Ford Motor (China) Company

视频延迟优化方案说明文档

Version <1.1>

免责申明

本文档中的内容仅供参考。福特汽车中国对本服务内容的错误或遗漏概不负责。在任何情况下，福特汽车中国均不对因使用本文档而产生或与之相关的任何特殊，直接，间接，间接或偶然的损害赔偿或任何损害负责，无论是在合同，疏忽，其他侵权行为中服务或服务的内容。福特汽车中国保留随时对本文档内容进行补充，删除或修改的权利，恕不另行通知。

Disclaimer

The contents contained in this document are for general information purposes only. Ford Motor China assumes no responsibility for errors or omissions in the contents on the Service.

In no event shall Ford Motor China be liable for any special, direct, indirect, consequential, or incidental damages or any damages whatsoever, whether in an action of contract, negligence or other tort, arising out of or in connection with the use of the Document or the contents of the Document.

Ford Motor China reserves the right to make additions, deletions, or modification to the contents on the Service at any time without prior notice.

编制	Beyondsoft	日期	2017-11-14	版权	署名-相同方式共享 4.0 国际
审核	Ford	日期	2017-11-14	管理	Ford

Project Name: SDL 视频延时优化	Version: <1.1>
SDL 视频延时优化方案说明	Date: <14/11/2017>
<document identifier>	

修改历史

版本	日期	说明
1.0.0	2017-9-19	文档创建
1.1.0	2017-11-14	更新 SDL_CORE 视频延迟优化技术

目录

1 引言

1.1 背景

SDL 是连接车载信息娱乐系统的智能手机应用程序，允许汽车制造商为客户提供高度集成的连接体验，和应用程序开发人员提供与客户连接的新的和令人兴奋的方式。随着越来越多智能手机应用接入，优化 SDL 视频延时的需求也越来迫切，本方案的目标就是降低 SDL 在传输高分辨率，高码率的实时视频时的延时。

1.2 内容

本文档描述的主要内容为 SDL 视频延迟优化的方案及测试方法进行简单描述。

1.3 适用范围

本文档的主要描述了：视频流传输功能概述、视频流特性、视频流传输功能当前优化方案和视频流传输功能优化研究方向，视频延时测试方法。

1.4 术语

缩写	含义
SDL	SmartDeviceLink
SDL_CORE	The Core component of SDL runs on a vehicle's computing system (head unit).
SDL_ANDROID	SmartDeviceLink mobile library for Android
SDL_HMI	HTML5 based utility to see how the SDL works. It connects via WebSocket to SDLCore
GSTREAMER	GStreamer is a library for constructing graphs of media-handling components. The applications it supports range from simple Ogg/Vorbis playback, audio/video streaming to complex audio (mixing) and video (non-linear editing) processing.

Mbps	Million bits per second

1.5 参考资料

https://smartdevicelink.com/

2 SDL 视频功能简介

2.1 概述

SDL 视屏流功能指手机 App 端将视屏流数据传输到 SDL_CORE，SDL_CORE 将数据写入管道、文件或 Socket，GSTREAMER 等平台进行同步播放视屏流数据的过程。

2.2 系统运行环境

该系统为 C/S 三层结构,它的运行环境分为手机客户端、SDL_Core 服务端、SDL_HMI 用户、GSTREAMER 视屏流显示端四个部分。

如下是系统的软件环境。

（1）手机客户端

操作系统：Android 5.0 及以上

USB 支持：USB_AOA

手机 App：SPT

（2）SDL_Core 服务端

开发板：Firefly-RK3288

操作系统：Ubuntu16.04

USB 支持：USB_AOA

（3）SDL_HMI 用户

操作系统：Ubuntu16.04

依赖库：QT 5.3.1

（4）GSTREAMER 视屏流显示端

开发板：Firefly-RK3288

操作系统：Ubuntu16.04

3 SDL 视频流传输用例

3.1 SDL 视频流传输及显示过程中的角色

本系统本要用于下列角色：

（1）手机 App，完成 App 的注册，视频流的启动、停止，视屏流数据的传输。

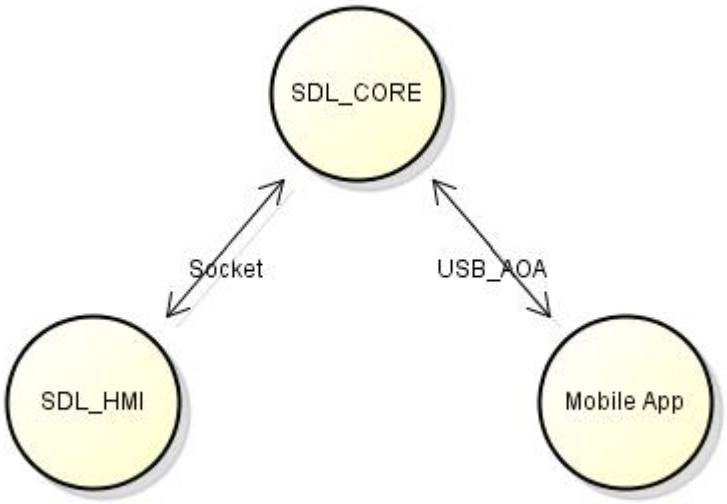
（2）SDL_HMI 用户，其行为包括手机 App 的注册，视频流启动、停止命令的确认等。

（3）SDL_CORE，完成手机 App 和 SDL_HMI 的命令交互，视频流数据的存储等。

（4）GSTREAMER，完成视频流数据的显示。

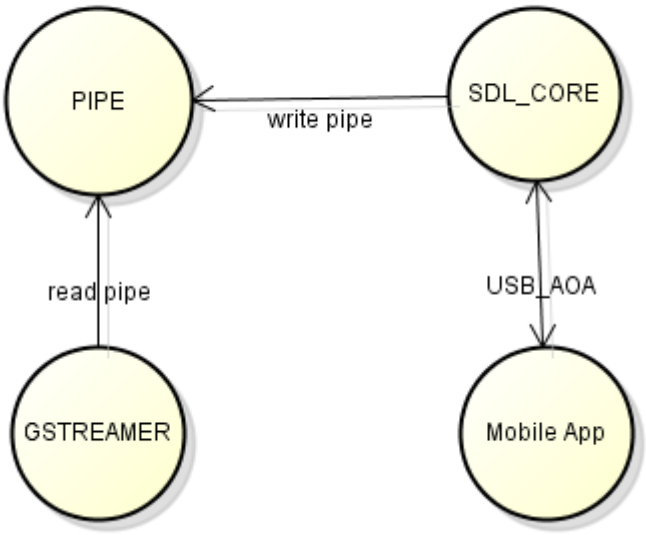
3.2 SDL 视频流传输用例图

- 手机 App、SDL_HMI 用户、SDL_CORE 之间的控制帧传输见图一：



图一 控制帧传输用例图

- 手机 App、SDL_CORE、GSTREAMER 之间的视频流传输见图二：



图二 视频流数据帧传输用例图

4 SDL 实时视频数据分析

实时视频流程包括：视频采集，视频编码，视频传输，视频解码，视频显示 5 大部分，其中

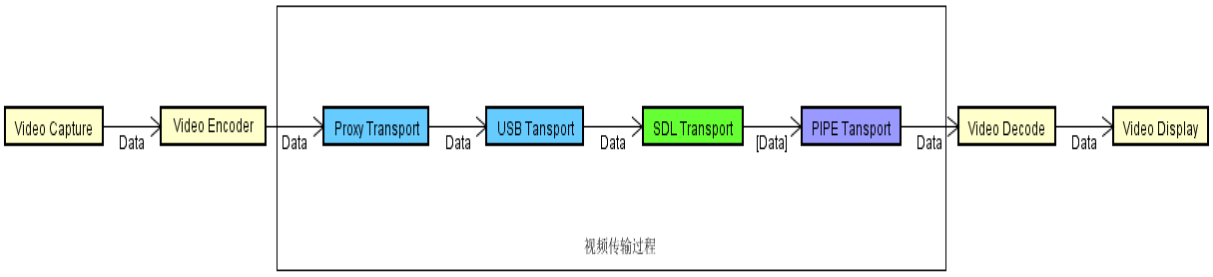
(1) 视频采集：主要由手机端摄像硬件系统和底层驱动组成，此部分技术成熟度高，产生的视频延时值较小且固定。由于此部分技术主要由第三方提供，故不作为优化方案考虑范围。

(2)视频编码：主要由 Proxy 提供的视频编码模块构成。此模块采用的成熟的 H264 编码算法，其视频延时优化空间小，故也不作为优化方案考虑范围。

(3)视频传输：一般市场上实时视频延时偏高的解决方案中，视频传输延时高是其主要原因，故此部分作为优化方案中的重点优化部分。

(4)视频解码： 主要采用 gstreamer 开源的多媒体框架。此框架采用成熟的 H264 解码算法，其视频解码延时优化空间小，故也不作为优化方案考虑范围。

(5)视频显示：主要由显示驱动和显示设备构成。此部分技术成熟度高，产生的视频延时值较小。由于此部分技术主要由第三方提供，故不作为优化方案考虑范围。



图三 当前实时视频流程图

图五为 SDL 实时视频流流向图，其中绿色标记为 SDL 传输优化部分，蓝色标记为 USB 传输优化部分，紫色标记为解码传输优化部分。
根据以上分析，目前阶段，我们重点对视频传输模块做相应调查和优化。具体优化方案见第五章。

5 SDL_CORE 视频延迟优化技术方案

5.1 SDL_CORE 视频传输瓶颈分析

当前 SDL_CORE 传输瓶颈的主要原因是视频帧在通过 Proxy 传输到 SDL_CORE 的 Transport 时，被大量的分片。大量的分片在经过 Transport 层的 event_queue_队列传递到 Protocol 时，由于当前队列采用的是读写线程异步互斥方式读写数据，频繁读写导致的读写线程互斥频繁，读写线程由于互斥导致线程被系统内核频繁的挂起到等待队列，浪费了大量的 CPU 时间。大量的分片也导致合帧时需要更多 CPU 时间。

例如一个 64KB 的视频帧，在经过 proxy（按 16KB 分片）时，被分为 4 个 proxy 包；在经过 Transport（按 512 字节分片），被分为 128 个 RawMessage 包；在经过 event_queue_队 128 次写读传递到 Protocol。

不同视频码率，队列读写次数统计表：

视频码率	Transport 队列每秒读写次数
1Mbps=128KB	256
2Mbps=256KB	512
3Mbps=384KB	768
4Mbps=512KB	1024
5Mbps=640KB	1280
6Mbps=768KB	1536

上述情况在 protocol 到 Media 传输队列和 Media 到 Gstreamer Pipe 传输队列中也存在。

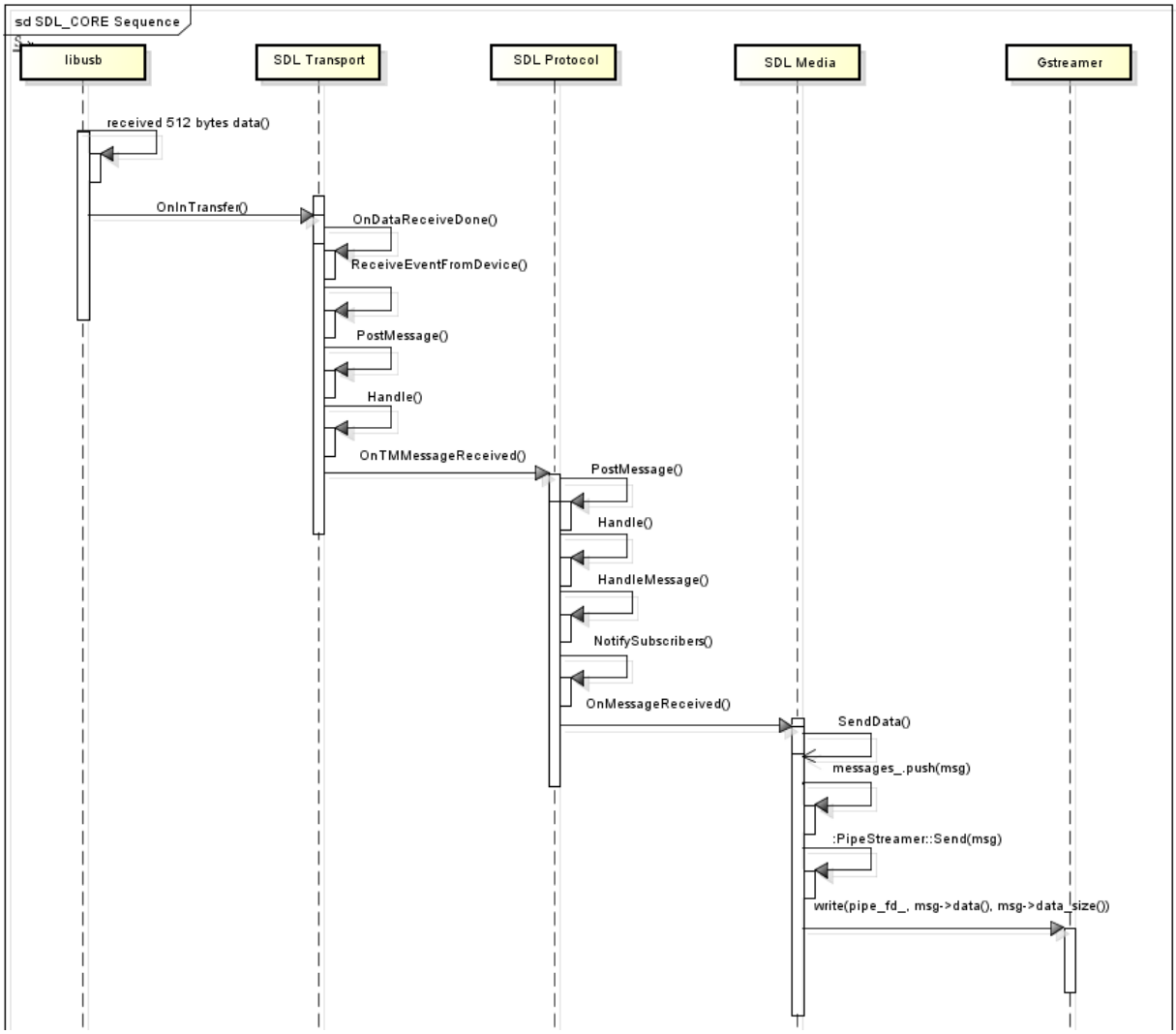
5.2 当前 SDL_CORE 优化目标

当前解决 SDL_CORE 传输瓶颈目标：在不修改 SDL 原有框架的情况下，减少 Transport 对视频帧的分片，减少队列读写次数，减少读写线程互斥次数，提高线程数据读写效率。

5.3 SDL_CORE 优化方案

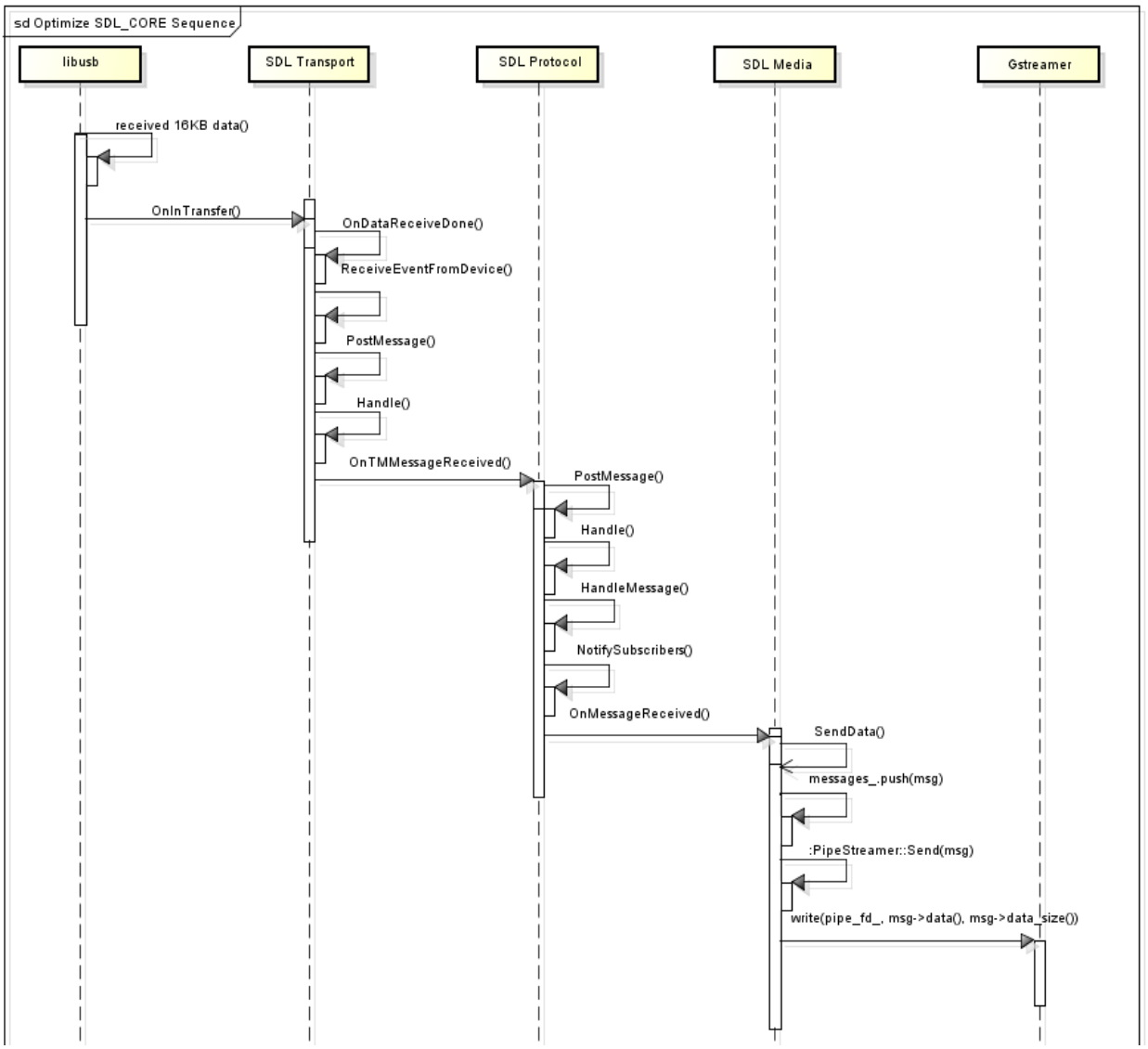
当前 SDL_CORE 的 Transport 注册到 libusb 数据接收缓存大小设置为当前 USB 设备一次读写数据的最大值（当前测试硬件使用的值为 512 字节）。由于 USB 在接收大数据包（例如实时视频流）时，被 512 字节的缓存分片为多个包，这些包在通过 Transport 的 event_queue_队列传递到 Protocol 时，需要通过大量的异步读写，然后在 protocol 合帧，浪费了大量的 CPU 处理时间，造成 SDL_CORE 在大数据流的情况下，数据传输效率降低。

优化前 SDL_CORE 数据传输序列图：



新的优化方案，设置 Transport 注册到 libusb 数据接收缓存大小为 16KB，大于当前 USB 设备一次读写数据的最大值，即允许 libsub 一次缓存多个 usb 数据包后，通过 SDL_CORE Transport 一次写入 event_queue_队列，再传递到 Protocol 处理。新的方案大量减少了视频帧的分片次数，减少了队列的读写次数，提高了 SDL_CORE 实时视频传输时的数据带宽。

优化后 SDL_CORE 数据传输序列图：



新的优化方案以不改变 SDL_CORE 传输构架为前提，只需要增加 SDL_CORE Transport 注册到 libusb 的缓存至 16KB，即可的明显提升 SDL_CORE 视频数据传输的带宽，减小 SDL 视频延时。

6 SDL_ANDROID 视频传输延迟优化技术方案

6.1 PipedStream 优化

改进目标：优化 SDL_ANDROID 数据传输策略，提高 SDL_ANDROID 传输负载数据量，减少视频数据传输延时。此部分工作已开始，现在阶段修改了 Proxy 发送策略，由单字节方式发送改为数据块方式发送。

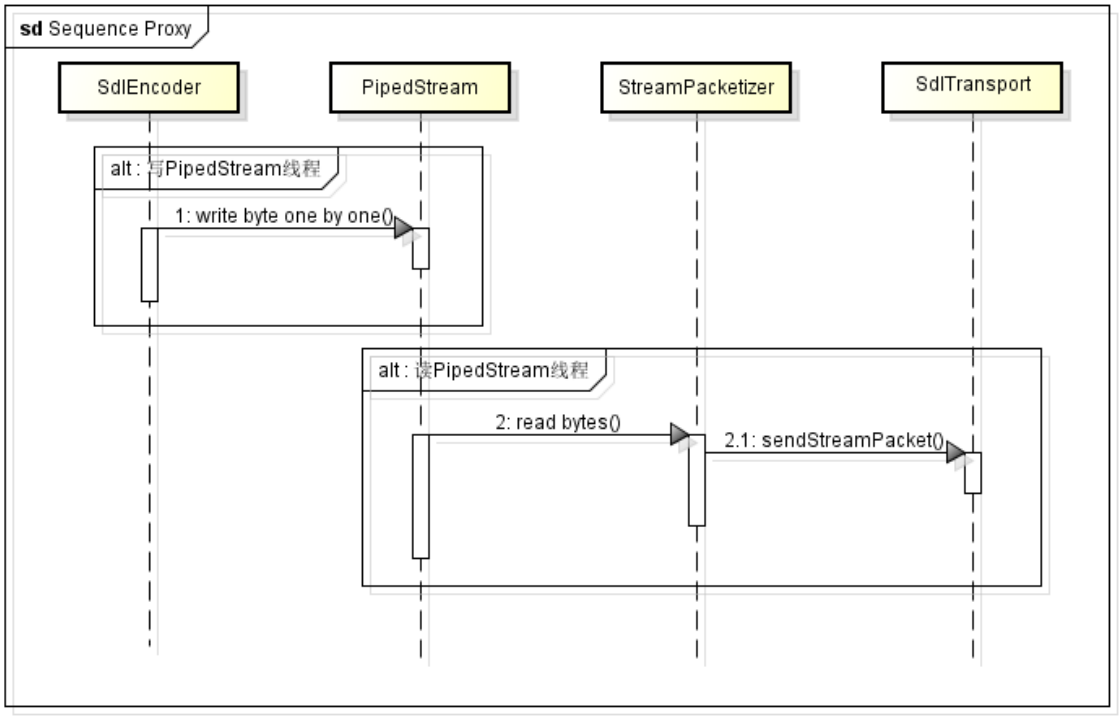
6.2 Proxy 视频流传输修改前后对比

（1） 优化方案比当前方案增加了每次写入 PipedStream 数据量；

- 修改前

当前编码后的视频数据通过 PipedInputStream 一个字节一个字节的写入数据，StreamPacketizer 通过 PipedOutputStream 读取数据。

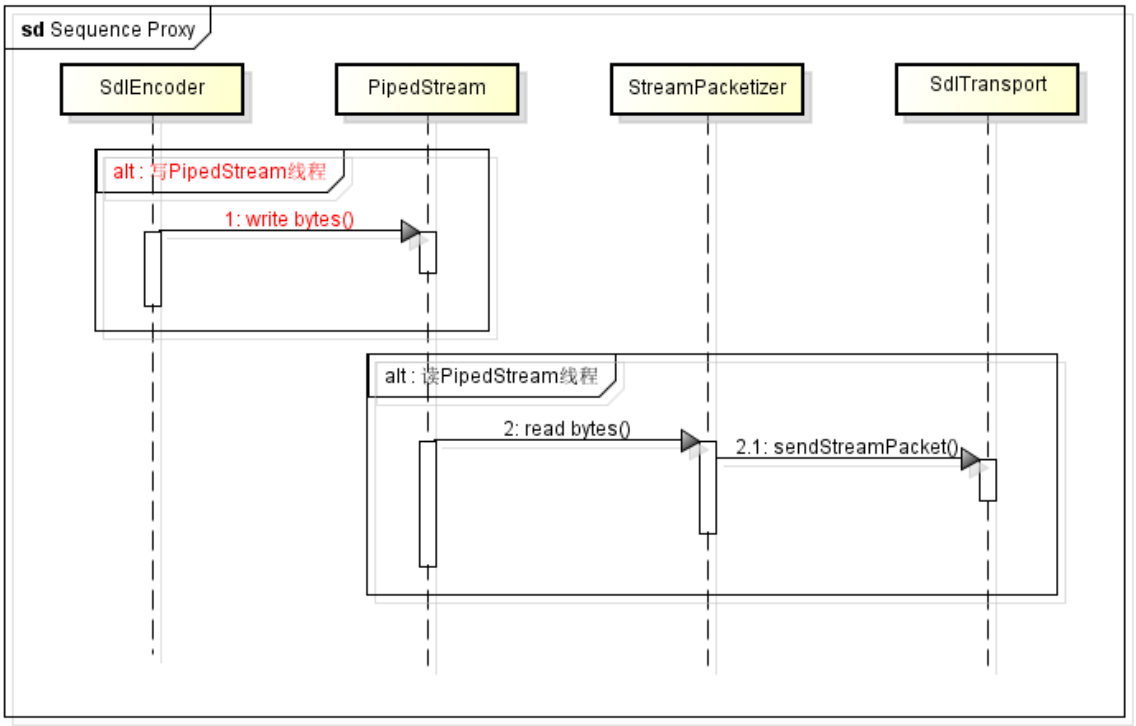
Proxy 当前视频流数据传输序列图



- 修改后

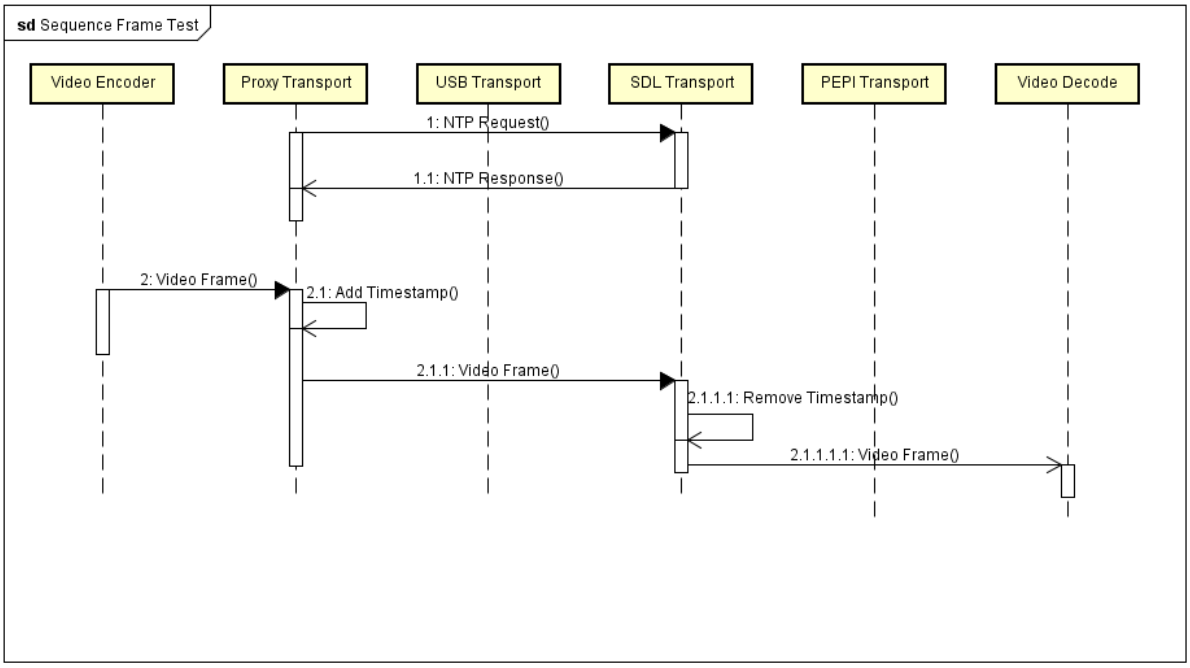
优化后编码的视频数据通过 SdlPipedInputStream 一块一块的写入数据，StreamPacketizer 通过 SdlPipedOutputStream 读取数据。

Proxy 优化后视频流数据传输序列图



6.3 测试

增加手机端与 SDL 设备端 NTP 对时功能，为 Proxy 传输添加分帧功能，USB 传输视频帧添加时间戳，用于传输延时测试，测试流程见视频帧 SDL_ANDROID 传输延时测试序列图



视频帧 SDL_ANDROID 传输延时测试序列图六

7 Gsteamer 解码延时优化技术方案

改进目标：分析 gstream 视频缓存策略，解决等待视频缓存充满才解码，导致的视频延时问题。

7.1 Gstreamer 版本更新

SDL 设备端的 Gstreamer 版本由 V201705250940 更新到 V20170910_1700， 经过初步测算发现解码性能有较大提升，详细结果参见《视频延时优化测试报告》

Gstreamer 版本更新说明：

由于官方给出具体解释说明，源代码一直在开发进行中，上周验证硬件解码性能较之官方 5 月 25 日发布的文件系统有较大提升。

新版本主要更新日志 ：

package: gstreamer: rebuild gstreamer1.0-plugin-bad with gles2 enabled

package: gstreamer-rockchip-exra: update for rgaconvert

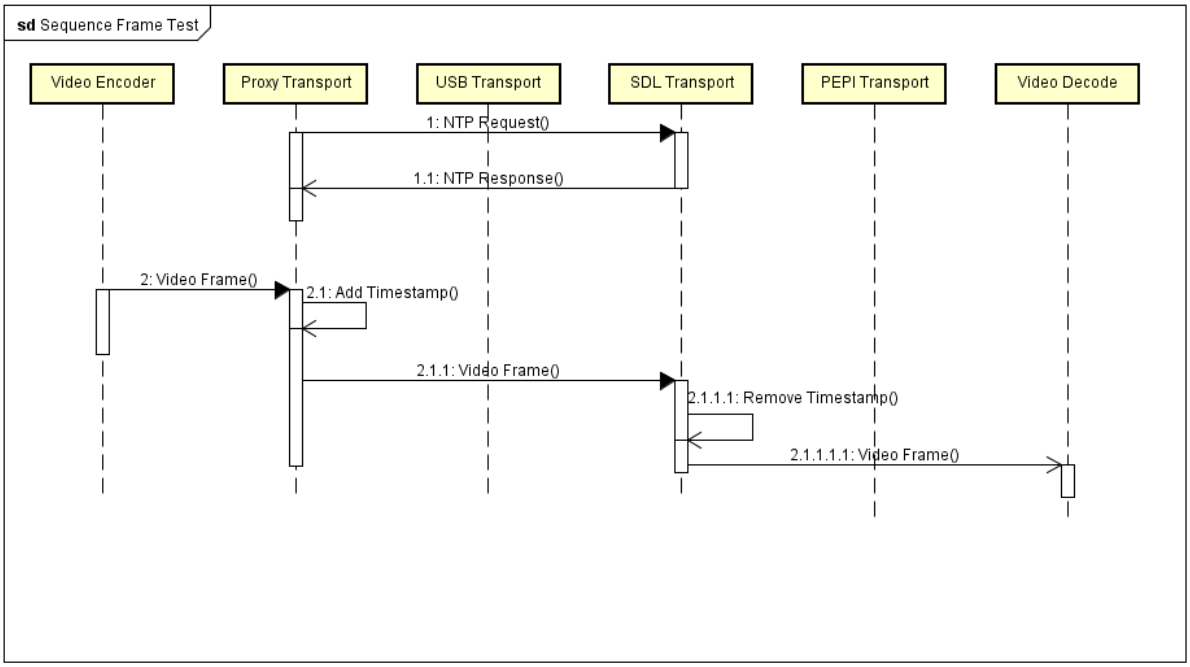
add gstreamer-rockchip-extra package

```
gstreamer: 20170811: update
overlay: debug: remove old cif_isp driver files
packages: mpp: release-20170811: fix buffer leakage
packages: gstreamer: release-20170811: add kmssink
overlay: debug: test_rga: fix double quotes
overlay: debug: use dmabuf for camera
packages: libmali: fix for rk3328
overlay: debug: add gstreamer camera scirpt
```

详细信息可查看链接：https://github.com/rockchip-linux/rk-rootfs-build/commits/master

8 视频延时测试设计方案

依据视频延时测试需求，设计方案增加了手机端与 SDL 设备端 NTP 对时功能，为 Proxy 传输添加分帧功能，USB 传输视频帧添加时间戳，用于传输延时测试，测试流程见视频帧 USB 传输延时测试序列图



视频帧 USB 传输延时测试序列图七

8.1 基于 USB 的 NTP 对时模块设计

8.1.1 NTP 定义

NTP 网络时间协议，英文名称：Network Time Protocol（NTP）是用来使计算机时间同步化的一种协议，它可以使计算机对其服务器或时钟源做同步化，它可以提供高精准度的时间校正（LAN 上与标准间差小于 1 毫秒，WAN 上几十毫秒）。

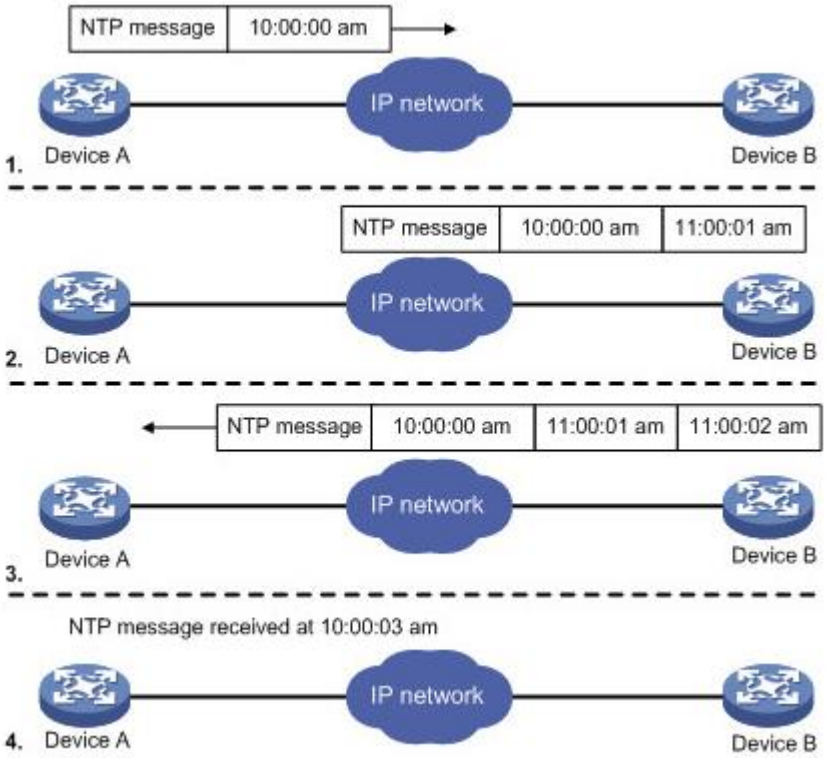
8.1.2 NTP 原理

NTP 的基本工作原理如图所示。Device A 和 Device B 通过网络相连，它们都有自己独立的系统时钟，需要通过 NTP 实现各自系统时钟的自动同步。为便于理解，作如下假设：

在 Device A 和 Device B 的系统时钟同步之前，Device A 的时钟设定为 10:00:00am，Device B 的时钟设定为 11:00:00am。

Device B 作为 NTP 时间服务器，即 Device A 将使自己的时钟与 Device B 的时钟同步。

NTP 报文在 Device A 和 Device B 之间单向传输所需要的时间为 1 秒。



系统时钟同步的工作过程如下：

Device A 发送一个 NTP 报文给 Device B，该报文带有它离开 Device A 时的时间戳，该时间戳为 10:00:00am（T1）。

当此 NTP 报文到达 Device B 时，Device B 加上自己的时间戳，该时间戳为 11:00:01am（T2）。

当此 NTP 报文离开 Device B 时，Device B 再加上自己的时间戳，该时间戳为 11:00:02am（T3）。

当 Device A 接收到该响应报文时，Device A 的本地时间为 10:00:03am（T4）。

至此，Device A 已经拥有足够的信息来计算两个重要的参数：

NTP 报文的往返时延 Delay=（T4-T1）-（T3-T2）=2 秒。

Device A 相对 Device B 的时间差 offset=（（T2-T1）+（T3-T4））/2=1 小时。

这样，Device A 就能够根据这些信息来设定自己的时钟，使之与 Device B 的时钟同步。

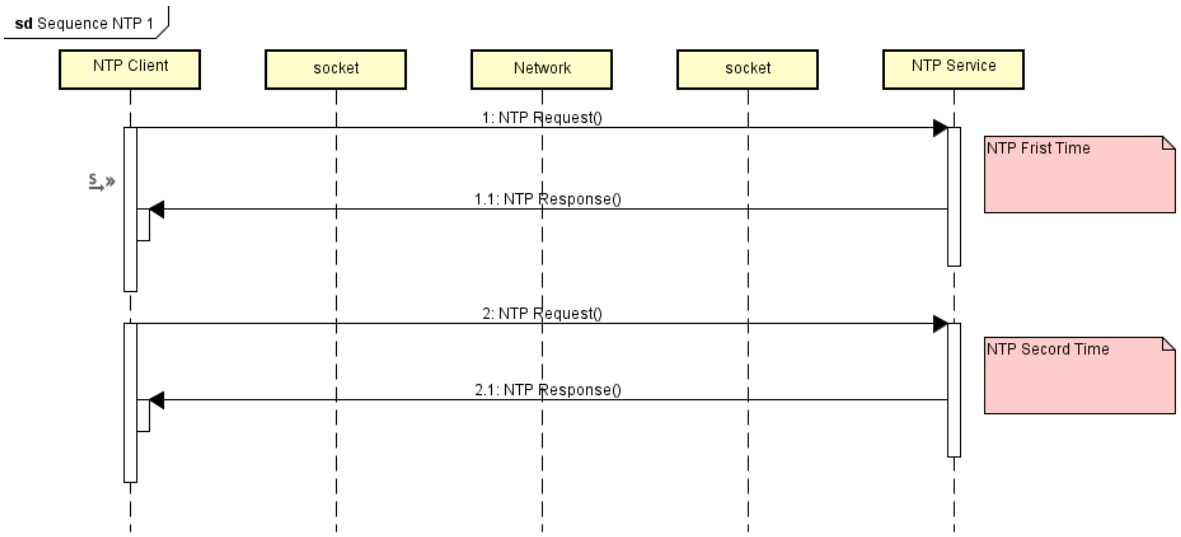
8.1.3 NTP 实现

NTP 是基于网络 UDP 协议的对时协议，为了满足手机 APP 和 SDL 设备之间时间同步的需求，我们在视频延时优化测试代码实现：

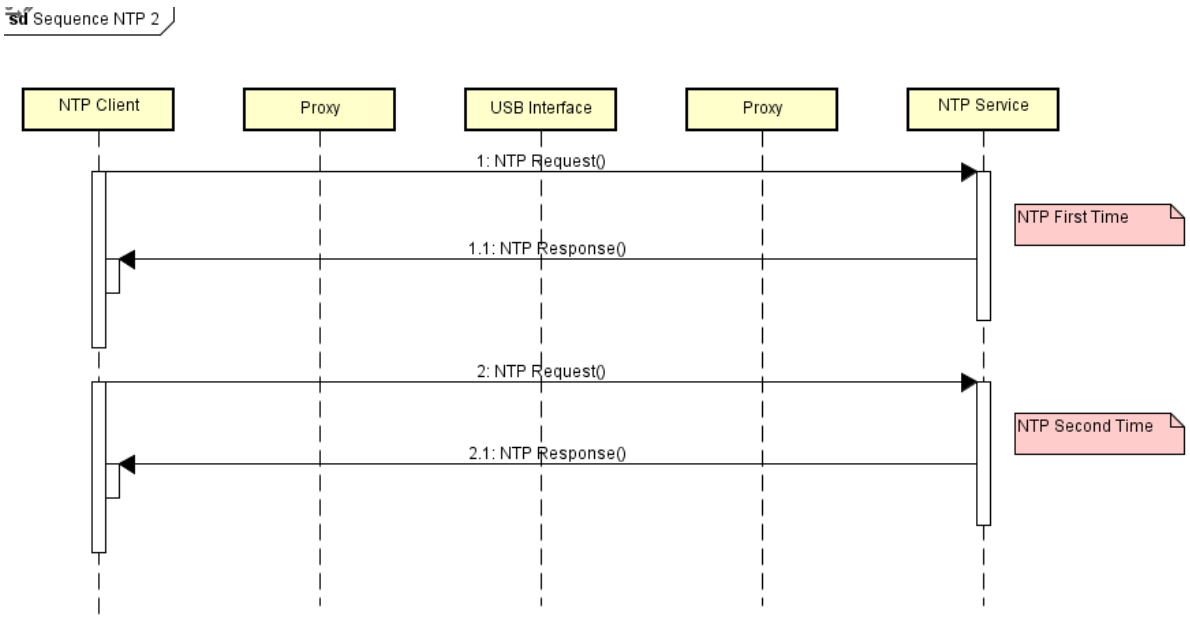
1 在手机 APP 端中实现了 NTP Client 功能模块， 在 SDL 设备端实现了 NTP Service 功能模块。

2 NTP 由 socket 方式改为采用 Proxy 传输接口发送和监听 NTP 协议数据包。

3 在测试开始前，手机 APP 自动发送 NTP 请求，SDL 设备端监听 NTP 请求并应答，手机 APP 接收到 NTP 应答后，依据 NTP 时间计算算法，实现手机 APP 向 SDL 设备时间对时功能。此过程将连续持续 2 次，以提高时间对时精度(对时精度为毫秒)。



NTP socket 传输序列图八



NTP Proxy 传输序列图九

8.2 视频帧的识别和时间戳的实现

8.2.1 视频帧的识别

设计目标：对经过 Proxy 和 SDL 的视频流进行处理，实现视频帧的识别。

实现：基于 H264 帧格式，实现对视频流中帧的识别和分帧功能。

8.2.2 视频帧时间戳

设计目标：为视频帧增加时间戳，支持视频传输延时的计算（精度毫秒）。

实现：定义了一个新的视频测试报文格式，用于对视频帧添加时间戳功能。

0	4
Magic Byte	
Timestamp Hight Bite	
Timestamp Low Bite	
Frame Data	

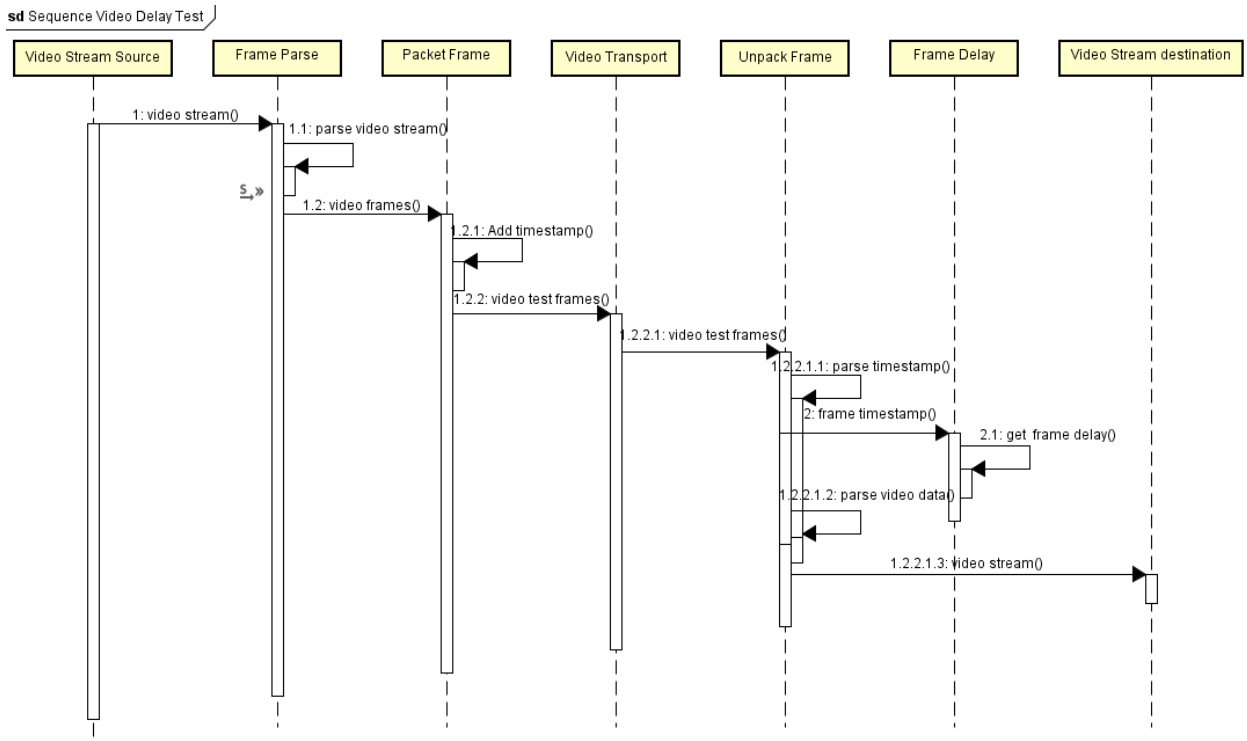
. . .

其中魔法字定义为 4 字节，为常量值 0x12CD34EF,用于标识视频测试报文开始；Timestamp 时间戳定义为 8 字节，高 4 字节保存从 1900 年 1 月 1 日 0 时 0 分 0 秒起到现在时间的秒数，低 4 字节保存现在时间的毫秒值报文。

8.2.3 视频流延时测试流程

视频延时测试流程包括：

- 视频流的识别与分帧
- 添加时间戳
- 时间戳和帧数据打包为视频测试帧
- 视频测试帧的传输（视频传输延时测试目标）
- 视频测试帧的解包，解析出时间戳和帧数据
- 计算视频延时
- 视频流恢复

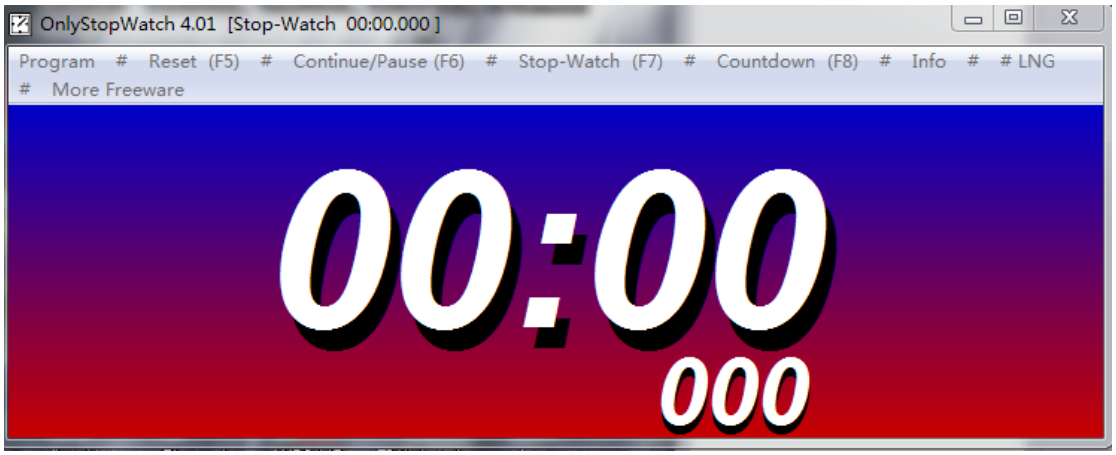


视频延时测试序列图十

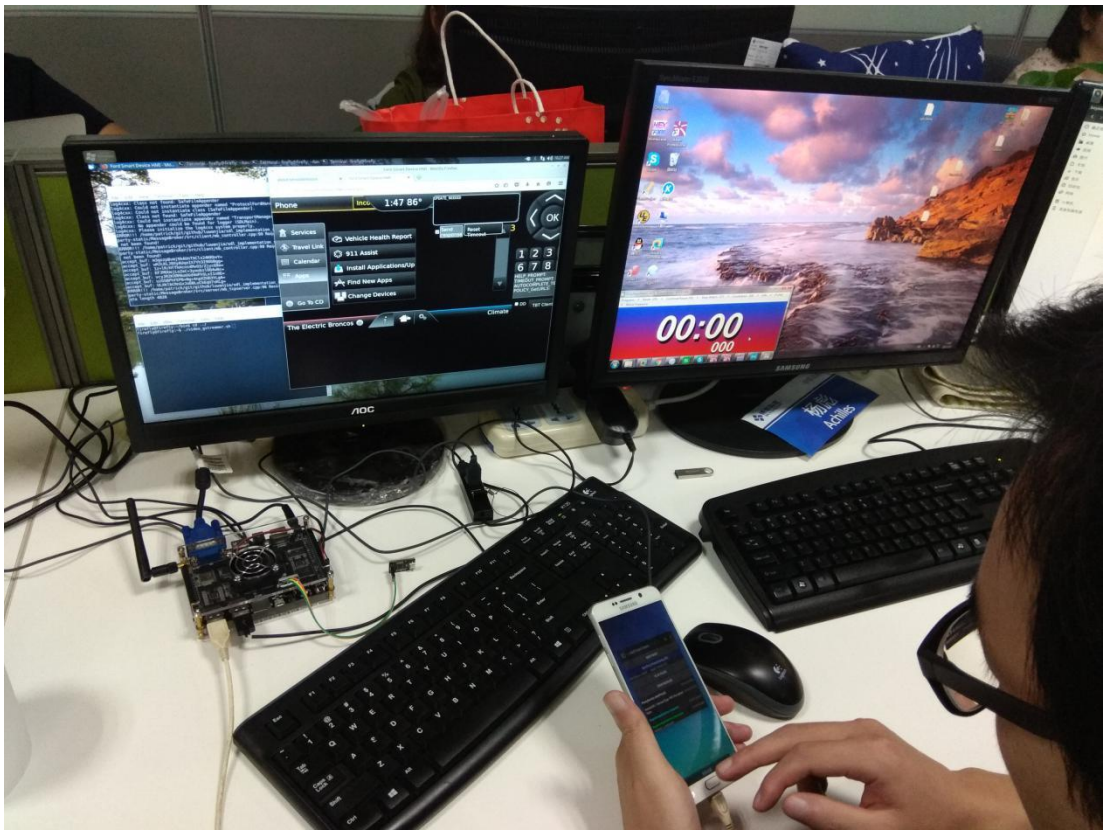
9 实时视频网络延时测试方法

9.1 测试步骤

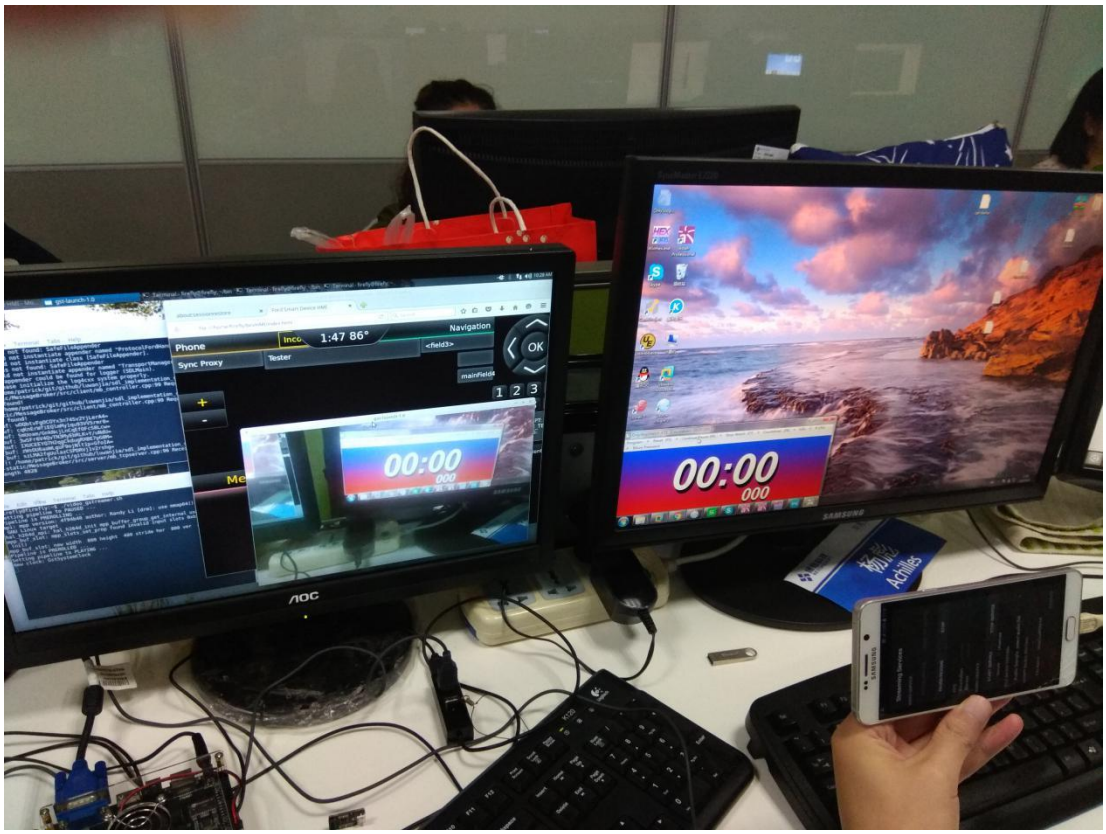
第一步 在 windows 上运行计时器软件 OnlyStopWatch_x64：



第二步 分别运行开发板 SDL， 手机测试应用 SyncProxyTester。SyncProxyTester 发送视频流到开发板，开发板调用 gstream 解码视频在开发板显示器上显示。



第三步 控制测试手机摄像头对准开发板显示屏与 windows 屏，然后点击计时器软件记时开始。（注：下图左边屏幕为开发板显示屏， 右边屏幕为 window 显示屏）



第四步 使用另外一部手机对准 windows 显示屏与开发板显示屏拍照。（注：精度单位毫秒）

第五步 计算视频显示延时。上图开发板时间为 20 秒 393 毫秒，windows 屏幕时间为 20 秒 640 毫秒，计算视频显示延时时间为 247 毫秒。

9.2 测试原理

以 Windows 显示屏显示的时间为参照时间,开发板显示屏显示的时间为实时视频播放显示时间。测试计算公式：实时视频显示延时（毫秒）= 视频参照时间 - 视频显示时间

9.3 视频流分析工具简介

Elecard StreamEye 工具：为用户提供了一系列的视频编码功能和流媒体结构分析,视频序列分析功能,支持格式:MPEG-1/2/4 or AVC/H.264 Video Elementary Streams (VES),MPEG-1 System Streams (SS),MPEG-2 Program Streams (PS) 和 MPEG-2 Transport Streams (TS)。

Elecard Stream Analyzer 工具：是一款简单小巧的码流分析工具，通过该软件，用户可以快速的分析查看视频序列码流；软件操作简单，使用方便，用户只需将视频文件导入软件内，系统就会自动帮您分析文件，分析后就会显示视频码的文件大小、码流类型、数据包数等内容了，方便用户对视频的质量进行初步的评估，有效的改善视频的拍摄质量以及制作相应的修改方案。