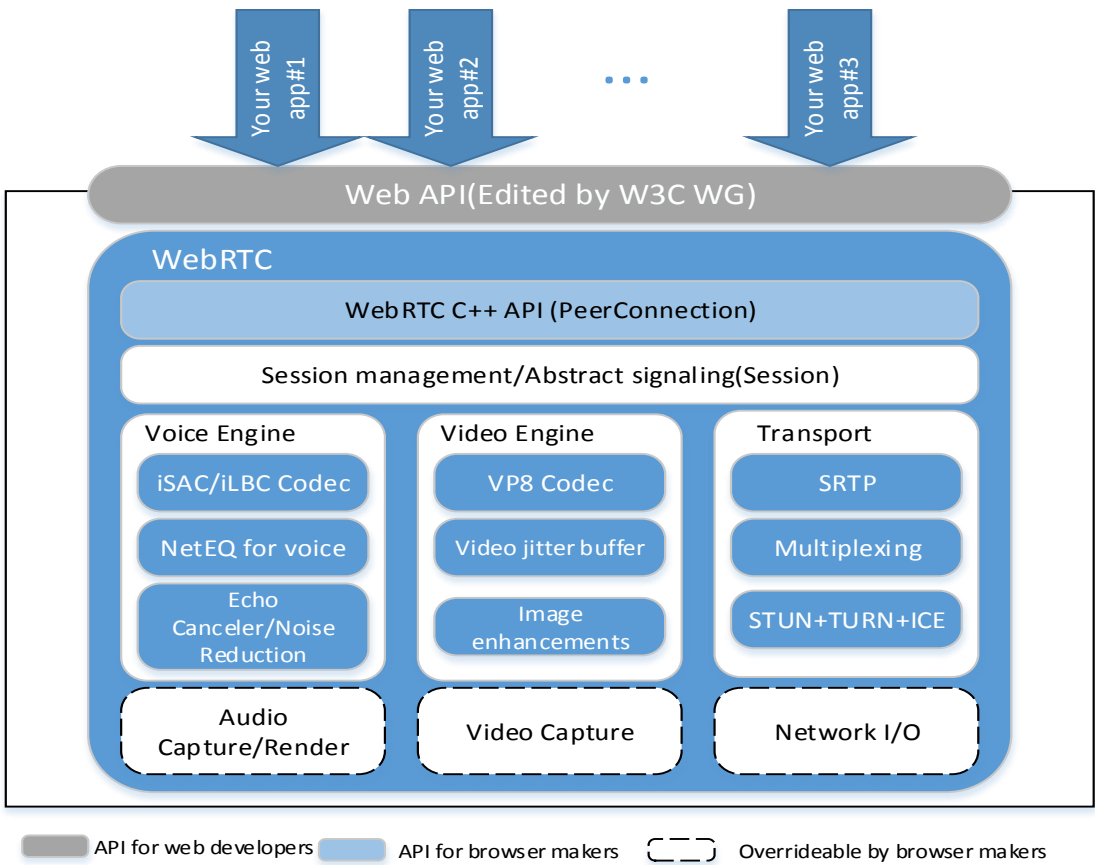


图 2.1 WebRTC 整体框架<sup>[1]</sup>

WebRTC 整体框架中,除了 WebRTC 的 Web API 之外,值得研究的地方是 WebRTC 的三大模块:音频引擎、视频引擎和传输模块。

### 1. 音频引擎

音频引擎是一系列音频多媒体处理的程序<sup>[20]</sup>。从功能上讲,该音频引擎主要依赖五大功能模块:音频数据采集和播放、音频数据预处理、音频数据编解码、音频数据缓冲区 NetEQ<sup>[21]</sup>、音频数据混音。其中比较重要的是音频数据预处理,音频数据缓冲区 NetEQ 和音频数据编解码。

音频预处理包括回声消除(Echo Cancellation, EC)、噪声消除(Noise Cancellation, NS)、自动增益控制(Automatic Gain Control, AGC)、语音活动检测(Voice Activity Detection, VAD)、舒适噪声产生(Comfort Noise Generator, CNG)。声学回声消除从音频通道中消除声学回声。噪声消除用来消除背景噪声或环境噪声。自动增益控制是当经过回声消除和噪声消除之后音频能量受损时,通过自动增益,增大音频电平,达到自适应调节音量的作用。语音活动检测是检测是否有人的语音。舒适噪声生成指的是在通话过程中由于出现短暂静音,而产生的背景噪声,通常舒适噪声生成是语音活动检测的一部分,二者结合使用,可用来维持一个能感受到的可接受的服务品质,同时尽可能降低传输成本和宽带使用<sup>[22]</sup>。其中 AEC 算法也是本文主要的研究内容。

音频数据缓冲区 NetEQ 是目前性能最为优良的抖动消除技术,基本上包含四大部分:自适应缓冲器、语音解码器、抖动控制和丢包隐藏、播放<sup>[21]</sup>。核心算法是自适应抖动控制算法和语音包丢失隐藏算法。该模块可以快速适应不断变化的网络状况,为音频引擎拥有较小的延迟和很好的语音质量提供保障。

音频数据编解码模块是 WebRTC 的音频编解码器,主要包含 iSAC 和 iLBC 两个编解码器。其中 iSAC 是默认的编解码器。

### 2. 视频引擎

视频引擎是包括一系列视频处理的程序,主要包括三大功能模块:视频编解码器、视频抖动缓冲器和图像质量增强。WebRTC 的视频编解码器默认使用的是 VP8 编解码器,可自行集成兼容 H264 编解码器,VP8 在低码率的状况下,具有较好的低延迟性,能够很好地作用于实时通信的应用场景。视频抖动缓冲器,类似于音频引擎的 NetEQ 模块,主要分为抖动和缓冲两部分,抖动部分主要是根据当前帧的大



小和延时评估出抖动延迟,再结合其他延迟,如编码延迟、渲染延迟和音视频同步延迟等,得到最后的渲染时间,来平稳地控制视频帧的渲染。缓冲部分主要是对丢包、乱序、延迟到达等异常情况进行处理。该模块可以处理由于视频抖动和视频信息包丢失所造成的不良效果。图像质量增强模块是对摄像设备采集到的图像进行处理,提高视频效果的质量,一般包括明暗度检测、颜色增强和降噪处理等<sup>[23, 24]</sup>。

### 3. 传输模块

传输模块包含安全实时传输协议(Secure Real-time Transport Protocol, SRTP)、多路复用(multiplexing)、NAT 三个部分<sup>[25]</sup>。其中 WebRTC 引入数据包传输层安全性协议(Datagram Transport Layer Security, DTLS)用于传输 SRTP 数据包时的安全密钥交换和加密传输。SRTP 用于音视频数据的传输,涉及实时传输协议(Real-time Transport Protocol, RTP)和 RTP 控制协议(RTP Control Protocol, RTCP),RTP 负责媒体数据的打包、解包和传输,WebRTC 将媒体数据封装在 RTP 数据包中,使用 RTCP 来报告媒体流的性能。通过 RTCP 发送的报告,有助于不同媒体流的播放同步以及调整发送端媒体编码速率等参数。WebRTC 中多路复用的作用是在同一条 RTP 会话中,可以复用多条媒体流,前提是接收方必须支持和接受。NAT 主要包括 STUN、TURN 和 ICE,是用于 NAT 网络和防火墙穿越的,可以通过连接 STUN/TURN 服务器,生成当前用户的网络信息,但 WebRTC 内部未集成这类服务器,在外网环境下,需要开发者自行解决或部署 STUN/TURN 服务器。

#### 2.1.2 浏览器端核心接口

WebRTC 整体架构中有一层是 Web API,即浏览器端接口,Web 开发者利用 WebRTC 标准中定义的接口可以实现音视频实时通信应用<sup>[26]</sup>。WebRTC 在浏览器端有三个重要的 Javascript 接口:getUserMedia、RTCPeerConnection、RTCDataChannel<sup>[27]</sup>。其他的接口对象可以通过 WebRTC 开源网站或 MDN Web docs 的 WebRTC 文档查阅。

getUserMedia 接口是为用户提供直接访问终端媒体设备的接口,使用 getUserMedia 可以在不依赖任何插件的情况下,直接访问硬件终端的媒体设备。该接口在前期标准未制定时,在不同浏览器厂商的命名也不同,一般以浏览器内核的名称为前缀,如在谷歌浏览器中是 webkitgetUserMedia,在火狐浏览器中是

mozgetUserMedia。近几年随着 WebRTC 标准的完善和更新，最初的 navigator.getUserMedia 接口逐渐被抛弃，由 navigator.mediaDevices.getUserMedia 接口代替。使用前提是需要定义一个 containers 对象，用于指定要请求的媒体类型以及每种类型的任何要求，一般包含 audio 和 video 两个对象。该接口被调用后，会返回一个 mediaStream 对象，表示媒体内容的数据流，调用 getUserMedia 接口之后，将媒体流通过 video 标签在网页显示。值得关注的是，该接口无法在非安全的浏览器页面内调用，必须配置为 HTTPS。

RTCPeerConnection 接口的作用是创建 WebRTC 连接，在两个浏览器之间建立端到端的数据传输链路，即 P2P 连接。同样，由于不同浏览器厂商的具体实现，该接口的命名也随浏览器的不同而有所差异，可利用 adapter.js 来解决 WebRTC 在浏览器中的兼容问题。创建 RTCPeerConnection 对象，需要 STUN/TURN 服务器地址信息，而在传输链路建立之前，需要获取并分享两端的网络信息，以及需要获取并分享本地和远程描述即 SDP(Session Description Protocol)。然后，将通过 getUserMedia 获取到的 mediaStream 对象，传入本地创建的 RTCPeerConnection 对象，根据两端之间共享的网络信息和会话描述信息，则可以建立端到端的 WebRTC 媒体通信。

RTCDataChannel 接口的作用是在两端之间建立一个双向的数据通道。该接口是建立在 RTCPeerConnection 对象之上来发送和接收自定义的数据。该接口有点类似 WebSocket，由于 RTCDataChannel 接口通信是在浏览器之间直接连接，而 WebSocket 接口需要服务器中转，所以比 WebSocket 更快。由于所有 WebRTC 组件都需要使用加密，因此使用数据报传输层安全性(Datagram Transport Layer Security，DTLS)自动保护在 RTCDataChannel 上传输的任何数据，同时使用流控制传输协议(Stream Control Transport Protocol，SCTP)控制数据发送和接收。使用时，通过 RTCPeerConnection 对象创建 datachannel 对象来通知远端用户 RTCDataChannel 被加载到连接上，并使用 RTCDataChannel 的 onopen 属性在数据通道建立或重新建立时触发 open 事件，使用 onmessage 属性触发发送消息事件时所调用的函数，使用 onclose 属性触发收到 close 事件时所调用的函数，表示数据通道已关闭。RTCDataChannel 的属性都是事件处理器(EventHandler)，但由于标准的制定，有些浏览器厂商并未实现该功能，所以在使用时需要注意<sup>[28]</sup>。

2.1.3 协议分析

在介绍 WebRTC 浏览器端核心的 Javascript 接口时，涉及到了接口实现所使用的协议，如 NAT、Media、Signal 等方面的协议。

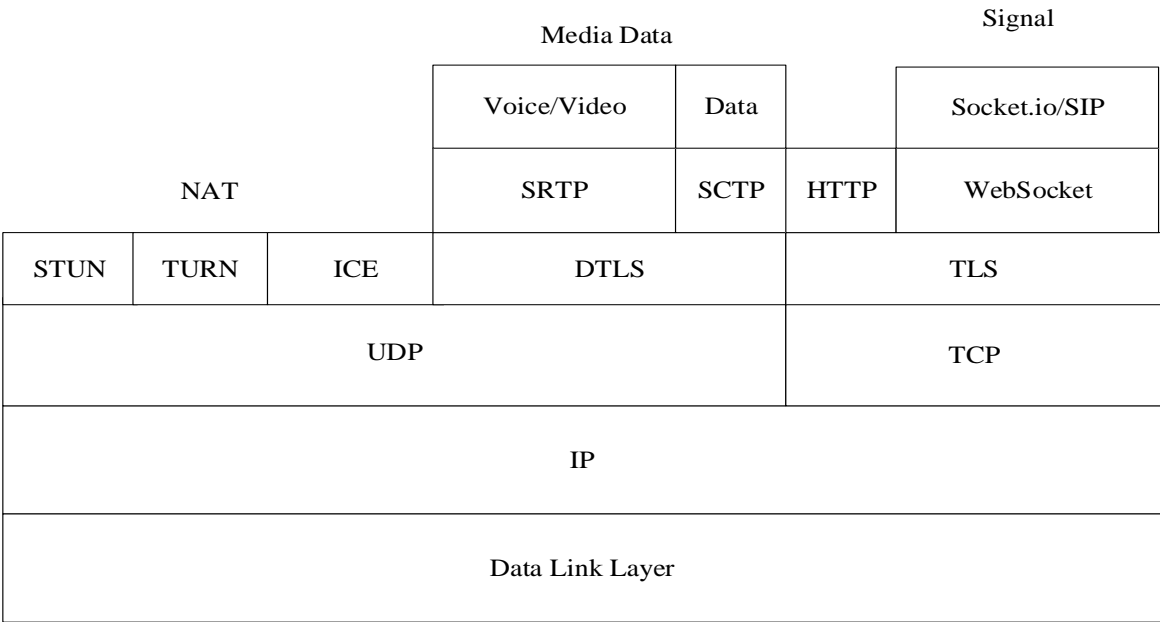


图 2.2 WebRTC 协议栈

如图 2.2 所示的是 WebRTC 的协议栈，即 WebRTC 通信过程中所涉及的各类协议。WebRTC 在建立 P2P 连接之前，需要通过信令来完成会话管理，信令的传输需要信令协议以及网络来完成，例如图 2.2 中基于 WebSocket 的 socket.io 和 SIP over WebSocket，信令的作用是协调和管理会话，通过交换会话控制信息、网络配置信息、媒体适配信息等信令消息来达到会话管理的目的。信令的传输是基于 TCP 和 TLS 协议执行的<sup>[29]</sup>。

NAT 部分的作用是使用 STUN、TURN 和 ICE 协议，获取当前用户的网络配置信息，从而穿越公网，为用户实现与对等端建立连接。STUN 协议作用是确定用户是否在 NAT 之后，并发现自己的公网映射地址和端口。TURN 协议是 STUN 协议的扩展，在打洞失败时，提供中继服务与对端进行报文传输。ICE 协议其实是一种综合性的 NAT 穿越技术框架，集成了 STUN 和 TURN，该技术可以利用各种 NAT 穿越方式打通远程的防火墙。STUN 协议的运行可基于 TCP 也可基于 UDP，TURN 协议是基于 UDP 传输。

WebRTC 核心部分音频、视频主要是基于 SRTP 协议的，即 RTP over DTLS。另外，RTP/RTCP 协议是流媒体通信的基础，RTP 主要负责定义流媒体数据在传输时的数据包格式，而 RTCP 协议主要负责为可靠传输、流量控制和拥塞控制等服务质量提供保证，DTLS 用来确保两端之间的数据传输的安全。SCTP 是一种可靠的通用传输协议，RTCDataChannel 接口使用 SCTP 来传输的数据，同样也是利用 DTLS 来确保其安全性。

## 2.2 WebRTC 会议系统相关技术

### 2.2.1 信令技术

WebRTC 音视频通信的实现是基于 P2P 连接的，而这一过程是需要通过信令来协商两端的通信。针对 WebRTC 通信而言，信令消息包括三类：交换会话控制信息、网络配置信息、媒体适配信息。信令消息的传递需要信令协议或信令技术来实现，而 WebRTC 未定义信令层的标准，所以有多种实现信令传递的技术，常见的信令技术包括 SIP over WebSocket、基于 WebSocket 的 socket.io、XHR(XMLHttpRequest)等 [30]。

XHR 是 Web 信令的经典方法，通过 HTTP 协议实现，是基于长轮询的服务器推模型，即浏览器端主动发起请求，与服务器端连接成功后保持一条连接，需要等待服务器端推送数据。其优点在于有较好实时性，性能好，但缺点也很明显，长时间占用连接，对服务器端的负载有所增加，丧失了无状态高并发的特点。

SIP(Session Initiation Protocol)是由 IETF 制订的多媒体通信协议。该协议与 HTTP 类似，是基于文本的协议，也是 VoIP 应用中常用协议。SIP 协议本身是一个成熟的经过考验的网络通话协议，但实现过于复杂，与 WebRTC 结合，需要搭建在 WebSocket 的基础上，如果没有跨平台的需求，一般无需使用此类协议，类似的协议还有 XMPP/Jingle。

socket.io 是基于 WebSocket 的框架，可以在 WebSocket 作为信令通道的基础上，看做信令协议。而 WebSocket 协议是 HTML5 中新提出的通信协议，功能强大，是基于 Web 的 C/S 通信协议，实现浏览器与服务器之间全双工通信<sup>[31]</sup>，针对实时通信场景，其性能优于传统的轮询方式。WebSocket 的传输层是 TCP，该协议也是双向

通信协议，即在连接建立之后，浏览器端和服务端都可主动发送或接收数据，像 Socket 一样，极大地方便了实时应用的 Web 开发<sup>[32]</sup>。

### 2.2.2 NAT 穿越技术

众所周知，早在 2011 年互联网数字分配机构 (The Internet Assigned Numbers Authority, IANA)宣布，IPv4 的地址空间已经被分配完毕。意味着 IPv4 地址基本消耗殆尽，而网络地址转换(Network Address Translation, NAT)技术的出现，延缓了 IPv4 地址不足的紧急情况<sup>[33]</sup>。网络地址转换，就是通过将内部网络的私网 IP 地址替换为出口的公网 IP 地址，通常 NAT 部署在一个组织或机构的网络出口位置。

根据 NAT 端口映射方式可分为非对称/锥形 NAT 和对称 NAT，非对称 NAT 可细分为全锥形 NAT、限制锥形 NAT、端口限制锥形 NAT。事实上，这些术语是描述一种工作方式，而不是一个设备。现实中很多 NAT 设备是将各种转换方式配置在一起进行运作的。

在互联网环境下，绝大多数的 IP 节点都会在 NAT 之后，当处于私网的两台终端要直接进行通信时，即所谓的 P2P 通信，则需要 NAT 穿越(NAT Traversal)技术来解决 IP 端到端在 NAT 环境下的会话问题。对于 WebRTC 来说，常见的 NAT 穿越技术是基于 UDP 的技术，如 2.1.3 协议分析所描述的 STUN、TURN、ICE 协议<sup>[34]</sup>。

STUN 协议，根据 RFC3489 中的定义，全称为 Simple Traversal of User Datagram Protocol Through Network Address Translators，即简单的用 UDP 穿透 NAT，其作用是使处于私网中的终端应用程序检测到它们与公网之间存在的 NAT 和防火墙的类型，让终端程序获取到 NAT 分配的公网 IP 地址和端口号对。另外，根据 RFC5389，STUN 的全称修改为 Session Traversal Utilities for NAT，即 NAT 环境下的会话传输工具，STUN 自身不再是一个完整的 NAT 传输解决方案，而是成为一种处理 NAT 传输的工具。另外，STUN 是一种 Client/Server 的协议，支持两种传输类型，一种是请求/响应(request/respond)类型，由客户端给服务器发送请求，等待服务器返回响应，这是应用程序中常用到的类型。另一种是指示类型(indication transaction)，该类型可以由服务器端或客户端中的任意一端主动发送指示，而另一端不会产生响应。

TURN 协议全称是 Traversal Using Relays around NAT，即使用中继穿透 NAT，是 STUN 协议的一个扩展，主要添加了中继功能。针对 NAT 类型，STUN 协议可以

处理大多数的锥形 NAT，对于对称 NAT，为了保证通信能够建立，可以使用中继方法，向对等端转发来往的数据。TURN 协议允许主机控制中继的操作，使用中继与对等端交换数据。通常，只有在 STUN 协议穿越 NAT 失败的情况下，才会使用 TURN 的方式去穿越 NAT。

ICE 全称是 Interactive Connectivity Establishment，即互动式连接建立，跟 STUN/TURN 不一样，本质上不是一种协议，而是一种整合了 STUN 和 TURN 框架。在 ICE 协议中如果使用 TURN，则中继地址会作为一个候选地址，由 ICE 在多个候选中进行综合评估，选取最合适的候选地址。一般来说，中继的优先级是最低的。STUN 能解决大多数的锥形 NAT 穿越，TURN 作为 STUN 的扩展，针对对称 NAT 实现穿越，而 ICE 框架综合了 STUN 和 TURN，支持各种 NAT 穿越，统一了 NAT 穿越技术<sup>[26]</sup>。

## 2.3 本章小结

本章针对 WebRTC 整体框架进行研究，介绍了各个模块所包含的功能，并对所涉及到的协议进行简要分析。重点研究了 WebRTC 浏览器端的核心接口和 WebRTC 相关技术，如信令机制、NAT 穿越等，为后续本文第五章的系统设计与实现提供理论基础。

## 第3章 回声消除与自适应滤波器原理

### 3.1 回声产生与回声消除原理

#### 1. 回声产生

会议系统通话过程中，回声是一个影响通话质量的重要因素<sup>[36]</sup>。回声通常分为电路回声和声学回声，电路回声是由二四线转换阻抗不匹配导致能量泄露等原因产生的，通过硬件电路的合理设计可以消除，本文不做研究。声学回声是近端讲话者通过麦克风发送给其他远端的语音，在远端扬声器播放之后，被远端的麦克风直接或者间接捕获到，又重新发送回自己的听筒的现象。需要注意的是声学回声易受环境的影响，可能通过不同回声信道产生多路回声，间接被麦克风捕获，这种间接反射回声，由于产生回声的信道不同，延迟也就不同，所以消除比较困难。

回声会对会议系统产生严重的干扰，造成通话过程不稳定，通过质量不佳，必须想办法进行消除。

#### 2. 回声消除原理

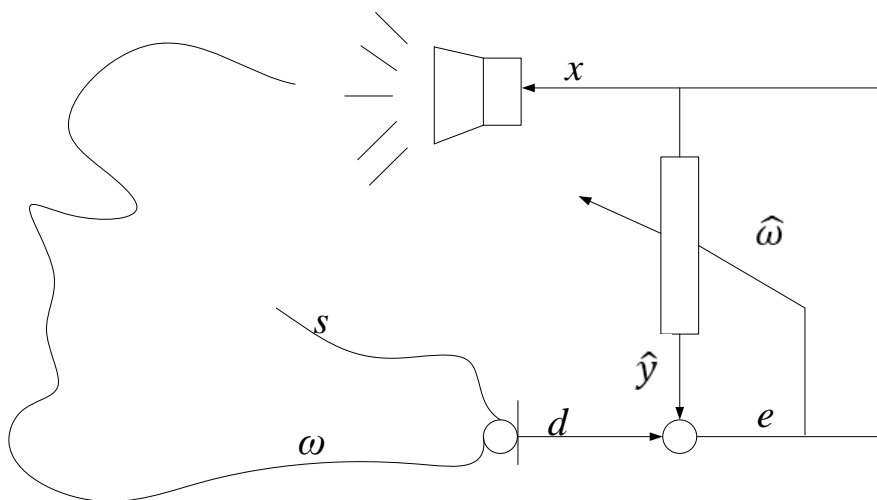


图 3.1 回声消除原理图

现有的回声消除技术主要有移频技术、子带中心削波技术、语音控制开关技术、梳妆滤波器、话筒阵列技术和自适应滤波器技术。前五种技术因各自的弊端一

直难以推广<sup>[14]</sup>。自适应滤波器技术是目前国内外的主流技术，也是自适应信号处理比较活跃的研究领域之一。本文主要研究自适应滤波器技术。

图 3.1 所示的是利用自适应滤波器进行回声消除的原理图。式(3.1)-式(3.4)是图 3.1 中相关变量的表达式：

$$echo = x * \omega \quad (3.1)$$

$$d = s + echo \quad (3.2)$$

$$\hat{y} = x * \hat{\omega} \quad (3.3)$$

$$e = d - \hat{y} \quad (3.4)$$

如图 3.1 所示，回声消除基本原理是使用自适应滤波器对未知的回声信道 $\omega$ 进行参数辨识，根据远端参考信号 $x$ 和产生的多路回声的相关性，建立远端信号模型，得到模拟回声路径 $\hat{\omega}$ ，通过滤波过程获取回声估计值 $\hat{y}$ ，以及通过自适应算法调整滤波器权值更新，使其冲击响应和真实的回声路径 $\omega$ 相逼近，然后在麦克风接收到的信号 $d$ 中去除估计值，得到误差信号 $e$ ，即可实现回声消除功能。

### 3.2 自适应滤波器原理

本节将主要研究基于 LMS 算法的自适应滤波器原理，介绍 LMS 族的自适应滤波算法，为后续的 WebRTC 自适应回声消除算法的分析与 NLMS 算法的改进，提供重要理论支撑。

滤波器主要包括线性滤波器和非线性滤波器两种。如果滤波器输出端的量是它输入量的线性函数，则认定该滤波器是线性的。如果不是，则是非线性的。在使用统计方法解线性滤波问题时，会设定已知有用信号和含噪数据的某些统计参数，将含噪信号作为输入量，并在某种统计准则下，使得噪声对滤波器的影响最小。实现滤波器最优的解决方案是使误差信号量的均方值最小，一般对于平稳输入，该滤波器就是维纳滤波器。但是维纳滤波器一方面不适合用于信号的非平稳性，另一方面它设计要求输入量的统计特性与滤波器所设计的某一先验知识相匹配，该滤波器方能达到最优，得到最优解，也称维纳解。而实际中，当信息完全未知时，就不可能将滤波器设计为维纳滤波器<sup>[36]</sup>，也就是说在实际中无法得到所需的统计参数，需要滤波器能够自我调节，即对输入信号进行过滤的同时，使用自适应算法不断改进滤



波器的性能,使其输出量和期望量之间的误差越来越小,最终收敛于最优,这就是本文所要研究的自适应滤波器<sup>[37]</sup>。

自适应滤波器依靠递归算法,使得该滤波器在有关信号特征无法完整得到的情况下,完成滤波运算。自适应滤波器又可以称为自适应滤波器系统,而该系统主要包括两部分,即自适应滤波架构和自适应滤波算法,

自适应滤波架构可以分为有限脉冲响应(Finite Impulse Response, FIR)滤波器和无限脉冲响应(Infinite Impulse Response, IIR)滤波器<sup>[38]</sup>。本文所涉及到的自适应滤波器均属于有限脉冲响应滤波器。

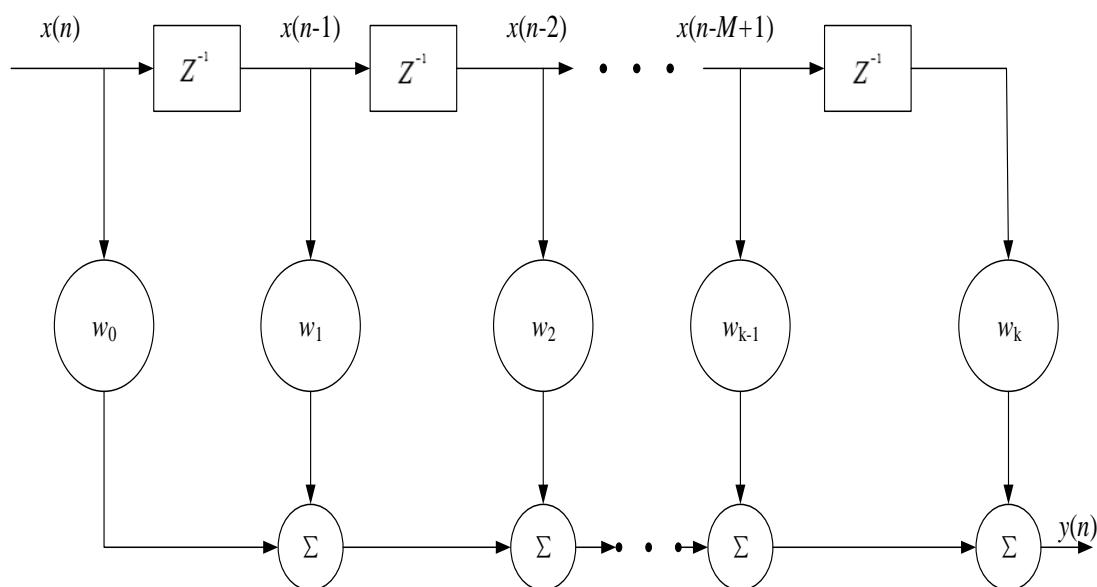


图 3.2 滤波器结构图

如图 3.2 是有限脉冲横向滤波器的结构图,  $z^{-1}$  是延迟单元, 延迟单元的个数是滤波器的阶数,  $w_k$  是抽头权值, 其中  $k=0, 1, \dots, M$ ,  $x(n)$  是抽头输入, 用滤波器的系数或称为抽头权值与对应输入相乘, 通过加法器对各个乘法器输出求和, 产出总的滤波器输出  $y(n)$ 。

自适应滤波器算法主要有最小均方(Least Mean Square, LMS)自适应滤波器算法、归一化最小均方(Normalized Least Mean Square, NLMS)自适应滤波器算法、最小二乘法算法(Recursive Least Square, RLS)等几类。其中 RLS 算法属于最小二乘算法类别, 虽然性能较优越, 但由于其算法复杂性, 无法得到较广泛的应用。因此, 本文主要研究属于随机梯度类别的 LMS 和 NLMS 算法。

### 3.2.1 最速下降算法

最速下降算法，又称为梯度下降算法，在了解 LMS 算法和 NLMS 算法之前，需要先研究一下最速下降算法，因为 LMS 算法总是联合最速下降算法使用。本节将描述最速下降法，来理解基于梯度的自适应方法。

首先，定义一个代价函数  $J(\mathbf{w})$ ，它是某个未知向量  $\mathbf{w}$  的连续可微函数。 $J(\mathbf{w})$  可以将  $\mathbf{w}$  映射为实数。从某一个初始状态  $\mathbf{w}(0)$  开始，产生一系列的权向量  $\mathbf{w}(1), \mathbf{w}(2), \dots, \mathbf{w}(n)$ ，为了使代价函数在算法的每一次迭代中呈现下降趋势，数学表示为：

$$J(\mathbf{w}(n+1)) < J(\mathbf{w}(n)) \quad (3.5)$$

其中  $\mathbf{w}(n)$  是权向量的过去值， $\mathbf{w}(n+1)$  是其更新值。如果希望得到一个最优解  $\mathbf{w}_o$ ，则对任意  $\mathbf{w}$  满足如下条件：

$$J(\mathbf{w}_o) < J(\mathbf{w}) \quad (3.6)$$

利用最速下降法，沿着最速下降方向连续调整权向量  $\mathbf{w}$ ，梯度向量表示为：

$$\mathbf{g} = \nabla J(\mathbf{w}) = \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \quad (3.7)$$

因此，使用  $n$  表示迭代进程， $\mu$  表示步长参数，最速下降算法可以表示为：

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \frac{1}{2} \mu \mathbf{g}(n) \quad (3.8)$$

最终，算法会收敛到最优值  $\mathbf{w}_o$ ，也称为维纳解<sup>[36]</sup>。

### 3.2.2 最小均方自适应滤波算法

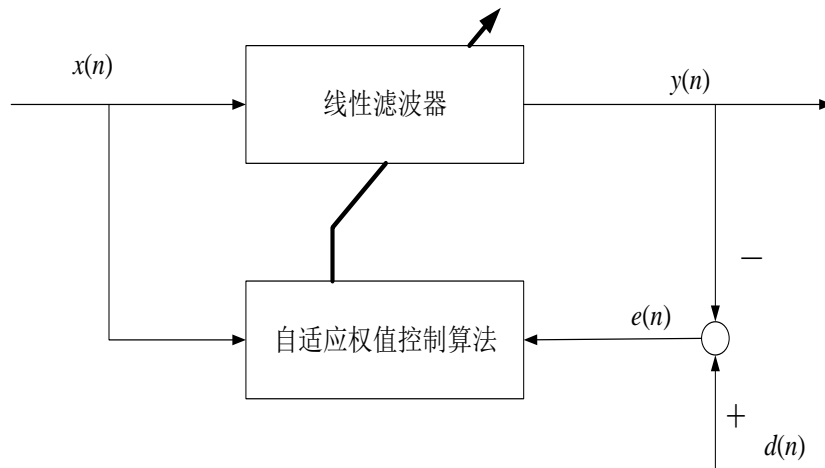


图 3.3 自适应滤波器图

最小均方自适应滤波(LMS)算法, 于 1960 年, 由 Widrow 和 Hoff 共同完成, 是基于维纳滤波原理, 在最速下降法的基础上发展而来<sup>[35]</sup>。LMS 算法属于线性自适应滤波算法<sup>[40]</sup>, 通常包括两个过程: 滤波过程和自适应过程。

图 3.3 所示的是最小均方自适应滤波器的两个过程, 一个是滤波过程, 用来计算滤波器输出对输入信号的响应, 并比较输出结果和期望响应来得出估计误差。另一个是自适应过程, 是根据估计误差自动调整滤波器的参数<sup>[37]</sup>。

由先验知识可知, 最速下降算法获取到的权值向量将收敛于最优解, 即维纳解, 是根据梯度向量的精确测量所得到的, 但实际上是无法实现的。当算法在未知情况下, 需要根据可用数据, 估计梯度向量, 可以表示如下:

$$\nabla J(n) = -\mathbf{P} + 2\mathbf{R}\mathbf{w}(n) \quad (3.9)$$

其中  $\mathbf{P}$  是抽头输入向量与期望响应之间的互相关向量,  $\mathbf{R}$  是抽头输入的相关矩阵, 分别定义为:

$$\mathbf{P}(n) = \mathbf{X}(n)d(n) \quad (3.10)$$

$$\mathbf{R}(n) = \mathbf{X}(n)\mathbf{X}^H(n) \quad (3.11)$$

结合梯度 $\nabla J(n)$ 和最速下降算法权值更新表达式, 得到新的一个递归关系式:

$$\mathbf{W}(n+1) = \mathbf{W}(n) + \mu\mathbf{X}(n)[d(n) - \mathbf{X}^H(n)\mathbf{W}(n)] \quad (3.12)$$

最后, 可以根据算法流程中所需要的滤波输出、估计误差和抽头权值更新, 用三个基本关系式表示, 如下:

$$y(n) = \mathbf{W}^H(n)\mathbf{X}(n) \quad (3.13)$$

$$e(n) = d(n) - y(n) \quad (3.14)$$

$$\mathbf{W}(n+1) = \mathbf{W}(n) + \mu\mathbf{X}(n)e(n) \quad (3.15)$$

式(3.13)是滤波器的滤波结果, 由维纳-霍夫方程所得, 式(3.14)是根据期望信号与输出信号对比, 得到的误差信号, 将用于式(3.15)的权值更新, 通过递归算法不断调整抽头权值,  $\mu$ 为步长因子, 为固定常数。

LMS 算法是通过最速下降法演变过来的, 在该算法中步长因子 $\mu$ 起到很重要的作用, 步长因子的选择, 直接关系到算法整体的收敛性能, 即步长数值越大其算法收敛速度越迅速, 而步长数值越小其算法收敛速度越缓慢<sup>[14]</sup>。同时, LMS 算法的稳态误差与步长因子之间的函数关系成正比, 即如果步长越小, 则稳态误差越小。反之, 误差越大。

因此,在 LMS 自适应滤波算法中,步长因子与收敛性能之间存在一个显著的矛盾<sup>[39]</sup>。如果步长因子较大,该算法收敛速度较快,跟踪能力不错,但此时算法的稳态误差较大,无法对信号进行一个较为准确的滤波。反之,如果步长因子较小,则算法稳态误差小,虽然稳态性能较好,但得到的收敛速度则较慢,无法对实时性要求较高的信号进行滤波处理。所以,如果对滤波器的收敛速度有较高需求,则需要牺牲一定的稳态性能。如果对滤波器的稳态误差有较高的需求,就需要牺牲一定的收敛性能。收敛速度和稳态误差之间的矛盾,一直以来就是 LMS 算法的一个核心问题。针对这个问题,人们也提出了各自的改进算法<sup>[38]</sup>。

### 3.2.3 归一化最小均方自适应滤波算法

通过最速下降算法计算得到的抽头权向量  $\mathbf{w}(n)$ ,最终收敛于维纳解  $\mathbf{w}_o$ ,而 LMS 算法由于梯度噪声的存在,其解始终无法收敛到维纳解。在均方误差(mean square error, MSE)的准则下,LMS 算法得到的稳态解与维纳解之间相差的程度,被成为失调,而在 LMS 算法中,失调直接与滤波器输入量  $x(n)$ 成正比。因此,当  $x(n)$ 较大时,LMS 滤波器会遇到梯度噪声放大的问题,不过可以使用归一化 LMS 滤波器来解决这一问题<sup>[37]</sup>。

归一化最小均方(Normalized Least Mean Square, NLMS)自适应滤波算法是对 LMS 算法的改进,并在实际应用中取代了 LMS 算法的地位。而归一化 LMS 滤波器的设计准则,可以表述为约束优化问题。更新的抽头权值的增量如下:

$$\delta \mathbf{W}(n+1) = \mathbf{W}(n+1) - \mathbf{W}(n) \quad (3.16)$$

使该增量的欧式范数最小化,且约束条件如下:

$$\mathbf{W}^T(n+1)\mathbf{X}(n) = d(n) \quad (3.17)$$

接着,使用拉格朗日乘子法,来解决这个约束优化问题,考虑代价函数为:

$$J(n) = \|\delta \mathbf{W}(n+1)\|^2 + \text{Re}[\lambda(d(n) - \mathbf{W}^T(n+1)\mathbf{X}(n))] \quad (3.18)$$

求代价函数值最小时的最优权向量,需要式(3.18)对  $\mathbf{W}(n+1)$ 求导后,当其等于 0 时,得到最优解为:

$$\mathbf{W}(n+1) = \mathbf{W}(n) + \frac{1}{2}\lambda \mathbf{X}(n) \quad (3.19)$$

将式(3.19)代入约束条件式(3.17),可得到等式:

$$d(n) = \mathbf{W}^T(n)\mathbf{X}(n) + \frac{1}{2}\lambda\|\mathbf{X}(n)\|^2 \quad (3.20)$$

然后求得 $\lambda$ ，如下：

$$\lambda = \frac{2e(n)}{\|\mathbf{X}(n)\|^2} \quad (3.21)$$

将式(3.19)和式(3.21)代入式(3.16)，得到：

$$\delta\mathbf{W}(n+1) = \frac{1}{\|\mathbf{X}(n)\|^2}\mathbf{X}(n)e(n) \quad (3.22)$$

等价于：

$$\mathbf{W}(n+1) = \mathbf{W}(n) + \frac{1}{\|\mathbf{X}(n)\|^2}\mathbf{X}(n)e(n) \quad (3.23)$$

这就是归一化 LMS 算法的递归公式，为了对抽头权值向量的增量变化进行控制而不改变向量的方向，引入一个正的实数标度因子 $\tilde{\mu}$ ，则上式为：

$$\mathbf{W}(n+1) = \mathbf{W}(n) + \frac{\tilde{\mu}}{\|\mathbf{X}(n)\|^2}\mathbf{X}(n)e(n) \quad (3.24)$$

在克服了 LMS 算法梯度噪声的影响之后，人们的关注的问题在于归一化 LMS 算法本身所引起的问题，即当输入向量 $\mathbf{X}(n)$ 较小时，可能出现数值计算有困难的情况。因此，为了防止计算过程中发散，在分母中引入一个数值较小的正数 $\delta$ 。进而 NLMS 算法权值更新公式如下：

$$\mathbf{W}(n+1) = \mathbf{W}(n) + \frac{\tilde{\mu}}{\delta + \|\mathbf{X}(n)\|^2}\mathbf{X}(n)e(n) \quad (3.25)$$

其中 $\tilde{\mu}$ 的取值范围是在 0~2 之间的固定值。通常可以认为 NLMS 算法是一种改进的变步长 LMS 算法，解决了 LMS 算法由于输入权向量过大导致的失调增大的问题。

但是，针对传统的 NLMS 算法本身，仍有一个固定步长因子的问题，因此各种改进的变步长 NLMS 算法也相继被提出。

### 3.2.4 块自适应滤波算法

对于传统的 LMS 算法和 NLMS 算法来说，通常处理一个样值便更新一次滤波器系数，而块自适应滤波是将输入数据分成 $L$ 点的块，块被一次一块地加到长度为 $M$ 的滤波器，使得滤波器的自适应一块一块地进行。

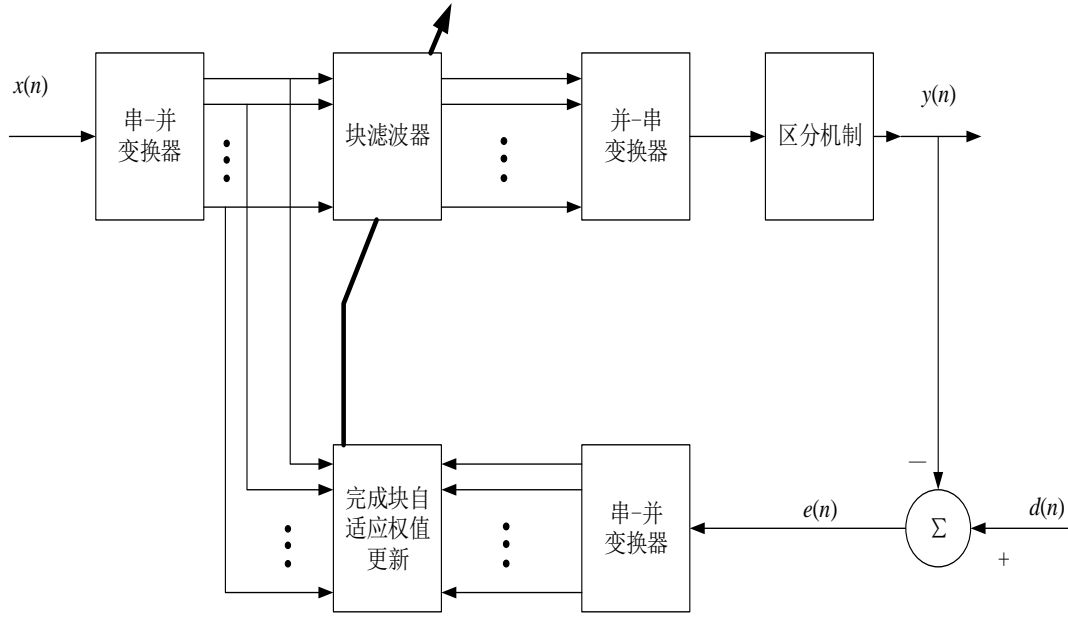


图 3.4 块自适应滤波器图

图 3.4 所示的是块自适应滤波器的结构。其中  $n$  时刻输入信号向量表示为：

$$\mathbf{X}(n) = [x(n), x(n-1), \dots, x(n-M+1)]^T \quad (3.26)$$

相应地， $n$  时刻抽头权向量，表示为：

$$\mathbf{W}(n) = [w_0(n), w_1(n), \dots, w_{M-1}(n)]^T \quad (3.27)$$

下面对  $\mathbf{X}(n)$  分块，设  $k$  表示块的下标，它与原始值时间  $n$  的关系为：

$$n = kL + i \quad i = 0, 1, 2, \dots, L-1; k = 1, 2, \dots \quad (3.28)$$

其中  $L$  是块的长度，则第  $k$  块的输入信号向量可定义为：

$$\mathbf{A}(k) = [x(kL), x(kL+1), \dots, x(kL+L-1)]^T \quad (3.29)$$

根据块 LMS 算法，针对输入信号向量  $\mathbf{x}(kL+i)$  做出的输出信号向量定义为：

$$\mathbf{y}(kL+i) = \mathbf{W}^T(k) \mathbf{A}(k) = \sum_{j=0}^{M-1} w_j(k) x(kL+i-j) \quad i = 0, 1, \dots, L-1 \quad (3.30)$$

设期望信号为  $\mathbf{d}(kL+i)$ ，则误差信号定义为：

$$\mathbf{e}(kL+i) = \mathbf{d}(kL+i) - \mathbf{y}(kL+i) \quad (3.31)$$

在块 LMS 算法中，每一块的抽头权向量的更新公式为：

$$\mathbf{W}(k+1) = \mathbf{W}(k) + \mu \sum_{i=0}^{L-1} \mathbf{x}(kL+i) \mathbf{e}(kL+i) \quad (3.32)$$

另外，块 LMS 算法使用了如下的梯度向量的估计：

$$\hat{\mathbf{v}}(k) = -\frac{2}{L} \sum_{i=0}^{L-1} \mathbf{x}(kL+i) \mathbf{e}(kL+i) \quad (3.33)$$

按照梯度向量来更新块 LMS 算法的权值更新公式，如下：

$$\mathbf{W}(k+1) = \mathbf{W}(k) - \frac{1}{2} \mu_B \hat{\mathbf{v}}(k) \quad (3.34)$$

其中常数  $\mu_B$  可以看作块 LMS 算法中的“有效”步长参数。

块 LMS 算法本质上是分块的 LMS 算法在时域上的计算，所以，经过快速傅里叶变换(Fast Fourier Transformation, FFT)到频域之后，同样也能进行自适应滤波。式(3.30)、式(3.33)使用了线性卷积和线性相关的计算，利用 FFT 算法为快速卷积和快速相关计算提供了支持，以及数字信号处理中的重叠存储方法和重叠相加方法，也为快速卷积计算提供了两种效率高的方法<sup>[37]</sup>。这些都为后续 WebRTC 的分段块频域自适应算法原理分析，提供了理论基础。

### 3.3 本章小结

本章根据回声产生的原因，简要分析了回声消除的原理，引出自适应滤波器算法。首先介绍了基于维纳原理的最速下降算法，在此基础上，推出 LMS 自适应滤波算法，为克服最小均方自适应滤波算法失调问题，进一步讲述 NLMS 自适应滤波算法，而后为了提供算法在实际应用中的运行效率，结合 LMS 算法，论述了块自适应滤波算法原理，该算法的分析将促进后续对 WebRTC 自适应滤波器的理解。

## 第 4 章 WebRTC 自适应回声消除算法分析及改进

现有的回声消除主要使用自适应滤波器来进行处理，其中涉及回声延迟估计、残留回声处理等算法。本文通过研究 WebRTC 中自适应回声消除(Adaptive Echo Cancellation, AEC)算法，归纳总结了该过程所涉及的算法原理、数学公式和计算过程。本章将讲述 WebRTC 自适应回声消除模块总结的成果，同时，根据该模块所使用的固定步长 NLMS 算法进行改进，提出一种改进的变步长 NLMS 算法。

### 4.1 自适应回声消除算法原理与分析

本章通过分析可知，WebRTC 的自适应回声消除算法主要包括三个重要模块，分别是回声延迟估计、线性自适应滤波器和非线性处理(NonLinear Processing, NLP)。回声延迟估计的作用是根据估计出来的回声延迟，对齐所要处理的语音信号，以便消除回声。线性自适应滤波器的作用是根据远端参考信号得出估计回声信号，用于消除回声，主要有两个基本过程：滤波过程和自适应过程。非线性处理的作用是对残留回声信号进行处理，消除残留回声。

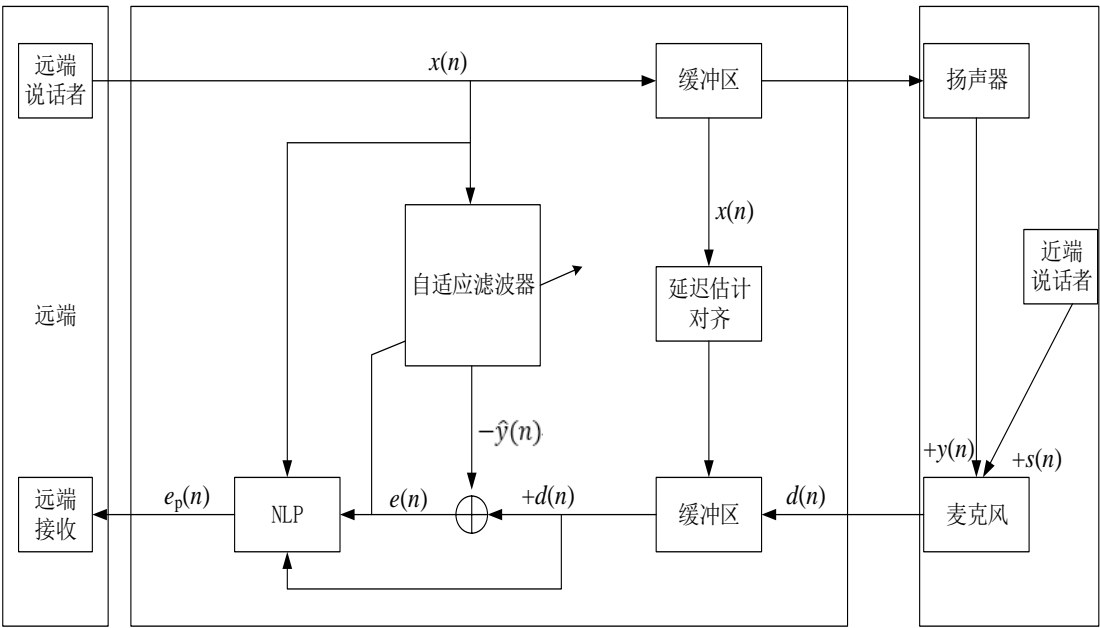


图 4.1 WebRTC 回声消除框架图



如图 4.1 所示,  $x(n)$  表示远端参考信号,  $y(n)$  表示回声信道产生的回声信号,  $s(n)$  表示近端说话者的语音信号,  $d(n)$  表示近端麦克风获取的近端语音信号, 包括  $s(n)$  和  $y(n)$ ,  $\hat{y}(n)$  表示自适应滤波器根据远端参考信号  $x(n)$  所得的回声估计信号,  $e(n)$  表示误差信号, 也称残留回声信号, 即近端语音信号去除回声估计信号所得的信号,  $e_p(n)$  表示残留回声信号经过非线性处理之后的信号。图 4.1 清晰地表示出 WebRTC 的回声消除的流程, 远端信号通过近端扬声器播放, 经过一定的延迟, 与近端讲话者的语音信号一起被近端麦克风捕获, 在缓冲区经过延迟估计对齐, 然后去除回声估计信号, 得到误差信号, 再经过残留回声处理, 得到最终的语音信号, 发送到远端。

#### 4.1.1 延迟估计方法

由于实际中, 远端参考信号经扬声器播放之后, 经过多路径反射之后, 到被麦克风捕获处理, 是有一定延迟的, 而回声延迟估计模块的作用是估计此延迟, 方便回声消除器能够准确地去除对应的回声, 所以回声延迟估计模块对整个回声消除过程有较大的影响<sup>[41]</sup>。

该估计过程需要远端参考信号和近端接收信号作为输入, 分别用连续时间的函数  $x(n)$ 、 $d(n)$  表示, 设时间间隔  $T_p$  和频带  $F_q$ 。

对每一个信号值和每个时间间隔计算周期图, 在时间间隔  $T_p$  中,  $x(n)$  的周期图是非负的实数矢量  $\boldsymbol{\varepsilon}_p = (\varepsilon_{1,p}, \varepsilon_{2,p}, \varepsilon_{3,p} \dots \varepsilon_{Q,p})$ , 其值计算如下:

$$\varepsilon_{q,p} = \int_{F_q} \left| \int_{T_p} e^{-j2\pi f n} x(n) w(n) dn \right|^2 df \quad (4.1)$$

其中  $w(n)$  是汉宁窗。同样地,  $d(n)$  的周期图是  $\boldsymbol{\lambda}_p = (\lambda_{1,p}, \lambda_{2,p}, \lambda_{3,p} \dots \lambda_{Q,p})$ , 其值计算如下:

$$\lambda_{q,p} = \int_{F_q} \left| \int_{T_p} e^{-j2\pi f n} d(n) w(n) dn \right|^2 df \quad (4.2)$$

对于远端参考信号值, 使用固定阈值  $\tilde{\varepsilon}_1, \tilde{\varepsilon}_2, \dots, \tilde{\varepsilon}_Q$  其每个值表示信号功率的周期图的平均值, 则每个周期图被量化为二进制矢量  $\mathbf{X}_p = (X_{1,p}, \dots, X_{Q,p})$ , 其中:

$$X_{q,p} = \begin{cases} 1 & \text{if } \varepsilon_{q,p} \geq \tilde{\varepsilon}_q \\ 0 & \varepsilon_{q,p} < \tilde{\varepsilon}_q \end{cases} \quad (4.3)$$

同样地，对近端语音信号值使用阈值 $\tilde{\lambda}_1, \tilde{\lambda}_2, \dots, \tilde{\lambda}_Q$ ，获得二进制矢量 $\mathbf{D}_p = (D_{1,p}, \dots, D_{Q,p})$ ，其中：

$$D_{q,p} = \begin{cases} 1 & \text{if } \lambda_{q,p} \geq \tilde{\lambda}_q \\ 0 & \lambda_{q,p} < \tilde{\lambda}_q \end{cases} \quad (4.4)$$

上述过程，通过与适合的阈值比较，分别对远端参考信号和近端语音信号进行量化，简化到二进制的形式，获得的两个二进制序列，对应语音信号的实际情况，二进制 1 表示有说话音，二进制 0 表示无说话音（静音或者很弱的音），则存在四种可能的共生：(0, 0), (0, 1), (1, 0), (1, 1)。

缓冲区将远端参考信号和近端语音信号的二进制矢量收集到二进制矩阵 $\mathbf{X} = (X_1, \dots, X_{K+D_m})$ 和 $\mathbf{D} = (D_1, \dots, D_K)$ 中，用于最优候选延迟 $Delay_m$ 的计算。

设  $M$  个候选延迟 $Delay_1 < Delay_2 < \dots < Delay_M$ ，其每个值表示为连续的时刻的距离的倍数，即延迟单位。通过缓冲区收集的二进制矩阵，对每个候选延迟计算平均代价。

对于第  $q$  个量、第  $m$  个候选延迟和在  $K$  个时刻内的平均代价，计算如下：

$$\Delta_{q,m} = \frac{1}{K} \sum_{k=1}^K \text{penalty}(X_{q,k+Delay_m}, D_{q,k}) \quad (4.5)$$

在所有候选延迟的平均代价计算完成后，按照下式进行加权和计算：

$$\Delta_m = \sum_{q=1}^Q C_q \Delta_{q,m} \quad (4.6)$$

其中系数 $C_1, C_2, \dots, C_q$ 优选地反应每个量的重要性。在生成加权和 $\Delta_1, \Delta_2, \dots, \Delta_m$ 之后，对应候选延迟 $Delay_1, Delay_2, \dots, Delay_M$ 当中的最小的值，将是最好的估计 $Delay_M$ 。

根据该回声延迟估计模块得到的延迟估计值，缓冲区会进行延迟估计对齐，这样回声消除器中的线性自适应滤波器就可以合理地处理包含对应回声的近端语音信号<sup>[47]</sup>。

#### 4.1.2 线性自适应滤波器

经过延迟对齐的语音信号，将用于去除自适应滤波器估计所得的估计回声。WebRTC 回声消除算法中所用到的自适应滤波算法，通常被称为分段块频域自适应

滤波算法<sup>[42, 43]</sup>，有时也被称为多延迟分块频域滤波算法<sup>[44]</sup>。其本质是在时域分块之后，将信号经快速傅里叶变换到频域，再在块自适应滤波器上一块一块的进行 NLMS 处理，得到估计的回声信号。

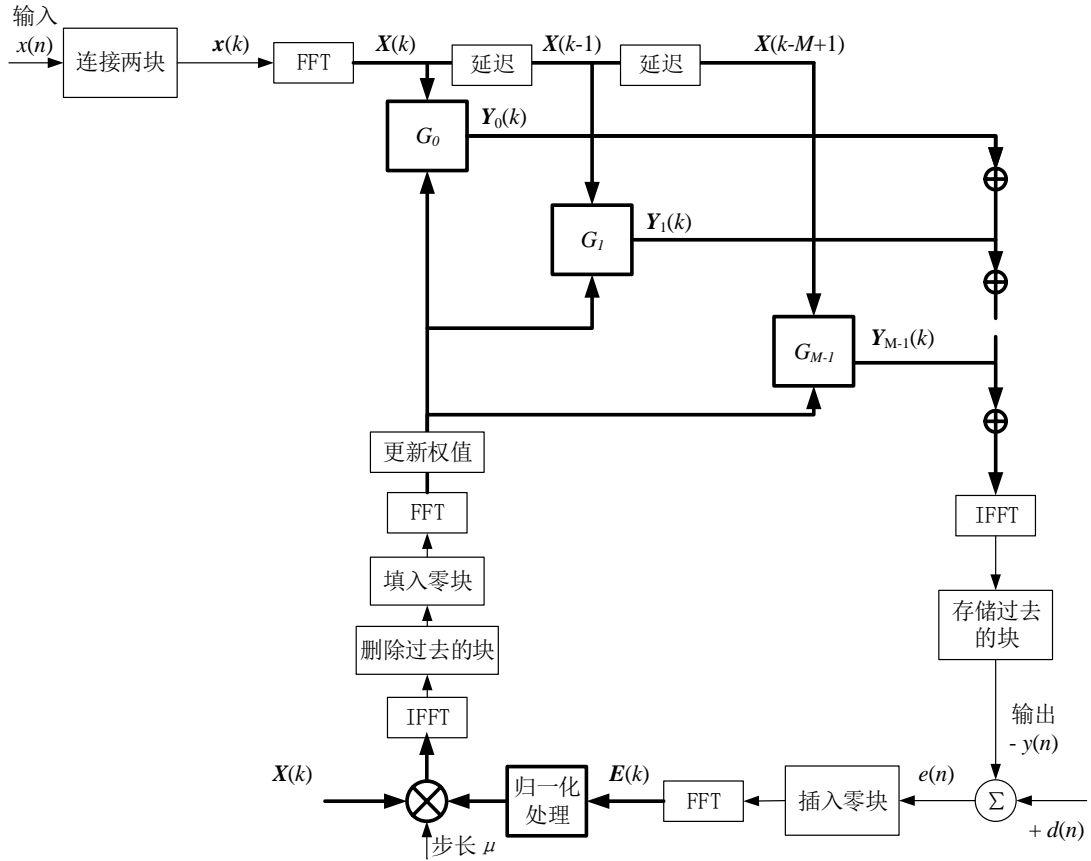


图 4.2 分段块频域处理方法图

如图 4.2 所示，细线表示时域中操作，粗线表示频域中操作，该方法对远端参考信号  $x(n)$  分块后，连接两块数据，做 FFT 变换，再分别对各块数据做频域滤波，累加之后再 IFFT 变换，由于使用重叠存储法，所以只取后  $N$  点作为有效结果，得到估计回声  $y(n)$ ，再将所得估计信号从近端获取信号  $d(n)$  中去除，得到误差信号  $e(n)$ 。而误差信号再插入零块，进行 FFT 变换后，参与自适应滤波器的权值更新，根据 NLMS 公式要求，进行归一化处理，与固定的步长因子和输入信号在频域的共轭矩阵相乘，再经 IFFT 变换后，废除过去的块，插入零块，做 FFT 变换，用作滤波器的抽头权重更新后，再分块进行 NLMS 处理<sup>[44]</sup>。

设总滤波器的长度  $L=M*N$ ，在滤波器的长度范围内有  $G_m$  块或分区，块或分区内有  $N$  个样本。对远端参考信号  $x(n)$  进行分块后，连接两个块，经过 FFT 转换，同时由于转换分块，会导致信号延迟，所以 FFT 转换后矩阵的表达式如下：

$$\mathbf{X}(k-m) = \text{diag}(\text{FFT } x(k-m)) \quad (4.7)$$

而  $\mathbf{X}(k-m)$  乘以滤波器分区，得到估计的回声，如下：

$$\mathbf{Y}_m(k) = \mathbf{X}(k-m)\mathbf{W}_m(k) \quad m = 0, 1, \dots, M-1 \quad (4.8)$$

将  $\mathbf{Y}_m(k)$  作 IFFT 转换，并获取信号总和的最后  $N$  个样值，估计的回声信号在时域中的表示如下：

$$y(n) = [\mathbf{0}_N \quad \mathbf{I}_N] \text{IFFT} \sum_{m=0}^{M-1} \mathbf{Y}_m(k) \quad (4.9)$$

其中  $\mathbf{I}_N$  是  $N*N$  的单位矩阵， $\mathbf{0}_N$  是  $N*N$  的零矩阵。然后在时域中得到误差信号  $e(n)$ ，计算如下：

$$e(n) = d(n) - y(n) \quad (4.10)$$

得到的就是误差信号，即残留回声信号。该信号将用于调整滤波器系数，即把误差信号作插入零块处理后，进行 FFT 转换，得到  $\mathbf{E}(k)$ ，表示为：

$$\mathbf{E}(k) = \text{FFT} \begin{bmatrix} \mathbf{I}_N \\ \mathbf{0}_N \end{bmatrix} e(n) \quad (4.11)$$

根据 NLMS 算法式(3.25)要求，对误差信号  $\mathbf{E}(k)$  进行归一化处理得到  $\mathbf{N}(k)$ ，再经过与固定的步长因子  $\mu_0 = 0.6$  和输入向量  $\mathbf{X}(k-m)$  相乘，做 IFFT 转换之后，删除过去块，插入零块，再 FFT 转换后，用于滤波器系数更新，公式如下：

$$\mathbf{W}_m(k+1) = \mathbf{W}_m(k) + \mu_0 \mathbf{N}(k) \mathbf{X}(k-m) \quad (4.12)$$

为了表达简便，式(4.12)中省略了 FFT 和 IFFT 的表达符号。至此，线性滤波器的过滤和自适应两个过程完成。

该方法过滤和权值更新是在频域中执行，这样时域卷积将被乘法所替代，计算复杂性大大降低<sup>[45]</sup>。核心算法是 NLMS 算法，只是将数据分段块后，在频域中过滤。由于该方法使用固定步长因子 0.6 的 NLMS 算法，本文后续将提出改进的变步长 NLMS 算法。

### 4.1.3 基于互相干性的非线性处理

在 WebRTC 的回声消除算法中, 经过去除估计回声的信号, 仍然包含部分残留回声信号, 所以需要使用非线性处理将残留回声进行抑制, 获得最终的语音信号。其原理是将基于块的信号转换为频域, 针对每个频带, 计算至少两个信号之间的一个或者多个相干性, 基于这些相干性的值, 计算每个频率带相应的抑制因子, 通过抑制因子消除语音信号中的残留回声<sup>[46]</sup>。

如图 4.1 所示, 该模块需要远端参考信号  $x(n)$ 、误差信号  $e(n)$  和近端语音信号  $d(n)$  作为输入信号。同样采用块长度  $N$ , 使用重叠相加法, 将连续的块串联, 通过加汉宁窗, 再转换成为频域。对于误差信号  $e(n)$ , 第  $k$  块的转换块  $\mathbf{E}_k$  可表示为:

$$\mathbf{E}_k = FFT \mathbf{w}_{2N} \circ \begin{bmatrix} \mathbf{e}_{k-1} \\ \mathbf{e}_k \end{bmatrix} \quad k \geq 0, \mathbf{e}_{-1} = \mathbf{0}_N \quad (4.13)$$

其中  $\circ$  表示元素乘积算子,  $\mathbf{e}_k$  是长度为  $N$  的时域采用列向量,  $\mathbf{w}_{2N}$  是长度为  $2N$  的平方根汉宁窗列向量, 其值可表示为:

$$w(n) = \sqrt{\frac{1}{2} \left[ 1 - \cos\left(\frac{2\pi n}{2N}\right) \right]} \quad n = 0, 1, \dots, 2N-1 \quad (4.14)$$

汉宁窗的作用在于可以保证重叠的分段符合下列等式, 以提供完美的重构:

$$w^2(n) + w^2(n-N) = 1 \quad n = N, N+1, \dots, 2N \quad (4.15)$$

同样的,  $\mathbf{X}_k$ 、 $\mathbf{D}_k$  分别为第  $k$  块的远端参考信号块、近端语音参考信号块的频域表示。

首先, 计算每个信号的功率谱密度(Power Spectral Density, PSD), 远端参考信号、近端语音信号和误差信号的功率谱密度分别表示为  $\mathbf{S}_x$ 、 $\mathbf{S}_d$ 、 $\mathbf{S}_e$ 。

再次, 需要计算分别计算两个信号之间的互功率谱密度, 即远端参考信号  $x(n)$  和近端语音信号  $d(n)$  之间的互功率谱密度  $\mathbf{S}_{xd}$ , 近端语音信号  $d(n)$  和误差信号  $e(n)$  之间的互功率谱密度  $\mathbf{S}_{de}$ 。互功率谱密度  $\mathbf{S}_{xd}$  的计算公式如下:

$$\mathbf{S}_{\mathbf{X}_k \mathbf{D}_k} = \lambda_s \mathbf{S}_{\mathbf{X}_{k-1} \mathbf{D}_{k-1}} + (1 - \lambda_s) \mathbf{X}_k \circ \mathbf{D}_k \quad k > 0, \mathbf{S}_{\mathbf{X}_0 \mathbf{D}_0} = \mathbf{0}_N \quad (4.16)$$

互功率谱密度是按照指数方式平滑变化, 其中指数平滑系数  $\lambda_s$  表示如下:

$$\lambda_s = \begin{cases} 0.9 & \text{if } fs = 8000 \\ 0.93 & \text{else} \end{cases} \quad (4.17)$$

同样的, 互功率谱密度  $\mathbf{S}_{de}$  的计算如公式(4.16)。

在非线性处理阶段，同样需要选择最佳区块进行残留回声处理。可使用最大能量值，计算出延迟指数，用于远端参考信号的最佳区块，如下列公式：

$$\tilde{d} = \arg \max(\|W_m\|^2) \quad (4.18)$$

此外，考虑到线性自适应滤波器会有偏离回声路径估计的情况，一般通过检测散度，来选择非线性处理的信号。具体如下，如果：

$$\|S_{E_k E_k}\|_1 > \|S_{D_k D_k}\|_1 \quad (4.19)$$

则进入发散状态，令  $E_k = D_k$ ，直接处理近端语音信号，如果散度过于高，如：

$$\|S_{E_k E_k}\|_1 > \sigma_1 \|S_{D_k D_k}\|_1 \quad \sigma_1 = 19.05 \quad (4.20)$$

则使线性自适应滤波器恢复到初始状态，即：

$$W_m(k) = \mathbf{0}_N \quad m = 0, 1, \dots, M-1 \quad (4.21)$$

如果满足下列判断条件，则退出发散状态：

$$\sigma_0 \|S_{E_k E_k}\|_1 < \|S_{D_k D_k}\|_1 \quad \sigma_0 = 1.05 \quad (4.22)$$

退出发散状态之后，进入正常的非线性处理过程。

通过式(4.16)计算所得的功率频谱密度  $S_{de}$ 、 $S_{xd}$  分别用于计算远端参考信号和近端语音信号之间的频率带相干性  $c_{xd}$ ，以及近端语音信号与误差信号之间的频率带相干性  $c_{de}$ ，计算公式分别如下：

$$c_{xd} = \frac{S_{X_{k-d} D_k} \circ S_{X_{k-d} D_k}}{S_{X_{k-d} X_{k-d}} \circ S_{D_k D_k}} \quad (4.23)$$

$$c_{de} = \frac{S_{D_k E_k} \circ S_{D_k E_k}}{S_{D_k D_k} \circ S_{E_k E_k}} \quad (4.24)$$

由式(4.23)和式(4.24)计算所得相干性  $c_{xd}$  和  $c_{de}$  分别用于增加非线性处理的稳定性和用于直接抑制误差信号得到输出信号。非线性处理过程的输出表示为：

$$\tilde{Y}_k = E_k \circ c_{de} \quad (4.25)$$

通过重叠相加转换，逆转为长度为  $N$  的时域输出信号  $\tilde{y}_k$ ，公式如下：

$$\begin{bmatrix} \tilde{y}_k \\ \tilde{y}'_{k+1} \end{bmatrix} = IFFT w_{2N} \circ \tilde{Y}_k + \begin{bmatrix} y'_k \\ \mathbf{0}_N \end{bmatrix}, k \geq 0, y'_0 = \mathbf{0}_N \quad (4.26)$$

综上所述，本节是 WebRTC 自适应回声消除算法的原理分析。根据分析可知，回声估计算法通过共生计数，附加代价值来统计最小值来获得最佳估计的，由于实际源码实现与算法流程略有不同，本文未详细描述。另外，非线性处理过程较为复

杂，尤其是相干性作为抑制因子之前，会使用多种数据分析的方法来微调相干性，理论上过于复杂，本文未展开分析。重点是线性自适应滤波器部分，该部分使用的是固定步长因子 NLMS 算法，由于固定步长自身存在一定的局限性，基于此，本文提出改进的 NLMS 算法。

## 4.2 改进的变步长 NLMS 算法

由第三章和本章可知，自适应回声消除的核心部分是靠自适应滤波器的滤波过程和自适应过程来完成的，而当前流行的自适应滤波算法便是 NLMS 算法。传统的 LMS 算法存在稳态误差和收敛速度之间具有矛盾的问题，在自适应过程中，步长的取值对算法的收敛速度和稳态误差有着不同的影响<sup>[48]</sup>。如步长较小时，稳态误差也会随之较小，但收敛速度会降低，而步长较大时，收敛速度会提升，但稳态误差便会增大。而传统的 NLMS 算法，虽然定义了步长与参考信号之间的动态函数关系<sup>[49]</sup>，但是仍有一个固定步长因子，如 WebRTC 回声消除模块在分段块频域自适应算法中使用的便是固定因子的 NLMS 算法，它在获取最优步长<sup>[50]</sup>时对收敛速度也会产生较大的影响，因此，基于此问题的变步长 NLMS 算法相继提出。

文献[51]中提出利用  $\text{atan}(g)$  函数来建立步长与梯度之间的函数关系，其中定义的梯度是利用瞬时信号估计得到，而瞬时信号的估计看做是误差  $e(n)$  和参考信号  $x(n)$  的互相关函数，加入归一化算法，从而得到新的变步长的 NLMS 算法。步长公式如下：

$$\mu(n) = \mu_0 \alpha \text{atan}(\beta \|e(n)x(n)\|) \quad (4.27)$$

基于此方法，在以均方误差为准则，求自适应滤波器算法过滤过程中的最优解时，定义均方误差的代价函数  $J$ ，该代价函数  $J$  为最小值时，则是最优解。对于这个条件，有一个等效的条件就是正交性原理：

$$E[x(n)e_0(n)] = 0 \quad (4.28)$$

其中  $e_0(n)$  是最优条件下估计误差的特定值。因此， $e(n)$  和  $x(n)$  的互相关函数随着迭代会不断减小，在算法初始收敛阶段会很大，而当算法趋于稳定之后，其值会很小，所以可以用来反应步长的变化。

基于上述文献参考与分析，本文针对传统 NLMS 中固定步长因子，提出了一种新的变步长的 NLMS 算法，抽头权值更新公式如下：

$$\mathbf{W}(n+1) = \mathbf{W}(n) + \frac{\mu(n)}{\varepsilon + \|\mathbf{X}(n)\|^2} e(n)\mathbf{X}(n) \quad (4.29)$$

初始时，因子 $\alpha=0.9$ ， $\beta=0.85$ ，用来防止信号突变的因子，而变步长公式如下：

$$\mu(n) = \frac{\alpha\mu_1(n)}{\mu_2(n)} \quad (4.30)$$

该变步长公式本质上也是使其根据误差信号和输入信号的变化，进行迭代更新，类似于权值更新。其中 $\mu_1(n)$ 是误差信号量值和输入信号量矩阵的欧式范数来表达，公式如下：

$$\mu_1(n) = \|e(n)\mathbf{X}(n)\| \quad (4.31)$$

而 $\mu_2(n)$ 是随误差信号和输入信号的迭代更新，初始值为0，定义公式如下：

$$\mu_2(n) = \beta\mu_2(n) + \|e(n)\mathbf{X}(n)\| \quad (4.32)$$

以上公式即本文提出的变步长的 NLMS 算法。

### 4.3 实验结果与分析

本节将使用 MATLAB(R2017b)软件对本章的算法进行仿真实验<sup>[52,53]</sup>，每条仿真曲线均独立仿真 100 次以上。通过测试不同步长对 NLMS 算法的影响，对比算法的收敛速度和跟踪能力以及在回声消除中的应用，来证明本文改进的 NLMS 算法优于传统的 NLMS 算法。

#### 1. 不同步长对 NLMS 算法的影响

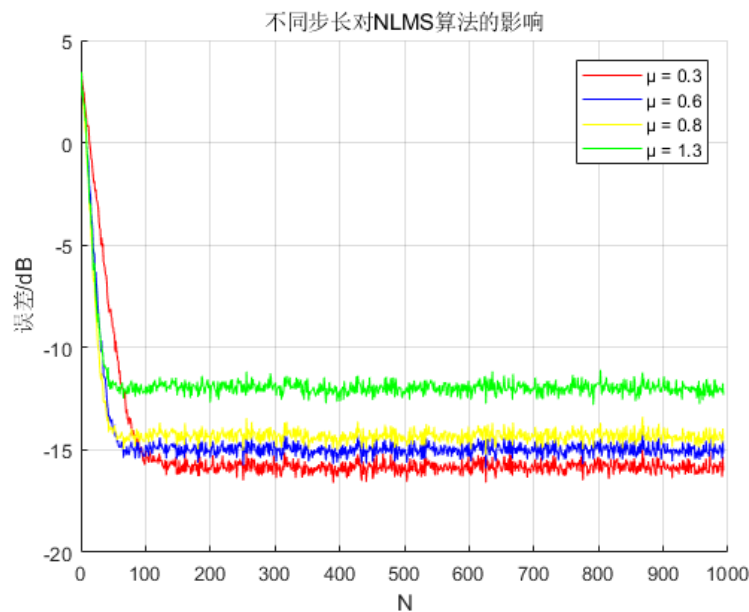


图 4.3 不同步长对 NLMS 算法的影响



将步长因子 $\mu$ 分别取固定值 0.3、0.6、0.8、1.3，来分析算法的收敛特效，得到的误差曲线如图 4.3 所示。

由图 4.3 可知，下降曲线的斜率反应了算法的收敛速度，稳定后的误差反应了算法的精度。由结果可知，不同的步长对 NLMS 算法是由不同的影响，随着步长的增大，起始时算法的收敛速度是有所提升，而最后达到稳态时算法的精度值也在增大。由实验可以看出，该算法自适应收敛时收敛速度和精度是存在固有矛盾的，步长越大，收敛速度可能有所提升，但精度会增大，反之亦然。

所以，可以通过采用变步长的方式在确保精度较小的情况下，缩短自适应收敛过程。

## 2. 对比改进的 NLMS 算法

通过实验对比，证明本文中改进的 NLMS 算法，在最小均方误差准则下，所得到的收敛效果是最佳的。测试数据采用随机模拟语音信号，自适应滤波器阶数为 9，采样点数为 500。

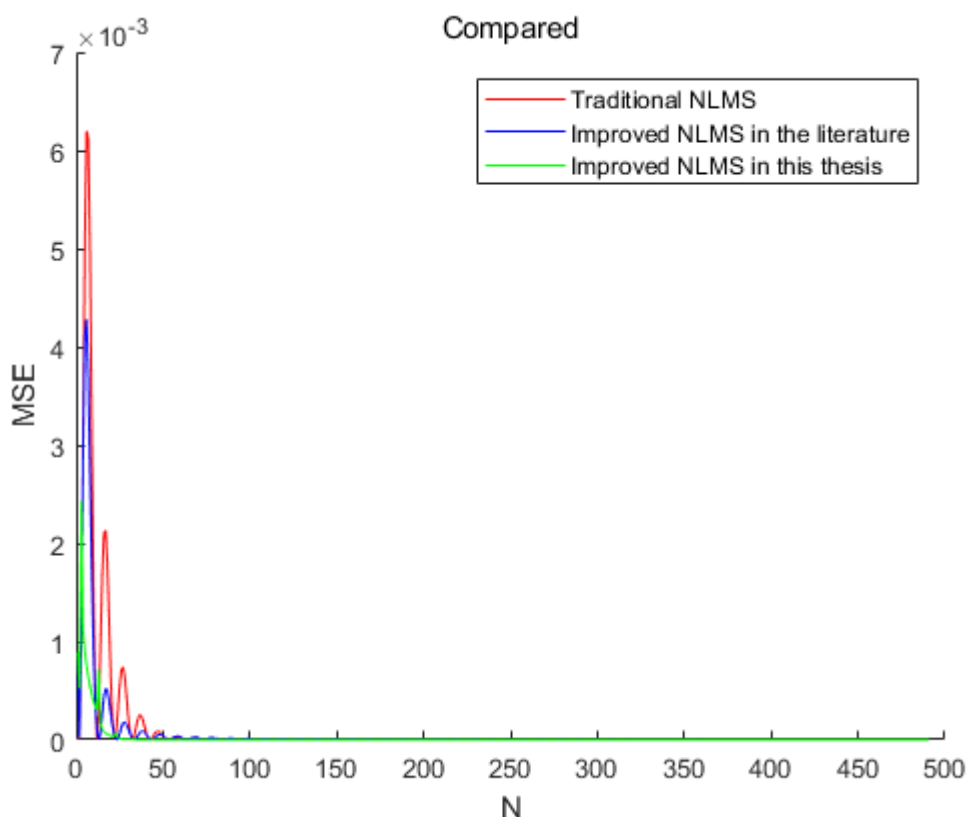


图 4.4 改进的 NLMS 对比

如图 4.4 所示, 红线是固定步长因子  $\mu = 0.3$  时的收敛曲线, 蓝线是文献[51]改进的 NLMS 算法收敛曲线, 绿色是本文改进的 NLMS 收敛曲线, 其中  $\alpha = 0.9$ ,  $\beta = 0.8$ ,  $\mu_2(n) = 0$ 。由图 4.4 可看出, 本文改进的算法在起始时的误差量均小于对比算法, 随后传统 NLMS 算法的收敛级数到 50 左右趋于稳定, 而文献[51]改进的 NLMS 和本文改进的 NLMS 在 40 左右趋于稳定, 另外, 从图中可以看出本文改进的算法收敛状态略稳定。

另外, 在相同的稳态失调的条件下, 验证算法的跟踪能力, 即信号经过突变之后, 对比传统 NLMS 算法和本文改进的 NLMS 算法。NLMS 算法的固定步长因子  $\mu = 0.1$ , 本文改进的 NLMS 算法  $\alpha = 0.9$ ,  $\beta = 0.85$ ,  $\mu_2(n) = 0$ 。

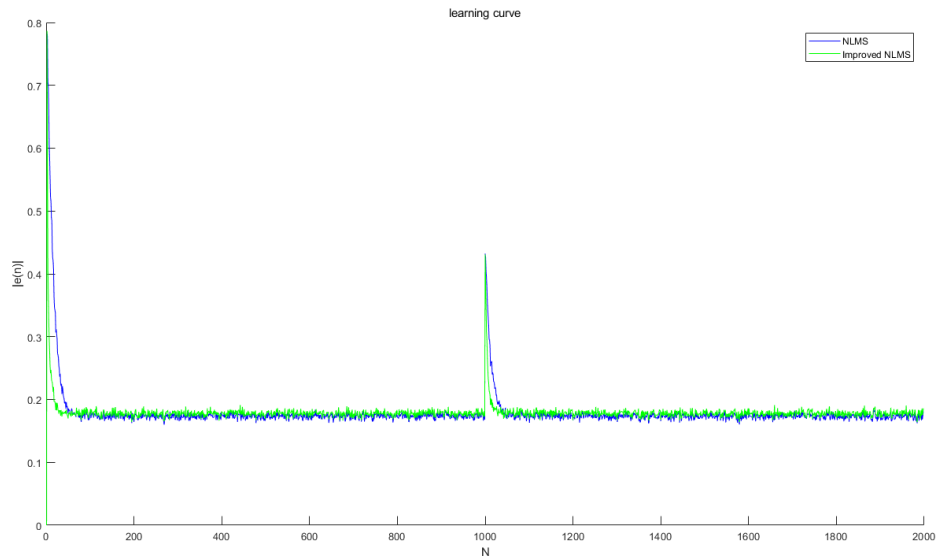


图 4.5 跟踪能力学习曲线

如图 4.5 所示, 蓝线是传统 NLMS 算法, 绿线是本文改进的算法。采样点为 2000, 当  $n = 1000$  时, 系统发生突变, 从图 4.5 可以看出, 本文改进的算法跟传统的 NLMS 算法相比有较强的跟踪能力。

本文在传统的 NLMS 算法的权值更新中, 引入变步长的定义, 通过误差信号和输入信号的互相关函数, 来动态调整步长, 根据仿真实验, 可以看出改进的 NLMS 算法在获得较小的稳态误差时, 其收敛速度也优于传统的 NLMS 算法, 并且当系统引入发生突变时, 拥有较强的跟踪能力。

### 3. 回声消除实验

基于上述的改进，将改进的 NLMS 算法应用于回声消除，验证经过改进算法处理所得到的语音信号，去回声效果更佳。测试数据为语音电话通信过程中，截取的一段远端语音信号和近端语音信号，本测试并未经过非线性处理残留回声。

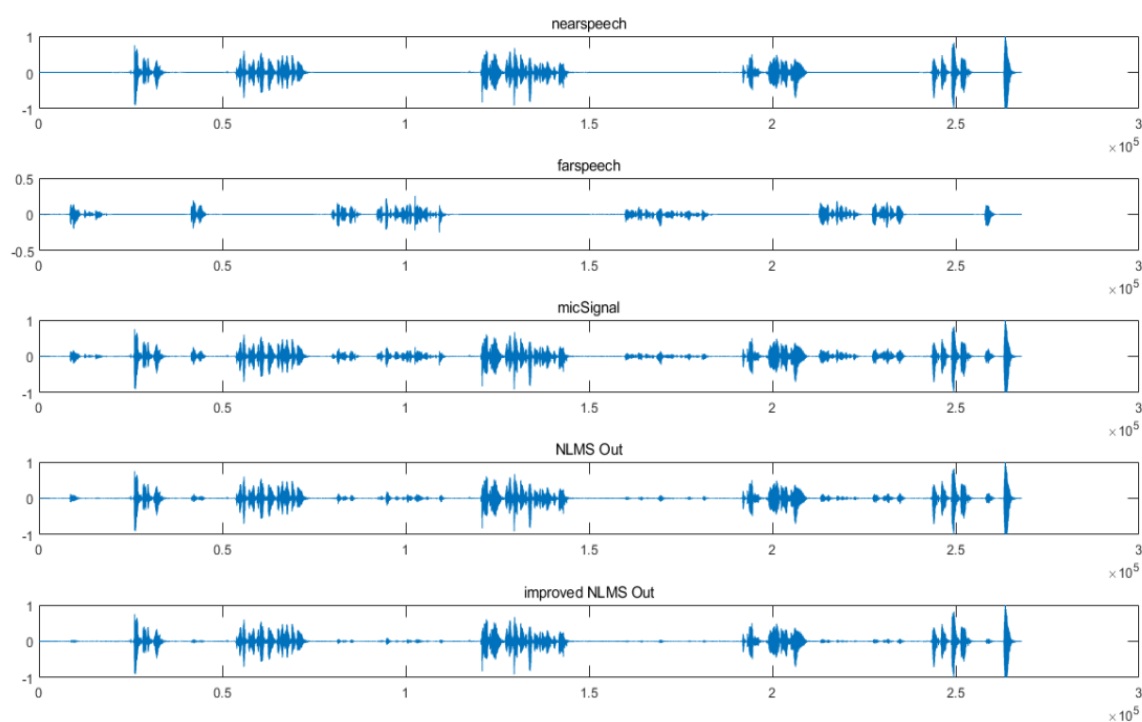


图 4.6 回声消除结果对比

图 4.6 中，第一幅图是近端语音信号。第二幅图是远端语音信号。第三幅图是通过 DSP 工具模拟出近端语音信号混入远端语音信号后麦克风获得的信号。第四幅图是传统固定步长 NLMS 算法处理后得到的语音信号，可以看出残留回声较多，实际效果仍有人耳可识别的回声。第五幅图是本文改进的变步长 NLMS 算法处理后所得到的语音信号，通过图 4.6 可以看出，处理效果明显优于传统 NLMS 算法的处理效果。

综上所述，本文所提出的改进 NLMS 算法，通过建立步长与误差信号和输入信号互相关性的函数关系，使得算法在获得较小稳态误差的情况下，收敛速度也较快，同时面对系统信号突变，拥有较好的跟踪能力，可以快速收敛。并通过实际语音处理对比残留回声，经改进的 NLMS 算法，获得的语音效果优于传统 NLMS 算法。

## 4.4 本章小结

本章主要研究分析了 WebRTC 框架中的自适应回声消除算法,分别对其延迟估计、线性自适应滤波器、非线性处理三个模块进行归纳总结,其中线性自适应滤波器是核心,滤波器结构上使用分段块频域处理方法,重点算法是 NLMS 算法。本章在文献研究的基础上,根据误差信号和输入信号与步长之间的函数关系,提出改进的变步长 NLMS 算法,改善了由于固定步长算法本身所带来的矛盾问题,在较小的稳态误差情况下,改进的算法拥有较快的收敛速度和较强的跟踪能力,在处理回声消除的效果也较好,并通过 MATLAB 仿真软件进行实验验证,证明了本章改进的变步长 NLMS 算法具有较好的优势。

## 第 5 章 基于 WebRTC 的会议系统的设计与实现

本章在第二章的基础上，同时结合会训无线投影云管理系统项目需要线上会议场景的使用需求，完成基于 WebRTC 的多人音视频会议系统的设计与实现。本系统主要分为服务器端和客户端，服务器端包括 Web 服务器、信令模块、NAT 穿越服务器模块、房间服务器模块等，客户端包括会议房间进入界面模块、会议房间主界面模块等。

### 5.1 系统简介与需求分析

本系统主要是满足会训无线投影云管理系统的线上会议的需求，使用户可以通过浏览器，在任意地点都能参加线上会议<sup>[54]</sup>。如表 5.1 所示，对当前市场上主流的音视频通信应用进行对比，同时根据项目实际情况，决定本系统限定允许 8 人以内的用户可以同时参加，保证视频画面清晰，音频响亮无杂音。

表 5.1 对比		
来源	软件名称	参与者数量限制
国外	Google Hangouts	10
	Houseparty	8
	Skype	25
	appear.in	12
	Amazon Chime	桌面版 16, IOS 版 8
国内	WeChat	9
	QQ	50

#### 5.1.1 功能需求分析

基于目前市场上现有的 Web 端会议系统所提供的功能，提出本系统的功能需求。本会议系统的功能需求主要有：用户可以创建会议房间、系统可以打开用户浏览器的音视频设备、用户进入会议房间后可进行视频聊天、用户在会议房间可进行即时消息发送<sup>[55]</sup>、用户可以实现屏幕共享。

### 5.1.2 非功能需求分析

#### 1. 系统的实用性

首先，对于用户来说，使用基于 WebRTC 的 Web 音视频应用，不需要安装任何插件，只需要允许程序调用本地的音视频设备。其次，随着 WebRTC1.0 标准的制定，WebRTC 在 Web 端的使用已成大势所趋。2018 年，目前市面上主流的浏览器都已经相继支持或者兼容 WebRTC 的标准接口<sup>[56]</sup>，如微软的 Edge 浏览器、Safari 浏览、Google 浏览器、Firefox 浏览器和 360 浏览器。基于 WebRTC 的音视频会议应用，可以很方便地通过任意一款浏览器进行线上会议。所以该 WebRTC 会议系统的实用性还是很好的。

#### 2. 系统的安全性

当前的安全浏览器页面使用 TLS 传输机制，WebRTC 使用 TLS 来保证信令 and 用户界面的安全，另外，根据 WebRTC 的技术标准，浏览器会在 WebRTC 的音视频通信过程中使用 DTLS 协议，即 Datagram TLS。根据 WebRTC 的安全要求，本系统使用的 Web 服务器需要部署为 HTTPS 模式，才能保证整个 WebRTC 音视频通信的安全<sup>[57]</sup>。所以，从 WebRTC 技术自身来看，就已经保证了该会议系统的安全性。

#### 3. 系统的稳定性

本会议系统的音视频模块的设计使用的是目前已经较为成熟和流行的 P2P 网状拓扑结构。根据本会议系统的音视频架构的设计，服务器端的压力不大，主要是用户端的网络带宽压力较大。由于本系统是结合会训无线投影云管理系统使用，除到场人员之外，在线上参加会议的人数较少，以及根据目前我国国内运营商所提供的光纤网络带宽提高到 100M、500M 甚至更高，所以基本可以保证 8 人以内的人员进入同一会议房间时，能够正常通信，其稳定性也是可以保障的。

## 5.2 系统整体架构设计

本节主要从设计的角度，讲述本会议系统的整体模块划分、会话管理信令消息类型、客户端模块和服务端架构。

5.2.1 整体模块设计

本会议系统从整体流程上看，系统通过 node.js 保证系统 Web 页面的运行，然后由客户端产生房间号、信令、媒体流、文本消息等数据，并将各类数据通过不同类型的信令消息发送到服务器端的信令模块，系统通过 WebSocket 保证信令数据的正常传送，利用 socket.io 提供对信令消息的支持，服务器端根据不同的信令消息触发不同的事件，完成不同模块的功能。

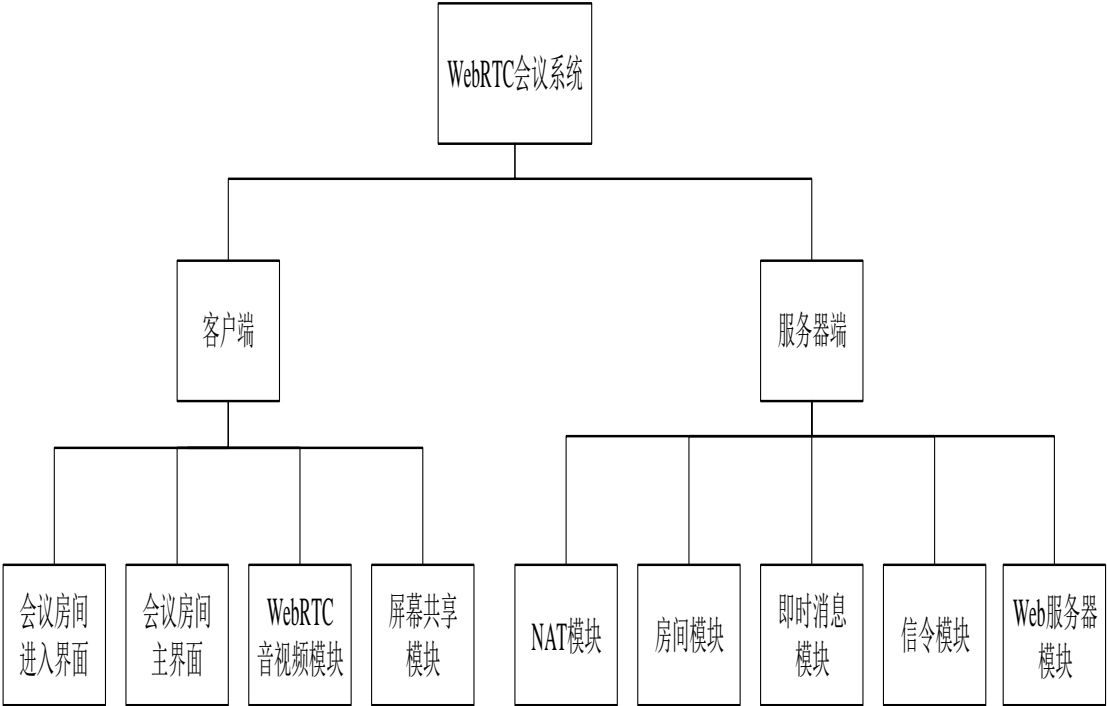


图 5.1 整体功能模块框架图

如图 5.1 所示，从宏观上展示了本会议系统的整体功能模块图，主要分为客户端和服务端。客户端主要有两个界面模块和两个功能模块组成，分别是会议房间进入界面、会议房间主界面、WebRTC 音视频模块、屏幕共享模块。重点是两个功能模块，WebRTC 音视频模块主要负责端到端的音视频通信，屏幕共享模块负责共享用户屏幕，方便会议中分享内容。服务器端主要包括信令模块、NAT 模块、房间模块、即时消息模块、Web 服务器模块五大模块。重点是信令模块，该模块是本会议系统的会话管理机制，负责为用户创建、加入或者离开会议房间，以及辅助即时消息发送，贯穿整个系统实现。

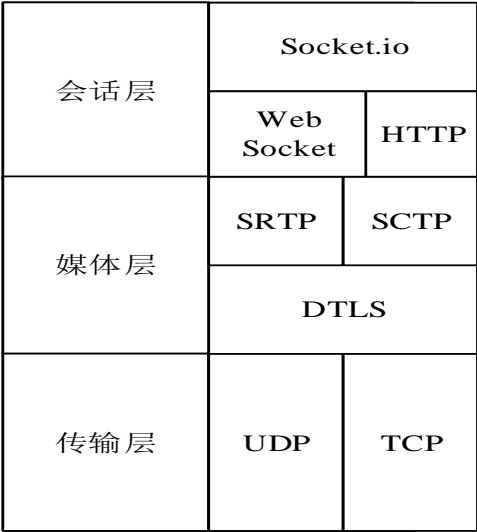


图 5.2 逻辑协议框架图

如图 5.2 所示，本系统在逻辑上可分为三层架构：会话层、媒体层、传输层，图 5.2 中展示了每层所使用到的协议。会话层负责会话管理，利用信令，保证浏览器之间的 SDP 和 candidate 数据的传递，管理用户的加入和退出。媒体层负责获取用户媒体信息和媒体数据流。传输层分为 TCP 和 UDP 协议，分别对应 WebSocket 和 RTCPeerConnection，一个为信令通道确保信令消息的传输，一个为浏览器媒体流之间的传输提供服务。

5.2.2 会话管理信令消息设计

本系统需要通过服务器对用户之间进行会话管理，在客户端与服务器端之间信令消息交互的过程中，根据不同类型的信令信息，进行相应的操作。针对本系统，下面自行设计会话管理机制的信令消息，传输过程中信令消息以 JSON 格式存储。

表 5.2 createOrJoinRoom 类型			
数据字段	数据类型	长度	说明
id	String	20	用户标识
nickname	String	20	用户昵称
room	String	20	房间号
peers	Array	10	房间内的用户

表 5.2 所示的是 createOrJoinRoom 类型的信令消息，作用是服务器根据该信令使客户端用户创建或加入会议房间，是用于前后端会话交互的信令，其格式与 created



类型相同。而 `created` 信令则是根据服务器上的数据，返回当前会议房间的信息，供客户端程序使用。`id` 是由 `WebSocket` 所生成的 `socket` 的唯一标识，`nickname` 是用户在浏览器页面显示的名称，默认是 `me`，`room` 是用户输入的房间号，`peers` 是当前房间内其他的用户，每个用户的 `peers` 都不包含自己。

当用户等到他人进入房间后，根据 `WebRTC` 的提议/应答协商的通信机制，产生 `offer/answer` 和 `candidate` 数据，为浏览器之间创建 `RTCPeerConnection` 连接，提供 `sdp` 数据交互，而数据的交互需要 `offer`、`answer` 和 `candidate` 类型的信令消息来完成。

表 5.3 offer 类型

数据字段	数据类型	长度	说明
from	String	20	发送 offer 的用户
to	String	20	接收 offer 的用户
room	Array	10	房间号
sdp	String	不限	会话描述协议信息

表 5.3 所示的是 `offer` 类型的信令消息，该类型信令的作用是当有其他用户进去房间之后，将用户创建的请求建立连接的数据即 `offer` 数据，通过 `offer` 信令消息发送给其他用户。`from` 是设置 `offer` 的用户的标识，`to` 是在信令服务器上需要中继/转发给的用户标识，`room` 是指用户所在的房间号，`sdp` 是会话描述协议，是用户端 `RTCPeerConnection` 变量通过 `createOffer` 方法创建的 `offer` 数据中的 `sdp` 属性值，包含用户端的音视频信息，该信息获取后，用于 `RTCPeerConnection` 实现端到端的连接。

表 5.4 answer 类型

数据字段	数据类型	长度	说明
from	String	20	发送 answer 的用户
to	String	20	接收 answer 的用户
room	Array	10	房间号
sdp	String	不限	会话描述协议信息

表 5.4 所示的是 `answer` 类型的信令消息，当用户接收到 `offer` 数据之后，该类型信息将用户创建的 `answer` 数据返回到发送 `offer` 数据的用户端，用于实现端到端的连接。`from` 是设置 `answer` 数据的用户的标识，`to` 是返回 `answer` 数据到某个用户的

标识，room 是指用户所在房间号，sdp 是会话描述协议，是 createAnswer 方法创建的 answer 数据中的 sdp 属性值。

在用户双方建立媒体流之前，必须在二者之间进行协商会话。offer 数据和 answer 数据的交换，可以确保双方都知道要发送和接受的媒体类型以及如何正确解码和处理该媒体。

表 5.5 candidate 类型

数据字段	数据类型	长度	说明
from	String	20	发送 candidate 用户
to	String	20	接收 candidate 用户
sdpMid	String	10	描述协议的 id
sdpMLineIndex	Integer	10	描述协议的行索引
sdp	String	不限	会话描述协议信息
config	String	不限	STUN/TURN 地址

表 5.5 所示的是 candidate 类型的信令消息。该信令是在建立对等连接之前，为获取到对方 candidate 数据，而定义的 candidate 类型的信令消息，作用是保存对方的网络信息。from 是发送 candidate 数据的用户的标识，to 是接收 candidate 数据的用户的标识，sdpMid 和 sdpMLineIndex 是 RTCIceCandidate 的属性，sdpMid 是用于唯一标识候选者从中提取数据的源媒体组件，如果候选者不存在此类关联，则为 null，sdpMLineIndex 是包含提供媒体描述的 m 行集合中的基于 0 号索引的数字，指示哪个媒体源与候选者相关联，sdp 是会话描述协议，config 是在 RTCPeerConnection 变量创建时，通过 addIceCandidate 方法将获取到的 candidate 数据中需要关联到 RTCPeerConnection 变量上的 NAT 服务器信息。

表 5.6 message 类型

数据字段	数据类型	长度	说明
from	String	20	发送 message 用户
to	String	20	接受 message 用户
room	String	20	房间号
peers	Array	10	房间内的用户
message	String	100	消息内容
nickname	String	20	from 用户昵称

表 5.6 所示的是 message 类型的信令消息，该信令是为房间内即时通信而设计。from 是发送 message 的用户的标识，to 是接收 message 的用户的标识，room 是用户

所在房间号,peers 是当前房间内其他用户的标识,message 是用户发送的文本信息,nikename 是 from 用户的昵称。

表 5.7 joined 类型

数据字段	数据类型	长度	说明
from	String	20	加入用户的标识
room	String	20	房间号

表 5.7 所示的是 joined 类型的信令消息,当有用户加入时,服务器端会发送 joined 类型的消息到其他客户端,告知有新的客户端加入。form 表示加入用户的 socket 标识,room 表示加入的房间号。

表 5.8 exit 类型

数据字段	数据类型	长度	说明
from	String	20	离开用户的标识
room	String	20	房间号

表 5.8 所示的是 exit 类型的信令消息,当有用户离开时,服务器端会发送 exit 类型的信令消息到其他客户端,告知有客户端退出。form 表示要离开房间用户的 socket 标识,room 表示房间号。

5.2.3 客户端模块设计

从图 5.1 可知,客户端主要有会议房间进入界面、会议房间主界面、WebRTC 音视频模块、屏幕共享模块。会议房间进入界面需要用户通过输入房间号,点击进入房间,跳转到会议房间主界面。会议房间主界面用来进行线上会议,用户视频窗口布局使用魔方矩阵样式,并将功能栏置于屏幕最下方,所有功能按键自左向右按顺序排列。

从用户交互角度来设计,用户通过浏览器,打开会议房间进入界面,输入房间号,点击进入房间,跳转到会议主页面,在用户允许程序通过浏览器调用音视频设备之后,完成创建房间和进入房间的操作,等待他人进入会议房间,进行音视频会议。从系统程序设计来看,前端页面上的程序,将房间号、SDP 数据、即时消息等数据相继发送到后台服务器,后台服务器接收前端传来的数据后,调用进行一系列

方法，如创建房间和记录当前进入房间的用户信息等。同时，通过 WebRTC 的 API 保证媒体流的建立和传输，来完成用户的操作。

WebRTC 音视频模块主要是由 WebRTC API 实现，主要是通过 `getUserMedia` 获取到用户音视频设备的权限，从而获取音视频媒体流。通过会话管理机制，获取到对等端的媒体和网络信息，利用 `RTCPeerConnection` 建立用户之间视频流的传输通道。

另外，根据项目需求，本系统设计扩展插件，来实现屏幕共享的功能，用户可以更好地体验线上会议所带来的方便性。屏幕共享模块目的是实现共享用户浏览器网页、桌面和应用的屏幕，而屏幕媒体流是通过前端页面的内容脚本与插件的背景脚本进行通信，调用 chrome 内置 api 来获取的。

#### 5.2.4 服务端架构设计

由 5.2.1 节可知，本系统服务器架构主要包括信令模块、NAT 模块、房间模块、即时消息模块、Web 服务器模块。信令模块主要负责协商媒体功能和设置，标识和验证会话参与者的身份。NAT 模块主要使用 STUN/TURN 服务器为用户之间实现 NAT 穿越，从而保证用户浏览器之间完成端到端的流传输。房间模块主要用于记录和标识用户所进入的房间号，存储当前房间号内所有的用户标识。即时消息模块主要用于用户之间广播发送即时消息。Web 服务器主要负责用户通过浏览器获取页面以及系统数据的传输。

图 5.3 所示的是整体架构设计图，体现出服务器端所涉及的功能模块。本系统设计 Web 服务器使用的是 `node.js`，以及使用 `Express` 应用开发框架。利用 `socket.io` 框架，设计会话管理系统，实现本系统的信令模块。利用 `socket.io` 框架的内置对象在逻辑上形成房间概念，从而实现房间模块。结合会话管理机制和房间模块，完成会议系统中即时消息模块。

另外，NAT 模块是负责网络穿越的，本系统使用的 NAT 穿越服务器是自行部署的 Coturn 开源服务器。另外，使用 Google 免费的 STUN/TURN 服务器地址作为备用地址，以防止 Coturn 服务器穿越失败后无法建立连接。

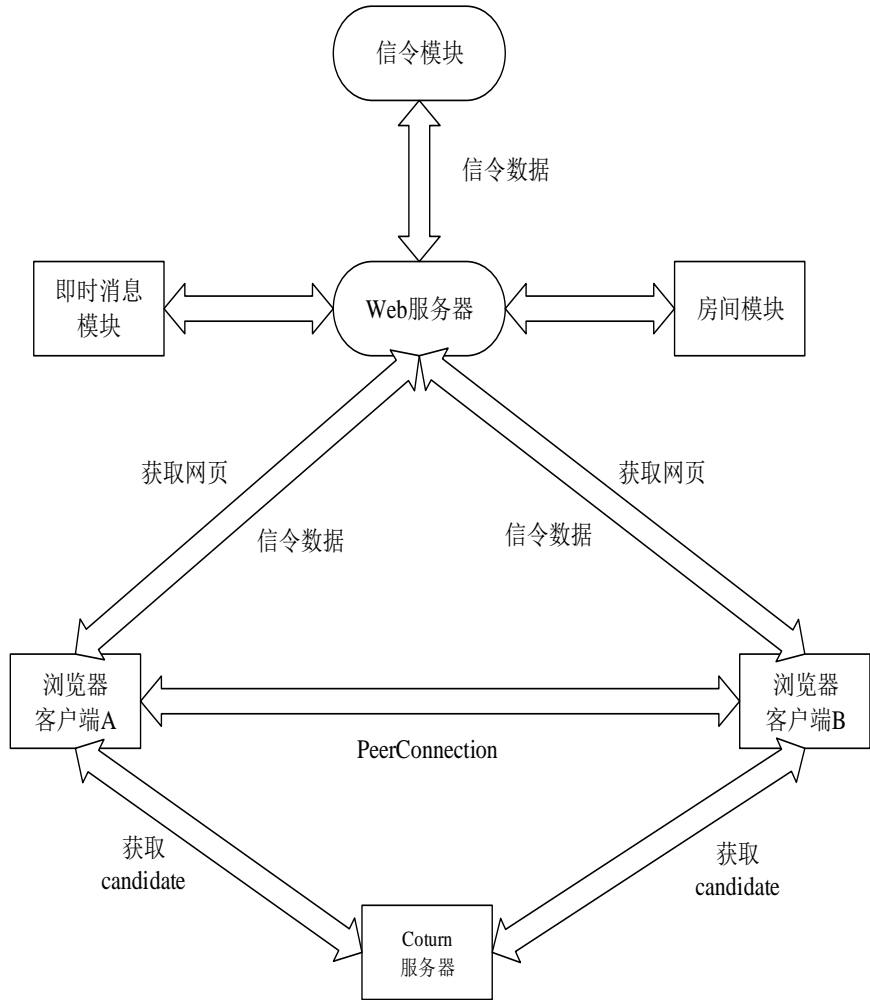


图 5.3 架构图

### 5.3 系统关键技术实现

本节将详细描述系统的关键技术实现，涵盖了客户端的 WebRTC 音视频模块和屏幕共享插件的实现，以及服务器端信令模块、房间模块、NAT 模块和即时通信模块的实现。将通过时序图，流程图等方式直观地展示出整个实现过程。主要技术包括前端页面的 HTML5、JavaScript、CSS 和服务端端的 socket.io、node.js、Express。NAT 模块使用开源服务器 Coturn，通过自行下载源码包进行编译、安装和部署。系统使用的 openssl 生成自签名的免费证书。服务器使用的是阿里云的公有云服务器，公网 IP 为 123.57.243.85。

### 5.3.1 会话管理机制

本系统通过会话管理机制以管理用户进入和退出会议房间，以及对必要数据进行传输。会话管理机制需要利用信令在浏览器中建立并管理多个连接，使用信令在需要建立连接的两个浏览器端之间进行交换媒体信息和候选地址，这两个信息是至关重要的，通过会话描述协议(Session Description Protocol, SDP)来存储，并决定着用户之间是否连接成功。

WebRTC 的 API 一方面提供 `RTCPeerConnection.createOffer` 接口供 web 应用程序调用并生成相关 SDP 信息，另一方面也提供了包含候选地址属性的 ICE 功能接口，如 `RTCIceCandidate` 接口，而信令协议和信令通道则可以使用基于 `WebSocket` 的 `socket.io`。

```
v=0
o=- 7863037465416800258 2 IN IP4 127.0.0.1
s=-
t=0 0
a=group:BUNDLE audio video
a=msid-semantic: WMS YxkPr2GPQIG29BEg5KIDHWlqrg1QAbVFufpz

m=audio 9 UDP/TLS/RTP/SAVPF 111 103 104 9 0 8 106 105 13 110 112 113 126
c=IN IP4 0.0.0.0
a=rtcp:9 IN IP4 0.0.0.0
a=ice-ufrag:y0og
a=ice-pwd:/4qkiGvANuTZAjTlWeDj/rQW
a=ice-options:trickle
a=fingerprint:sha-256 63:37:84:A1:C6:2D:D6:7D:1A:05:D5:C1:4C:6C:23:41:15:8F:AA:05:81:D7:25:96:EF:2D:F8:33:93:25:BB:A7
a=setup:active
a=mid:audio
a=sendrecv
a=rtpmap:111 opus/48000/2
a=rtpmap:103 ISAC/16000s
a=rtpmap:104 ISAC/32000
a=rtpmap:9 G722/8000
a=ssrc:3975027615 cname:cGi8sdnc3hOAY1Ds
a=ssrc:3975027615 msid:YxkPr2GPQIG29BEg5KIDHWlqrg1QAbVFufpz e8f9bf61-1a96-483a-9ee5-6fb5ade0b89e

m=video 9 UDP/TLS/RTP/SAVPF 96 97 98 99 100 101 102 123 127 122 125 107 108 109 124
c=IN IP4 0.0.0.0
a=rtcp:9 IN IP4 0.0.0.0\r\na=ice-ufrag:y0og
a=ice-pwd:/4qkiGvANuTZAjTlWeDj/rQW
a=ice-options:trickle
a=fingerprint:sha-256 63:37:84:A1:C6:2D:D6:7D:1A:05:D5:C1:4C:6C:23:41:15:8F:AA:05:81:D7:25:96:EF:2D:F8:33:93:25:BB:A7
a=setup:active
a=mid:video
a=sendrecv
a=rtcp-mux
a=rtpmap:98 VP9/90000
a=rtpmap:100 H264/90000
a=ssrc:1073447924 cname:cGi8sdnc3hOAY1Ds
a=ssrc:1073447924 msid:YxkPr2GPQIG29BEg5KIDHWlqrg1QAbVFufpz 2461553a-ea77-4c23-bf1b-0373cbeaf13b
```

图 5.4 SDP 格式图

通过浏览器的 WebRTC 接口 `createOffer` 会获得本地所支持的音视频格式，以及传输相关参数信息，如 RTP 包头的扩展选项，并将所有与音频、视频和传输相关的

描述信息添加到请求端的会话描述 SDP 中。如图 5.4 所示是一个 SDP 的示例,限于篇幅,只展示了主要的一部分。由图中展示可知,SDP 是由多部分信息组成,每部分信息占一行。其中 v 表示版本, sdp 版本号一直为 0,是由 rfc4566 规定的; o 表示会话源; s 表示会话名,没有会话时可用-代替; t 表示会话的起始时间和结束时间; a 表示会话属性; m 表示媒体协议信息。通过 createAnswer 接口获取 SDP 过程同 createOffer 一样。

另外还需要注意设置本地和添加对端的 candidate 数据、sdpMid 属性和 sdpMLineIndex 属性。本地程序通过 RTCPeerConnection.onicecandidate 触发 icecandidate 事件,该事件将 candidate 数据用 SDP 存储,如表 5.5 中 candidate 类型的 sdp 字段。远端程序通过 RTCIceCandidate 接口,创建 rtcIceCandidate 变量,并通过 RTCPeerConnection.addIceCandidate 方法将变量绑定到 RTCPeerConnection 上。

利用上述 SDP 和候选地址数据,设计本会议系统的会话管理机制,即信令交互过程,通过将 SDP 添加到 5.2.2 节所对应的数据格式中,完成会话管理的任务。

如图 5.5 是本会议系统实现的会话管理机制时序图,使用 JS 事件回调函数机制,根据不同类型的信令信息触发对应的事件,并返回相应的数据。图中箭头线上的名称,是会话管理数据交互过程中信令消息的类型名称,而信令消息所包含的消息字段,已在 5.2.2 节设计。以用户 A 和用户 B 通信为例,会话管理的具体流程如下:

步骤 1: 当用户 B 第一个打开浏览器输入房间号,进入会议房间时,向服务器端发送 createOrJoinRoom 类型的信令消息。服务器收到该类型的信令消息,通过房间模块,创建房间,返回 created 类型的信令消息到浏览器端。

步骤 2: 当用户 A 进入用户 B 创建的房间时,同样会向服务器端发送 createOrJoinRoom 类型的信令消息,房间模块检测到该房间已经创建,会发送 joined 类型的信令消息给用户 B,通知有用户 A 加入房间,同时返回 created 信令消息给用户 A 告知房间已经创建,房间内有用户 B。

步骤 3: 用户 A 根据 WebRTC 的 offer/answer 机制,创建 offer 类型的信令消息,发送给服务器,服务器转发给用户 B。用户 B 收到 offer 类型的信令消息,创建 answer 信令消息,经服务器端转发给用户 A。

步骤 4: 经过步骤 3,用户 A 和用户 B 都将 icecandidate 数据通过回调函数存储到 candidate 类型的信令消息中,并相互发送,来完成网络信息的交换。

步骤 5: 当用户之间完成数据交互之后, 就可以直接建立端到端连接, 进行音视频通信。连接建立期间, 如果用户 A 在房间内发送文本消息, 会向服务器发送 message 类型的信令消息, 经服务器接收处理, 并广播给房间内的所有用户。

步骤 6: 当用户 A 退出, 会向服务器发送 exit 类型的信令消息, 房间模块将用户 A 的信息去除, 再转发 exit 信令消息给房间内所有用户, 通知用户 A 已退出。

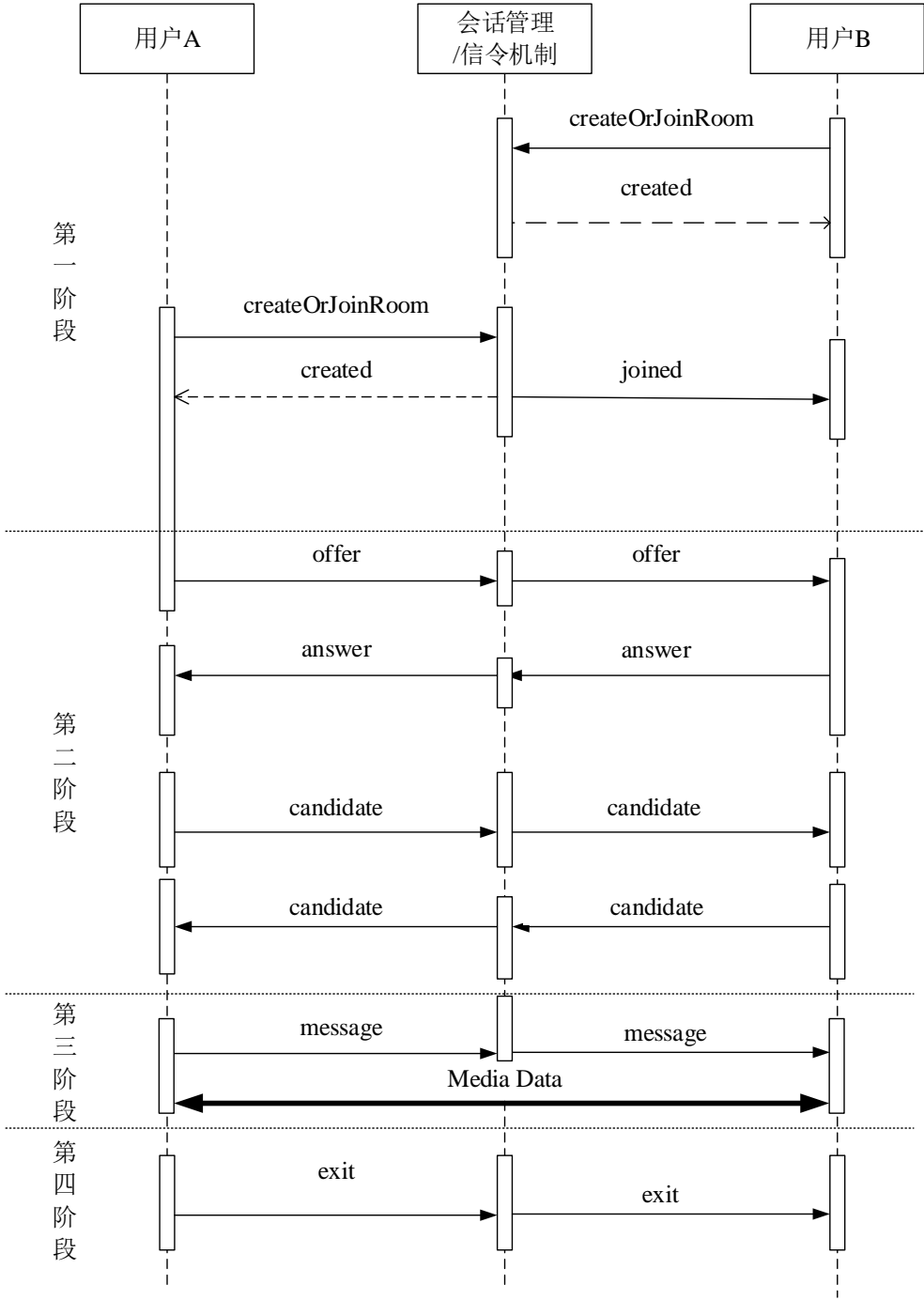


图 5.5 会话管理机制时序图



总的来说，本系统的会话管理机制可以概括为四个阶段。第一个阶段是初始化阶段，这一阶段用户之间通过信令服务器根据房间号，创建房间，管理房间内用户之间的会话，包括步骤 1 和步骤 2。第二个阶段是 offer/answer 和 candidate 数据交互阶段，包括步骤 2、步骤 3 和步骤 4。第三个阶段是通信阶段，可以进行音视频通信，以及基于房间的即时消息通信，如步骤 5。第四个阶段是用户退出阶段，如步骤 6。

本会议系统的会话管理机制，可以有效地管理用户创建房间、进入房间、退出房间和在房间内发送即时消息。图 5.6 和图 5.7 所示的是程序运行时，服务器端 console 输出，显示的是服务器端接收和转发的信令消息。

```
123.57.243.85
Received createAndJoinRoom: 444
Room 444 now has 0 client(s)
Client ID u25honB1P0afdZCbAAAA created room 444
Received createAndJoinRoom: 444
Room 444 now has 1 client(s)
[ 'u25honB1P0afdZCbAAAA' ]
Socket length 1
{ '444': Room { sockets: { u25honB1P0afdZCbAAAA: true }, length: 1 },
  u25honB1P0afdZCbAAAA: Room { sockets: { u25honB1P0afdZCbAAAA: true }, length: 1 },
  'o0pydoh7Xw-xrM-AAAAAB': Room { sockets: { 'o0pydoh7Xw-xrM-AAAAAB': true }, length: 1 } }
Client ID o0pydoh7Xw-xrM-AAAAAB joined room 444
Received offer: o0pydoh7Xw-xrM-AAAAAB room:444 message: {"from":"o0pydoh7Xw-xrM-AAAAAB","to":"u25honB1P0afdZCbAAAA","room":"444","sdp":{"v=0\r\no=- 8911269640350891244 2 IN IP4 127.0.0.1\r\ns=-\r\n
```

图 5.6 服务器端 console 图

```
123.57.243.85
Received candidate: o0pydoh7Xw-xrM-AAAAAB message: {"from":"o0pydoh7Xw-xrM-AAAAAB","to":"u25honB1P0afdZCbAAAA","room":"444","candidate":{"sdpMid":"video","sdpMLineIndex":1,"sdp":"candidate:1289607982 1 tcp 1518149375 10.21.13.80 9 typ host tcptype active generation 0 ufrag xv7U network-id 3"}}
Received answer: u25honB1P0afdZCbAAAA room:444 message: {"from":"u25honB1P0afdZCbAAAA","to":"o0pydoh7Xw-xrM-AAAAAB","room":"444","sdp":{"v=0\r\no=- 1948374508249107904 2 IN IP4 127.0.0.1\r\ns=-\r\n
```

图 5.7 服务器端 console 图

5.3.2 WebRTC 音视频交互

WebRTC 音视频交互主要是实现浏览器之间的音视频通信，通过 5.3.1 节的会话管理机制可知，整个通信过程中具体涉及到 WebRTC 的 offer/answer 协商机制以及通过 STUN/TURN 服务器获取 ice 候选地址信息。根据会话管理机制的设计完成会话协商，具体使用 WebRTC 接口实现浏览器用户端到端的音视频交互。

图 5.8 所示的是实现 WebRTC 音视频交互的过程，以及与其他服务器端模块的通信过程。

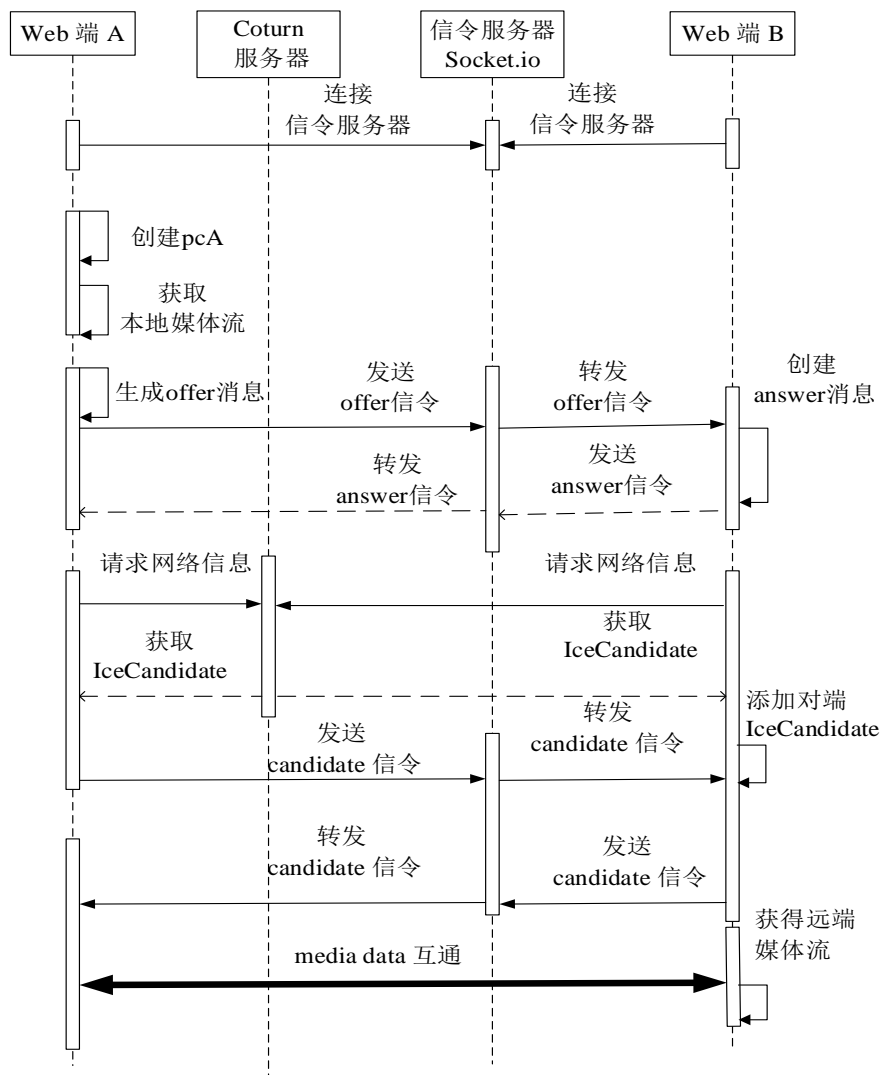


图 5.8 WebRTC 音视频交互时序图

以 Web 端 A 与 Web 端 B 通信为例，具体过程如下：

步骤 1：Web 端 A 和 Web 端 B 首先都会连接信令服务器。此时 Web 端 B 先于 Web 端 A 连接服务器。后连接的 Web 端 A 创建 offer，主动与 Web 端 B 进行连接。

步骤 2：首先 Web 端 A 需要创建变量 pcA，通过 API 方法 `RTCPeerConnection()`，使用参数 `iceServer`，即 Coturn 服务器地址信息。其次，使用 `addTrack` 方法，将 `getUserMedia` 获得的媒体流与 pcA 关联，并设置远端媒体流 stream 数据回调。再次，使用 `createOffer()` 设置 offer 类型的信令消息，并发送到信令服务器，经信令服务器转发到 Web 端 B。

步骤 3：Web 端 B 接收到 offer 数据，同样使用或创建与 Web 端 A 对应的 pcB 变量，设置本地媒体流绑定，设置远端媒体流 stream 数据回调。然后设置远端会话

描述 `setRemoteDescription`，同时使用 `createAnswer` 方法，创建 `answer` 类型的信令消息，发送给信令服务器，由信令服务器转发给 Web 端 A。

步骤 4：当 Web 端 A 收到 `answer` 数据之后，使用 `RTCSessionDescription` 接口定义变量 `rtcDescription`，传入 `type` 值为 `answer` 和 `sdp` 值为 `answer.sdp` 的参数。将 `rtcDescription` 传入 `pcA` 变量，来设置远端会话描述 `setRemoteDescription`。

步骤 5：Web 端 A 和 Web 端 B 会与 Coturn 服务器连接，请求并获取 ICE 候选地址，通过信令服务器转发给彼此，双方根据收到的 `candidate` 数据，通过 `RTCIceCandidate` 方法创建变量 `rtcIceCandidate`，变量 `pcA` 和 `pcB` 通过 `addIceCandidate` 方法，使用变量 `rtcIceCandidate` 作为传入参数，使其关联在一起。

步骤 6：当 Web 端 A 和 Web 端 B 对应的变量 `pcA` 和 `pcB` 之间，`offer/answer` 机制完成，并获取到对方的 `candidate` 数据，此时 Web 端 A 和 Web 端 B 可以实现端到端的连接，彼此的媒体 `stream` 也会到达对方浏览器中，通过 `video` 标签在相应的布局中显示出来。

由于本会议系统是网状拓扑架构，所以在会话机制的控制下完成，每两个浏览器之间都会经过上述过程，进行端对端的连接。如图 5.9、图 5.10 和图 5.11 所示的是浏览器端 `console` 输出图。图 5.9 所示是用户标识 `EQP-h7A8LkINNptgAAAA` 接收到的 `created` 信令消息。图 5.10 是用户标识为 `yuX83ODLMRqLDf1AAAB` 的接收的 `created` 和发送的 `offer` 信令信息，图 5.11 所示的是接收的 `answer` 信令信息。

```
Received local stream
created: {"id":"EQP-h7A8LkINNptgAAAA","room":"444","peers":[],"nickname":"me"}
joined: {"id":"yuX83ODLMRqLDf1AAAB","room":"444"}
offer: {"from":"yuX83ODLMRqLDf1AAAB","to":"EQP-h7A8LkINNptgAAAA","room":"444","sdp":"v=0\r\no=-
2048470369825461337 2 IN IP4 127.0.0.1\r\ns=-\r\nnt=0 0\r\na=group:BUNDLE audio video\r\na=msid-semantic: WMS
kVTVQ-7u4Qs7sk6V4xj1JzDshasRkwn499a\r\nm=audio 9 UDP/TLS/RTP/SAVPF 111 103 104 9 0 8 106 105 13 110 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200\r\na=ice-ufrag:6VJb\r\na=ice-pwd:5qK2ekqSYb824skZOTJMN2J7\r\na=ice-
options:trickle\r\na=fingerprint:sha-256
01:68:93:87:9D:4E:09:8D:F7:96:E9:FA:A9:A0:56:D3:EF:2B:A2:64:08:48:EE:16:EA:CA:DF:7D:38:68:37:2F\r\na=setup:actpa
o\r\na=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level\r\na=sendrecv\r\na=rtcp-mux\r\na=rtpmap:111 opus/480
fb:111 transport-cc\r\na=fmtp:111 minptime=10;useinbandfec=1\r\na=rtpmap:103 ISAC/16000\r\na=rtpmap:104 ISAC/320
G722/8000\r\na=rtpmap:0 PCMU/8000\r\na=rtpmap:8 PCMA/8000\r\na=rtpmap:106 CN/32000\r\na=rtpmap:105 CN/16000\r\na
CN/8000\r\na=rtpmap:110 telephone-event/48000\r\na=rtpmap:112 telephone-event/32000\r\na=rtpmap:113 telephone-
```

图 5.9 浏览器 B console 图

```
Received local stream
created: {"id":"yuX83ODLMRqLDf1AAAB","room":"444","peers":[{"id":"EQP-h7A8LkINNptgAAAA"}],"nickname":"me"}
createOffer: success id:EQP-h7A8LkINNptgAAAA offer {"type":"offer","sdp":"v=0\r\no=- 2048470369825461337 2 IN
IP4 127.0.0.1\r\ns=-\r\nnt=0 0\r\na=group:BUNDLE audio video\r\na=msid-semantic: WMS
hvIX0aZy4pQqJrh6V4xj1JzDshasRkwn499a\r\nm=audio 9 UDP/TLS/RTP/SAVPF 111 103 104 9 0 8 106 105 13 110 112 113 126
0.0.0.0\r\na=rtcp:9 IN IP4 0.0.0.0\r\na=ice-ufrag:6VJb\r\na=ice-pwd:5qK2ekqSYb824skZOTJMN2J7\r\na=ice-
options:trickle\r\na=fingerprint:sha-256
01:68:93:87:9D:4E:09:8D:F7:96:E9:FA:A9:A0:56:D3:EF:2B:A2:64:08:48:EE:16:EA:CA:DF:7D:38:68:37:2F\r\na=setup:actpa
o\r\na=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level\r\na=sendrecv\r\na=rtcp-mux\r\na=rtpmap:111 opus/480
fb:111 transport-cc\r\na=fmtp:111 minptime=10;useinbandfec=1\r\na=rtpmap:103 ISAC/16000\r\na=rtpmap:104 ISAC/320
G722/8000\r\na=rtpmap:0 PCMU/8000\r\na=rtpmap:8 PCMA/8000\r\na=rtpmap:106 CN/32000\r\na=rtpmap:105 CN/16000\r\na
CN/8000\r\na=rtpmap:110 telephone-event/48000\r\na=rtpmap:112 telephone-event/32000\r\na=rtpmap:113 telephone-
```

图 5.10 浏览器 A console 图

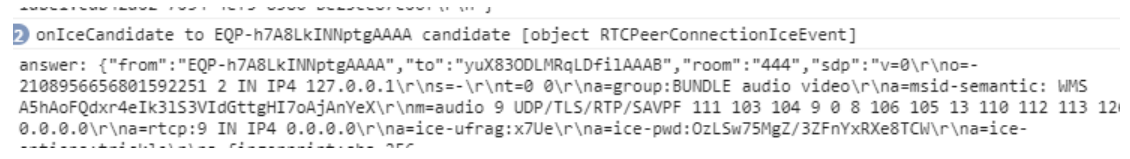


图 5.11 浏览器 A console 图

### 5.3.3 共享屏幕功能

本节主要讲述系统共享屏幕功能的实现。因为 chrome 浏览器允许网页程序捕获屏幕媒体流,但只能通过扩展插件实现,所以根据 chrome 提供了扩展程序的 API,来实现网页程序与插件脚本的通信,并完成此功能。

如图 5.12 所示的是屏幕共享扩展插件实现流程图。在 chrome 中,扩展插件的主要部分包括内容脚本、背景脚本(background.js)和 manifest.json。其中内容脚本运行在 Web 页面的上下文中,属于 Web 程序的组成部分。背景脚本是负责与内容脚本交互数据,返回页面程序所需要的数据。manifest.json 是每个扩展插件中必须存在的唯一文件,包含基本元数据,极其重要。

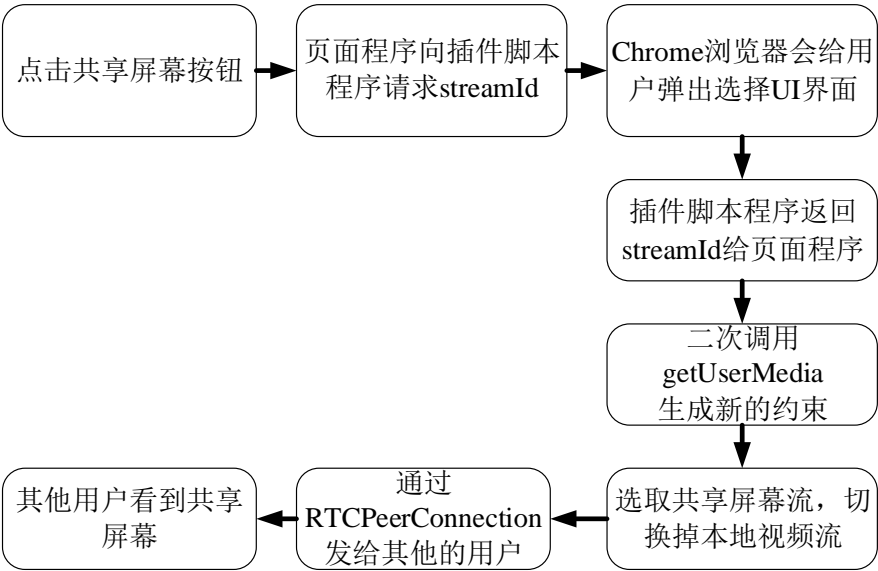
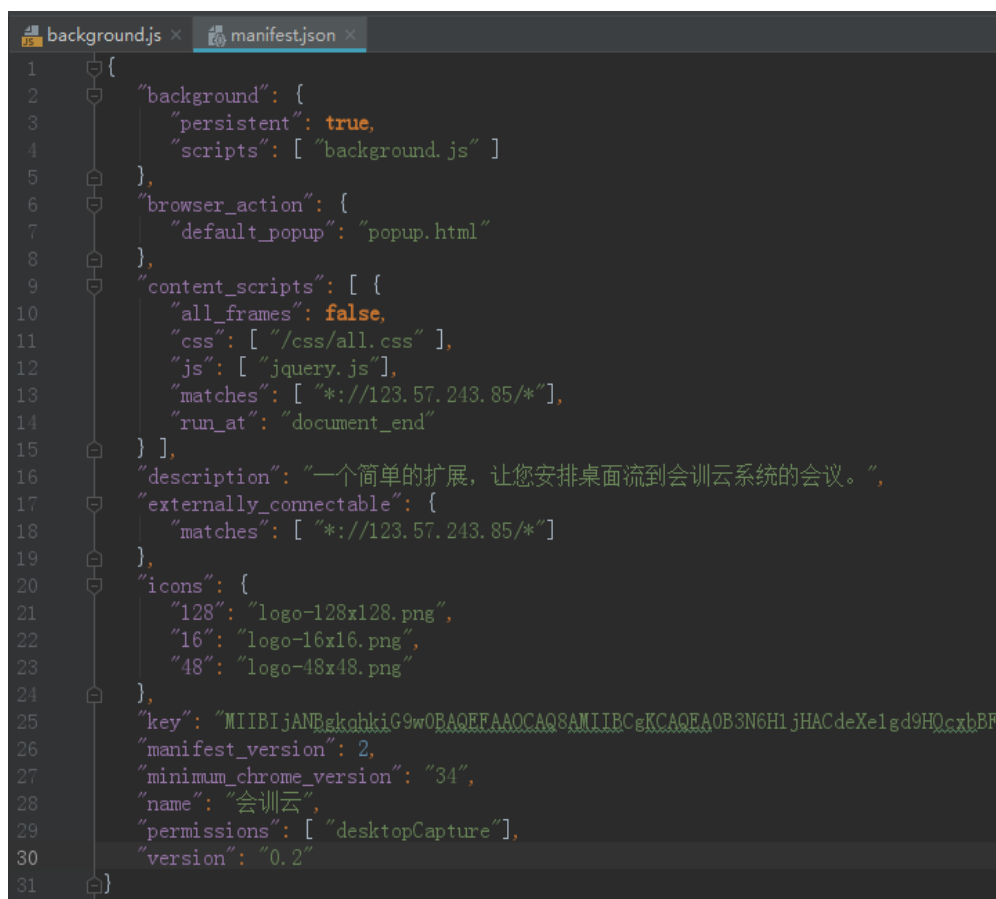


图 5.12 屏幕共享扩展插件实现流程图

如图 5.13 所示,实现扩展插件的之前,需要在 manifest.json 中配置关键的属性值 background 指定脚本程序, content\_scripts 的子属性 matches 指定匹配地址。



```
1 {
2   "background": {
3     "persistent": true,
4     "scripts": [ "background.js" ]
5   },
6   "browser_action": {
7     "default_popup": "popup.html"
8   },
9   "content_scripts": [ {
10     "all_frames": false,
11     "css": [ "/css/all.css" ],
12     "js": [ "jquery.js" ],
13     "matches": [ "*/123.57.243.85/*" ],
14     "run_at": "document_end"
15   } ],
16   "description": "一个简单的扩展，让您安排桌面流到会训云系统的会议。",
17   "externally_connectable": {
18     "matches": [ "*/123.57.243.85/*" ]
19   },
20   "icons": {
21     "128": "logo-128x128.png",
22     "16": "logo-16x16.png",
23     "48": "logo-48x48.png"
24   },
25   "key": "MIIBIjANEgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA0B3N6H1jHACdeXelgd9H0CxbBF",
26   "manifest_version": 2,
27   "minimum_chrome_version": "34",
28   "name": "会训云",
29   "permissions": [ "desktopCapture" ],
30   "version": "0.2"
31 }
```

图 5.13 manifest.json 文件内容图

有了 manifest.json 文件，就可以按照图 5.12 的实现流程图来实现扩展插件，系统程序执行具体过程如下：

步骤 1：首先确认点击屏幕共享功能的按钮，页面程序中内容脚本，则调用 chrome.runtime.sendMessage() 方法，携带参数 extensionId 和 openUrlInEditor，一个是扩展插件的 ID，一个调用该扩展的 url 地址，并使用 chrome.runtime.connect 与扩展插件程序建立通信。

步骤 2：扩展插件经过 manifest.json 中配置的属性 matches 的匹配，允许该 url 调用扩展。同时背景脚本中使用 chrome.runtime.onMessageExternal.addListener() 方法，监听发送消息事件。

步骤 3：扩展插件获得发送消息之后，浏览器弹出共享 UI 界面，有 screen、window、tap 三类共享，供用户选择。

步骤 4：插件生成 streamId 参数，chrome.desktopCapture.chooseDesktopMedia() 方法中的回调方法，再将此 streamId 参数返回到内容脚本，即页面程序。

步骤 5: 页面程序获取到 `streamId`, 将值传给 `getUserMedia()` 方法, 获得新的 `media constraints`, 切换 video 的媒体流。

步骤 6: 通过已有的 `RTCPeerConnection` 接口, 发送给其他的用户, 然后其他用户接收到屏幕数据。

通过上述实现, 共享屏幕的功能已经完成, 如图 5.14 和 5.15 所示, 背景脚本成功获取到 123.57.243.85 网址发来的请求, 网页程序也成功获取到背景脚本返回的 `streamId`, 并生成新的约束。

```
Got request ▶ {getVersion: true} ▼ {url: "https://123.57.243.85/444", tab: {...}, frameId: 0} ⓘ
  frameId: 0
  ▼ tab:
    active: true
    audible: false
    autoDiscardable: true
    discarded: false
    favIconUrl: "https://123.57.243.85/images/favicon.ico?v=1"
    height: 758
    highlighted: true
    id: 344
    incognito: false
    index: 1
    ▶ mutedInfo: {muted: false}
    pinned: false
    selected: true
    status: "loading"
    title: "伏守会训云"
    url: "https://123.57.243.85/444"
    width: 1440
    windowId: 240
    ▶ __proto__: Object
    url: "https://123.57.243.85/444"
    ▶ __proto__: Object
```

图 5.14 背景脚本 console 图

```
Response from extension: ▼ Object ⓘ
  streamId: "FL8dG1DmHLxpPgyOoDTo8A=="
  ▶ __proto__: Object

Get media constraints ▼ Object ⓘ
  audio: false
  ▼ video:
    ▼ mandatory:
      chromeMediaSource: "desktop"
      chromeMediaSourceId: "FL8dG1DmHLxpPgyOoDTo8A=="
      maxFrameRate: 3
      maxHeight: 900
      maxWidth: 1440
      ▶ __proto__: Object
    ▶ optional: []
    ▶ __proto__: Object
    ▶ __proto__: Object
```

图 5.15 浏览器页面 console 图

### 5.3.4 基于会议房间的即时消息

用户通过本系统的起始页面输入房间号，进入会议房间。在会议房间内，可以实现如 5.3.2 节所述的音视频通信，也可以实现房间内文本消息的即时通信。

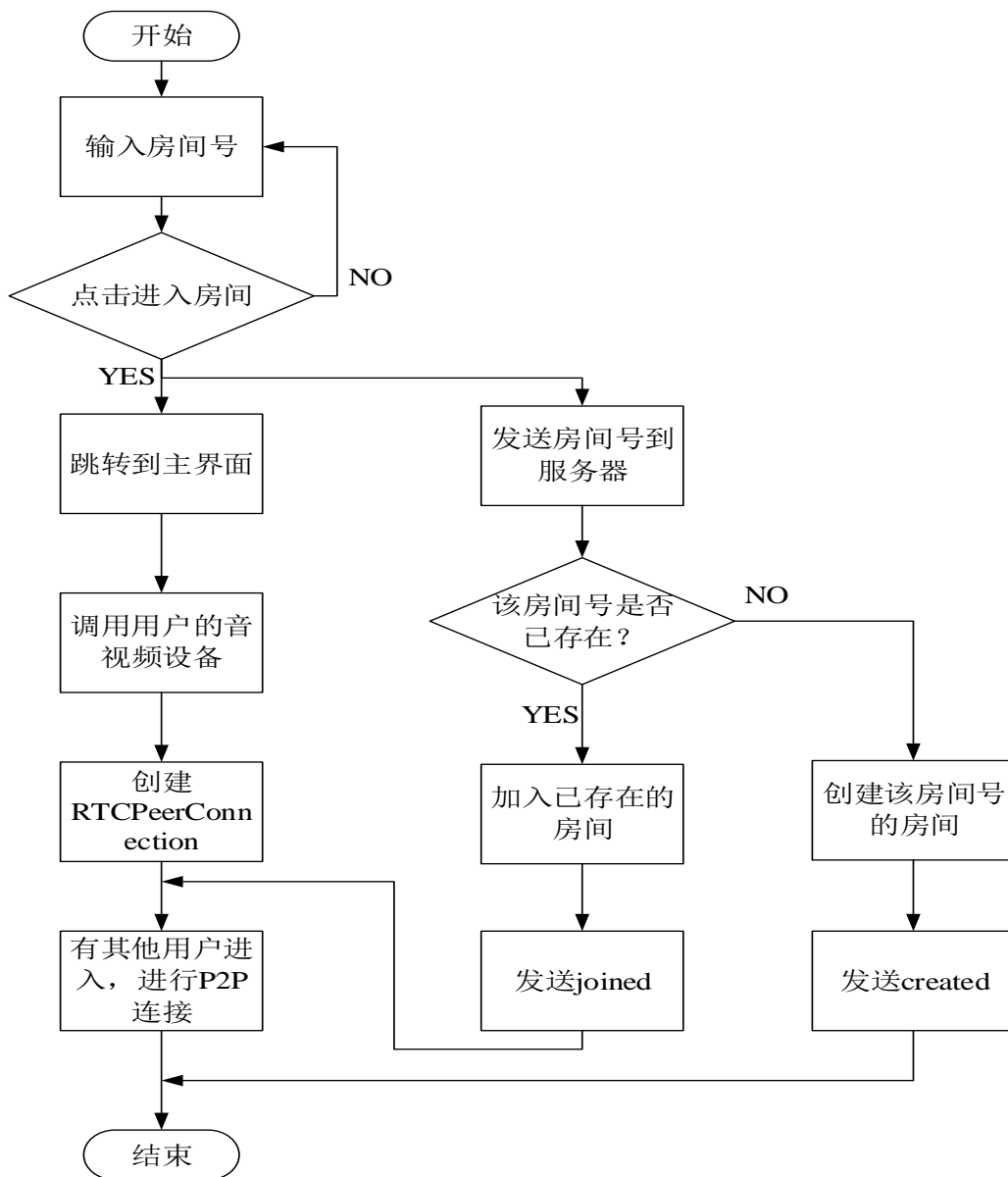


图 5.16 房间创建流程图

图 5.16 所示的是服务器端接收到前端页面发送的数据，服务器程序进入或创建房间的流程。后台服务器收到前端页面传来的数据，首先，判断此 room 号的房间是否存在，如果存在，便加入房间，将用户的 socket 标识存入 peers 字段中，向所有房间内的用户发送 joined 类型的信令消息，通知有新用户进入会议，与其他用户创

建 P2P 连接。如果不存在，则服务器端房间模块创建 room 房间，用 peers 字段存储进入房间的用户，将 created 类型的信令消息返回前端。客户端程序主要是调用音视频设备，创建 RTCPeerConnection 变量，根据服务器端返回的信息，循环访问 peers 字段的 socket 标识，使用 RTCPeerConnection 与房间内每个用户建立端对端的连接。

房间模块的 rooms 对象和 room 对象是 socket.io 内置对象。socket.io 框架中有一个 adapter.rooms 对象，其格式是 JSON 格式，属性名为 id，属性值为 room 对象。而 room 对象同样是 JSON 格式，有 sockets 和 length 两个属性，前者 sockets 属性存储的是 JSON 数据，属性名为 socketId，属性值为布尔值，而后者 length 属性存储的是数值。利用 rooms 对象中第一个变量所包含的内容，逻辑上可以实现为一个房间的概念，构建房间模块。

```
{
  "room_id" : Room {
    sockets : {"socket_id_1" : true , "socket_id_2": true },
    length : 2 },
  "socket_id_1" : Room {
    sockets : {"socket_id_1" : true },
    length : 1 },
  "socket_id_2" : Room {
    sockets : {"socket_id_2" : true },
    length : 1}
}
```

图 5.17 rooms 对象 json 格式图

如图 5.17 所示的是 rooms 对象的 json 格式图，表示房间号为 room\_id 的房间内有两个用户，用户标识分别为 socket\_id\_1 和 socket\_id\_2。

接着，在房间内实现文本消息的即时发送。如图 5.18 所示，用户在客户端页面上，输入文本消息，点击发送后，将包含文本消息的 message 类型的信令消息发送到服务器的信令模块。服务器接收到 message 类型的消息之后，首先根据 peers 字段，判断房间内是否用户数量为 0，如果不为 0，则利用房间模块的 socket.io 的 in 函数，在指定房间内进行广播 message 类型的信令消息，其他用户的页面程序接到该信令消息，读取 message 字段并显示文本消息内容。如果为 0，则服务器端不做任何操作，用户只在客户端页面上显示自己发送的文本消息。具体流程如图 5.18 所示。



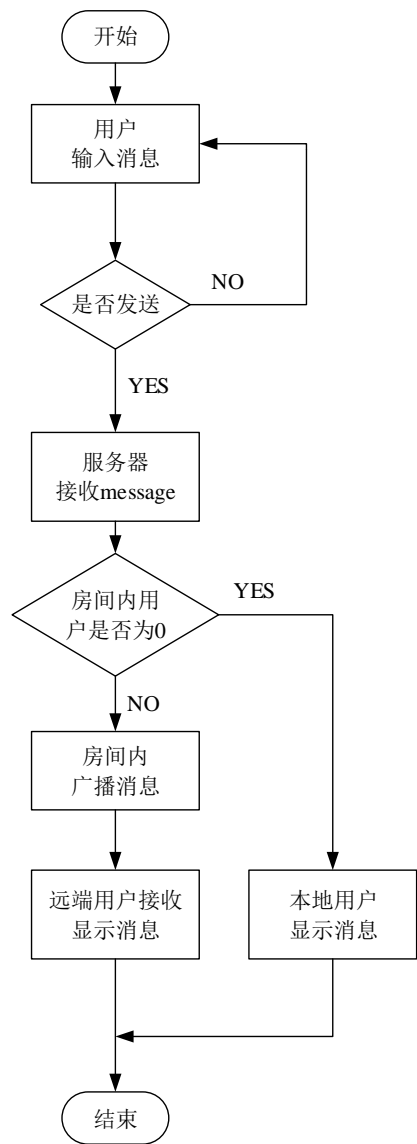


图 5.18 即时通信模块流程图

5.4 系统测试与分析

本节主要是对系统基本功能的测试，以及对系统性能的分析。本系统部署在阿里云服务器 123.57.243.85，实验室使用系统测试用机如表 5.9 所示，浏览器使用 chrome 72.0.3626.119 版本。

表 5.9 系统测试用机

	Dell PC	ASUS PC	技嘉台式 1	技嘉台式 2
系统	Win7	Win10	Ubuntu14	Win7
CPU	Intel i5	Intel i3	PentiumG2020	PentiumG2020
内存	6G	4G	4G	8G

5.4.1 系统基本功能测试

1. 会议房间进入界面

当用户打开浏览器，输入特定 URL，进入会议房间进入页面，用户可在输入框内输入地址和房间号后，点击 GO 按钮，便跳转到主页面。

如图 5.19 所示，房间进入页面主要包括背景图片部分、系统 LOGO 标签部分、房间号输入框和进入房间的 GO 按钮。

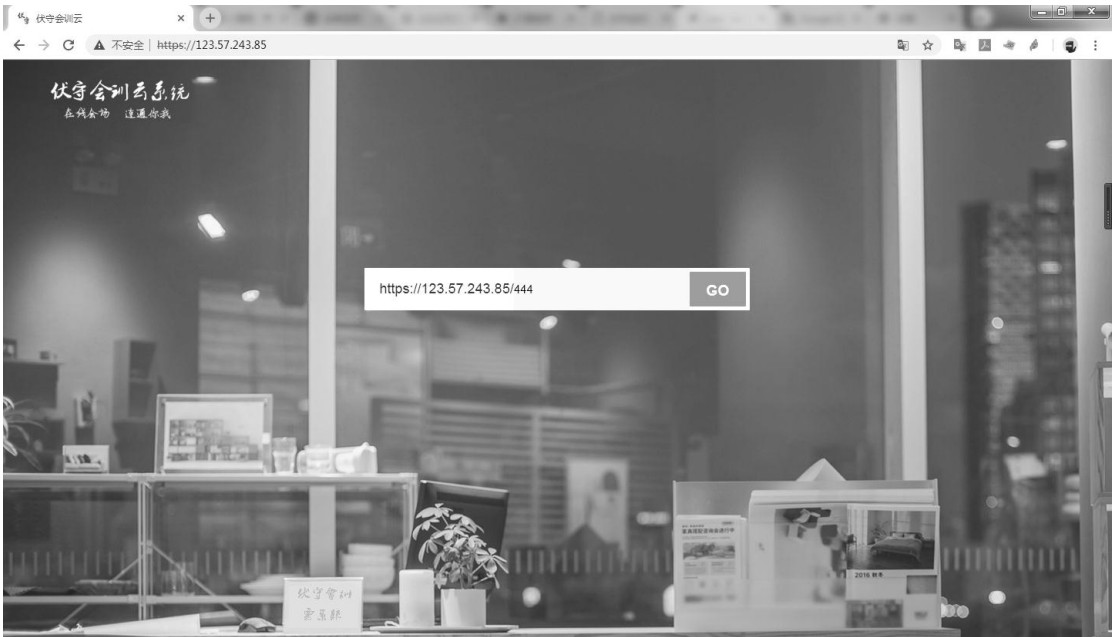


图 5.19 房间进入页面图

2. 会议房间主界面

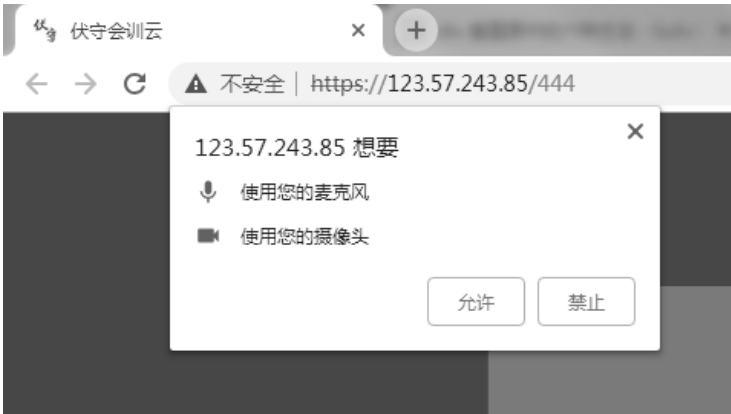


图 5.20 获取音视频设备权限图

当用户输入房间号之后,会跳转到会议房间主页面。如图 5.20 所示。点击进入按钮之后,跳转到主界面的第一步,需要请求用户允许调用终端音视频设备的权限,当用户点击允许之后,用户进入会议房间主界面后,该系统才能进行正常的音视频通信,否则只能接收对方的音视频。

当用户允许系统调用 PC 的音视频设备之后,第一个用户进入会议房间的主界面如图 5.21 所示。

会议房间主界面主要由视频窗口和功能栏两部分组成。功能栏分布在界面正下方,主要有共享屏幕按钮、即时消息窗口按钮、退出房间按钮、音频和视频开关按钮。

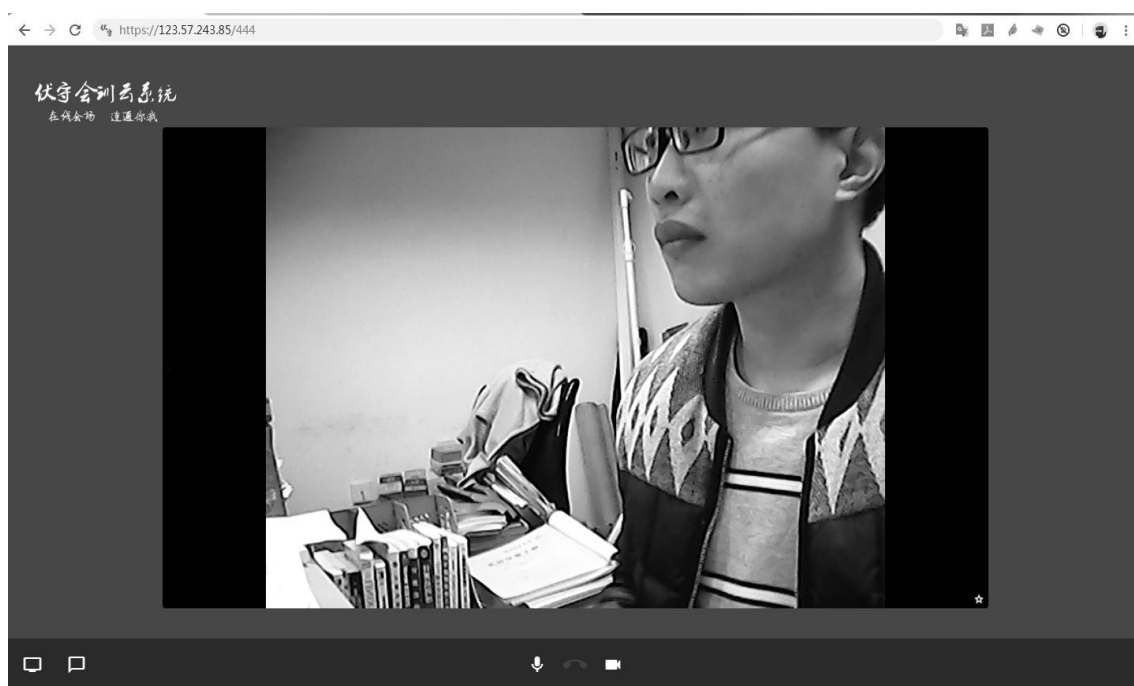


图 5.21 会议房间主界面图

当前,流行的会议系统主界面通常有两种样式:hangouts 和 magic matrix,本系统采用魔方矩阵样式。由于采用魔方矩阵样式,后续加入的用户窗口会以矩阵方块的形式,均匀分布在会议房间内,默认最后一个窗口是本地用户窗口,其余用户窗口随着远端用户接入的顺序进行排列。如图 5.22 所示,多人同时参会的界面。由于本系统限制 8 人以内参会,人数最多时会以 3\*3 矩阵居中对齐显示,但第九个用户会因为房间已满被拒绝进入。



图 5.22 多人参会界面

3. 屏幕共享

根据实际需求，当参加会议时，需要进行文本资料展示，本系统通过插件的形式，实现屏幕共享的功能。接下来，测试屏幕共享功能，点击功能栏中第一个 PC 状的按钮，浏览器会跳出选择标签页面，如图 5.23 所示。



图 5.23 共享选择界面

如图 5.24 所示的屏幕共享效果界面。通过选择整个屏幕、应用窗口、浏览器标签页中任意一个进行共享，选择共享后，本地用户窗口则显示分享的屏幕。



图 5.24 共享屏幕界面

4. 即时消息

本系统基于信令机制，实现会议房间内即时消息的通信，即时消息功能是对音视频通信进行补充，可以通过现场文字讨论，更有效的推进会议进行。点击屏幕下方功能栏中的即时消息窗口按钮，在屏幕的左侧弹出消息窗口，在本文框中输入消息后，点击键盘回车键进行发送。效果如图 5.25 所示。



图 5.25 即时消息通信界面

5.4.2 系统性能分析

本系统用户之间通过 P2P 网络架构进行连接，在本地网络带宽良好的状况下，用户终端的 CPU 消耗会随对等端用户的增加而变化，从而影响音视频效果。本系统程序在管理用户时，是根据 websocket 的 id 来判断一个用户，从而进行端对端的连接，即打开一个浏览器标签页就相当于一个用户进入系统。由于实验条件有限，选择表 5.9 中的一台 PC 作为数据读取的测试终端，其余每台电脑模拟多个用户。在检测终端，通过资源管理器获取 CPU 消耗率和内存使用率的平均值。

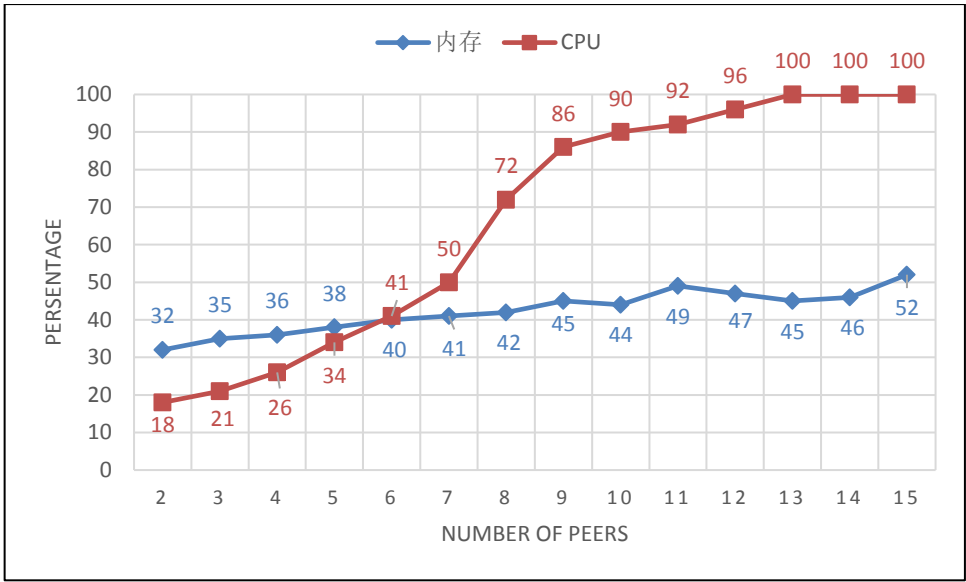


图 5.26 内存和 CPU 性能分析

图 5.26 所示，随着用户数增加，测试终端的 CPU 消耗率呈线性增长，内存保持在一定的水平范围内。当超过 9 人之后，CPU 消耗已经到达 80% 上，此时测试终端已经出现卡顿现象，但人数控制在 8 人以内时，终端处理能力基本满足本系统正常的音视频通信。测试说明，终端的内存不是系统使用的局限，而是 CPU 性能。

表 5.10 音视频直观效果表现

	对端数量	维持时间	音频质量	视频质量
直观效果	2-6	3min	良好	良好
	7-9	3min	可接受	可接受
	10-15	3min	不可接受	不可接受

在测试音视频通信的过程中，同时记录随用户量增多，用户对音视频通信质量的直观表现，如表 5.10 所示，当用户量在 2-6 人时，音视频质量良好，基本能维持正常的通信。当用户量在 7-9 人时，音视频会出现卡顿，原因是随着 CPU 消耗增大，浏览器处理音视频流出现延迟。当超过 10 人时，音视频严重卡顿，无法正常通信，原因是 CPU 消耗过高，测试终端的浏览器已无法处理音视频流。

另外，在系统进行音视频正常通信过程中，可以利用 chrome 浏览器自带的 webrtc-internals 实时检测工具，本质上是使用 WebRTC 的统计 API 分析系统运行时的状态，通过部分数据图，进一步分析。

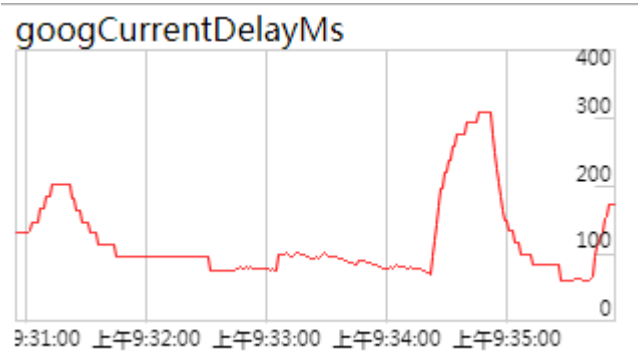
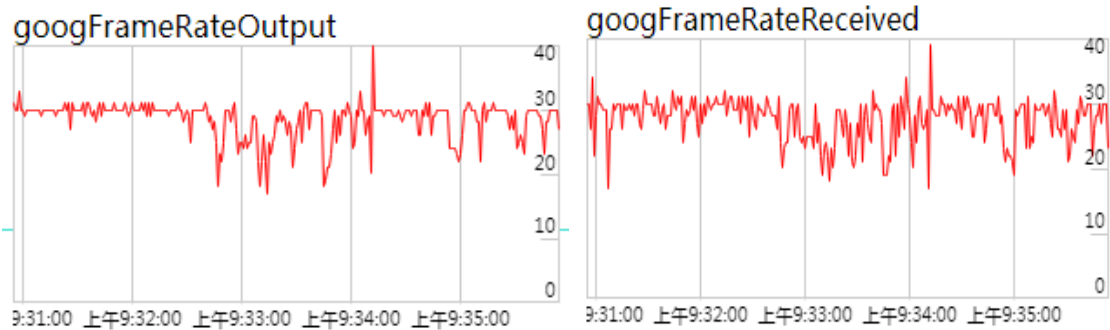


图 5.27 延迟参数

图 5.27 所示的是系统网络延迟，通过数据图观察延迟平均维持在 100ms 左右，说明在网络带宽良好的情况下，除了当用户进入时，会引起一定的延迟波动，但整体不会对音视频通信造成影响。



(a) 帧发送率参数

(b) 帧接收率参数

图 5.28 帧收发率参数

图 5.28 所示的是视频每秒传输的帧数，左边是系统每秒发送的帧数，右边是每秒接收的帧数。通过数据图可以观察，系统每秒发送和接收 25 帧左右，随着用户的进入同样也会有所波动，但基本能保持视频的流畅度，不会有严重卡顿。

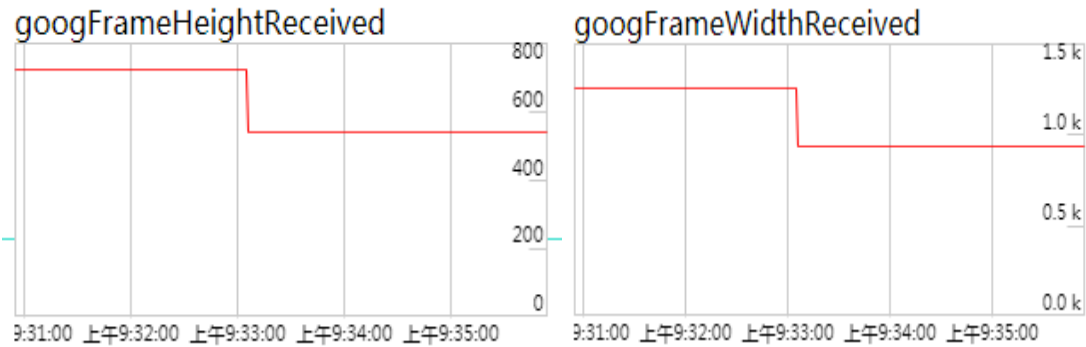


图 5.29 分辨率参数

图 5.29 所示的是视频接收的分辨率大小，通过数据图可知，随着用户量在可接受范围内增加，起始分辨率也会有所下降，最终仍然保持在较好的水平即 900\*500 左右，不会出现太模糊的画面。

综上所述，通过系统基本功能测试，本会议系统的音视频通信功能，即时消息功能，屏幕共享功能，均使用正常。通过系统性能分析，在网络带宽良好的环境下，普通终端的 CPU 性能基本满足系统正常运行的需求，音视频的质量从直观效果、帧收发率和分辨率等方面分析，基本满足用户的使用需求。

### 5.5 本章小结

本章在第二章的基础上，介绍了基于 WebRTC 的音视频会议系统的设计与实现，依托会训云项目，根据系统需求分析，设计系统模块和整体框架，客户端包括会议进入界面、会议房间主界面、WebRTC 音视频交互模块、屏幕共享模块。主要结合 CSS 和 JavaScript 完成样式布局，使用 HTML5 标签、WebRTC JavaScript API 和 Chrome 内置 API 实现音视频交互和屏幕共享功能。服务器端架构包括信令服务器、NAT 服务器、房间模块、即时消息模块、Web 服务器，重点设计了基于 socket.io 的会话管理机制，通过 Coturn 服务器实现 NAT 穿越功能，以及实现基于房间模块的即时消息通信。最后，通过对系统功能效果进行测试，在网络带宽良好的情况下，



本系统所实现的功能基本正常，满足使用要求，通过性能测试进行分析，本系统在限制的 8 人以内的通信，普通终端的 CPU 消耗基本满足对音视频流处理的需求，音视频质量同样也可以满足用户的使用需求。

## 第6章 总结与展望

### 6.1 总结

本文主要围绕 WebRTC 技术进行深入研究，重点工作是研究 WebRTC 的回声消除算法并提出基于自适应滤波器算法的改进，以及实现基于 WebRTC 的多人音视频会议系统。现对本文的主要研究工作进行总结。

首先，研究了回声消除算法的发展现状，基于当前主流的自适应滤波器算法，讨论和总结了最速下降算法、LMS 算法、NLMS 算法和块 LMS 算法原理以及对应的滤波器结构。在此基础上，对 WebRTC 自适应回声消除算法按模块进行分析，归纳总结了其延迟估计方法、线性自适应滤波器和基于相干性的非线性处理的原理。针对其核心模块线性自适应滤波器中使用的固定步长 NLMS 算法，由当前文献中成果的启发，结合步长与误差信号和输入信号之间的关系，建立新的步长函数表达式，理论上，改善了由于固定步长所产生的问题，即稳态误差与收敛速度之间的矛盾，通过 MATLAB 仿真软件进行实验对比验证，在一定程度上，较传统的 NLMS 和文献中改进的算法要好，即在较小稳态误差的情况下，算法拥有较快的收敛速度，并在信号突变时，比传统的 NLMS 算法有较好的跟踪能力。

其次，根据 WebRTC 在国内外发展的现状，结合 WebRTC 整体框架，研究了 WebRTC 在浏览器端的核心接口，围绕实现基于 WebRTC 的会议系统进一步研究了信令机制、NAT 穿越技术。在此基础上，设计基于 socket.io 信令的会话管理机制，从而结合 WebRTC 核心接口 RTCPeerConnection 以及相关 JS 接口实现 WebRTC 音视频通信，会议主界面显示样式为魔方矩阵样式，同时，基于会议房间，依靠会话管理机制，实现会议用户之间的即时消息发送。另外，根据项目需求，利用 Chrome 的扩展程序插件以及所提供的 API，由浏览器网页与插件中的背景文件进行通信，实现共享屏幕的功能。最后测试系统功能使用正常，通过分析系统在多人通信时的 CPU 消耗和内存使用率，说明在良好的网络环境下，普通终端可以满足系统运行需求，系统运行时音视频效果良好，满足系统 8 人以内使用的人数需求。

## 6.2 展望

总的来说,通过 6.1 节所述的具体工作,本文取得一定的研究成果。但是由于时间以及实验条件有限,本文同样存在一些不足之处,未来需要进一步改进和研究。

一方面,虽然本文对 WebRTC 自适应回声消除算法展开研究,但只针对其核心部分线性自适应滤波器算法中的 NLMS 算法这一部分提出改进,并通过仿真实验在理论上验证其改进效果,由于时间、能力有限,并未将实现的改进算法整合到 Chromium 源码的 WebRTC 音频引擎模块中,后续将继续完成这部分的代码编译工作。在此基础上,可以再对 ANS、GCC 等算法进行研究,会得到更多的研究成果。

另一方面,本文围绕会训云项目的需求,实现了基于 WebRTC 的会议系统。在功能上,应该更注重人性化交互设计,实现更多的功能如 P2P 文件共享,增加用户粘性。在系统网络架构上,由于实验条件和项目经费有限,目前所提供的云服务器仅为 1 核 CPU, 2G 内存,固定带宽 4Mbps,无法提供较高带宽的转发服务器或高性能的混流服务器,所以采用网状架构,在良好的网络环境下,虽然能保证系统正常使用,但是一旦出现网络高峰期,网速下降,则需要关闭视频流,来保持音频通话,所以后续可以在该架构中结合新的策略或者技术,在客户端根据网络带宽的大小调整 WebRTC 发送速率以及分辨率,可以得到更好的效果。

## 参考文献

- [1] 屈振华, 李慧云, 张海涛, 等. WebRTC 技术初探[J]. 电信科学, 2012, 28(10): 106-110.
- [2] Ristic D. Learning WebRTC[M]. Birmingham: Packt Publishing, 2015: 2-6, 27-46.
- [3] Johnston A B, Burnett D C. WebRTC 权威指南[M]. 声网 Agora.io, 译. 北京: 机械工业出版社, 2016: 45-60, 160-231.
- [4] 蔡亚东. 基于 WebRTC 和 SDN 的多人视频会议系统的设计与实现[D]. 北京邮电大学, 2017.
- [5] Vashishth S, Sinha Y, Babu K H. Addressing challenges in browser based P2P content sharing framework using WebRTC[C]//Jara A. 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA). Piscataway: IEEE Press, 2016: 851-857.
- [6] Pasha M, Shahzad F, Ahmad A. Analysis of challenges faced by WebRTC video conferencing and a remedial architecture [J]. International Journal of Computer Science and Information Security, 2016, 14(10): 698 -705.
- [7] 文昊翔, 乐彦杰. 针对回声消除应用的自适应算法评价标准研究[J]. 韶关学院学报, 2015, 36(10): 21-25.
- [8] 康健. 音频质量评价和语音识别预处理技术的研究及实现[D]. 北京邮电大学, 2017.
- [9] 姚力, 刘强. VoIP 中一种基于 WebRTC 的回声消除改进算法[J]. 计算机科学, 2017, 44(S1): 309-311+318.
- [10] 张向辉, 黄佳庆, 吴康恒, 等. 基于 WebRTC 的实时视音频通信研究综述[J]. 计算机科学, 2015, 42(2): 1-6+32.
- [11] 吴晨, 林伟. 基于 WebRTC 的音视频通信的设计与实现[J]. 有线电视技术, 2017, 332(8): 88-90.
- [12] Edan N M, Sherbaz A A, Turner S J. Design and implement a hybrid WebRTC signalling mechanism for unidirectional & bi-directional video conferencing[J]. International Journal of Electrical and Computer Engineering, 2018, 8(1): 390-399.

- [13] Garcia B, Gortazar F, Fernandez L L, et al. WebRTC testing: challenges and practical solutions[J]. IEEE Communications Standards Magazine, 2017, 1(2): 36-42.
- [14] 文昊翔. 面向实时通信系统的自适应回声消除算法研究[D]. 浙江大学, 2013.
- [15] 彭娟, 洪小斌. 回声消除基于 NLMS 改进算法的研究[J]. 现代电子技术, 2013, 36(22): 35-38.
- [16] 曾召华, 刘贵忠, 赵建平. LMS 和归一化 LMS 算法收敛门限与步长的确定[J]. 电子与信息学报, 2003, 25(11): 1469-1474.
- [17] 张健. 基于 NLMS 算法回波消除的研究与实现[D]. 哈尔滨工业大学, 2015.
- [18] 高鹰, 谢胜利. 一种变步长 LMS 自适应滤波算法及分析[J]. 电子学报, 2001, 29(8): 1094-1097.
- [19] 周龙龙, 胡启国. 自适应噪声对消的归一化 LMS 算法[J]. 电声技术, 2018, 42(5): 74-77.
- [20] 王亚辉. 基于 WebRTC 语音引擎的会议混音技术研究[D]. 西安电子科技大学, 2013.
- [21] 吴江锐. WebRTC 语音引擎中 NetEQ 技术的研究[D]. 西安电子科技大学, 2013.
- [22] 何海, 付仕明, 叶顺舟. 回声抵消技术中 DTD 算法综述[J]. 中国新通信, 2015, 17(17): 1-2.
- [23] 熊雨新. 基于 WebRTC 引擎的音频视频交互系统设计与实现[D]. 电子科技大学, 2014.
- [24] 曾照成. 基于 WebRTC 的视频会议系统的设计与实现[D]. 中南林业科技大学, 2016.
- [25] Boldt D, Kaminski F, Fischer S. Decentralized bootstrapping for WebRTC-based P2P networks[C]//Westphall C M, Marghitu D. The Fifth International Conference on Building and Exploring Web Based Environments. Red Hook: IARIA Conference, 2017:17-23.
- [26] Apu K I Z, Mahmud N, Hasan F, et al. P2P video conferencing system based on WebRTC[C]//Bhuiyan M A M. 2017 International Conference on Electrical, Computer and Communication Engineering (ECCE). Piscataway: IEEE Press. 2017: 557-567.
- [27] Weinrank F, Becke M, Flohr J, et al. WebRTC data channels[J]. IEEE Communications Standards Magazine, 2017, 1(2): 28-35.

- [28] Eskola R, Nurminen J K. Performance evaluation of WebRTC data channels[C]//Daneshmand M. 2015 IEEE Symposium on Computers and Communication (ISCC). Piscataway: IEEE Press, 2015: 676-680.
- [29] Garcia B, Fernandez L L, Gortazar F, et al. Analysis of video quality and end-to-end latency in WebRTC[C]//Cui S R. 2016 IEEE Globecom Workshops (GC Wkshps). Piscataway: IEEE Press, 2016: 1-6.
- [30] 才鑫. 基于 WebRTC 的多方多媒体通信系统的设计与实现[D]. 北京邮电大学, 2015.
- [31] 胡萍. 基于流媒体视频会议移动终端的系统研究[D]. 北京交通大学, 2014.
- [32] Sredojevic B, Samardzija D, Posarac D. WebRTC technology overview and signaling solution design and implementation[C]//Biljanovic P. 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). Piscataway: IEEE Press, 2015: 1006-1009.
- [33] Edan N M, Sherbaz A A, Turner S J. WebNSM: A novel scalable WebRTC signalling mechanism for many-to-many video conferencing[C]//Aberer K. 2017 IEEE 3rd International Conference on Collaboration and Internet Computing (CIC). Piscataway: IEEE Press, 2017: 27-33.
- [34] Ouya S, Seyed C, Mbacke A B, et al. WebRTC platform proposition as a support to the educational system of universities in a limited Internet connection context[C]//Abraham A, Haqiq A. 2015 5th World Congress on Information and Communication Technologies (WICT). Piscataway: IEEE Press, 2015: 47-52.
- [35] 张炳婷. 自适应滤波算法的研究及应用[D]. 曲阜师范大学, 2016.
- [36] Haykin S. 自适应滤波器原理. 第四版[M]. 郑宝玉, 译. 北京: 电子工业出版社, 2010: 1-25, 159-298.
- [37] 陈泓宇. 自适应滤波算法的研究及在 VoIP 系统上的应用[D]. 浙江工业大学, 2017.
- [38] 管四海. LMS 类自适应滤波算法的研究[D]. 西安电子科技大学, 2017.
- [39] Paliwal K, Schwerin B, Wojcicki K. Speech enhancement using a minimum mean-square error short-time spectral modulation magnitude estimator[J]. Speech Communication, 2012, 54(2): 282-305.
- [40] 陈国志, 乐彦杰, 张磊, 等. 用于回声消除系统的自适应延时估计算法研究[J]. 科学技术与工程, 2015, 15(3): 244-249.

- [41] Valin J M. On adjusting the learning rate in frequency domain echo cancellation with double-talk[J]. IEEE Audio, Voice and Language Processing Transactions, 2007, 15(3): 1030-1034.
- [42] 胡坚, 樊可清, 刘洋. 基于分段块频域自适应滤波算法的长延时回声消除[J]. 数据采集与处理, 2009, 24(S1): 11-14.
- [43] Soo J S, Pang K K. Multidelay block frequency domain adaptive filter[J]. IEEE Transactions on acoustics, speech and signal processing. 1990, 38(2): 373-376.
- [44] Estermann P, Kaelin A. A hands-free phone system based on a partitioned frequency-domain adaptive echo canceler[C]//1996 8th European Signal Processing Conference (EUSIPCO 1996). 1996: 1-4.
- [45] Eneman K, Moonen M. Iterated partitioned block frequency-domain adaptive filtering for acoustic echo cancellation[J]. IEEE Transactions on Speech and Audio Processing, 2003, 11(2): 143-158.
- [46] Borralló J M P, Otero M G. On the implementation of a partitioned block frequency domain adaptive filter for long acoustic echo cancellation[J]. Signal Processing, 1992, 27(3): 301-315.
- [47] 赵万能, 何峰, 郑林华. 一种新的变步长 NLMS 盲自适应多用户检测算法[J]. 信号处理, 2010, 26(3): 413-416.
- [48] 刘志骋, 杨博. 一种改进的变步长 NLMS 算法[J]. 现代电子技术, 2015, 38(22): 12-13+16.
- [49] 谷源涛, 唐昆, 崔慧娟, 等. 新的变步长归一化最小均方算法[J]. 清华大学学报(自然科学版), 2002, 42(1): 15-18.
- [50] 张兰勇, 王帮民, 刘胜, 等. 一种新的变步长自适应噪声消除算法[J]. 电子学报, 2017, 45(2): 321-327.
- [51] 沈再阳. 精通 MATLAB 信号处理[M]. 北京: 清华大学出版社, 2015: 56-85.
- [52] 万永革. 数字信号处理的 MATLAB 实现. 第二版[M]. 北京: 科学出版社, 2016: 126-256.
- [53] Edan N M, Sherbaz A A, Turner S J. WebNSM: a novel WebRTC signalling mechanism for one-to-many bi-directional video conferencing [C]//Moaid E N. IEEE Computing Conference 2018. Piscataway: IEEE Press, 2018: 558-568.

- 
- [54] Cengiz T, Albert L. WebRTC based augmented secure communication[C]//Ozturk E. 2016 24th Signal Processing and Communication Application Conference (SIU). Piscataway: IEEE Press, 2016: 1621-1624.
- [55] 崔健. 基于 WebRTC 的 P2P 视频会议系统设计与实现[D]. 北京工业大学, 2016.
- [56] Edan N M, Sherbaz A A, Turner S J. Design and evaluation of browser-to-browser video conferencing in WebRTC[C]//Lorenz P. 2017 Global Information Infrastructure and Networking Symposium (GIIS). Piscataway: IEEE Press, 2017: 75-78.



## 致谢

转眼间，三年研究生生涯即将结束。此时回想当初，独自一人踏上远程之路，来到陌生的学校，在这三年的时光里，努力过，辛苦过，快乐过，曾经陌生的一切逐渐熟悉，而今又要离开。在重邮的经历，无论是学习上还是生活上，都将是我今后人生中不可或缺的一部分，并激励我继续前进。

在我的研究生生涯中，我要感谢实验室里敬爱的老师和同甘共苦的同窗，是他们在在学习上给予指导，始终如一地鞭策我前进；在生活上互帮互助，一起共进退。

首先，感谢我的导师龙昭华教授，龙老师既是一位知识渊博、科研精神十足的大学教授，也是一位项目经验丰富、善于团队管理的领导者。龙老师在科研路上，永远以身作则，走在前列，带领我们朝着既定方向前进。无论春夏秋冬，龙老师经常不分昼夜的阅读国际文献，加班加点地搞科研。龙老师根据每位学生的特长，指定安排其在某一方向学习，攻克难题，我们经常会收到龙老师凌晨三四点的邮件，将他在学习过程中的资料和感悟即时发送给我们，这让我们年轻人感到惭愧。龙老师身上的高尚品质，值得我们终身学习，即将离开校园，走上工作岗位，我将以龙老师为楷模，努力做一名奋斗者。

其次，我要感谢实验室的指导老师张林老师，张老师是一位技术实力雄厚的青年老师，在生活上与学生达成一片，在学习上亲自指导我们，加快项目的整体开发。既是我们学业上的良师，也是我们日常的益友。

还有，我要感谢实验室的同学们，感谢会训云小组中的同门师兄黄军华、周鹏、董瑞芳、师弟杨宗靖、梅文博、乔焕宇、崔永明、师妹邓青青的帮助，以及感谢师兄蒋纬昌、杨鑫、陈江南、熊华为、李国领、叶二伟、汤浩、师弟范天文、宋杨、马俊、胡波、余快、师妹沈励芝。还有，感谢重庆邮电大学为我们学生提供优良的教学质量，维持良好的学习环境以及为我们提供优越的就业条件。

感谢我的家人，感谢我的父母多年来对我学业上、生活上的支持，相信在不久的将来，我会用我努力的成果来回报您们。

最后，真诚地感谢审阅论文和参加答辩委员会的老师，感谢各位老师对学生论文进行批评与指导。

## 攻读硕士学位期间从事的科研工作及取得的成果

### 参与科研项目：

- [1] 会训无线投影云管理系统的推广(KJZH17118), 重庆市教委高校优秀成果转化项目, 2017 年 1 月-2019 年 12 月。
- [2] ISO 国际标准研究项目(ISO/IEC TR 29181-8-2017): Future Network -- Problem Statement and Requirements -- Part8: Quality of Service.

### 发表及完成论文：

- [1] **Ziqiang Wei**, Zhaohua Long. Improved Variable Step Size NLMS Algorithm[C]// 2019 7th International Conference on Computer-Aided Design, Manufacturing, Modeling and Simulation (CDMMS 2019), 2019年1月已录用.
- [2] 龙昭华, **魏自强**. 多分屏无线投影系统V1.0. 中国. 计算机软件著作权. 登记号: 2018SR624860.
- [3] 龙昭华, 周鹏, **魏自强**. 多用户控制无线投影系统V1.0. 中国. 计算机软件著作权. 登记号: 2018SR626527.
- [4] Guoling Li, Zhaohua Long, **Ziqiang Wei**. Future Network with Multi-layer IP Address[C]// Lin Liu. Proceedings of the 2th International Conference on Advances in Materials, Machinery, Electronics. Melville: AIP Press, 2018: 040186-1-4.