

# 基于 Media Source Extension 的直播系统的研究

王金环<sup>1</sup>, 李宝敏<sup>2</sup>

(1. 西安培华学院 智能科学与信息工程学院计算机系, 陕西 西安 710125;

2. 西安工业大学 计算机学院, 陕西 西安 710021)

**摘要:**在浏览器中,所有的数据交换都必须遵守 HTTP 协议,数据必须在完全传输完毕后才能由 JavaScript 进行处理。在这种情况下,是无法来实现实时的流数据传输。对于传统的云直播服务则需要提供视频云转码的功能,以便于兼容各种设备。随着 WebSocket 技术的开发应用,借助该技术,可以在浏览器上实现流传输,使得视频实时直播成为了可能。该系统实践了这个过程,使用 JavaScript 在浏览器上对视频流进行 De-multiplexing、Decoding、Encoding、Multiplexing,并根据浏览器对视频格式的支持情况,针对直播云服务器提供的视频转发功能,基于最新的接口,将视频的解码、编码、渲染等操作集成到客户端,通过自动选择视频流的处理方式,最后实现实时直播。视频直播服务器不再需要处理视频流,客户端通过 WebSocket 接收视频数据并根据浏览器所支持的编码格式进行重编码。该系统的实践具有直播带宽要求低、服务端低配置、低延迟、无法盗链等优点。

**关键词:**WebSocket; 视频编码、解码; 信号处理; JavaScript; 云服务器

中图分类号: TP399

文献标识码: A

文章编号: 1673-629X(2020)12-0210-06

doi:10.3969/j.issn.1673-629X.2020.12.037

## Research on Making Live System Based on Media Source Extension

WANG Jin-huan<sup>1</sup>, LI Bao-min<sup>2</sup>

(1. Department of Intelligent Science and Information Engineering, Xi'an Peihua University, Xi'an 710125, China;

2. School of Computing, Xi'an Technological University, Xi'an 710021, China)

**Abstract:** In the browser, all data exchange must comply with the HTTP, and the data must be processed by JavaScript after it is completely transmitted. In this case, it is impossible to achieve real-time streaming data transmission. For traditional cloud live broadcast services, the video cloud transcoding function needs to be provided in order to be compatible with various devices. With the development and application of WebSocket technology, with this technology, streaming can be realized on the browser, making it possible for live video to be broadcast live. This system has practiced this process, using JavaScript to perform De-multiplexing, Decoding, Encoding, and Multiplexing on the browser, and according to the browser's support for the video format, based on the video forwarding function provided by the live cloud server, the video decoding, encoding, rendering and other operations are integrated into the client based on the latest interface. By automatically selecting the processing method of the video stream, the real-time live broadcast is finally achieved. The video live server no longer needs to process the video stream. The client accepts the video data through WebSocket and re-encodes it according to the encoding format supported by the browser. The practice of this system has the advantages of low live broadcast bandwidth requirements, low server configuration, low latency, and inability to steal the chain.

**Key words:** WebSocket; video encoding and decoding; signal processing; JavaScript; cloud server

## 0 引言

目前,网页上缺乏能够替代 Flash 技术的直播技术,随着 Flash 技术的淘汰,Apple 公司的 HLS 技术延迟过高,网页实时直播就成了技术上的空白区域。该系统的研究填补了这片空白,得以解决网页上直播困

难的问题。

同时,弹幕作为一种新兴的交流方式,已经被大众广为接受。该系统整合了弹幕与实时直播,结合系统低成本的优点,允许任何没有技术能力的人可以便捷地搭建自己的直播云平台。

收稿日期:2020-01-19

修回日期:2020-05-20

基金项目:陕西省 2018 年度陕西省教育专项科研计划项目(18JK1085)

作者简介:王金环(1979-),女,硕士研究生,副教授,研究方向为计算机网络、语义网、数据挖掘、图像处理、模式识别等。

传统的云直播服务<sup>[1-2]</sup>需要提供视频云转码的功能,以便于兼容各种设备。而该项目的直播云服务器提供的视频转发功能<sup>[3]</sup>,则基于最新的接口,将视频的解码、编码、渲染等操作集成到客户端,这样做不仅能降低服务端的硬件需求,更能降低视频直播延迟。因此,该项目的核心意义在于:降低直播服务端的硬件需求,将计算压力分散到客户端,是一种新型的直播方式。基于 WebSocket<sup>[4]</sup>与 HTML 5 的 Uint8Array 以及 Media Source Extension API,为用户提供高实时性的视频互动直播,相对于其他直播系统而言,成本低,延迟少,进而将“弹幕”这种新兴高效的交流方式推广给大众。

## 1 系统概述

### 1.1 研究内容

(1)通过 JavaScript<sup>[5-6]</sup>与 WebSocket 处理二进制数据流;

(2)通过 JavaScript 对视频进行 De-multiplexing、Decoding、Encoding、Multiplexing;

(3)通过网页显示 RGB 图像信息;

(4)通过网页 API 对 De-multiplexing 后的视频封包进行 Decoding 并显示;

(5)通过 WebSocket,将视频弹幕广播给观众。

### 1.2 拟解决的关键问题

通过 JavaScript 对视频进行解码与重编码<sup>[7]</sup>,自动判断浏览器所支持的视频格式,然后重新编码成为浏览器所支持的格式。

由于 JavaScript 目前已经加入了 Uint8 Array 的数据类型,拥有了对字节数据流的处理能力,基于以上前提,服务端可以通过 WebSocket 下发视频数据包,由服务端解包,并根据浏览器所支持的格式解码,提供给浏览器。

关于视频的解码部分,该系统的工作流程为:首先,所有的视频流都是由推流端发送的原样数据,因此,服务端必须存储该视频流的格式信息,以便给客户端选择 Decoder。服务端在客户端连接后,立刻先发送视频流的格式信息,然后发送视频流的元数据信息。客户端根据视频流的格式信息选择 Decoder,然后通过 Decoder 解码视频流的元数据,再根据元数据中的视频编码信息选择 Codec,最后,通过 Codec 对视频流的每一个 Packet 进行解码,实现视频的解码。

而关于判断浏览器所支持的编码格式方面,该系统采用了遍历测试的方式,对所有系统中支持的编码进行逐一检测。检测的方式是通过生成一段有效的某一编码的视频数据,然后交给浏览器播放,等待一段时间后,判断该视频是否开始播放,如果没有开始播放,

则说明浏览器不支持这种编码。对于所有编码都不支持的浏览器,该系统将视频流解码为 RGB 数据,由浏览器的 canvas 绘制出来,实现视频的播放。

对于弹幕数据,首先,弹幕数据需要进行敏感字审核才能在客户端显示,但为了保证服务端的最低硬件需求,服务端会下发所有的敏感字列表到客户端;其次,由于 WebSocket 的高实时性,系统依然使用 WebSocket 作为弹幕数据的传输方式,由其他客户端将弹幕的文字、颜色、位置数据格式化为 JSON 消息,发送到服务端,服务端通过 WebSocket 转发给其他客户端,客户端根据关键字列表自动审核。

## 2 傻瓜式弹幕直播系统的总体架构

(1)推流端与服务端进行直播鉴权;

(2)推流端选择视频封装格式;

(3)推流端将封装格式发送到服务端;

(4)推流端对推流设备摄像头上的图像数据进行编码压缩 Encoding;

(5)推流端将 Encoded 的数据进行 Multiplexing 打包;

(6)推流端将 Multiplexed 的视频封包发送到服务端;

(7)服务端接受视频封包格式并采用这种格式对后面发来的视频封包进行 Decoding;

(8)服务端根据推流端的鉴权数据,首先获取第一个封包元数据封包并保存;

(9)服务端把视频封包以及音频封包根据直播鉴权信息,将数据包转发到其他相同通道的客户端;

(10)客户端根据浏览器兼容性选择重编码格式;

(11)客户端根据服务端发来的视频封装格式进行 Decoding;

(12)客户端对视频数据进行 De-Multiplexing;

(13)客户端转码;

(14)由 Media Source Extension API 将视频数据提交给浏览器,由浏览器播放;

(15)由客户端对用户发送的弹幕信息进行 JSON 编码,并发送到服务器;

(16)服务器下发给其他客户端;

(17)其他客户端进行 JSON 解码,然后根据数据显示弹幕。

## 3 编码解码流程概述

视频文件的解码分为两个步骤<sup>[8]</sup>,首先,将原始图像信息进行压缩 Encoding,然后,将压缩后的数据封装入数据包 Multiplexing。

该系统默认采用 MPEG-Video<sup>[9]</sup>进行压缩,按照

MPEG 格式进行编码<sup>[10-11]</sup>。

### 3.1 MPEG-Video 视频压缩算法

#### 3.1.1 颜色压缩

首先, MPEG-Video 规定, 在压缩之前, 必须将图像像素格式转换为  $Y'CbCr$  格式( $Y'$  = 明度,  $Cb$  = 蓝色色度,  $Cr$  = 红色色度)。由于大部分设备摄像头的图像数据均为 RGB 格式, 因此, 这个转换必不可少。

$Y'CbCr$  信号被称为  $YPbPr$ ,  $YPbPr$  信号是通过如下公式定义的:

$$Y' = K_R \cdot R' + K_G \cdot G' + K_B \cdot B' \quad (1)$$

$$P_B = \frac{1}{2} \cdot \frac{B' - Y'}{1 - K_B} \quad (2)$$

$$P_R = \frac{1}{2} \cdot \frac{R' - Y'}{1 - K_R} \quad (3)$$

其中,  $R'G'B'$  表示应用了 Gamma 矫正后的 RGB 值,  $K_R K_G K_B$  是定义的 RGB 颜色空间, 并且满足:  $K_R + K_G + K_B = 1$ 。

明度与色度是分离储存的, 色度需要以 4 : 2 : 2 进行抽样, 抽样后的图像大小会变为原来的四分之一。色度抽样的原理是由于人眼相对于色度, 对明度更加敏感, 所以减少色度信息可以在保持画面质量微降的情况下大幅降低图像尺寸。不同于红绿蓝三原色的颜色表示法, 在视频领域中通常使用明度和两个色度通道来表示颜色, 色度和明度是由 Gamma 矫正后的  $R'G'B'$  分量的加权和形成的。因此, 明度与亮度并不相同。4 : 2 : 0 是指, 在明度上, 分辨率是 100% 的, 在色度上, 水平方向 50% 分辨率, 垂直方向 50% 分辨率, 由于人眼对色度不如亮度敏感, 这样的压缩并不会太多地降低画面质量, 但能将图像尺寸大幅降低。示例见图 1, 其中, 灰圈代表  $CbCr$  颜色像素, 白圈代表亮度像素。

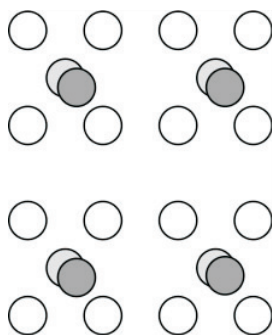


图 1 4 : 2 : 0 颜色抽样示例

#### 3.1.2 帧压缩

MPEG-Video 有多种不同的帧以便应对不同的情况。

##### (1) I-frames。

I-frames 是关键帧, 它保存了完整的图像信息, 在 seek 视频进度的时候, 只能 seek 到最附近的关键帧,

因为只有关键帧才保存了完整的图像信息。

##### (2) P-frames。

P-frames 是 Predicted-frame 的缩写, 还被称为前向预料帧, P-frame 不保存完整的图像信息, 只保存与前一帧的差异信息。

##### (3) B-frames。

B-frames 是 Bidirectional-frame 的缩写, 还被称为后向预料帧, B-frame 与 P-frame 非常类似, 但 B-frame 保存了自己与前一帧和后一帧的差异信息。

##### (4) D-frames。

D-frames 也是一种关键帧, 但是它的画面经过了非常严重的有损压缩, 在播放的时候会跳过 D-frame, 但在 seeking 的时候, D-frame 会用来显示当前 seek 的画面。主要用途是在 seeking 的时候既能让用户看到当前画面, 又能节约预览图像所花费的时间。

#### 3.1.3 DCT 图像压缩

每 8×8 像素的块会被应用离散余弦变换 (DCT), 离散余弦变换类似于只使用实数的离散傅里叶变换, 然后再消除变换后的小的高频信息就可以得到压缩后的图像数据<sup>[12]</sup>。图 2 是一个编码后的 8×8 DCT 块示例。

-415	-30	-61	27	56	-20	-2	0
4	-22	-61	10	13	-7	-9	5
-47	7	77	-25	-29	10	5	-6
-49	12	34	-15	-10	6	2	2
12	-7	-13	-4	-2	2	-3	3
-8	3	2	-6	-2	1	4	2
-1	0	0	-2	-1	-3	4	-1
0	0	-1	-4	-1	0	1	2

图 2 编码后的 8×8 DCT 块示例

通常来说, DCT 是一个线性的, 可反的函数  $f: R^N \rightarrow R^N$  (其中  $R$  是实数集), 也可以说是一个可反的  $N \times N$  的矩阵。它们都是根据下面的某一个公式  $n$  个实数  $x_0, x_1, \dots, x_{n-1}$  变换到另外  $n$  个实数  $f_0, f_1, \dots, f_{n-1}$  的操作:

DCT-I:

$$X_k = \frac{1}{2} (x_0 + (-1)^k x_{N-1}) + \sum_{n=1}^{N-2} x_n \cos\left(\frac{\pi}{N-1} nk\right) \quad (4)$$

$$k = 0, 1, \dots, N-1$$

DCT-I 的边界条件是:  $x_k$  相对于  $k = 0$  点, 偶对称, 并且相对于  $k = n-1$  点偶对称; 对  $f_m$  的情况也类似。

DCT-II:

$$f_m = \sum_{k=0}^{n-1} x_k \cos\left[\frac{\pi}{n} m\left(k + \frac{1}{2}\right)\right] \quad (5)$$

DCT-II 的边界条件是:  $x_k$  相对于  $k = -\frac{1}{2}$  点偶对

称,并且相对于  $k = n - \frac{1}{2}$  点偶对称;对  $f_m$  相对于  $m = 0$  点偶对称,并且相对于  $m = n$  点奇对称。

DCT-III:

$$f_m = \frac{1}{2}x_0 + \sum_{k=1}^{n-1} x_k \cos\left[\frac{\pi}{n}\left(m + \frac{1}{2}\right)k\right] \quad (6)$$

DCT-III 的边界条件是:  $x_k$  相对于  $k = 0$  点偶对称,并且相对于  $k = n$  点奇对称;对  $f_m$  相对于  $m = -\frac{1}{2}$  点偶对称,并且相对于  $m = n - \frac{1}{2}$  点偶对称。

DCT-IV:

$$f_m = \sum_{k=0}^{n-1} x_k \cos\left[\frac{\pi}{n}\left(m + \frac{1}{2}\right)\left(k + \frac{1}{2}\right)\right] \quad (7)$$

DCT-IV 边界条件是:  $x_k$  相对于  $k = -\frac{1}{2}$  点偶对称,并且相对于  $k = n - \frac{1}{2}$  点奇对称;对  $f_m$  类似。

### 3.2 多路复用

在视频编码领域,一般采用分时复用,在不同时间发送不同类型的数据,以便实现多路复用的功能。

MPEG 的 Program Stream 是 MPEG 的多路复用方式。关于 Program Stream 的定义如下:

(1) 协议头。

首先,每一个 Stream 都必须由一个 32 位的起始码开头,第 0 到第 3 字节是起始码前缀,第 4 字节为 Stream ID。

(2) 传输数据包。

传输数据视频包有: Sync byte, Transport Error Indicator (TEI), Payload Unit Start Indicator (PUSI), Transport Priority, PID, Transport Scrambling Control (TSC), Adaptation field control, Continuity counter, Adaptation field, Payload Data。

## 4 视频渲染

### 4.1 媒体源扩展

Media Source Extension 允许 JavaScript 从 <video> 标签或 <audio> 标签动态地构建媒体流<sup>[13-14]</sup>。并定义了一个可以用来作为媒体数据源的 Media Source 对象[]。Media Source 对象拥有一个 Source Buffer 对象,可以用来适配一些基于系统性能以及其他参数的附加的数据。Source Buffer 中的数据将会被当作媒体数据被解码和播放<sup>[15]</sup>。具体流程如图 3 所示。

该项目通过将视频文 Re-Multiplexing 为浏览器所支持的格式,并提供给浏览器播放的方式来实现视频文件的渲染。

Media Source Extension 定义了如下几个接口:

MediaSource: 代表被 HTMLVideo/Audio 标签所播放的媒体源对象;

SourceBuffer: 代表传递给 HTML Video /Audio 的一部分媒体数据;

SourceBufferList: 一个 Source Buffer List 列表;

VideoPlaybackQuality: 包含了被 Video/Audio 标签所播放的媒体的质量信息,例如被抛弃的或不正确的帧的数量等;

TrackDefault: 提供关于 SourceBuffer 的类型、标签、语言等信息;

TrackDefaultList: TrackDefault 的列表。

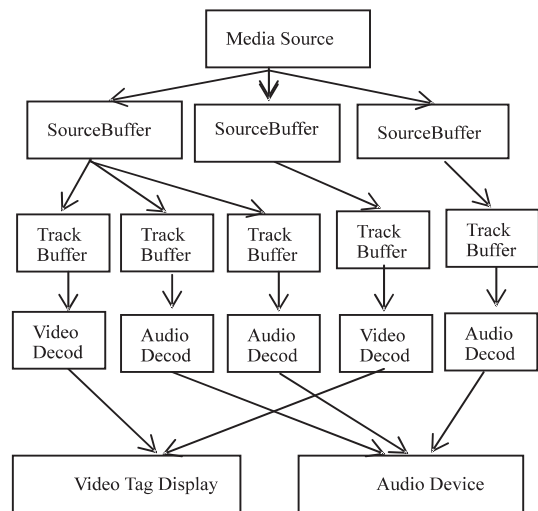


图 3 Media Source Extension 的工作示例

MediaSource 代表了被播放的媒体源,对象可以附加到一个 HTML Video/Audio 标签上。媒体源到事件目标的关系如图 4 所示。

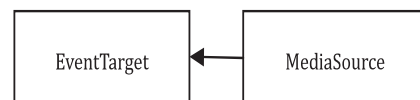


图 4 媒体源到事件目标的关系

#### 4.1.1 构造函数

MediaSource(): 构造一个新的 Media Source 对象<sup>[16-17]</sup>。

#### 4.1.2 属性

MediaSource.sourceBuffers: 一个只读属性值, 返回了 SourceBufferList。

MediaSource.activeSourceBuffers: 一个只读属性值, 返回了所有的被激活的 Source BufferList。

MediaSource.readyState: 一个只读属性值, 返回了一个表示当前媒体源的状态, 有三种值: closed 表示媒体尚未打开或已经关闭, open 表示媒体已经打开, ended 表示媒体已经播放完成。

MediaSource.duration: 获取或设置当前所播放的媒体源的长度。



#### 4.1.3 事件

`MediaSource. onsourceclose`: 媒体源关闭的时候会触发。

`MediaSource. onsourceended`: 媒体源结束的时候会触发。

`MediaSource. onsourceopen`: 媒体源打开的时候会触发。

#### 4.1.4 方法

`MediaSource. addSourceBuffer()`: 通过一个 MIME 类型创建一个新的 Source Buffer, 并且添加到 Source BufferList。

`MediaSource. removeSourceBuffer()`: 从 Source BufferList 中删除一个 Source Buffer。

`MediaSource. endOfStream()`: 结束一个媒体流。

`MediaSource. setLiveSeekableRange()`: 设置用户可以拖动的时间线的范围。

`MediaSource. clearLiveSeekableRange()`: 清空用户可拖动的时间线的范围。

#### 4.1.5 静态方法

`MediaSource. isTypeSupported()`: 输入 MIME 类型, 返回浏览器是否支持该类型。

### 4.2 Canvas Based Video Rendering

对于不支持 Media Source Extension 的浏览器, 或找不到适合的编码格式的浏览器, 该项目将视频转换为 rgb 数据, 提供给 Html 的 Canvas, 通过 Canvas 来绘制视频帧, 实现视频渲染。

该项目通过 `createImageData` 来创建帧, 然后通过 `canvas` 的 `data` 属性赋值的方式在图片上绘制像素, 最后, 通过 `drawImage` 来实现在 Canvas 上画图。最终实现视频的渲染。

### 4.3 WebGL Based Video Rendering

WebGL 是一套浏览器端的硬件图形 API, 它提供了类似于 OpenGL、DirectX 的功能, 使通过 JavaScript 运行游戏成为可能。

WebGL 渲染视频可以加速视频的渲染速度, 然而不同于 Native 平台上的硬件渲染, 这里的 WebGL 渲染, 由于目前 WebGL 只能提供有限的图形 API, 所以只能作为视频像素格式的转换器, 但可以提供由 Pixel Shader 实现的硬件运算的更高效率的视频滤镜。

## 5 视频传输

WebSocket 在浏览器与服务器之间传输流媒体<sup>[18]</sup>, 通过 HTTP 或 RTMP 协议在推流端传输流媒体。

WebSocket 协议:

### 5.1 握手

客户端发送握手请求, 服务端返回握手回复, 握手协议大致如下:

客户端发送 HTTP 请求, 并带有 Upgrade、Connection、Sec-WebSocket-Key、Sec-WebSocket-Protocol、Sec-WebSocket-Version 这五个字段, 服务端会根据 Sec-\* 的三个字段, 确定 WebSocket 的协议版本和加密 Key。

客户端请求:

GET /chat HTTP/1.1

Host: server.example.com

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Key: x3JJHMbDL1EzLk9 GBh

XDw==

Sec-WebSocket-Protocol: chat, superchat

Sec-WebSocket-Version: 13

Origin: http://example.com

服务端返回:

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm

5OPpG2HaGWk=

Sec-WebSocket-Protocol: chat

### 5.2 字段说明

Connection 必须设置 Upgrade, 表示客户端希望连接升级。

Upgrade 字段必须设置 WebSocket, 表示希望升级到 WebSocket 协议。

Sec-WebSocket-Key 是随机的字符串, 服务器端会用这些数据来构造出一个 SHA-1 的信息摘要。把“Sec-WebSocket-Key”加上一个特殊字符串“258EAF5 - E914 - 47DA - 95CA - C5AB0 DC 85B11”, 然后计算 SHA-1 摘要, 之后进行 BASE-64 编码, 将结果作为“Sec-WebSocket-Accept”头的值, 返回给客户端。如此操作, 可以尽量避免普通 HTTP 请求被误认为 WebSocket 协议。

Sec-WebSocket-Version 表示支持的 WebSocket 版本。RFC6455 要求使用的版本是 13, 之前草案的版本均应当弃用。

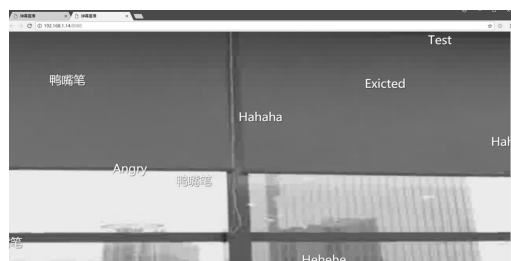
Origin 字段是可选的, 通常用来表示在浏览器中发起此 WebSocket 连接所在的页面, 类似于 Referer。但是, 与 Referer 不同的是, Origin 只包含了协议和主机名称。

其他一些定义在 HTTP 协议中的字段, 如 Cookie

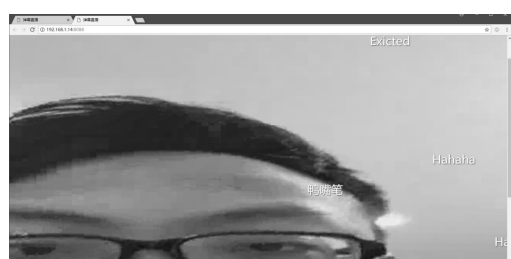
等,也可以在 WebSocket 中使用。

## 6 实验效果

电脑网页端播放效果如图 5 所示。



(a)

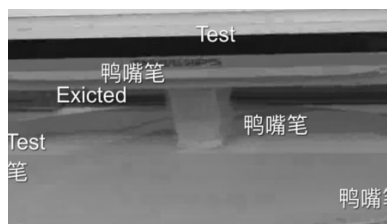


(b)

图 5 电脑上播放视频的效果  
IOS 手机端播放效果如图 6 所示。



(a)



(b)

图 6 手机上播放的效果

## 7 结束语

该项目实现了手机上媒体的编码,基于 Web-Socket 的流媒体传输,以及网页上通过 JavaScript 将流媒体解码,并且通过 Media Source Extension 的 API 将解码的数据发送给浏览器等功能。主流的手机都可以

流畅地编码解码视频,并且带宽要求相对于 HLS 协议更低。

但是压缩算法还有一些优化空间,如果能进一步优化压缩算法,就能让用户在带宽更低的环境中流畅播放视频,并减小服务器的带宽压力;此外该项目还缺乏对足够多的视频格式的支持,如果支持更多的格式,推流端的格式选择就会更加自由。

### 参考文献:

- [1] 张晓洲. 云计算关键技术及发展现状研究[J]. 网络与信息, 2011, 25(9): 36-37.
- [2] 陈全, 邓倩妮. 云计算及其关键技术[J]. 计算机应用, 2009, 29(9): 2562-2567.
- [3] 张强. 云计算时代的平台战略[J]. 电脑知识与技术, 2010, 6(6): 1350-1352.
- [4] 阮一峰. WebSocket 教程[EB/OL]. 2017-05-15. [www.ruanyfeng.com/blog/2017/05/websocket.html](http://www.ruanyfeng.com/blog/2017/05/websocket.html).
- [5] FLANAGAN D. JavaScript 权威指南[M]. 北京: 机械工业出版社, 2007.
- [6] ZAKAS N C. JavaScript 高级程序设计[M]. 李松峰, 曹力, 译. 第3版. 北京: 人民邮电出版社, 2012.
- [7] 张春元, 文梅, 苏华友, 等. 流计算和视频编码[M]. 北京: 科学出版社, 2012.
- [8] 朱秀昌, 刘峰, 胡栋. 视频编码与传输新技术[M]. 北京: 电子工业出版社, 2014.
- [9] 尼古拉斯. JavaScript 面向对象精要[M]. 北京: 人民邮电出版社, 2015.
- [10] THEISEN K J. Programming languages in chemistry: a review of HTML5 /JavaScript[J]. Journal of Cheminformatics, 2019, 11: 1-19.
- [11] 惠苗, 赖道健. 基于 WebSocket 协议的即时通讯系统的开发[J]. 榆林学院学报, 2019, 29(6): 76-79.
- [12] 张杰. 视讯技术—中小型视频监控系统[M]. 西安: 西安电子科技大学出版社, 2018.
- [13] 齐俊杰, 胡洁, 麻信洛. 流媒体技术入门与提高[M]. 第2版. 北京: 国防工业出版社, 2009.
- [14] 罗祥远. 流媒体技术应用教程[M]. 北京: 清华大学出版社, 2014.
- [15] 姚建东, 施云青, 胡庆喆, 等. 新媒体编播技术与应用[M]. 北京: 清华大学出版社, 2018.
- [16] 周明全, 耿国华, 韦娜. 基于内容图像检索[M]. 北京: 清华大学出版社, 2007.
- [17] 徐锋, 陈暄. UML 面向对象建模基础[M]. 北京: 中国水利水电出版社, 2006.
- [18] 钟玉琢, 向哲, 沈宏. 流媒体和视频服务器[M]. 北京: 清华大学出版社, 2003.