# Android 实时流媒体监控的关键技术研究

周晶晶[1],舒 翔[2],龙 涛[2],桂良启[2]
(1.武汉军械士官学校 火控雷达教研室,湖北 武汉 430074;
2.华中科技大学 电子与信息工程系,湖北 武汉 430074)

**摘 要**:以视频监控系统在物联网中的应用为背景,介绍了如何在 Android 平台上进行实时的视频监控系统的开发。在对 Android 操作系统进行深入分析的基础之上,提出了一个基于 Android 的流媒体监控方案,此方案通过移植 X264 开源库,实现了 Android 视频的 H.264 编码,并通过双缓冲文件搭建流媒体服务器对实时视频流进行发布。通过对系统的测试,指出了值得改进的方向,为今后的研究工作提供参考。

**关键词**:Android;视频监控;H.264;双缓冲技术;流媒体服务器
**中图分类号**:TM316.2 **文献标识码**:A **文章编号**:1674-7720(2013)19-0004-04

# Research on the key technology of a real-time streaming media monitoring based on Android operating system

Zhou Jingjing[1], Shu Xiang[2], Long Tao[2], Gui Liangqi[2]
(1.The Department of the Fire Controling, Wuhan Ordnance N.C.O.Academy of PLA, Wuhan 430074, China;
2.Department of Electronics and Information Engineering, Huazhong University of Sicence and Technology, Wuhan 430074, China)

**Abstract**: This paper demonstrates developing video monitoring system in Internet of Things and introduces how to do it based on Android platform.This paper put forward a kind of streaming media monitoring solution based on a deep analysis of the Android operating system.This solution realize the Android video coding with H.264 through the transplant X264 open source library, and constructs a real-time video streaming media server through the double buffering file structures. Finally, the test of the system, and points out the direction that is to be improved int the future and is to provide the reference for future research work.

**Key words**: Android; video monitoring; H.264; double-buffering; streaming server

终端平台的智能化和 3G 网络的覆盖带来了移动互联网时代,这对当今不断壮大的物联网带来了很多的便利。比如对物流车辆进行随时、随地、随身的视频监控,相比传统的 PC 机监控更加方便和高效。在移动终端上进行视频监控系统的开发,由于其硬件资源和网络环境的限制,开发难度远大于 PC 机,并且对移动终端和网络都有很高的要求。本文通过分析流媒体服务器的特点,在服务器上实现了一个双缓冲机制来达到实时发布流媒体的要求,通过服务器多线程的方式实现边采集边传输并实时的发布。

本文采用 Android 操作系统作为终端视频采集的平台,借助 Android 系统平台开发的优点,可以很好地进行推广及后期应用。此外,为了保证数据传输的质量,本文通过在 Android 中移植 X264 开源库来实现流媒体视频的编码,并通过双缓冲文件优化流媒体传输机制以实现实时视频和移动监控的融合。

## 1 系统分析与设计

### 1.1 系统总体架构设计

系统由视频移动终端、流媒体服务器、视频监控端 3 部分组成,系统组成框图如图 1 所示。


图 1 系统组成框图

其中,视频移动终端通过 Anrdoid 平台提供的 api 实时获取摄像头捕获的视频流,并通过 JNI 的方式调用底层的 native 代码以完成 H.264 的编码工作,之后通过

socket 传输到视频监控服务器。

视频监控服务器包括应用服务器和流媒体服务器两部分。其中应用服务器作为整个系统的服务端，用于处理视频监控端的视频请求以及接收视频移动终端发来的实时视频流数据。流媒体服务器则用于将应用服务器接收的视频流数据封装成流媒体格式并实时发布。

视频监控端采用 Android 平台构建，可以通过 RTSP 和 HTTP 两种协议访问流媒体服务器以获得观看实时视频的效果。

### 1.2 视频采集和编码

本系统采集的实时视频来源于 Android 系统支持的摄像头。考虑到无线网络带宽的限制，本系统采用 H.264 标准进行压缩编码。由于 H.264 编码对硬件要求较高，编码的速度会受到一定的影响，这样采集到的视频可能不够连贯。本系统采用多线程加缓冲队列的方法进行采集和编码。视频采集线程将捕获的每一帧数据放入一个缓冲队列中，视频编码线程从队列中获取视频帧集合来完成编码和传输的工作。具体流程图如图 2 所示。

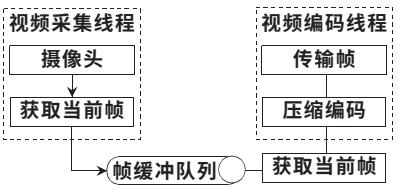


图 2 视频采集编码流程图

其中视频采集线程的伪代码如下：

```
Begin；
//初始化摄像头，设置视频采集参数；
While（视频采集处于激活状态）{
从摄像头获取一帧数据；
while（缓冲队列已满）
wait；                    //将线程挂起以等待队列可写
将一帧数据压入帧缓冲队列；
Notify；                  //通知编码线程队列可读
}
End
```

视频编码线程的伪代码如下：

```
Begin；
//设置编码参数（H.264），初始化编码对象；
While（编码标志位为真）{
While（缓冲队列为空）
wait；                    //将线程挂起以等待队列可读
从缓冲队列取一帧数据；
Notify；                  //通知采集线程队列可写
JNI 调用 native 代码对数据帧进行编码；
If（编码成功）{
调用 RTP 组件对数据打包；
通过 UDP 传输 RTP 包；
}
}
End
```

### 1.3 H.264 视频流传输控制模型

H.264 的定义由视频编码层（VCL）和网络提取层（NAL）两部分组成。其中 VCL 作为 H.264 的核心算法引擎对视频数据进行压缩编码和解码；NAL 层则根据不同的网络把数据打包成相应的格式并通过网络传送出去。为了保证较低的延时，需要将 H.264 视频流数据打包成 RTP 包，并加上时间戳和序列号等信息，然后通过 UDP 传输到服务器。RTP 的打包模式有 3 种：单 NAL 单元模式、非交错模式和交错模式。本文根据系统的要求，采用非交错模式按照编码的视频流顺序进行组包，适用于延时较低的实时系统。

由于 H.264 编码对 CPU 消耗较大，如果放在 java 层则会大幅度影响系统性能。本系统将 H.264 编码模块放在 native 层，用 C/C++实现，通过 jni 调用编码接口，然后通过 RTP 传输。

### 1.4 服务器设计

服务器端采用双缓冲文件实现流媒体的生成和发布。传统的实时视频监控一般采用 socket 连接实现边传输边播放视频，视频数据并不会被缓存，并且如果要有新的客户端加入监控，则必须要新建一个 socket 连接。本系统通过加入流媒体服务器作为视频中转，很好地解决了这一问题。由于流媒体服务器发布实时流媒体需要实时的视频源，本系统的应用服务器将接收的实时视频流数据写入一个缓冲文件作为流媒体服务器的视频源，此缓冲文件通过 linux 命名管道实现。此外，流媒体服务器发布流媒体也需要一个缓冲文件，用于存放被编码成流媒体格式后的视频流数据，以供客户端调用。服务器的工作流程图如图 3 所示。

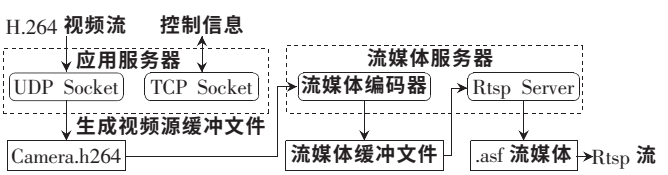


图 3 服务器工作流程图

## 2 系统实现

### 2.1 Android Camera 视频采集

为了实时捕获 Android 摄像头的画面，需要用到 Android Camera。Android Camera 包含取景和拍照两个功能，它实际上是建立在 C/S 架构上的。Camera 运行的时候，可以大致分成服务器和客户端两个部分，他们分别运行在两个不同的进程中，通过 Binder 机制来完成进程间通信。这种 Android 特有的 Binder 机制可以保证客户端和服务器独立的变化，客户端调用接口 AIDL 定义的接口，功能则在服务器中实现，并且进程间通信的部分对上层程序不可见。

具体实现中，通过 Android Framework 提供的android.hardware.Camera 类来完成和 Camera Service 服务端通信。由于需要对采集到的每一帧画面进行压缩编码处理，可以通过调用 Camera 对象的 setPreviewCallback 函数来设置一个回调对象，此回调对象中的 onPreviewFrame 函数可用于完成对当前帧的捕获，这样就可以在此函数中对采集到的视频进行编码的处理了。

### 2.2 JNI 调用 X264 库

考虑到无线网络环境的不稳定性和带宽有限的问题。本文对采集到的视频进行 H.264 标准的高效压缩编码。H.264 是目前一个广泛使用的具有高压缩比的视频编码格式，具有较高的视频压缩性能，适合用窄带传输，适用于移动互联网和流媒体播放。本文采用 X264 开源库来对实时视频进行压缩编码。首先需要在 Android 操作系统上移植 X264 库。

由于 X264 是用 C 语言写的一个开源库，为了能够被 Android 平台使用，需要用到 NDK 工具对其进行编译。NDK 是用来编译本地代码的工具，作为 Android SDK 的一个补充，用于将原生的 C/C++代码集成到应用中，并通过 JNI 的方式被上层 java 程序调用。本文采用 JNI 方式调用 X264 库的步骤如下所示。

(1)JNI 接口设计

设计调用本地代码的函数接口。本文需要对采集的视频进行 H.264 编码，编码部分需要定义 3 个接口。分别如下：

```
private native long CompressBegin(int width,int height);
private native int CompressBuffer(long encoder,int type,byte[] in,int insize,byte[]out);
private native int CompressEnd(long encoder);
```

其中 native 关键表明这 3 个函数来自于 native 代码。CompressBegin 接口是编码初始化接口，通过传递视频画面的宽和高来对 H.264 编码器进行初始化设定；CompressBuffer 接口是编码接口，通过传递 encoder 编码器结构和视频流字节数组来对当前帧的视频流数据进行 H.264 编码，编码后的结果存储在 out 数组中；CompressEnd 接口用于释放编码资源。

(2)实现本地方法

JNI 接口设计完毕之后，需要用 C 语言实现接口。创建一个 H264Android.c 的文件，用来实现第一步中定义的 3 个接口。

(3)生成动态链接库

实现 JNI 接口之后，需要生成.so 的动态链接库以供 java 程序调用。为了生成动态链接库，需要编写 Android.mk 文件并通过 NDK 工具对本地代码进行交叉编译。交叉编译时需要针对 X264 库编写相应的 Android.mk 文件，核心内容如下所示：

```
include $(CLEAR_VARS)
LOCAL_C_INCLUDES+=libx264/include
LOCAL_MODULE:=H264Android
LOCAL_SRC_FILES:=H264Android.c
LOCAL_LDFLAGS+=$(LOCAL_PATH)/libx264/lib/libx264.a
LOCAL_LDLIBS:=-L$(SYSROOT)/usr/lib-lgcc
include$(BUILD_SHARED_LIBRARY)
```

其中，LOCAL_C_INCLUDES 标明了编译需要的外部头文件路径；LOCAL_MODULE 标明了当前生成模块名称；LOCAL_SRC_FILES 标明了编译需要用到的源文件；LOCAL_LDFLAGS 标明了编译需要用到的外部静态库；LOCAL_LDLIBS 标明了引用的外部库文件。

通过引用 X264 的静态库，即可将 X264 编译到 native 代码中，并被上层 java 程序调用。编译成功之后，会在 Android 项目的根目录下的 libs 文件夹中形成一个 libH264Android.so 的动态链接库，编译完成。

(4)Java 程序调用本地代码

Android 中的 Java 程序可以通过 System.loadLibrary("H264Android")函数调用第 3 步中生成 libH264Android.so 动态链接库。并使用声明的 native 方法来完成视频编码的功能。

### 2.3 视频传输的实现

视频传输通过 TCP 和 UDP 两种方式配合实现。为了保证视频采集终端和服务器之间有可靠的通信机制，采用 TCP 连接来进行控制信息的传输，当视频终端收到有效的传输视频的控制信息之后，采用 UDP 连接发送实时的视频流到服务器。服务器视频接收线程会将收到的有效视频数据写入到一个缓冲文件 Camera.h264 中，此文件被作为流媒体服务器的视频源。

### 2.4 FFmpeg 流媒体服务器架设

服务器端采用 FFmpeg 作为流媒体服务器。FFmpeg 是一个开源免费跨平台的视频和音频流方案，属于自由软件，采用 LGPL 或 GPL 许可证。FFmpeg 既可以对视频进行编解码，也可以搭建基于 http 和 rtsp 协议的流媒体服务器。

本系统服务器利用 Camera.h264 缓冲文件作为 FFmpeg 的视频源进行流媒体的发布。由于 ffmpeg 发布流媒体需要用到其中的 FFserver 组件，ffserver 组件的启动需要编写相应的 ffserver.conf 配置文件，主要配置如下所示：

```
Port 8090                          //配置 RTSP 端口号
BindAddress 0.0.0.0                //绑定本地 IP 地址
MaxClients 1000                    //配置最大连接数
<Feed feed1.ffm>                   //配置流媒体缓冲文件
File/tmp/feed1.ffm
FileMaxSize 200 KB                 //缓冲文件大小为 200 KB
</Feed>
<Stream camera.asf>                //配置发布的流媒体格式
Feed feed1.ffm
```

```
Format asf
VideoFrameRate 15
VideoSize 352x240
</Stream>
```

其中，Port 指定了流媒体服务器绑定的端口。<Feed feed1.ffm>标签定义了流媒体服务器运行所需要的一个缓冲文件，大小为 200 KB。<Stream camera.asf>标签定义了流媒体服务器输出的视频格式以及视频相关的参数。如本系统输出的流媒体格式为.asf 格式。

在 ffserver 启动时，会根据 ffserver.conf 文件中的配置新建一个 feed1.ffm 缓冲文件。此缓冲文件用于存放来自视频源文件的实时视频流。FFmpeg 会根据配置文件中的视频输出格式将视频源中的文件进行转换，转换之后的数据会写入 feed1.ffm 文件中。本系统中，feed1.ffm 文件大小被限制在 200 KB，当 200 KB 的空间被用完后，新数据会从文件的开头进行写入。这样可以保证当有新的客户端加入监控时，观看到的是最新的视频。

## 3 系统测试

为了验证 H.264 视频在无线网络中的传输性能，本系统选取了 2 台 Android 2.3 系统的手机进行测试。测试环境如下：

视频终端：Google Nexus S

CPU 主频：1 GHz

内存：512 MB

操作系统：Android 2.3

服务器采用 Ubuntu10.04 搭建。

表 1 对双缓冲和无缓冲的流媒体传输机制进行了测试和对比，显示了随着分辨率和每秒传输帧数/(F/S)的变化导致的丢包率和延时的变化。

经过测试，在采用了双缓冲机制发布流媒体之后，客户端能够以更小的延时播放流媒体服务器发布的 H.264 视频流。由于丢包率主要取决于传输带宽，改变传输模式对丢包率的提升并不是很大。

在高速移动互联网的环境下进行视频监控成为了物联网行业一个比较热门的应用。本文在流媒体服务器的搭建上采用双缓冲文件技术，有效地保证了视频源的实时性，降低了网络传输的延时。此外，考虑到无线网络环境中视频数据传输的困难，本文采用 H.264 标准对实时视频进行压缩编码，有效地提高了带宽利用率。由于本文传输视频数据采用的 RTP 组包模式[5]并没有考虑到实际的应用背景，会产生一定程度的数据丢包，因此只是和应用与对实时画面要求不高的场景，比如物联网物流行业等，如果要实时传输更清晰的视频数据，则需要采用良好的失序和拥塞处理技术，并重写 RTP 的组包算法，这样可以保证视频数据的稳定性和完整性，这也是今后要研究和改进的方向。

表 1 双缓冲传输模式和无缓冲传输模式的测试结果比较

| 分辨率 | FPS | 传输模式 | 丢包率/% | 延时/s |
|---|---|---|---|---|
| 176×144 | 20 | 双缓冲 | 2.5 | 15 |
| | | 无缓冲 | 2.6 | 18 |
| | 15 | 双缓冲 | 1.9 | 10 |
| | | 无缓冲 | 2.0 | 12 |
| 352×240 | 20 | 双缓冲 | 2.8 | 17 |
| | | 无缓冲 | 2.8 | 20 |
| | 15 | 双缓冲 | 2.1 | 10 |
| | | 无缓冲 | 2.3 | 14 |

## 参考文献


[1] SCHULZRINNE H, CASNER S. RTP: A Transport Protocol for Real-Time Application[M]. RFC3550, 2003.

[2] WENGER S, HANNUKSEL M M. RTP Payload Format for H.264 Video[M]. RFC3984, 2005.

[3] 王立青.基于 X264 和流媒体的嵌入式视频监控系统[J].计算机安全，2010(7)：13-15.

[4] 任严.基于 FFMPEG 的视频转换与发布系统[J].计算机工程与设计，2007，28(20)：4962-4967.

[5] 魏聪颖.基于实时流媒体传输系统的 H.264 组包算法研究[J].计算机科学，2007，34(8)：41-44.


(收稿日期：2013-06-14)

作者简介：

周晶晶，女，1977 年生，硕士研究生，讲师，主要研究方向：数字信号处理。