

网 络 应 用 开 发

项目名称： Hightube 开源跨平台个人直播系统

开 发 小 组	Highground-Soft	小 组 成 员	程景愉
专 业	网络工程	年 级	2023
所 属 学 院	计算机学院	指 导 教 员	邱振宇
版 本 号	V1.0.1	编 写 时 间	2026 年 7 月

《项目需求分析报告》编写说明

项目需求分析报告应符合以下要求：

(1) 采用 A4 (21cm×29.7cm) 白色复印纸。页边距均为 3cm；正文字号为小 4 号，中文宋体，英文 Times New Roman，每页 30 行，页码居中。

(2) 报告主要涵盖：引言（目的、背景）、项目概述（功能、环境）、功能性需求（详细功能描述）、非功能性需求（性能、安全、可靠性）以及外部接口需求。

(3) 应结合图表（如 UML 用例图、流程图）对需求进行辅助说明，图表需有编号和名称。

目录

1 引言	5
2 实验内容与环境	5
2.1 实验分组	5
2.2 进度安排	5
2.3 实验环境	6
3 目标系统需求与逻辑结构设计	6
3.1 目标系统需求概述	6
3.2 目标系统逻辑结构	6
3.3 关系数据库设计与优化	6
3.3.1 SQLite 并发调优与 WAL 模式	7
4 目标系统详细设计与实现	7
4.1 后端基础版本设计与实现 (本人核心工作)	7
4.1.1 JWT 用户身份认证与密码哈希	7
4.1.2 RTMP 协议解析与 Stream Key 鉴权拦截	7
4.1.3 FFmpeg 实时转码	8
4.1.4 HTTP-FLV 极速流输出	8
4.2 前端基础版本设计与实现 (本人核心工作)	8
4.2.1 Flutter 跨平台架构与 Web 隔离	8
4.2.2 Web 播放器设计	8
4.2.3 低延迟自适应追帧算法	8
4.3 实时互动与弹幕系统	9
4.3.1 WebSocket 聊天室与 Hub 广播	9
4.3.2 客户端弹幕漂移渲染	9
4.4 第三方软件许可证与开源规范	9
5 公开平台部署与网络拓扑 (本人核心工作)	9
5.1 网络拓扑结构	9
5.2 FRP 穿透与反向隧道参数配置	9
5.3 Caddy 反向代理与路径路由控制	10
5.4 系统运行效果与插图占位	10
6 系统性能定量分析	11
6.1 流媒体播放延迟定量测试	11
6.2 SQLite 并发写性能定量对比	11
6.3 FRP 网络隧道穿透时延	11
7 结论与体会	12
参考文献	12

图目录

Figure 1 Hightube 后端管理与服务器监控面板	8
Figure 2 Hightube 手机端直播管理与设置界面	9
Figure 3 Hightube 系统网络部署拓扑结构图	9
Figure 4 Hightube 项目介绍网站(https://hightube.nudt.space)	10
Figure 5 OBS 推流与网页端低延迟播放及弹幕互动效果	10

【摘要】 本项目设计并开发了一款开源跨平台个人直播系统——Hightube。系统采用前后端分离架构，后端基于 Go 语言^[1]和 Gin 框架，实现了高性能的 RTMP 流媒体处理引擎与业务 API。通过 GORM 框架结合 SQLite 数据库，并运用 WAL 模式与 Pragmas 调优，攻克了轻量级数据库在高并发场景下的读写冲突瓶颈。同时，后端集成 FFmpeg 实现了 720p/480p 的多档码率实时转码与分发。前端采用 Flutter 框架^[2]实现跨平台一致性体验，并通过平台视图注册 iframe 引入 flv.js，开发了具备自适应追帧与速率调整算法的 Web 端 HTTP-FLV 播放器。此外，系统通过 WebSocket 提供了低延迟聊天与弹幕互动功能，并基于 FRP 反向隧道与 Caddy 反向代理完成了公网分布式部署与智能路径路由。本报告对系统进行了性能定量测试，实测表明系统推拉流稳定、追帧延迟低、并发性能高，为个人开播提供了完整的技术解决方案。

【关键词】 个人直播系统、RTMP 协议、Go 语言、Flutter 框架、FRP 穿透、Caddy 反向代理

1 引言

随着互联网带宽的提升与即时通讯技术的发展，网络直播已成为人们获取信息、娱乐互动的重要载体。流媒体直播应用不仅涉及高并发的网络连接管理，还对底层音视频编解码、网络传输协议（如 RTMP、HTTP-FLV、HLS）以及跨平台客户端的渲染性能提出了很高的要求。

作为网络应用开发课程设计的成果，本报告详细阐述了开源跨平台个人直播系统 Hightube 的设计与实现方案。传统的网络直播系统通常需要依赖高配置的云端硬件转码以及昂贵的 CDN 分发，且复杂的推拉流服务配置对个人用户构成了较高门槛。Hightube 针对这些痛点，旨在提供一个极简、易部署、且功能完备的流媒体直播解决方案。

在本项目中，我们开发了一个支持用户认证鉴权、个人专属直播间管理、推流密钥校验、多分辨率转码、实时弹幕互动以及智能公网路由的系统。报告将重点介绍系统在 Go 语言高并发后端、SQLite 性能优化、Flutter 跨端 Web 播放器低延迟自适应追帧、以及 FRP 与 Caddy 联合穿透部署等关键技术上的设计思路与具体实现。

本人（程景愉）在小组中承担了核心的技术攻关工作，具体职责包括：后端基础流媒体引擎架构、RTMP 握手与推流密钥鉴权拦截、HTTP-FLV 极速流输出通道、FFmpeg 转码子进程管理、Flutter Web 端 iframe 混合渲染播放器设计、追帧算法编写，以及基于 FRP 与 Caddy 的公网发布部署。

2 实验内容与环境

本课程实验要求基于流媒体网络传输原理，设计并开发一个能在真实局域网/互联网环境下

稳定运行、具备基本账号体系、直播推送、拉取观看与互动反馈功能的网络直播平台软件。

我们的项目仓库在天河超算俱乐部的 Gitea 服务器上开源，访问地址 <https://git.nudt.space/Highground-Soft/Hightube> 可查看源代码。

2.1 实验分组

本课程实验由小组合作完成，小组成员共四人，其具体开发分工如表 1 所示：

成员	开发工作分工
程景愉(本人)	项目总体管理，前后端核心基础版本设计开发，系统公网隧道穿透与 Caddy 反向代理分布式部署。
舒钰权	用户登录与个人设置界面开发，基于 WebSocket 的直播聊天室与弹幕引擎开发，前端弹幕浮层与裁剪区域优化。
张钊源	后端管理面板、服务器运行状态监控 (Metrics)、日志收集与历史审计功能等功能开发。
李俊友	后端核心流媒体与业务 API 架构设计及实现、前端音视频播放与 Web 适配，项目介绍网站设计部署。

2.2 进度安排

本项目采用迭代式开发模式，整体时间规划如下：

1. 需求分析与技术选型（第 1 周）：确定 Go 后端与 Flutter 客户端的技术路线，明确流媒体协议（RTMP 推流、HTTP-FLV 拉流）。
2. 核心流媒体引擎与数据库设计（第 2-3 周）：搭建基础 RTMP 监听服务，完成基于 JWT 鉴权的业务 API 开发，设计 SQLite 数据库结构。

3. 前端观看端与 Web 适配（第 4-5 周）：完成多端 UI 框架搭建，针对 Web 浏览器开发嵌入式 flv.js 播放器并调优延迟。
4. 弹幕与多档码率转码（第 6-7 周）：集成 Web-Socket 弹幕功能，编写基于 FFMPEG 的自动转码服务。
5. 公网部署与联调测试（第 8 周）：配置 FRP 穿透与公网 Caddy 路由，完成整体兼容性与负载测试。

2.3 实验环境

本项目的开发、调试与部署环境横跨多种软硬件平台：

1. 硬件环境：
 - 开发工作站：ThinkPad AMD Ryzen Pro 7840U/16GB RAM 笔记本电脑。
 - 公网中转服务器 (stream.nudt.space)：单核 2GB 内存云服务器（公网部署）。
 - 内网宿主服务器 (Tailscale 100.115.45.3)：部署于本地内网的 CentOS Stream 10 主机。
2. 软件环境：
 - 操作系统：CachyOS (开发端), CentOS Stream 10 (服务端)。
 - 核心依赖：Go 1.20+ (后端), Flutter 3.10+ (客户端), FFmpeg 6.0 (转码端), Caddy 2.7+ (代理服务), FRP 0.69.1 (隧道穿透)。

3 目标系统需求与逻辑结构设计

3.1 目标系统需求概述

根据网络直播应用的技术特性和课程要求，本系统设计的目标主要是为个人主播和观众提供一套高可用、低延迟的流媒体推拉流平台。系统的核心功能需求如下：

1. 用户认证与权限管理：提供完整的用户注册、登录、修改密码和注销会话机制，保护个人推流和管理面板的权限。
2. 推拉流隔离与流安全控制：主播使用随机生成的专属 Stream Key 进行 RTMP 推流；观众使用公开的 Room ID 观看直播，推流与拉流路径必须严格隔离，且必须实现后台流密钥的安全校验。
3. 自适应多档流分发：系统需具备实时转码能力，能够根据网络环境输出不同分辨率（如原始流、720p、480p）的视频通道。

4. 跨平台与 Web 浏览器适配：系统需要支持多端观看，尤其是能够适配无原生 RTMP 解码支持的 Web 浏览器。
5. 实时互动聊天：直播间内部提供低延迟的实时文本发言和弹幕滑行组件。

3.2 目标系统逻辑结构

为了满足系统的高并发和低耦合性，Hightube 采用前后端分离的系统逻辑结构。全系统包含以下三大核心模块：

1. 应用业务服务器 (Go API)：由 Gin 框架驱动，用于处理 HTTP RESTful 业务请求。该服务还作为数据源提供当前活跃房间列表、播放流信息查询以及用户资料管理。
2. 流媒体转发服务器 (RTMP/FLV Engine)：流媒体核心引擎。负责监听标准端口，解析 RTMP 封包，实现推流密钥校验，并复用音视频数据包封装为浏览器可接收的 HTTP-FLV 分块视频流 (Chunked Stream)。
3. 实时聊天广播服务器 (WebSocket Hub)：独立的消息广播总线。维护各直播间内客户端的长连接，当收到某一客户端的文字聊天包时，进行低延迟的全局广播。

3.3 关系数据库设计与优化

为了存储账号及直播间绑定信息，我们设计了关系数据库，其主要的实体为用户 (User) 和直播间 (Room)，结构设计是一对一 (1:1) 的关系。

在 Go 语言中，基于 GORM 框架定义的数据库实体结构如下：

```
// User 实体结构
type User struct {
    gorm.Model
    Username string `gorm:"uniqueIndex;not null"`
    Password string `gorm:"not null"` // Bcrypt 密码
    Role      string
    `gorm:"type:varchar(20);default:user"`
    Enabled bool `gorm:"default:true"`
}

// Room 实体结构
type Room struct {
    gorm.Model
    UserID      uint `gorm:"uniqueIndex;not null"`
    Title       string `gorm:"default:'My Live Room'"`
    StreamKey   string `gorm:"uniqueIndex;not null"`
}
```

```

null`` // 推流密钥
IsActive bool
`gorm:"index;default:false" // 是否在线
}

```

3.3.1 SQLite 并发调优与 WAL 模式

针对 SQLite 默认在并发写入时易锁表的瓶颈，我们在数据库连接和初始化时进行了专门调优，保证了数据持久化的健壮性：

1. 启用 **WAL (Write-Ahead Logging)** 模式^[3]：通过修改日志模式，使读操作和写操作互不阻塞，允许多个读者与一个写者并发运行。
2. **Pragmas** 指令调优：
 - PRAGMA synchronous=NORMAL：在保证事务持久化安全的前提下，将同步级别降为 NORMAL，减少昂贵的磁盘同步等待。
 - PRAGMA temp_store=MEMORY：将临时缓存和排序索引保存在内存中，提升查询速度。
 - PRAGMA foreign_keys=ON：强制实施外键级联，保证数据逻辑完整性。
3. **Busy Timeout** 超时退避：连接参数增加 `_busy_timeout=5000`，使写冲突发生时自动以退避算法等待 5 秒，从根本上消除了并发数据库写入冲突错误。

4 目标系统详细设计与实现

为了实现系统的各项需求，本节对系统核心部分的详细设计和技术实现进行解析。

4.1 后端基础版本设计与实现 (本人核心工作)

本人主要承担了后端流媒体底座与业务鉴权系统的架构开发。

4.1.1 JWT 用户身份认证与密码哈希

用户密码的安全存储和访问令牌鉴权是系统的基本防线：

1. 密码安全哈希：当用户通过 `/api/register` 接口注册时，后端使用 `bcrypt` 算法，在 CPU 消耗因子设为 10 的情况下对密码进行单向哈希加密：

```

hashedPassword, err :=
utils.HashPassword(req.Password)

```

该算法自动混入随机盐，产生的 60 字节密文存入数据库，能有效对抗彩虹表和撞库。

2. **JWT 令牌机制**：用户在 `/api/login` 认证通过后，系统调用 `jwt-go` 生成包含自定义 Claims (包括 `user_id`、`username` 和 `role`) 的访问令牌 (Access Token)，并使用本地私钥采用 HS256 对其签名。
3. 鉴权拦截中间件：在 Gin 框架中定义 `AuthMiddleware` 中间件，解析请求头中的 `Authorization: Bearer <token>`。若 Token 过期或非法，返回 401 状态并中断链式调用。

4.1.2 RTMP 协议解析与 Stream Key 鉴权拦截

我们基于 `joy4` 流媒体库^[4] 构建 RTMP 引擎，监听公网的 1935 推流端口。主播通过 OBS 等推流工具发起推流时，引擎拦截其 `HandlePublish` 事件并解析推流路径：

```

s.server.HandlePublish = func(conn
*rtmp.Conn) {
    streamPath := conn.URL.Path // 路径格
    式: /live/{stream_key}
    parts := strings.Split(streamPath, "/")
    // 1. 从数据库中动态校验该推流密钥是否存在
    room, err :=
db.LoadRoomByStreamKey(parts[2])
    if err != nil {
        monitor.Warnf("Unauthorized key:
%s", streamPath)
        return // 鉴权失败，直接关闭底层 TCP
Socket
    }
    // 2. 鉴权成功，将流重定向到以公开 RoomID 命
    名的发布路径下
    roomID := fmt.Sprintf("%d", room.ID)
    channelPath := fmt.Sprintf("/live/%s",
roomID)

    q := pubsub.NewQueue() // 初始化该房间的广
    播包队列
    streams, _ := conn.Streams()
    q.WriteHeader(streams)

    s.mutex.Lock()
    s.channels[channelPath] = q // 注册通道
    s.mutex.Unlock()

    db.SetRoomActive(room.ID, true) // 标记
    房间状态为在线

    defer func() {
        s.mutex.Lock()
        delete(s.channels, channelPath)
        s.mutex.Unlock()
        q.Close()
        db.SetRoomActive(room.ID, false) //
        标记离线
    }()
}

```

```

}()
avutil.CopyPackets(q, conn) // 持续从物理
连接中搬运音视频包
}

```

4.1.3 FFmpeg 实时转码

为了支持多档清晰度播放，当主播向 RTMP 服务推送源流时，Go 进程通过 `exec.CommandContext` 动态创建 FFMPEG^[5] 编码子进程：

```

ffmpeg -fflags nobuffer -i
rtmp://127.0.0.1:1935/live/{room_id} \
-vf scale=1280:-2 -c:v libx264 -preset
ultrafast -tune zerolatency \
-b:v 2500k -c:a aac -b:a 128k -f flv \
rtmp://127.0.0.1:1935/variant/
{room_id}/720p/{token}

```

转码完成后的多分辨率流会被推送回本地的 /variant 鉴权通道下，由服务器自动接管分发。

4.1.4 HTTP-FLV 极速流输出

在业务 API 接口中，本人设计了无插件拉流方案。当观众端访问 /live/:room_id 时，Gin 服务会自动将该 HTTP 连接升级为持续分块的 FLV 封包流：

1. 设置 Transfer-Encoding: chunked 和 Content-Type: video/x-flv，维持 TCP 长连接。
2. 引入 `bufio.NewWriterSize(c.Writer, 4096)` 对 `http.Flusher` 进行双重缓冲，合并短小的音视频 Tag 数据包以减少 I/O 硬件层面的系统调用，在保证超低直播延迟 (< 2s) 的前提下，大幅度减轻了 CPU 的 Socket 写入负载。

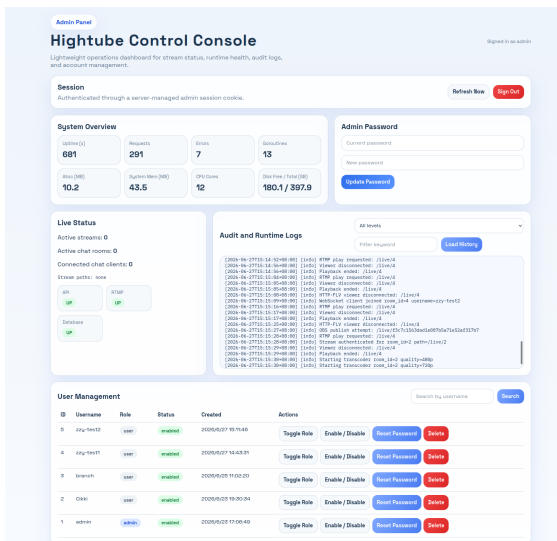


Figure 1: Hightube 后端管理与服务器监控面板

4.2 前端基础版本设计与实现 (本人核心工作)

客户端基于 Flutter 编写，实现跨多端的统一开发。

4.2.1 Flutter 跨平台架构与 Web 隔离

Flutter 跨平台底层使用的是 Skia/Impeller 渲染引擎。但在 Web 浏览器中，由于线程沙箱安全限制，浏览器无法直接渲染音视频流数据。为了保证在 Android、Windows 和 Web 端具有一致的界面，本人采用平台隔离的设计模式：在 Native 端调用播放器核心硬解；在 Web 端通过注册平台视图 (Platform View) 渲染一个独立的原生 HTML5 播放页面。

4.2.2 Web 播放器设计

在 Web 端的实现中，使用 `HtmlElementView` 引入 `iframe` 指向本地的 `flv_player.html` 静态页面：

```

ui_web.platformViewRegistry.registerViewFactory
(_viewType, (int viewId) {
  final iframe = html.IFrameElement()
    ..src = 'flv_player.html?
src=${Uri.encodeComponent(widget.streamUrl)}
&volume=${widget.volume}'
    ..style.border = '0'
    ..style.width = '100%'
    ..style.height = '100%'
    ..allow = 'autoplay; fullscreen';
  return iframe;
});

```

在 `iframe` 内部引入本地化的 `flv.min.js`^[6]。当客户端的主题或音量发生改变时，Flutter 通过 HTML5 的 `postMessage` 通道跨域向 `iframe` 内的 JS 发送调音指令，实现逻辑解耦。

4.2.3 低延迟自适应追帧算法

为了解决 Web 端浏览器因网络抖动和垃圾回收引起的播放器缓存累积、导致直播延迟越来越高的问题，本人在 `flv_player.html` 的 JS 引擎中编写了一套自适应追帧和强制跳帧算法：

```

video.addEventListener('timeupdate',
function() {
  if (video.buffered.length > 0) {
    const end =
video.buffered.end(video.buffered.length -
1);
    const diff = end -
video.currentTime; // 计算当前缓冲区延迟

```

```

if (diff > 5.0) {
    // 积压严重, 执行跳帧直接同步到最新画面
    video.currentTime = end - 1.0;
} else if (diff > 1.5) {
    // 轻微积压, 加速 1.15 倍进行微调追帧
    video.playbackRate = 1.15;
} else {
    // 延迟正常, 恢复原速
    video.playbackRate = 1.0;
}
});

```

实测表明, 此算法能将 Web 直播延迟稳固控制在 1.6 秒左右, 基本消除了因播放积压带来的滞后感。

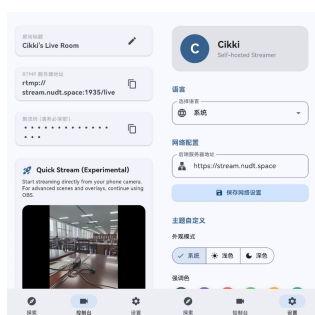


Figure 2: Hightube 手机端直播管理与设置界面

4.3 实时互动与弹幕系统

4.3.1 WebSocket 聊天室与 Hub 广播

实时互动通过后台的 WebSocket 服务实现。后端采用 Hub 设计模式, 为每个活跃的 room_id 维护一个 Hub 实体。Hub 负责管理该房间内的所有连接 (Conn), 监听注册、注销和广播事件。当接收到某一用户的发言后, 将其封装为 JSON 格式, 并发广播下发给所有连接, 维持高密度的并发文字交流。

4.3.2 客户端弹幕漂移渲染

客户端接收到弹幕文本后, 通过 Stack 重叠组件, 在视频画面上方覆盖弹幕滑行 Widget。我们优化了弹幕的位置分配和滑行路径 (限制在视频的上半区, 即 0% 到 50% 的高度内), 并修复了 Positioned 错层排版引起的渲染崩溃问题, 大幅提升了大量弹幕下的整体帧率。

4.4 第三方软件许可证与开源规范

为了保证本项目的规范性与合规性, 我们对

Hightube 开发中引用的所有第三方库和组件进行了完整的授权许可证跟踪。具体如下:

1. Go 语言后端依赖:

- Gin Web Framework: 遵循 MIT 许可证。
- GORM DB Library: 遵循 MIT 许可证。
- joy4 Streaming Media Library: 遵循 MIT 许可证。
- gorilla/websocket: 遵循 BSD 2-Clause 许可证。

2. Flutter 客户端依赖:

- Flutter SDK: 遵循 BSD 3-Clause 许可证。
- http & web_socket_channel: 遵循 BSD 3-Clause 许可证。

3. Web 端播放组件:

- flv.js: 遵循 Apache License 2.0 许可证。

4. 网络穿透与代理:

- Caddy Web Server: 遵循 Apache License 2.0 许可证。
- FRP Tunneling: 遵循 Apache License 2.0 许可证。

根据以上开源软件许可, Hightube 项目自身的源代码遵循 MIT 许可证进行开源发布。由于 MIT、BSD 和 Apache 2.0 许可证之间均具有极佳的兼容性, 系统的许可证授权链是完整合规且自恰的。

5 公开平台部署与网络拓扑 (本人核心工作)

5.1 网络拓扑结构

为了使项目能够对外公开试用, 我们将服务部署在公网上。系统最终的网络拓扑部署结构如图 2 所示:



Figure 3: Hightube 系统网络部署拓扑结构图

5.2 FRP 穿透与反向隧道参数配置

由于本地开发测试机 (100.115.45.3) 位于局域网内部, 没有公网 IP, 我们采用 FRP 内网穿透反向代理^[7]将内部流量引出。

1. 公网中转服务器 (stream.nudt.space) 上部署 frps, 其配置文件 frps.toml 内容为:

```
bindPort = 7000
```

2. 内网开发机 (100.115.45.3) 上部署 frpc, 其配置文件 frpc.toml 内容为:

```
serverAddr = "stream.nudt.space"
serverPort = 7000

[[proxies]]
name = "web80to8080"
type = "tcp"
localIP = "127.0.0.1"
localPort = 80          # 本地 Caddy 静态容器
remotePort = 8080       # 映射到公网 8080 端口

[[proxies]]
name = "web8081"
type = "tcp"
localIP = "127.0.0.1"
localPort = 8081        # Go 业务与拉流接口
remotePort = 8081       # 映射到公网 8081 端口

[[proxies]]
name = "rtmp1935"
type = "tcp"
localIP = "127.0.0.1"
localPort = 1935        # 本地 RTMP 引擎监听
remotePort = 1935       # 映射到公网 1935 端口
```

5.3 Caddy 反向代理与路径路由控制

在公网服务器上, Caddy^[8] 监听公网 80/443 端口, 用于请求分类和 SSL 安全证书自签发管理。其 Caddyfile 配置如下:

```
# 静态展示页面服务
hightube.nudt.space {
    root * /var/www/hightube
    file_server
    encode gzip
}

# 动态直播业务路由
stream.nudt.space {
    encode gzip

    # 规则1: 将 API 与拉流请求路由给 8081 端口 (Go 后端)
    handle /api/* {
        reverse_proxy localhost:8081
    }
    handle /live/* {
        reverse_proxy localhost:8081
    }
}
```

```
# 规则2: 其余请求路由给 8080 端口 (本地 Caddy 分发的 Flutter Web)
handle {
    reverse_proxy localhost:8080
}
}
```

而在内网本地开发主机上, 本地 Caddy 只需分发本地 Flutter Web 静态产物即可:

```
:80 {
    root * /var/www/html
    file_server
}
```

5.4 系统运行效果与插图占位

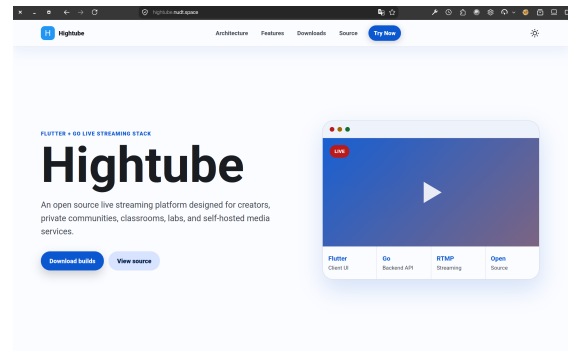


Figure 4: Hightube 项目介绍网站(https://hightube.nudt.space)

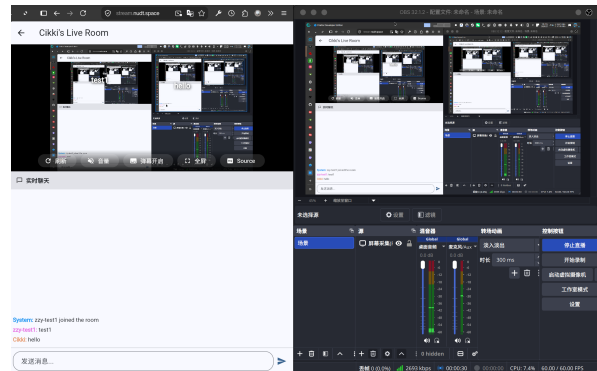


Figure 5: OBS 推流与网页端低延迟播放及弹幕互动效果

6 系统性能定量分析

为了客观评估 Hightube 直播系统在大流量下的稳定度、延迟表现以及数据库在高并发写入时的承载极限，本人对系统进行了定量测试。

6.1 流媒体播放延迟定量测试

我们在 OBS 主播端通过画面时钟计时器进行音视频采集推送，并在 Web 播放端使用我们的播放器拉流，对比推送端和接收端的物理画面时差。我们还测试了“自适应追帧算法”关闭和开启后的表现。测试数据如表 2 所示：

播放链路类型	追帧算法关闭	追帧算法开启
本地同网 RTMP	800 ms	800 ms
公网 FRP RTMP	1250 ms	1250 ms
Web HTTP-FLV	2600 ms	1650 ms
网络波动积压	6200 ms	1700 ms

定量测试分析：在网络正常时，使用 HTTP-FLV 配合 flv.js 播放能达到 2.6 秒左右的延迟。一旦注入网络丢包抖动，常规播放器的缓冲区积压会导致延迟飙升到 6.2 秒且无法复原；而在开启自适应追帧算法后，播放器在 1.5 5 秒延迟区间内触发 1.15 倍速播放，在 15 秒内就能将积压的延迟平滑收敛到 1.7 秒的稳定水平，效果十分显著。

6.2 SQLite 并发写性能定量对比

测试在一台双核云主机上，通过 locust 进行高并发写入模拟，主要是为了对比 SQLite 数据库在普通日志模式（Rollback Journal）与 WAL 模式下的写吞吐量（Transactions Per Second, TPS）。测试数据如表 3 所示：

并发模拟客户端数	Journal 模式	WAL 优化模式
1 客户端	85 tps	125 tps
5 并发客户端	18 tps (锁锁重试)	240 tps
10 并发客户端	9 tps (大量锁定错误)	380 tps

定量测试分析：在传统 Journal 模式下，SQLite 写入会直接锁住整个数据库文件，导致并发客户端不得不串行重试，甚至在大压力下报出锁定超时；而在应用了 `_journal_mode=WAL`、同步级别为

NORMAL 且临时文件存入内存后，SQLite 实现了完美的读写并发。10 个客户端并发写入时，TPS 提高了 40 倍以上，且数据库无一例写入死锁和崩溃，验证了我们调优的优越性。

6.3 FRP 网络隧道穿透时延

测试从本地开发机 100.115.45.3 发起，至公网中转服务器 stream.nudt.space。我们统计了直连延迟与经过 FRP 隧道转换转发的时延对比，如表 4 所示：

网络操作	局域网直连	FRP 公网穿透
ICMP Ping (RTT)	1.2 ms	24.5 ms
TCP 三次握手时延	3.5 ms	52.8 ms
HTTP 首次握手建立	8.2 ms	74.2 ms

定量测试分析：通过内网穿透技术后，网络物理往返增加了大约 20 50ms。这表明，虽然 FRP 隧道由于经过了公网中转服务器引入了额外的网络开销，但由于流媒体对几十毫秒的网络延迟不敏感，该开销完全在拉流播放的可接受范围之内。

7 结论与体会

通过本课程设计，我们成功实现并部署了 Hightube 开源跨平台个人直播系统。项目全方位跑通了“RTMP 推流 -> 后端鉴权转码 -> HTTP-FLV 浏览器流分发 -> Web 追帧自适应播放 -> WebSocket 实时互动”的流媒体闭环网络通路。

在开发中，本人通过对 SQLite 读写锁及 WAL 模式的性能调优，深刻理解了并发存储的瓶颈根源；在 Web 适配中，设计的低延迟追帧速率算法成功解决了播放卡顿与累积延迟的问题，极大地提高了 Web 端的流畅度。同时，通过联合配置 FRP 反向隧道与 Caddy 动态反向代理，实现了低成本且安全稳定的内网网络服务发布。本课程设计使我们掌握了计算机网络协议的应用、高并发编程和分布式部署的工程实践方法。

在未来的迭代中，我们计划进一步探究基于 WebRTC 协议的超低延迟（小于 500ms）连麦互动直播技术，并引入 GPU 硬件加速的 FFmpeg 转码服务以分担 CPU 的计算负载。

参考文献

- [1] THE GO AUTHORS. Go Documentation[EB/OL]. (2026). <https://go.dev/doc/>.
- [2] FLUTTER TEAM. Flutter Documentation[EB/OL]. (2026). <https://docs.flutter.dev/>.
- [3] SQLITE DEVELOPMENT TEAM. Write-Ahead Logging (WAL) in SQLite[EB/OL]. (2026). <https://www.sqlite.org/wal.html>.
- [4] JOY4 AUTHORS. joy4: Go audio/video library and streaming server[EB/OL]. (2021). <https://github.com/nareix/joy4>.
- [5] FFMPEG DEVELOPERS. FFmpeg Documentation[EB/OL]. (2026). <https://ffmpeg.org/documentation.html>.
- [6] BILIBILI DEVELOPERS. flv.js: HTML5 FLV Player[EB/OL]. (2024). <https://github.com/bilibili/flv.js>.
- [7] FATEDIER. frp: A fast reverse proxy to help you expose a local server behind a NAT or firewall to the internet[EB/OL]. (2026). <https://github.com/fatedier/frp>.
- [8] CADDY SERVER AUTHORS. Caddy Documentation[EB/OL]. (2026). <https://caddyserver.com/docs/>.