Computer Science and Application 计算机科学与应用, 2022, 12(6), 1641-1657
Published Online June 2022 in Hans. http://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2022.126165

Hans汉斯

基于H.265的低延迟优化策略研究

洪宇光[1]，叶兴扬[2]，潘丽浓[1]，余维栋[3]

[1]宁波海关技术中心，浙江 宁波
[2]杭州电子科技大学电子信息学院，浙江 杭州
[3]慈溪市质量技术监督检验检测服务中心，浙江 慈溪

收稿日期：2022年5月24日；录用日期：2022年6月22日；发布日期：2022年6月29日

摘　要

在有限的网络带宽下，传统的视频采集设备面临压缩率不足、帧率低、延迟高等技术瓶颈，这使得它们无法应用于超清图像的传输场景。因此，面对不断提升的视频质量，提高视频传输系统的编解码效率、降低传输延迟显得尤为重要。本文通过多种低延迟策略对视频传输系统整体延迟时间进行优化，测试结果表明：局域网内，在满足编码质量的前提下，对1080 P质量的视频进行传输，系统端到端的网络传输延迟时间为127 ms。与优化前的视频传输系统相比，优化后的视频传输系统总延迟时间减少了9.48%。

关键词

H.265，视频传输系统，低延时，拥塞控制

Research on Low Delay Optimization Strategy Based on H.265

Yuguang Hong[1], Xingyang Ye[2], Linong Pan[1], Weidong Yu[3]

[1]Ningbo Customs District Technology Center, Ningbo Zhejiang
[2]School of Electronic Information, Hangzhou Dianzi University, Hangzhou Zhejiang
[3]Cixi Quality and Technical Supervision, Inspection & Testing Service Center, Cixi Zhejiang

Received: May 24th, 2022; accepted: Jun. 22nd, 2022; published: Jun. 29th, 2022

Abstract

Under the limited network bandwidth, the traditional video acquisition equipment is faced with the technical bottleneck of insufficient compression rate, low frame rate and high delay, which makes it impossible for them to be applied in the ultra HD image transmission scene. Therefore, in

文章引用: 洪宇光, 叶兴扬, 潘丽浓, 余维栋. 基于 H.265 的低延迟优化策略研究[J]. 计算机科学与应用, 2022, 12(6): 1641-1657. DOI: 10.12677/csa.2022.126165

the face of constantly improving video quality, it is particularly important to improve the codec efficiency and reduce transmission delay of video transmission system. In this paper, the overall delay time of the video transmission system is optimized by a variety of low delay strategies. The test results show that the end-to-end delay time of the system is 127 ms when the 1080 P quality video is transmitted on the premise of satisfying the coding quality. Compared with the video transmission system before optimization, the total delay time of the video transmission system after optimization is reduced by 9.48%.

Keywords

H.265, Video Transmission System, Low Latency, Congestion Control

1. 引言

视频传输技术从早期的有线传输发展到现在的无线传输，伴随着 5G 网络的出现，我国对无线视频传输技术的发展又提出了更高的要求。图像的质量、码率、延迟等关键指标也需要更进一步的突破。这使得实时超清图像传输技术成为了研究重点。为了提升用户体验，图像的传输延迟需要进一步降低，这对视频编解码技术提出了更高的要求。H.265 编解码协议便在这样的需求环境下应运而生，它基于 H.264 编码框架进行开发，对多种帧内帧间预测算法进行优化改进，相比较于 H.264，压缩率提升了 50% [1]，码率降低了 50%。在高清、超高清视频编码需求不断增加的今天，H.265 编解码协议依靠其优秀的编码性能，逐步取代老一批的编解码协议，被广泛应用于各种实时视频传输系统中[2] [3]。

2. 系统设计与实现

2.1. 系统整体架构

系统采用经典 C/S 架构设计，由编码终端和解码终端共同组成。C/S 架构是一种比较早的软件架构，主要应用于局域网环境下，依托端到端直连的特点，能够保证较好的安全性和可靠性[4]。编码终端使用瑞芯微生产的 RK3399 芯片作为处理核心，根据功能需求搭载外围设备。解码端使用智能手机作为载体，采用开发 APP 的形式，通过代码层面做系统适配。系统架构图如图 1 所示。

2.2. 系统软件设计

编码终端需要实现调用 USB 摄像头获取 RGB 格式的原始视频数据，RGB 格式视频数据流无法直接应用于 H.265 压缩编码，需要先通过转码转换为 YUV420 的视频流，再经过 X.265 编码器进行压缩编码，压缩生成的数据一路通过 TCP 协议传输至 Android 手机，一路并行处理存储于 RK3399 的 SD 卡中，以备后续使用。解码终端需要通过开发 APP 来实现功能需求，基于 TCP 协议连接 RK3399 编码终端并接收 H.265 数据流，通过 MediaCodec 进行 H.265 解码，解码后的视频数据格式为 YUV420 格式，YUV420 格式的视频数据量较大，不利于控件的实时高帧率刷新显示。故需要将 YUV420 格式的数据流转码为 RGB 格式，转码之后通过 ImageView 控件进行渲染显示。系统软件功能流程图如图 2 所示。

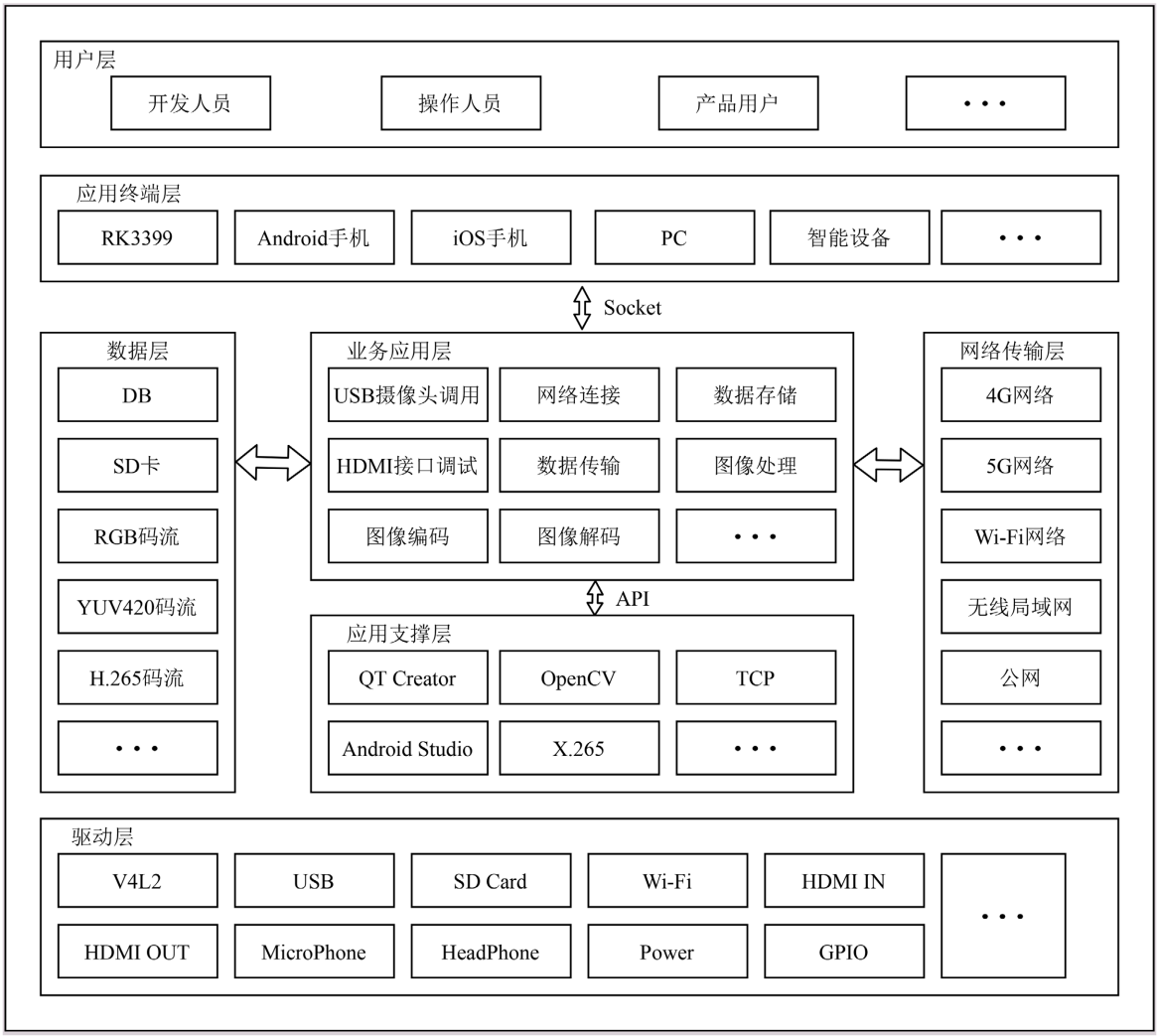


Figure 1. Overall system architecture diagram
图 1. 系统总体架构图

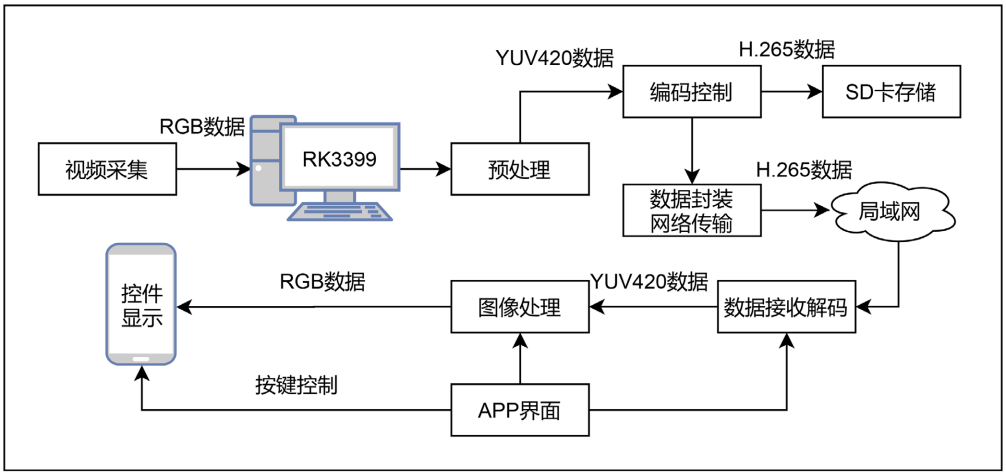


Figure 2. System software function flow chart
图 2. 系统软件功能流程图

编码终端由多个软件模块组成，其中视频采集模块作为编码终端软件起始模块，基于 V4L2 驱动进行实现。在软件功能的实现上，Linux 系统已经将 V4L2 驱动包装成了方便使用的 API 接口，通过调用对应接口函数，设置采集参数，就可以驱动采集设备的进行拍摄。编码模块使用 X265 编码器进行 H.265 编码，需要指定相应编码参数，系统初始化时设置的编码参数为：输入视频格式为 YUV420，视频分辨率为 1920*1080，帧率为 30 FPS，编码速率为 faster，码率分母为 30，码率分子为 1。系统的网络传输模块使用 Socket 通讯实现，协议选择可靠传输协议 TCP。编码终端作为 Socket 通讯的服务器使用，软件上需要提供 TCP 连接所需的端口号，并绑定同一个 IP 地址。通过指令开启监听，等待客户端的接入。存储模块用于存储编码后的 H.265 视频文件，写入 SD 卡时需要设置间隔时间为 33 ms。视频编码终端软件流程图如图 3 所示。

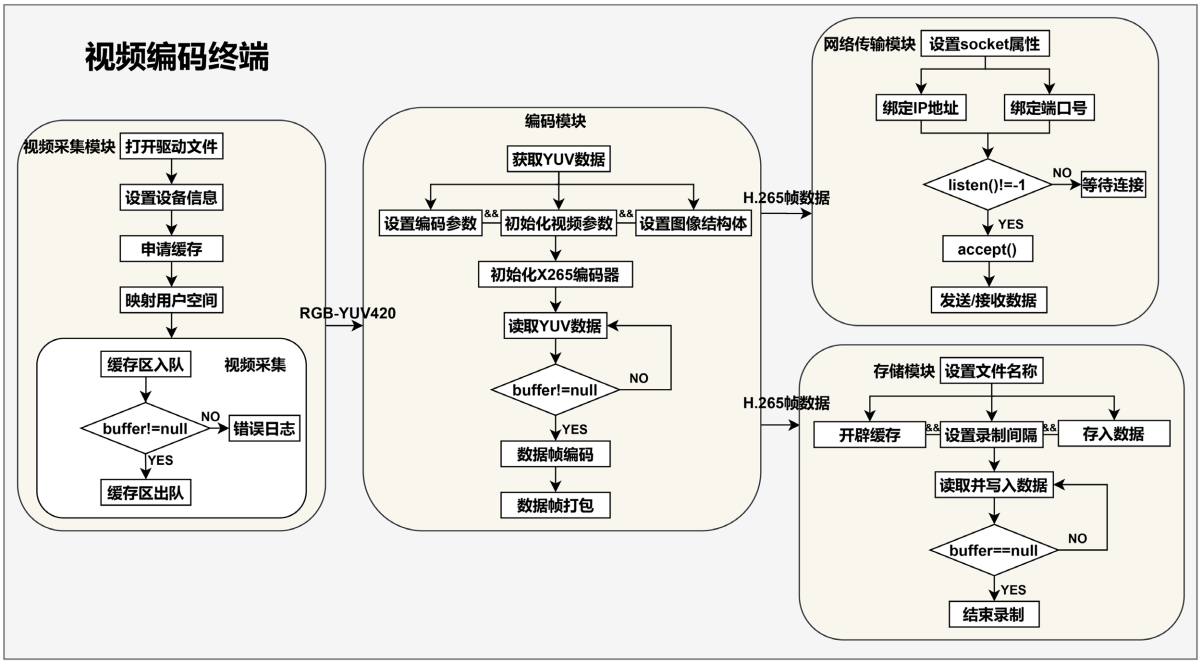


Figure 3. Coding terminal software flow chart
图 3. 编码终端软件流程图

解码终端采用搭载 Android 系统的智能手机作为硬件载体，使用 Android Studio 软件进行功能的开发与调试。其中视频接收模块作为解码终端的功能入口，需要实现网络连接与获取网络数据的功能。网络连接上基于 TCP 协议进行设计实现，与编码终端规定连接相同的 IP 地址和端口号，从而进行数据交互。解码模块使用 MediaCodec 进行 H.265 视频流的解码工作，MediaCodec 是 Android 提供的用于视频编解码的接口，通过对底层编解码器进行设置，与 MediaExtractor、MediaMuxer、AudioTrack 配合使用，可以高效实现 H.265 格式的编解码工作，且具有优秀的编解码效果。显像模块使用 ImageView 作为视频显示控件，通过 Bitmap 来承载 BGR 数据，之后将 Bitmap 通过 Handler 传递给主线程 Activity 进行渲染显示。解码终端软件流程图如图 4 所示。

2.3. 系统硬件设计

为了验证软件系统的功能性以及低延迟优化策略的适用性，故搭建视频传输系统。系统主要的硬件设计部分在于视频编码终端。视频编码终端由 RK3399 核心处理器以及扩展板组成。其中拓展板由视频

采集模块、音频采集模块、WiFi 网络通信模块、电源模块、SD 卡模块共同组成。总体硬件设计框图如图 5 所示。

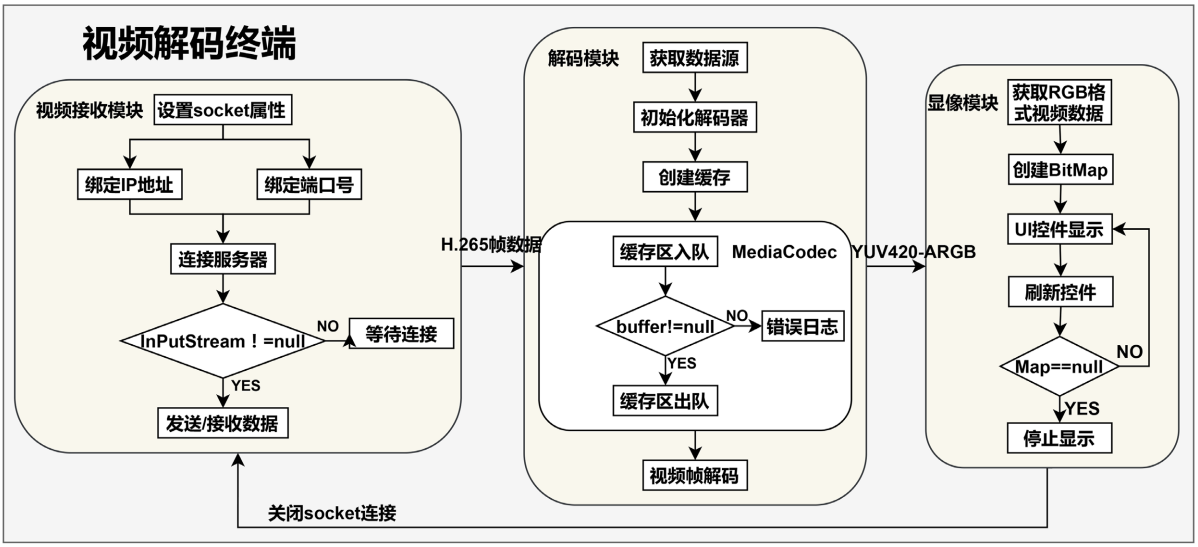


Figure 4. Decoding terminal software flow chart
图 4. 解码终端软件流程图

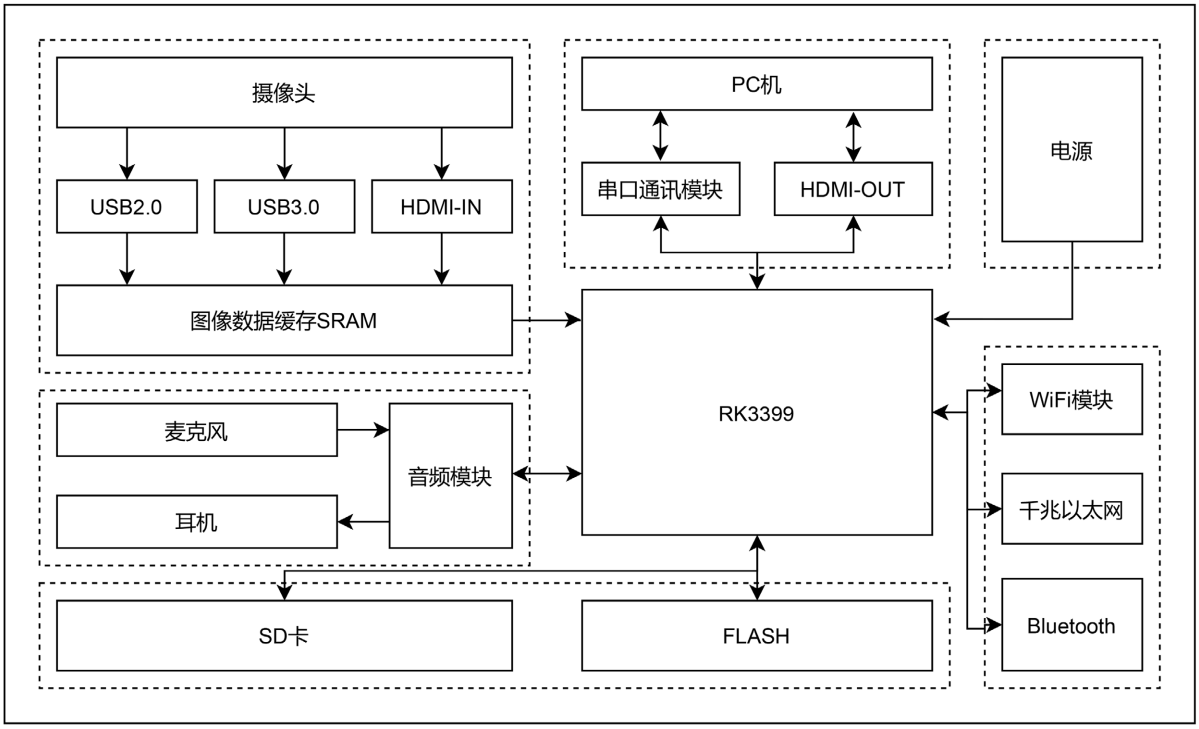


Figure 5. Video coding terminal hardware design block diagram
图 5. 视频编码终端硬件设计框图

系统采用瑞芯微 RK3399 芯片作为核心处理器，整个核心控制器模块由 RK3399 芯片以及必要的外围电路共同组成。核心控制器实物图如图 6 所示。

Figure 6. Physical drawing of core controller
图 6. 核心控制器实物图

视频采集模块采用 USB 接口模块进行设计，USB HUB 采用 VL817-Q7 芯片进行设计实现，VL817-Q7 是目前市场上主流的 4 端口 USB 3.0 的 HUB 方案。其芯片逻辑结构图如图 7 所示。

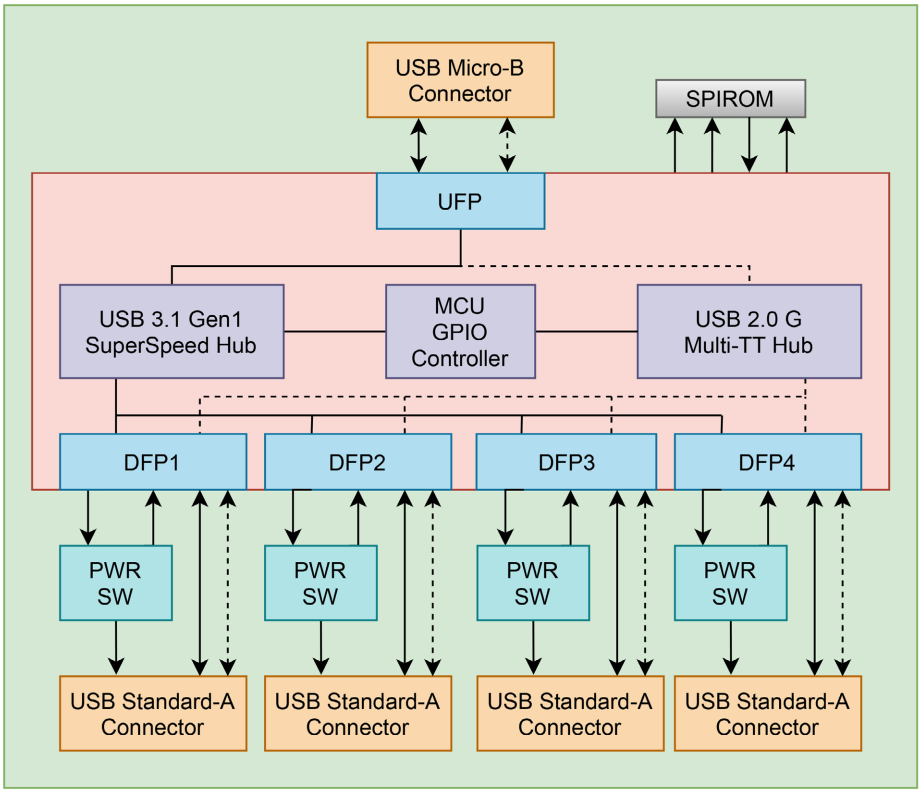


Figure 7. VL817-Q7 logical structure diagram
图 7. VL817-Q7 逻辑结构图

音频信号采集模块使用 ALC5651 芯片进行设计实现。ALC5651 是一个高性能，低功耗，双 I2S 接口的音频芯片。芯片的逻辑结构图如图 8 所示。

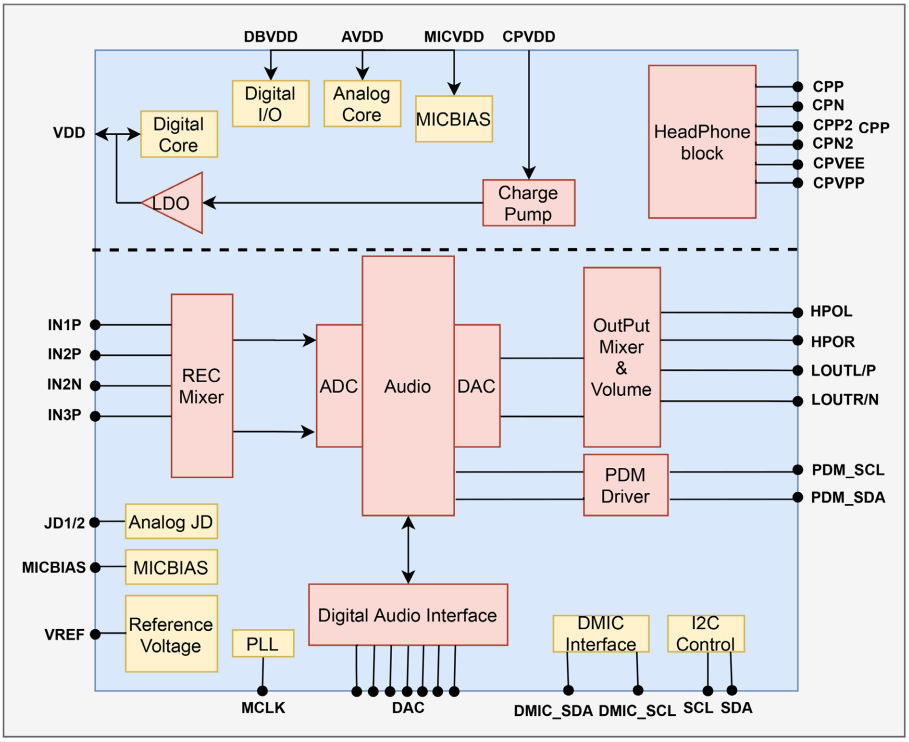


Figure 8. ALC5651 logical structure diagram
图 8. ALC5651 逻辑结构图

网络通信模块采用台湾 Realtek 公司生产的 RTL8188EUS 芯片进行设计实现。RTL8188EUS 具有功耗低，性能稳定的特点，且功率输出保持较好的线性状态。满足 IEEE802.11n 标准，同时兼容 IEEE802.11g、IEEE802.11b 标准。与所有满足该标准的无线设备均可以互联进行通讯。芯片逻辑结构图如图 9 所示。

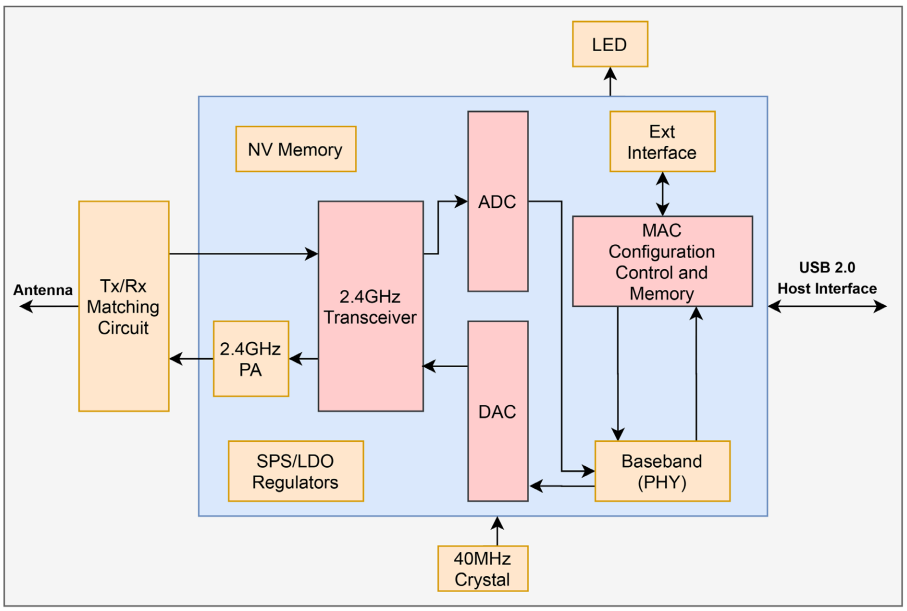


Figure 9. RTL8188EUS logical structure diagram
图 9. RTL8188EUS 逻辑结构图

存储模块需要放置 Linux 系统文件、应用程序以及 H.265 编码文件等需要长期保存的数据，故选择 SD 卡作为存储器，考虑到视频编码终端需要传输的数据量较大，故采用 SD 方式的 6 线模式进行电路设计实现。SD 方式下的逻辑结构图如图 10 所示。

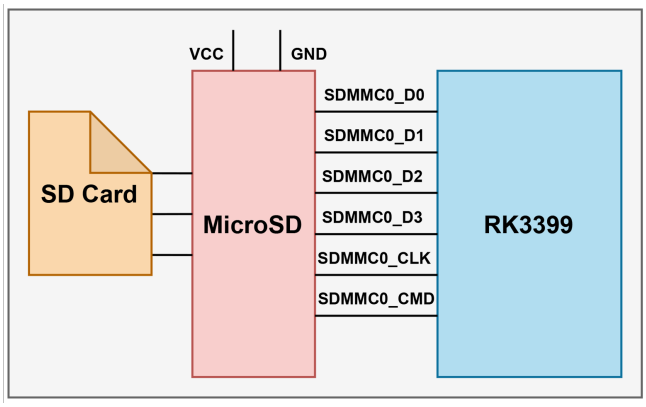


Figure 10. SD method logical structure diagram
图 10. SD 方式下的逻辑结构图

电源模块是视频编码终端工作的动力来源，NB679AGD 是 MPS 推出的一款大输出电流的高效同步整流的固定电压 DC-DC 转换器 IC，芯片采用 5.5 V~26 V 宽电压输入，输出电压为 5 V，最大输出电流 10 A。电源模块逻辑结构图如图 11 所示。

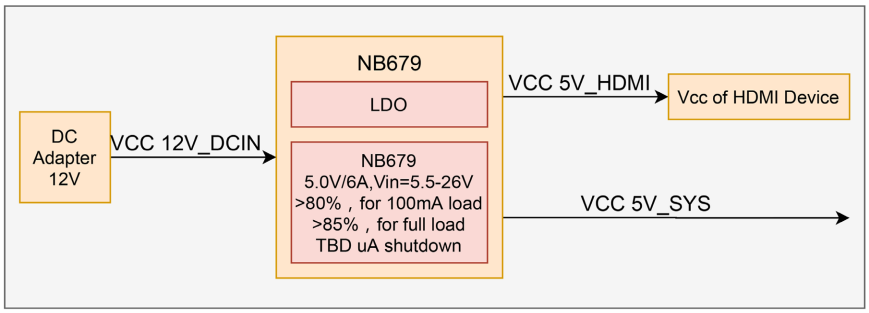
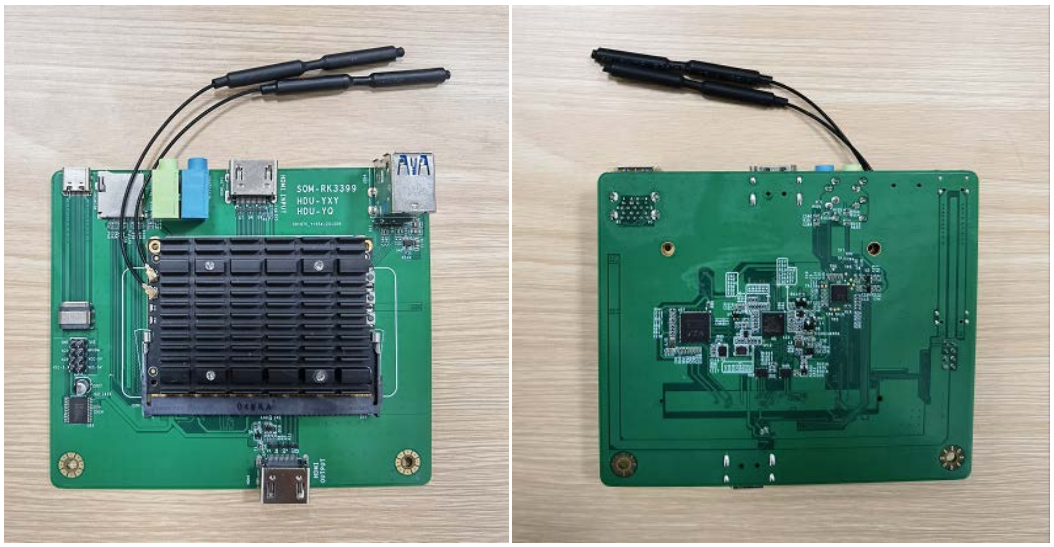


Figure 11. Power supply module logical structure diagram
图 11. 电源模块逻辑结构图

视频编码终端作为视频传输系统的主要硬件部分，通过设计制作拓展板来支持多种功能的拓展。设计过程中考虑到拓展板性能、体积、供电方式及使用场景等约束条件，将多种外围电路模块集成于 120 mm*100 mm*15 mm 大小的拓展板上，通过 5000 mA 锂电池进行供电，可以稳定工作 8 小时，具有体积小、方便携带的优点。视频编码终端实物图如图 12 所示。

3. 低延迟优化策略

系统从三个方向进行低延迟优化，分别为 X265 编码器配置优化，编码终端缓存优化和拥塞控制算法优化。X265 编码器配置优化用于提升编码器编码速率，减少帧间预测时间；编码终端缓存优化通过将数据结构应用于缓存设计，提升缓存区的利用率，加快数据流通速度；拥塞控制算法是 TCP 协议的核心算法，通过拥塞控制机制来保证传输可靠性，同时也是传输延迟的主要来源，通过对拥塞控制算法慢启动阶段进行优化控制，增加数据传输过程的稳定性，减少重传率，降低传输延迟。



(a) 正面　　(b) 反面

Figure 12. Physical picture of video coding terminal
图 12. 视频编码终端实物图

3.1. X265 编码器配置优化

X265 编码器使用时需要选择合适的编码器配置，其中 preset 参数与编码速度和质量有关，preset 参数包含十个级别，每个级别的 preset 对应一组编码参数。preset 定义了编码速度与编码质量的关系，通常编码速度越快，编码质量就会越差。在 X265 编码器中，preset 默认使用 medium 级别，为了提高编码速度，本文对 medium 和 faster 两种配置，选取六个经典测试序列进行测试。测试结果如表 1 所示。

Table 1. Comparison table of X265 in two configurations
表 1. X265 两种配置下的对比结果表

视频序列	medium			faster		
	BD-Rate	BD-PSNR	BD-V	BD-Rate	BD-PSNR	BD-V
Traffic	0.31%	−0.061 dB	1.21 FPS	0.49%	−0.092 dB	3.01 FPS
ParkScene	0.42%	−0.043 dB	2.43 FPS	0.62%	−0.106 dB	5.37 FPS
RaceHorses	0.54%	−0.044 dB	14.34 FPS	0.93%	−0.099 dB	27.23 FPS
BasketballPass	0.24%	−0.061 dB	61.21 FPS	0.65%	−0.112 dB	106.21 FPS
FourPeople	0.38%	−0.069 dB	5.92 FPS	0.70%	−0.092 dB	13.19 FPS
SlideEditing	0.51%	−0.072 dB	6.3 FPS	0.59%	−0.106 dB	10.05 FPS
AVERAGE	0.40%	−0.058 dB	15.24 FPS	0.66%	−0.101 dB	27.51 FPS

表 1 表明 X265 编码器在 medium 配置和 faster 配置下平均编码速度分别是 15.24 FPS 和 27.51 FPS，编码质量略微降低。在 medium 配置中 BD-PSNR 值虽然较低，但编码速度缓慢，不满足实时编码的需求。在 faster 配置中 BD-PSNR 值略微增加，但编码速度较快，对标每秒 30 FPS 的技术指标，基本满足实时编码需求。此外，使用 Inter VTune Amplifier XE 性能分析软件对视频序列 ParkScene 进行编码测试，获得 X265 编码器中各个关键技术的编码时间。各部分编码时间占比如图 13 所示。

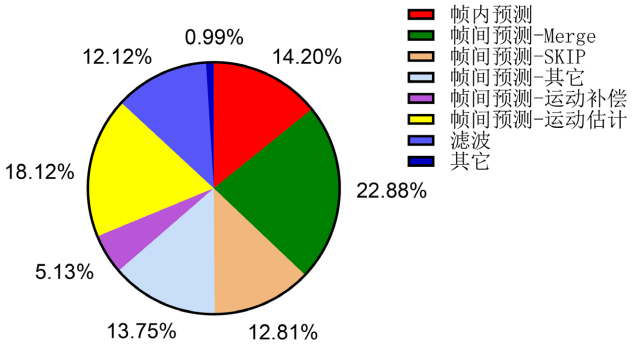


Figure 13. Ratio of encoding time to each part of the X265 encoder
图 13. X265 编码器各部分编码时间占比

可以看到帧内预测和滤波在总体编码时间中的占比为 26.32%，不足总编码时长的 1/3，帧间预测时间总和占总体编码时间的 72.69%。因此采用 faster 配置，加快编码器的编码速度，节省帧间预测时间，可以有效优化 X265 编码器在编码高清视频图像时编码速度较慢的问题，减少视频传输系统的延迟时间。

3.2. 编码终端缓存优化

编码终端使用 X265 编码器编码视频数据后，并不是直接通过 TCP 协议进行发送，而是先将数据存入缓存区中，在网络正常的前提下，不断从缓存区取出数据进行发送，同时释放已经发送的数据所占用的内存用于后续数据的缓存。这个过程能够正常运行的前提是缓存区足够大，这样才能保证数据在缓存区中流转时不会发生阻塞导致数据传输迟缓。但是一味地加大缓存并非明智之举，这会造成内存的浪费。编码终端在缓存区使用环形 FIFO (FIRST IN FIRST OUT)这种数据结构进行优化，其优点是可以保证缓存区内的数据按序入队出队，基于队列先进先出的特点，可以迅速清除已用的缓存，避免阻塞的发生[5]。在多线程调度中，提高了软件的并发能力和系统的运行效率，起到降低传输延迟的作用。

环形 FIFO 是将队列进行首尾相连所形成的数据结构，通过指针可以迅速判断队列中的数据情况。数据从队列头部进入，每插入一个数据，则整体往后移一位，当移动到尾部时，将转回到头部位置进行处理。这个转回操作通过取模来执行。代码实现上将数组的头部 q[0]和尾部 q[length-1]相连，指定两个指针 head 和 tail 分别指向可读的位置 q[x]和可写的位置 q[y]，其中 x 与 y 的区间均在[0, length-1]上。环形队列 FIFO 的关键是判断队列为空或者为满，当指针 tail 追上 head 时，证明队列已满，当指针 head 追上 tail 时，代表队列为空。队列不满，则写入数据。队列不为空，则读取数据。环形 FIFO 结构图如图 14 所示。

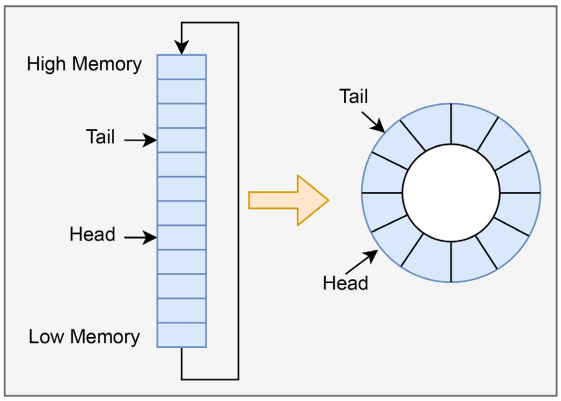


Figure 14. Annular FIFO structure drawing
图 14. 环形 FIFO 结构图

3.3. 拥塞控制算法优化

系统采用 TCP 协议作为网络传输协议，在视频传输系统中，协议的选择对传输延迟有着巨大的影响。拥塞控制算法在发生丢包重传时，会对慢启动门限值 ssthresh 和拥塞窗口 cwnd 进行参数调整，而这两个参数决定了传输数据量的大小。拥塞控制算法调度流程如图 15 所示。

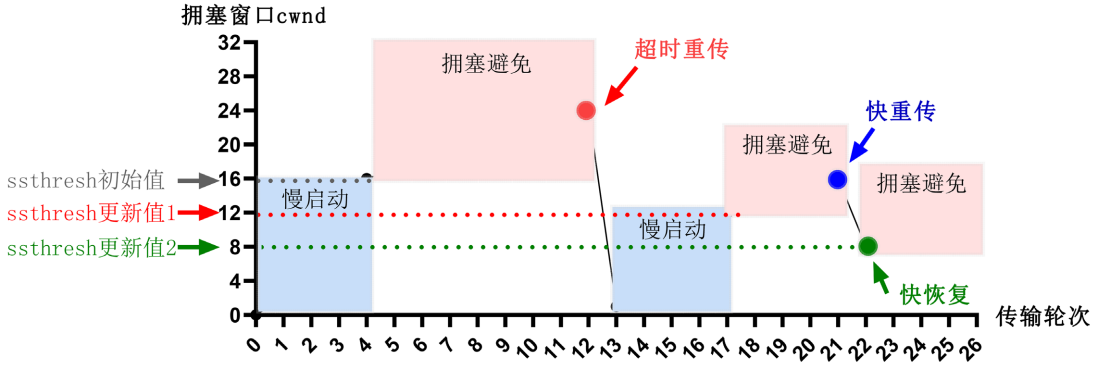


Figure 15. Congestion control algorithm scheduling process
图 15. 拥塞控制算法调度过程

拥塞控制算法可以按照丢包与延时进行划分，其中基于丢包的拥塞控制算法包括 TCPTahoe、TCPReno、TCPNewReno、HSTCP、STCP、HTCP 等，基于时延的拥塞控制算法主要是 TCPVegas。

TCPTahoe、TCPReno、TCPNewReno 的拥塞控制算法中：

慢启动状态下，发送端每收到一个 ACK，拥塞窗口 cwnd 累计加一，直到 cwnd 的值大于等于 ssthresh：

$$\text{cwnd} = \text{cwnd} + 1 \tag{1}$$

在一段时间后，拥塞窗口 cwnd 的值逐步增加，当其大于慢启动门限值 ssthresh 时，进入拥塞避免阶段：

$$\text{cwnd} = \text{cwnd} + \frac{1}{\text{cwnd}} \tag{2}$$

数据传输过程中出现丢包，则立即执行快重传，同时需要将慢启动门限值 ssthresh 降低，以应对数据量的变化，通常修改为拥塞窗口 cwnd 的一半，然后将 cwnd 置为 1：

$$\text{ssthresh} = \text{cwnd}/2 \tag{3}$$

$$\text{cwnd} = 1 \tag{4}$$

TCPReno 中增加了快恢复算法，对 TCPTahoe 进行了较大的优化。认定当收到 3 个重复 ACK 时，就假定网络产生了拥塞。然后进行快重传：

$$\text{ssthresh} = \max\left(2, \text{cwnd}/2\right) \tag{5}$$

$$\text{cwnd} = \text{cwnd}/2 \tag{6}$$

快重传之后不会重返慢启动阶段，而是进入快恢复阶段，当有 n 个重复 ACK (ndup)传输时：

$$\text{cwnd} = \text{ssthresh} + \text{ndup} \tag{7}$$

$$\text{cwnd} = \text{ssthresh} \tag{8}$$

TCPNewReno 继续对快恢复进行优化，相比较于 TCPReno 和 TCPNewReno，在收到部分 ACK 时，不会立刻退出快恢复阶段，而是等到数据包全部接收到为止。这使得当有大批数据包丢失时无需等待超

时重传就可以提前更正错误，提高了链路的利用率。

HSTCP 不同于上述的三种算法，通过在传统拥塞控制算法基础上对参数进行减小和增大的修正，让窗口一直运行在一定的数值范围内，提高了带宽利用率。当进入拥塞避免阶段时，若 cwnd 处于[ssthresh, W_{low}]区间内：

$$\text{cwnd} = \text{cwnd} + a(\text{cwnd}) \tag{9}$$

W_{low} 为 HSTCP 的门限值，通常取值为 38。若 cwnd > W_{low} 时，证明出现丢包，此时减小 cwnd 值：

$$\text{cwnd} = (1 + b(\text{cwnd})) * \text{cwnd} \tag{10}$$

这里引入 cwnd 增长函数与 cwnd 减少函数，分别使用 a (cwnd)和 b (cwnd)表示，二者公式如(11)和(12)所示：

$$a(\text{cwnd}) = 2 * \text{cwnd}^2 * b(\text{cwnd}) * p(\text{cwnd}) / (2 - b(\text{cwnd})) \tag{11}$$

$$b(\text{cwnd}) = \lg\left(\frac{\text{cwnd}}{W_{\text{low}}}\right) \Big/ \lg\left(\frac{W_{\text{high}}}{W_{\text{low}}}\right) 2 * (b_{\text{high}} - 0.5) + 0.5 \tag{12}$$

公式中 b_{high} 表示减小参数，W_{high} 表示最大 cwnd 值，一般默认设置为 83,000：

$$p(\text{cwnd}) = \left(\frac{\text{cwnd}}{W_{\text{low}}}\right)^{\frac{1}{8}} * p_{\text{low}} \tag{13}$$

$$p_{\text{low}} = \frac{1.5}{W_{\text{low}}^2} \tag{14}$$

HTCP 加入对网络抖动情况的判断，通过网络抖动强度，来判断网络拥塞的程度。意在发生重传之前就进行有效的拥塞控制。通过引入窗口增大参数 α 和窗口减小参数 β 来提高协议的公平性。

当发送方收到一个 ACK 时，

$$\alpha = \begin{cases} 1 & t < T \\ 1 + 10(t - T) + 0.25(t - T)^2 & t > T \end{cases} \tag{15}$$

$$\beta = 1 - 0.5\alpha \tag{16}$$

拥塞窗口 cwnd 与 α 之间的关系为：

$$\text{cwnd} = \text{cwnd} + \frac{\alpha}{\text{cwnd}} \tag{17}$$

当发送过程中出现丢包时：

$$\beta = \begin{cases} 0.5 & \varphi > 0.2 \\ \dfrac{\text{RTT}_{\min}}{\text{RTT}_{\max}} & \varphi \le 0.2 \end{cases} \quad \beta \in [0.5, 0.8] \tag{18}$$

$$\text{cwnd} = \beta * \text{cwnd} \tag{19}$$

通过上述分析可以看出基于丢包的拥塞控制算法均有一个特点，就是只能通过丢包重传来判断是否发生拥塞。但在实际的网络传输过程中，产生丢包的原因有很多种，比如代码链路错误也会导致丢包。因此单一的将丢包归结于拥塞是十分片面的。基于时延的拥塞控制算法 TCPVegas 则考虑了这一因素，将传输时延 RTT 作为网络拥塞的判断条件。若 RTT 增大，则证明出现了拥塞，此时减小 cwnd。当 RTT 减小，则证明拥塞缓解，此时增大 cwnd。虽然有效避免了链路错误导致的丢包误判，但对于传输时延

RTT 的误判又成为了新的问题。当网络开始传输时，数据量会持续增加，RTT 也会逐渐增大，会被误认为发生了持续拥塞，导致 cwnd 减小。这会使得整体传输数据量的减小，导致延迟迅速增加，不利于大量数据的传输。

综合两类拥塞控制算法的优缺点以及算法本身的实现流程，想要降低传输延迟，必须有效避免丢包重传的发生，降低传输过程中的重传率是优化传输延迟的关键。对于延迟优化而言，要尽可能减少超时重传的发生。超时重传会导致传输数据量的滑坡式下降，产生较大的延迟。编码终端通过降低慢启动门限值 ssthresh 的方式，减少慢启动过程时间，加快算法进入拥塞避免阶段的速度，进入拥塞避免阶段后，数据量由指数增长变为线性增长，从而减小了网络中的数据波动，增加传输稳定性，减少丢包重传的概率，以此来降低传输时延。

4. 低延迟测试与分析

4.1. 低延迟优化测试

系统对低延迟优化前后进行效果测试，对多种分辨率视频流进行传输，分别测量优化前后的延迟时间。如表 2 所示。

Table 2. Statistics of video transmission delay time before and after optimization
表 2. 优化前后视频传输延迟时间统计

图像分辨率	帧率 (FPS)	优化前	优化后	性能比较
		延时(ms)	延时(ms)	延时(ms)
640*480	30	130	121	6.92%
800*448	30	139	128	7.91%
800*600	30	155	142	8.38%
1280*720	30	195	177	9.23%
1920*1080	30	232	210	9.48%

通过 X265 编码器配置优化、编码终端缓存优化和拥塞控制算法优化三种优化手段，使得系统传输延迟平均降低 8.38%。在软件编码的编码方式下，将超清画质视频的传输延迟时间稳定控制在 210 ms 上下，符合实时的技术要求。可见这三种优化方式对系统实现实时编解码传输起到了重要的作用。

视频传输系统采用 H.265 软件编码与 RK3399 嵌入式开发相结合的方案设计实现，与目前主流的系统方案进行对比，在分辨率、帧率及传输延迟时间等关键技术指标上有着更加优秀的表现。不同方案下对低延迟视频传输系统的技术指标进行对比，结果如表 3 所示。类比于其他软件编码和硬件编码方案，本设计实现了超清 1080 P 画质的视频传输，系统延迟时间为 295 ms，具有较好的实时性。

Table 3. Comparison of system technical indexes under different schemes
表 3. 不同方案下系统技术指标对比

	H.265 + HI3516 [6]	FPGA [7]	ARM + HEVC [8]	GStreamer [9]	本设计
年份	2017	2017	2019	2020	2022
编码类型	硬件编码	硬件编码	软件编码	软件编码	软件编码
分辨率	1920*1080	720*576	640*480	640*480	1920*1080
帧率(FPS)	25	25	12	30	30
延迟(ms)	250	375	900	136	295

通过在编码终端和解码终端的代码中分别加上时间戳，经过软件计算，可以分别获取编码延迟时间、解码延迟时间、控件显示延迟时间和网络传输延迟时间。多次测量，求得延迟时间的平均值如表 4 所示。其中编码延迟时间为 85 ms，解码延迟时间为 68 ms，控件显示延迟时间为 15 ms。使用测得的总延迟时间减去三种延迟时间，可以得到网络传输延迟为 127 ms。

Table 4. System delay time test statistics
表 4. 系统延迟时间测试统计表

系统延迟时间分类	延迟时间(ms)
编码延迟时间	85
网络传输延迟时间	127
解码延迟时间	68
控件显示延迟时间	15
系统总延迟时间	295

4.2. 系统传输延迟优化分析

系统使用 wireshark 抓包软件进行网络测试，通过 wireshark 抓包分析，可以看到，视频数据经 TCP 协议进行传输，在传输过程中发生丢包重传，若判定当前重传为快重传时，则单个数据包的多次重传会导致整个传输网络产生剧烈波动。如图 16 所示，在慢启动阶段出现丢包重传，会重置拥塞窗口 cwnd，传输数据量从 5000 packets/s 下降到 900 packets/s，传输数据量陡降后需要经过一段时间恢复，导致传输延迟增加。

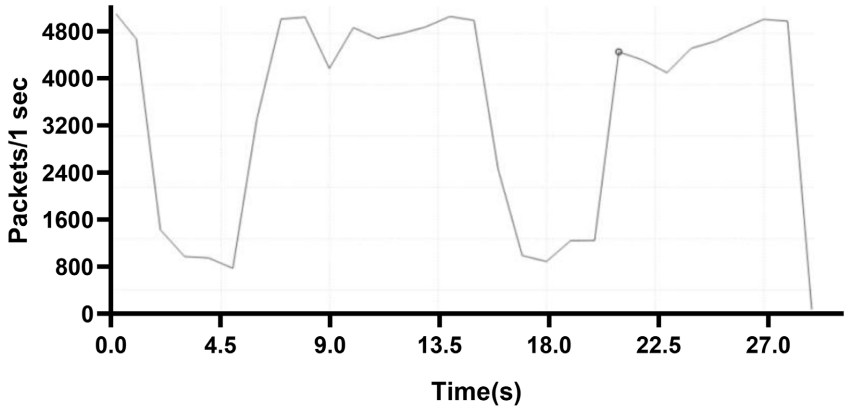
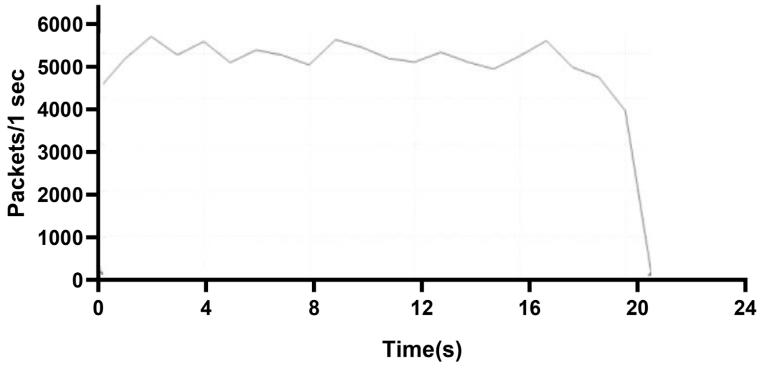
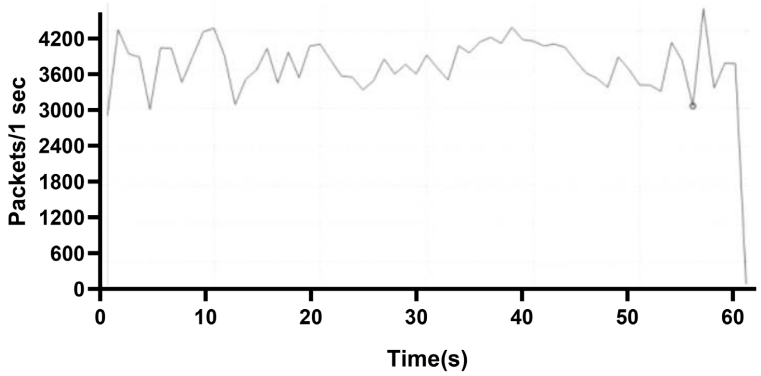


Figure 16. Data fluctuation under congestion control before optimization
图 16. 优化前拥塞控制下的数据波动

Linux 系统下慢启动门限值 ssthresh 默认值为 65,535，通过减小慢启动门限值 ssthresh，使得慢启动阶段时间变短，进入拥塞避免阶段后数据呈现线性增长，传输过程稳定性得到明显提升。在信号强度为−30 dBm 网络环境下，传输效果如图 17(a)所示。在信号强度为−90 dBm 网络环境下，传输效果如图 17(b)所示。数据的增长幅度虽然减小了，但是整体稳定性得到了明显提升。信号强度为−30 dBm 网络环境下，传输速率稳定在 4800 packets/s 到 5500 packets/s 的区间内；信号强度为−90 dBm 网络环境下，传输速率稳定在 3000 packets/s 到 4200 packets/s 的区间内；



(a) −30 dBm 网络环境下



(b) −90 dBm 网络环境下

Figure 17. Data fluctuation under optimized congestion control
图 17. 优化后拥塞控制下的数据波动

为了测试本系统中拥塞控制算法的优化效果，对传输过程的重传率进行了测试，重传率的定义如公式(20)所示。其中分子为单位时间内重传数据包的数量，分母为单位时间内发出数据包的总量。

$$\text{retrans} = \left(\text{RetransSegs} - \text{lastRetransSegs}\right) / \left(\text{OutSegs} - \text{lastOutSegs}\right) * 100\% \tag{20}$$

通过对不同分辨率，不同帧率的视频流进行传输测试，对比拥塞控制算法的优化效果。如表 5 所示。基于优化后的拥塞控制算法平均减少了 6.468%的重传率，对于高清 720 P 和超清 1080 P 视频图像的传输，其重传率分别减少了 7.19%和 6.39%。

Table 5. Statistics of retransmission rate before and after optimization
表 5. 优化前后重传率统计

图像分辨率	帧率(FPS)	拥塞控制算法	优化拥塞控制算法	性能比较
640*480	30	$3.23*10^{-3}$	$3.07*10^{-3}$	4.95%
800*448	30	$3.19*10^{-3}$	$2.99*10^{-3}$	6.23%
800*600	30	$3.30*10^{-3}$	$3.05*10^{-3}$	7.58%
1280*720	30	$5.14*10^{-3}$	$4.77*10^{-3}$	7.19%
1920*1080	30	$5.01*10^{-3}$	$4.69*10^{-3}$	6.39%

由于重传率统计的是历史数据，独立来看意义不大。因此引入 TCP 分段重传占比 pro，通过对一段时间内数据重传率的变化来判断优化的效果。该值越小越好，一般低于 20%。分段重传占比 pro 的计算

方式如公式(21)所示。

$$pro = \Delta RetransSegs / \Delta OutSegs \tag{21}$$

对优化前后的拥塞控制算法进行测试，结果如图 18 所示。相比于优化前，优化后的拥塞控制算法的分段重传比有明显减少。

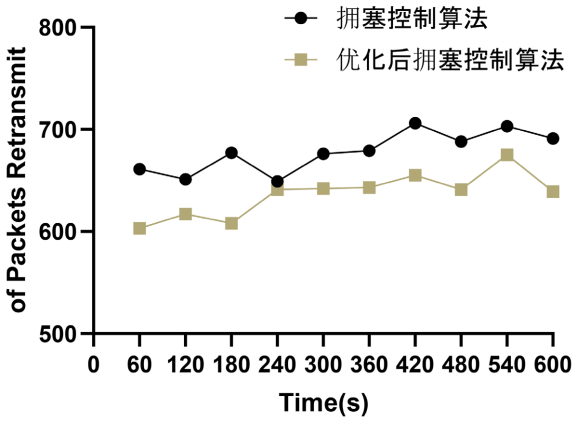


Figure 18. Segment retransmission ratio under congestion control before and after optimization
图 18. 优化前后拥塞控制下的分段重传比

5. 结论

本文对基于 H.265 的低延迟优化策略进行研究，结合时下流行的嵌入式技术和互联网技术，根据实际使用场景进行系统地开发实现。硬件上具有较好的适配性。数据传输支持 WiFi 和局域网环境，大大降低了网络传输的复杂度，具有良好的适用性。系统的编码终端使用 RK3399 核心控制器与自主设计的功能拓展底板进行构建，解码终端则采用 APP 形式进行软件开发，使用时下普及性最广的智能手机作为硬件载体。除了轻便的硬件设计外，系统从 X265 编码器配置、拥塞控制算法、缓存设计、多线程机制等多个方面进行优化，有效减少了系统的整体传输延迟，在 1080 P 图像的传输过程中，系统延迟时间为 300 ms，编码终端与解码终端之间的网络传输延迟时间仅为 127 ms。系统功能完整，可移植性强，在测试环境与实际使用过程中表现良好，具有一定的实用性。

参考文献

[1] Sullivan, G.J., Ohm, J.-R., Han, W.-J., *et al*. (2012) Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, **22**, 1649-1668. https://doi.org/10.1109/TCSVT.2012.2221191

[2] Sullivan, G.J., Boyce, J.M., Chen, Y., *et al*. (2013) Standardized Extensions of High Efficiency Video Coding (HEVC). *IEEE Journal of Selected Topics in Signal Processing*, **7**, 1001-1016. https://doi.org/10.1109/JSTSP.2013.2283657

[3] Yousfi, R., Ben Omor, M., Damak, T., Ben Ayed, M.A. and Masmoudi, N. (2018) JEM-Post HEVC vs. HM-H265/HEVC Performance and Subjective Quality Comparison Based on QVA Metric. 2018 4*th International Conference on Advanced Technologies for Signal and Image Processing* (*ATSIP*), Sousse, 21-24 March 2018, 1-4. https://doi.org/10.1109/ATSIP.2018.8364455

[4] 李路, 冉亮. 基于 CS 架构的雷电流信号智能检测系统设计[J]. 自动化与仪器仪表, 2019(7): 80-83.

[5] 彭瑶, 周端, 杨银堂, 等. 高速环形 FIFO 的设计[J]. 计算机辅助设计与图形学学报, 2011, 23(3): 488-495.

[6] 胡博, 赵旦峰, 王中刊. 基于 H.265 编解码高清视频传输系统[J]. 应用科技, 2017, 44(1): 27-32.

[7] 王静, 丁尧, 袁杰. 低延时视频传输系统的设计[J]. 自动化应用, 2017(12): 42-44, 47.

[8] 刘建新, 廖望, 严月浩. 基于 ARM 的无人机 HEVC 实时视频传输系统设计[J]. 南昌航空大学学报(自然科学版), 2019(2): 96-101.

[9] 王鑫, 左乐, 施振华, 等. GStreamer 音视频传输系统研究与实现[J]. 单片机与嵌入式系统应用, 2020, 20(9): 6-10.