摘 要

近年来，随着互联网技术的发展以及带宽的提高，移动互联网的兴起，使得互联网的媒体载体发生了深刻的变革。之前受制于网络的带宽，网络媒体主要以文本图像为主，辅以少量的音频媒体；而现在，视频逐渐成为了互联网媒体载体的主流形式。相较于文本、静止的图像及音频媒体，视频媒体具有生动、信息量大等特点。因此，其在远程教育、远程医疗、视频会议、在线直播、视频电话和航空航天等领域得到了广泛的应。

为了进一步扩展视频流媒体的应用范围，本文提出了一种基于移动终端的视频监控系统。该系统采用客户端/服务器（Client/Server，C/S）架构，由流媒体服务器和流媒体终端构成。在流媒体服务器端，研究了嵌入式Linux开发环境、H.264视频编码压缩算法和RTP/RTCP等内容，从而提出了基于嵌入式Linux环境的流媒体服务器的方案。在客户端，通过安卓（Android）移动操作系统的提供的JNI（Java Native Interface，Java原生接口），实现对于实时流媒体视频的解码回放。

在论文的最后一章，通过实验说明，本文提出的视频监控系统创新地实现了在移动终端上的实时解码回放，为以后流媒体视频应用范围的进一步拓宽打下了基础。

**关键词：**流媒体视频；移动终端；Linux；H.264；Android

Abstract

in recent years, with the improvement of technology, the development of Internet bandwidth and the rise of mobile Internet, the Internet media carrier has undergone profound changes. Before subject to network bandwidth, network media mainly consists of text, still images and a few of audio; and now video is becoming a mainstream form of Internet media carrier. Compared with text, still images and audio media, video media have vivid and informative features. Thus, stream video has been widely applied in distance education, telemedicine, video conferencing, online broadcasting, video phones, and aerospace and other fields.

In order to further expand the scope of application of video stream media, this paper proposes a video surveillance system based on mobile terminals. The system uses a client / server (Client / Server, C / S) architecture, consisting of the stream server and stream clients. In stream server, a new program of stream server based on embedded Linux operating system is presented after researching H.264c video encoding algorithm and RTP/RTCP. And the program about stream clients based on Android achieves the real-time stream video decoding playback using JNI that Android supplies.

At last, the experiment shows that the proposed video surveillance system based on mobile terminals achieves real-time video stream decoding playback on mobile devices and lays a foundation of expanding the scope of application in the future.

Keyword：Stream Video；Mobile Terminals；Linux；H.264；Android

第一章、绪论

近年来，移动互联网的兴起以及互联网带宽的逐渐提高使得实时视频监控技术成为可能。传统的监控系统包括前端摄像机、传输线缆、视频监控平台。摄像机可分为网络数字摄像机和模拟摄像机，可作为前端视频图像信号的采集。视频监控以其直观、准确、及时和信息内容丰富而广泛应用于许多场合。

当下，在信息技术领域正在发生着前所未有的变革。云计算、大数据以及物联网是这场变革中的三驾马车。而物联网利用其自身的传感器网络所获取的数据更是云计算与大数据的基石。同时，鉴于视觉是人类获取信息最为直观和最主要的途径，因此，研究如何利用逐渐兴起的移动互联网,方便、高效、便捷地捕捉图像（视频）信息具有非常重要的实践意义。

1.1 课题研究背景及意义

网络视频监控系统是一种利用计算机技术将视频信息数字化来进行传输的监控系统，其具有远程控制、监控便捷和易于扩展的优点。网络视频监控系统的优点正好克服了传统的模拟闭路电视监控诸如传输距离近、系统扩展能力差的局限性。

首先，数字化视频可以在计算机网络（局域网或广域网）上传输图像数据，基本上不受距离限制，信号不易受干扰，可大幅度提高图像品质和稳定性；其次，数字视频可利用计算机联网，网络带宽可以复用，无须重复布线，还便于集中监视，集中控制。

发展到今天，视频监控系统的发展经历了三个阶段，并朝着第四代智能监控系统演进。

第一代：模拟视频监控系统

20世纪90年代以前，主要是以模拟设备为主的闭路电视监控系统。系统由摄像机、视频矩阵、磁带录像机和监视器等组成。

第二代：数字视频监控系统

20世纪90年代中期，随着计算机处理能力的提高和视频监控技术的发展，人们利用计算机的高速数据处理能力进行视频的采集和处理，利用显示器的高分辨率实现图像的多画面显示，从而大大提高图像质量。第二代数字监控以DVS、DVR为代表，其主要特点是图像经过编码压缩后再进行传送。

第三代：网络视频监控系统

进入21世纪，随着网络技术的发展，数字视频监控系统进一步发展成为具有网络功能视频监控系统。其集成了多媒体技术、数字图像处理技术以及远程网络传输等最新技术，可以实现图像传输、远程控制、现场信号采集等监控功能，提供高质量的监控图像和便捷的监控方式。

第四代：智能视频监控系统

智能视频监控系统通过在监控系统中增加智能视频分析模块,借助计算机强大的数据处理能力,对视频进行一系列分析，从视频中提取运动目标信息，发现感兴趣目标与事件，并根据预设模板或用户预设的规则，自动识别出感兴趣目标并得到感兴趣数据，将这些信息及时反馈给监控员[1]。

其中，第四代只能视频监控系统以网络为基础，以数字视频的压缩、传输、存储和播放为核心，以智能实用的图像处理和分析为特色，引发了视频监控的技术革命，受到了学术界、产业界和使用部门的高度重视[2]。

1.2 国内外研究现状

随着近年来技术的发展，视频监测正在从以前的模拟监测演进到现在的数字监测；从现场监测演进到远程监测；从有人值守监测演进到现在的无人值守监测。视频监测正朝着数字化、网络化、集成化的嵌入式监测方向蓬勃发展。

数字化是网络化的前提，网络化又是系统集成化的基础，所以，视频监测发展的最大两个特点是数字化和网络化。数字化是21世纪的特征，是以信息技术为核心的电子技术发展的必然。系统的网络化在某种程度上打破了布控区域和设备扩展的地域和数量界限。

目前国内还较多地使用本地模拟图像监控系统，主要是以模拟设备构成的闭路监控系统。系统通常由前端和监控中心构成。前端设备有摄像机、云台等；监控中心设备包括监视器、控制设备、模拟录像机等。虽然本地模拟图像监控系统能够保证采集得到的图像清晰、不失真，但是存在着传输距离小、系统可扩展性差、后期处理能力不强以及无法利用现有网络等缺点。

为了克服上述模拟视频监控的缺陷，出现了基于嵌入式技术的网络视频监控系统。

1.3 课题研究的主要内容

由于本文提出的视频监控系统所使用的为C/S架构，故而系统整体分为两个部分：流媒体服务器端和流媒体接收终端。在流媒体服务器端主要的研究内容包括：

* + 通过摄像头采集视频数据；
  + 基于H.264标准进行的视频数据的编码压缩；
  + 基于RTP/RTCP协议的流媒体数据的网络传输；

而在流媒体客户端研究的主要内容包括：

* + 利用安卓（Android）平台提供的JNI接口进行混合编程；
  + 流媒体视频数据的接收与回放技术。

通过分别研究流媒体服务器与流媒体客户端的设计与实现，最终完成流媒体视频监控系统的设计与实现。

第二章、流媒体视频服务器的设计与实现

流媒体服务器在整体的视频监控系统中具有举足轻重的作用，是视频监控系统的主要组成部分。流媒体服务器的主要任务是通过外接的视频采集设备获取到原始的、未经处理的视频数据，然后在服务器内部完成视频的编码压缩，最终通过网络传输发送视频流媒体数据生成视频流。鉴于流媒体服务器在视频监控系统中的重要性，其设计与实现就显得尤为，故而本文提出了以Linux操作系统为基础，包含视频数据采集子系统、视频数据压缩子系统以及视频流数据传输子系统的视频流媒体服务器的架构，其架构如图1所示。

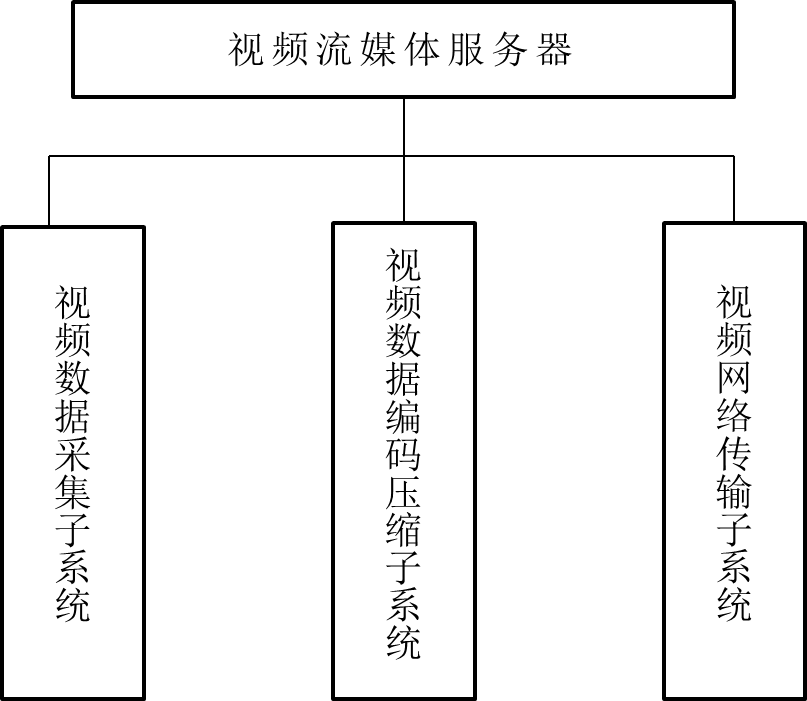
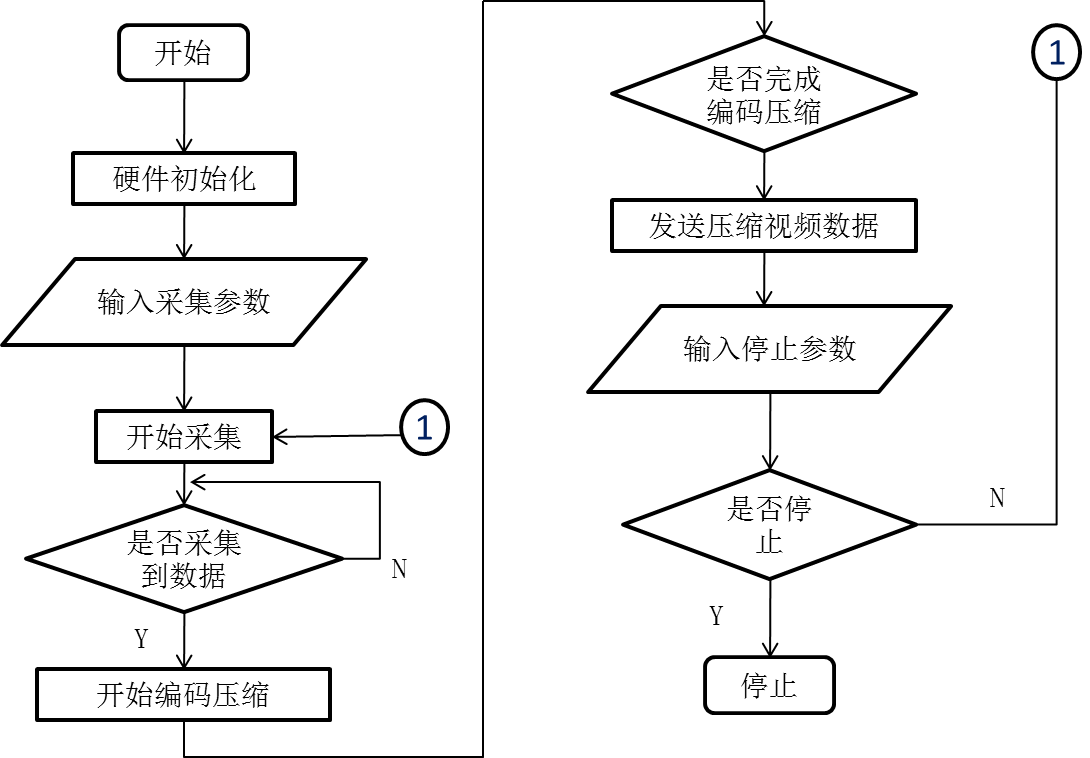
在该设计架构中，视频数据采集子系统利用Linux操作系统提供的V4L2(Video for Linux Two，简称V4L2)接口通过外接的视频采集设备来采集原始未经处理的视频数据，并送给视频编码压缩子系统；视频编码压缩子系统接收原始视频数据，并按照H.264标准进行视频数据的编码压缩；视频网络传输子系统接收已编码的视频数据，然后利用开源的RTP/RTCP开源协议库JRTPLIB进行网络封装，之后向外发送形成视频流数据。

图1. 流媒体服务器架构图

首先在流媒体服务器工作之前，需要进行必要的硬件初始化，包括摄像头初始化、准备视频采集内存、编码器的参数设置以及网络传输参数的设置等。然后，流媒体服务器开始工作，视频数据采集子系统通过外接视频采集设备采集视频数据，视频编码压缩子系统接收来自视频采集子系统的数据通过编码压缩算法得到压缩的视频数据，最终通过网络传输子系统进行发送。流媒体服务器软件流程图如图2所示。

图2. 流媒体服务器流程图

2.1 视频采集子系统

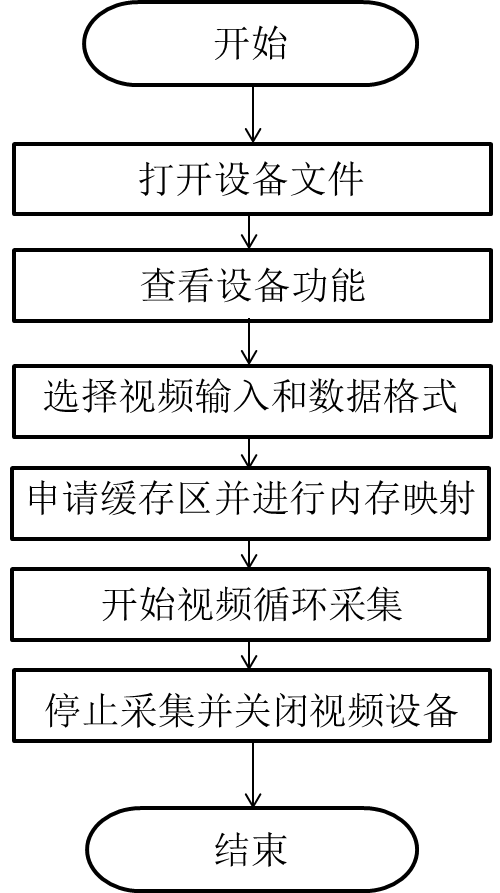
2.1.1 基于V4L2视频采集

Video4Linux（简称V4L）是由Linux操作系统提供的设备驱动框架以及应用编程接口的集合，其主要用于视频采集设备的驱动开发以及实时视频应用的开发。它为针对视频设备的应用程序编程提供一系列接口函数，这些视频设备包括现今市场上流行的TV卡、视频捕捉卡和USB摄像头等[3]。V4L2作为V4L接口的第二版修正了第一版当中的设计缺陷，并且提供了更为广泛的设备兼容性，支持更多的视频设备。

V4L2相比于V4L有更好的扩展性和灵活性，V4L2对V4L进行了彻底的改造，因而两者并不兼容。V4L2相较于V4L的最大优势是可以自定义设备驱动缓存数量，支持的硬件设备更多。V4L2最早出现在Linux 2.5 版本中，Linux 2.6发行版之后采用的接口大多是V4L2。

V4L2在视频采集中两个重要的系统调用时ioctl()和mmap()，ioctl()系统调用对设备的I/O通道进行控制，mmap()系统调用使得进程之间通过映射同一个普通文件来实现内存共享。普通文件被映射到进程地址空间后，进程可以像访问普通内存一样对文件进行访问，直接读写内存，不需要任何数据的拷贝，不必再调用read()、write()等操作，加速了I/O访问。采用共享内存的方式，大大提高了效率。

基于V4L2视频采集的软件流程图如图2所示。

图3. 视频采集流程图

V4L2视频采集步骤主要分为如下过程：

（1）打开视频采集设备，获取该设备的文件描述符。视频采集可通过应用程序使用阻塞方式或非阻塞方式打开。如果使用非阻塞方式，即使数据尚未准备完成，驱动程序将把已经缓存的视频数据返回给应用程序，但是由于已缓存的视频数据可能并非所需的有效数据，故而可能造成数据错误。如果使用阻塞方式，只有在数据准备完成之后，驱动程序可以将已缓存的数据返回给应用程序。本文中所采用的方式为非阻塞的方式。

（2）查询设备兼容性，即查看设备所具有的功能，如是否支持流采集、所支持视频捕获的格式等。

（3）设置视频帧的参数，本文中将视频帧的宽设为640像素，高设为480像素，像素格式设置为YUYV，即YUV422的采样方式。

（4）准备视频采集。首先向驱动申请缓存区，并使用mmap()函数将申请到的缓存区映射到用户空间，这样不必复制即可直接操作采集到的视频帧。

（5）将申请到的缓存区放入视频采集队列，用来存放采集到的数据。

for(int i = 0; i < n\_buffers; ++i){

struct v4l2\_buffer buf;

CLEAR(buf);

buf.type = V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE;

buf.memory = V4L2\_MEMORY\_MMAP;

buf.index = i;

if(-1 == xioctl(fd, VIDIOC\_QBUF, &buf))

errno\_exit("VIDIOC\_QBUF");

}

（6）开始采集视频。首先将数据写入申请的缓存区内，当一个缓存区的数据完成采集之后，驱动就会将其释放进输出队列，等待应用程序的处理。当所有的缓存区都进入输出队列后，驱动将停止采集，并等待缓存区重新放入采集队列。读取数据时，首先需要将一个缓存区出队列，驱动会从输出队列取出一个缓存区，并将其赋值给buf.index域，应用程序可以通过memcpy()函数将缓存区的数据复制到另一内存区域中。复制完成后，需要将缓存区重新放入采集队列队尾，如此即可实现循环采集。

（7）停止采集，首先使用VIDIOC\_STREAMOFF命令标识符停止视频采集，然后调用close()函数关闭视频采集设备。

2.1.2 YUV色彩空间

YUV色彩空间主要用于优化彩色视频信号的传输，使其向后兼容老式黑白电视。与RGB视频信号传输相比，它最大的优点在于使用的频宽极少，因为RGB视频信号要求三个独立的视频信号同时传输。其中“Y”表示明亮度（Luminance或者Luma），也就是灰阶值；而“U”和“V”表示的则是色度值（Chrominance或Chroma），作用是描述影像色彩及饱和度，用于指定像素的颜色。

YUV格式通常有两大类：打包（packaged）格式和平面（planar）格式。打包格式将YUV三个分量存放在同一个数组中，通常是几个相邻的像素组成一个宏像素；而平面格式使用三个数组分别存放Y、U、V三个分量，类似于一个三维平面。本文所设计的系统中使用的外接视频采集设备捕获到的视频数据为打包方式的YUV422。下述说明以8个像素为例，分为两行，每行4个像素来说明YUV422的采样及存储方式。YUV422的采样方式为：8个像素均采集亮度信息，即Y分量；每行4个像素中，若以数字0为起始，依次标识为像素0、像素1、像素2、像素3，则像素0和像素2采集色度信息U分量，而像素1和3则采集色度信息V分量。最终YUV422像素数据的排列为：

Y­­0U0 Y1V1 Y2U2 Y3V3

Y0U0 Y1V1 Y2U2 Y3V3

由上述说明可知，通过YUV422采集的图像数据，均保存在同一个数组当中，并不区分Y、U、V三个分量。每个像素的亮度信息和色度信息需要16个比特位进行保存，即每个像素的长度为2个字节。

而在视频压缩中所使用编码库x264只接受平面格式的YUV420P。下述说明依旧采用8个像素为例，分为两行，行4个像素。YUV420P的采样方式为：8个像素均采集亮度信息，即Y分量；以数字0为起始，依次标识每行的像素为像素0、像素1、像素2、像素3，在第一行像素数据中，只采集像素0和像素2的色度信息U分量，在第二行像素数据中，只采集像素1和 像素3的色度信息V分量。然后亮度信息Y分量、色度信息U分量、色度信息V分量的数据分别保存在3个数组中。最终通过YUV420P方式采集的图像数据，每个像素的亮度信息和色度信息共需要12个比特位进行保存，即每个像素的长度为1.5个字节。

因此，为了使用开源编码库x264，需要使用相应的算法将打包格式的YUV422视频数据转换为平面格式的YUV420P视频数据。该算法的思想如下：针对亮度信息Y分量，YUV420P原封不动地保存原有的YUV422的亮度信息；针对色度信息U分量，YUV420P仅保存原有YUV422数据中偶数行（行号排序从数字0开始）的U分量；针对色度信息V分量，YUV420P仅保存原有YUV422数据中奇数行（行号排序从数字0开始）的V分量。然后将重新采样后的Y、U、V分量分别存储于三个数组，最终保存一个文件，即可得到YUV420P格式的视频采样数据。具体的程序代码如下所示：

char \*y = yuv420p;

char \*u = &yuv420p[width \* height];

char \*v = &yuv420p[width \* height + width \* height / 4];

for(int j = 0; j < height; j++){

for(int i = 0; i < width \* 2; i++){

if(i % 2 == 0){

\*y = p[i+j\*width\*2];

++y;

}

if(j % 2 == 0){

if(i % 4 == 1){

\*u = p[j\*width\*2 + i];

++u;

}

}

if(j % 2 == 1){

if(i % 4 == 3){

\*v = p[j\*width\*2 + i];

++v;

}

}

}

}

上述代码中，yuv420p为预先定义的数组，大小根据视频帧的长宽进行计算，数组y、u、v即用来保存重新采样后的像素数据的三个数组；变量p为YUV422采样格式的原始视频数据；变量width及变量height即为视频帧的宽度与长度，单位为像素。

2.2 视频压缩子系统

通过视频采集设备捕获的原始数据含有大量的冗余信息，从而导致原始的视频数据量非常难以通过网络传输，即使可以传输也会耗费过多的网络带宽资源。因此需要特殊的视频编码压缩算法来对采集到的原始视频数据进行数据压缩，以实现视频数据的网络传输。

对于数据压缩技术而言，最基本的要求就是要尽量降低数字化后的数据量，同时要保持一定的信号质量。因此，所谓数据压缩，就是指在一定的数据存储空间要求下，将相对庞大的原始数据，重组为满足前述空间要求的数据集合，使得从该数据集合中恢复出来的额数据，能够与原始数据相一致，或者能够获得与数据一样的品质[5]。

目前主流的视频编码压缩标准主要有：MPEG-2，MPEG-4，H.264/AVC以及谷歌公司所提出的VP8标准。与其他压缩标准相比，在相同视频图像质量情况下，H.264能够节省50%左右的码率；同时，H.264/AVC标准在各种码率下能够提供稳定的视频质量，特别是在低码率条件下，也能够保持较高质量的视频图像。因此，H.264/AVC标准逐渐在实际应用中成为主流的编码压缩标准。

2.2.1 H.264/AVC标准的研究

H.264标准对网络传输具有非常好的支持功能。它引入了面向IP包的编码机制，有利于网络中的分组传输，支持网络中视频的流媒体功能。H.264具有较强的抗误码特性，可适应丢包率高、干扰严重的无线信道中的视频传输。H.264支持不同网络资源下的分级编码传输，从而获得平稳的图像质量。

H.264/AVC标准的压缩系统由视频编码层（Video Coding Layer，VCL）和网络抽象层（Network Abstraction Layer，NAL）两部分组成。VCL中包括VCL编码器与VCL解码器，主要功能是视频数据压缩编码和解码，它包括运动补偿、变换编码、熵编码等压缩单元。NAL则用于为VCL提供一个与网络无关的统一接口，它负责对视频数据进行封装打包后使其在网络中传送，它采用统一的数据格式，包括单个字节的包头信息、多个字节的视频数据与组帧、逻辑信道信令、定时信息、序列结束信号等。包头中包含存储标志和类型标志。存储标志用于指示当前数据不属于被参考的帧。类型标志用于指示图像数据的类型。

H.264标准可分为三挡：基本档次（其简单版本，应用面广）；主要档次（采用了多项提高图像质量和增加压缩比的技术措施，可用于SDTV、HDTV、和DVD等）；扩展档次（可用于各种网络的视频流传输）。H.264具有以下的技术特点。

1. 帧内预测编码

帧内编码用来缩减图像的空间冗余。为了提高H.264帧内编码的效率，在给定帧中充分利用相邻宏块的空间相关性，相邻的宏块通常含有相似的属性。因此，在对一给定宏块编码时，首先可以根据周围的宏块预测（典型的是根据左侧和上侧的宏块，因为这些宏块已经被编码处理），然后对预测值与实际值的差值进行编码，这样，相对于直接对该帧编码而言，可以大大减小码率。

2. 帧间预测编码

帧间预测编码利用连续帧中的时间冗余来进行运动估计和补偿。H.264的运动补偿支持以往的视频编码标准中的大部分关键特性，而且灵活地添加了更多的功能，除了支持P帧、B帧外，H.264还支持一种新的流间传送帧——SP帧，码流中包含SP帧后，能在有类似内容但有不同码率的码流之间快速切换，同时支持随机接入和快速回放模式。

3. 不同大小和形状的宏块分割

对每一个16x16像素宏块的运动补偿可以采用不同的大小和形状，H.264支持七种模式。小块模式的运动补偿为运动详细信息的处理提高了性能，减少了方块效应，提高了图像的质量。

4. 高精度的亚像素运动补偿

在H.263中采用的是半像素精度的运动估计，而在H.264中可以采用1/4像素或者1/8像素精度的运动估值。在要求相同精度的情况下，H.264使用1/4像素或者1/8像素精度的运动估计后的残差要比H.263采用半像素精度运动估计后的残差来得小。这样在相同精度下，H.264在帧间编码中所需的码率更小。

5. 多帧预测

H.264提供可选的多帧预测功能，在帧间编码时，可选五个不同的参考帧，提供了更好的纠错性能，这样更可以改善视频图像质量。这一特性主要应用于以下场合：周期性的运动、平移运动、在两个不同的场景之间来回变换摄像机的镜头。

6. 区块滤波器

H.264定义了自适应去除块效应的滤波器，这可以处理预测环路中的水平和垂直块边缘，大大减少了方块效应。

7. 整数变换

在变换方面，H.264使用了基于4x4像素块的类似于离散傅里叶变换（DCT）的变换，但使用的是以整数为基础的空间变换，不存在反变换因为取舍而存在误差的问题。与浮点运算相比，整数DCT变换会引起一些额外的误差，但因为DCT变换后的量化也存在量化误差，与之相比，整数DCT变换引起的量化误差影像并不大。此外，整数DCT变换还具有减少运算量和复杂度，有利于向定点DSP移植的优点。

8. 量化

H.264中可选32种不同的量化步长，这与H.263中有31个量化步长很相似，但是在H.264中，步长是以12.5%的复合率递进的，而不是一个固定常数。在H.264中，变换系数的读出方式也有两种：之字形（Zigzag）扫描和双扫描。大多数情况下使用简单的之字形扫描，双扫描仅用于较小量化级的块内，有助于提高编码效率。

9. 熵编码

视频编码处理的最后一歩是熵编码，在H.264中采用了两种不同的熵编码方法：通用可变长编码（UVLC）和基于文本的自适应二进制算术编码（CABAC）。

在H.263等标准中，根据要编码的数据类型如变换系数、运动矢量等，采用不同的可变长编码码表。H.264中的UVLC码表提供了一个简单的方法，不管符号表述什么类型的数据，都使用统一变字长编码表。其优点是简单；缺点是单一的码表是从概率统计分布模型得出的，没有考虑编码符号间的相关性，在中高码率时效果不是很好。

因此，H.264中还提供了可选的CABAC方法。算术编码使编码和解码两边都能使用所有句法元素（变换系数、运动矢量）的概率模型。为了提高算术编码的效率，通过内容建模的过程，使基本概率模型能适应随视频帧率而改变的统计特性。内容建模提供了编码符号的条件概率估计，利用合适的内容模型，存在于符号间的相关性可以通过选择目前要编码符号邻近的已编码符号的相应概率模型来去除，不同的句法元素通常保持不同的模型。

2.2.2 x264库介绍

x264是一个采用GPL（General Public License）授权的视频编码自由软件。x264的主要功能在于进行H.264/AVC的视频编码，而非作为解码器之用。x264提供了一个命令行接口以及一组应用程序接口（API）。前者在许多图形用户界面软件中得到应用，如Staxrip和MeGUI。后者也在许多其他类型的软件中得到应用，比如本文后续章节提到的FFmpeg开源音视频框架[21]。作为目前最为出色的有损视频压缩库，相较于其他的视频编码压缩库，x264具有如下特点：

* + 8x8与4x4自适应空间域转换
  + 自适应B帧选择
  + B帧可作为参考帧/自由的帧顺序
  + CALVC/CABAC熵编码
  + 自定义精确的矩阵模板
  + I帧：所有宏块模式（16x16，8x8,4x4以及所有全部预测的PCM）
  + P帧：所有的分割块（从16x16到4x4）
  + B帧：分割块从16x16到8x8（包括skip/direct）
  + 隔行扫描
  + 多个参考帧
  + 码率控制：固定量化，固定质量，一次或者多次编码的平均码率，可选的VBV参数
  + 场景变换检测
  + B帧时间域、空间域direct模式自适应选择
  + 可在多个CPU并行编码
  + 预测性的无损编码
  + 心理视觉优化，保留更多的细节
  + 可用于手动调整码率分配的zones参数

调用x264编码库进行视频编码的流程图如图4所示。

流程图中主要的函数如下所示：

x264\_param\_default()：设置参数集结构体x264\_param\_t为缺省值。

x264\_encoder\_open()：打开编码器。

x264\_encoder\_headers()：输出SPS，PPS，SEI等信息。

x264\_encoder\_encode()：编码输出一帧图像。

x264\_encoder\_close()：关闭编码器。

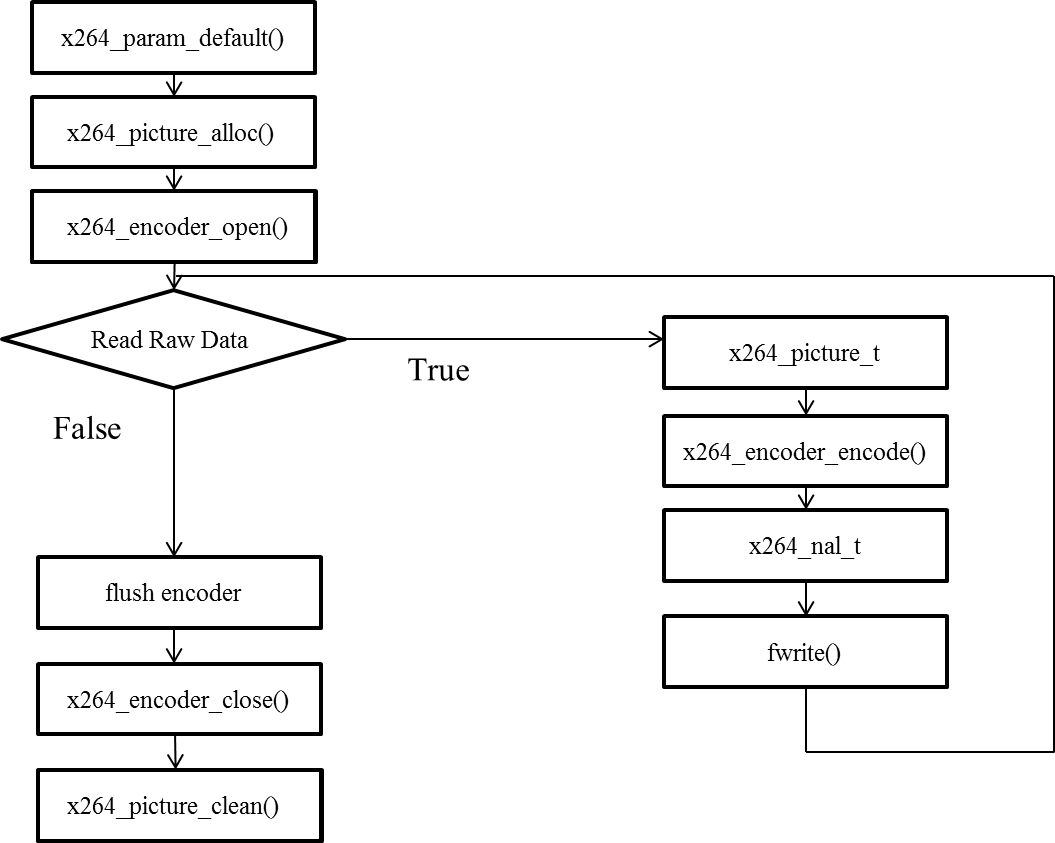


图4. 视频编码流程图

2.3 视频传输子系统设计

传统互联网的应用主要是传输文本或简单图形，传输数据量较小，对数据传输的实时性、同步性要求不高。但是通过流媒体技术所传输的对象主要是以音频内容为主体的多媒体信息，传统的互联网传输协议无法保证这些数据的准确、迅速和实时传送，必须出现新型的网络传输协议以满足高带宽、低传输延迟、支持组播模式和通道同步等特点[6]，才能保证流媒体技术在网络上的广泛应用。当前，互联网工程任务组（IETF）已经设计出几种支持流媒体传输的协议，主要有实时传输协议RTP（Real-time Transport Protocol）、实时传输控制协议RTCP（Real-time Transport Control Protocol）、实时流协议RTSP（Real-time Streaming Protocol）等。

2.3.1 RTP实时传输协议的研究

实时传输协议RTP被定义为传输音频、视频、模拟数据等实时数据的传输协议。最初设计是为了数据传输的多播，但是其也可以用于单播。与传统的注重高可靠的数据传输的传输层协议相比，它更加侧重数据传输的实时性。必须注意的是，RTP没有提供任何确保按时传送数据的机制，也没有提供任何质量保证的机制，因而要实现服务质量（Quality of Service，QoS）必须由下层网络来提供保证。同样需要需要注意的是，RTP同样不保证数据包序号传送，即使在下层网络能保证可靠性传送的条件下，也不保证数据包按序号传送。包含在RTP中的序号可供接收方用于重构数据包序列，也可用于包的定位[7]。

RTP协议负责对流媒体数据进行封包并实现实时传输，每一个RTP数据分组都由头部和负载两部分组成，其中头部前12个字节的含义是固定的，而负载则可以是音频或者视频数据[17,18]。每一个RTP包头数据格式如图5所示。

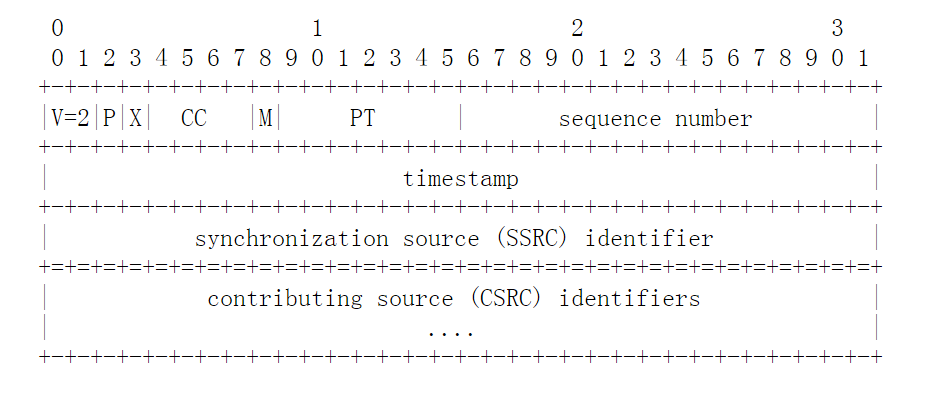


图5. RTP报文头数据格式

* V：版本号，2比特，标识RTP协议的版本号，要求通信双方的版本号相同，当前版本号为2；
* P：填充标识，1比特，表明数据区后是否有填充域，若该位被置位，则报文中就携带一位或者多位的填充字节，这些填充字节在报文的尾部，不算做有效载荷；
* X：扩展位，1比特，表明在固定头部后是否跟有扩展头，若该位被置位，则说明在固定报头后要跟且仅跟一个报文头扩展；
* CC：4比特，表示CSRC标识符在固定头后的数量；
* M：标志位，1比特，该位往往用来设置一帧视频图像中具有相同时间戳的一组数据包的最后一个。这样不必等到下个数据包的到来就能够判断一组数据包已经传输完毕；
* Timestamp：时间戳，32位，表示RTP数据包中第一个字节的采样时间。发送端在数据包中插入的一个即时时间标记，随时间的推移而增加。当数据包到达接受端后，接收端根据时间戳重新建立原始音视频的时序，也可用于同步多个不同的数据流，帮助接收方确定数据到达时间的一致性；
* Sequence number：顺序编号，16位，每发送一个RTP数据包，顺序号就加1。RTP协议通常使用UDP协议传输数据，但是UDP协议发送数据包时没有时间顺序，这就会造成数据包到达接收端时可能与发送端发出的数据包的顺序不同，为了对抵达的数据包进行重排就需要一种时间标记来进行标识。因此顺序编号主要是用来排序RTP分组，以消除重复分组，保持视频和音频流连续地播放；
* Payload Type：负载类型，7位，标识RTP分组中有效负载的格式。RTP可支持128种不同的有效载荷类型，可以根据需要更多的负载类型。对于音频流，这个域用来表示声音使用的编码类型；对于视频流，有效载荷类型可以用来表示视频编码的类型。如果发送端在会话或者广播的中途决定改变编码方法，发送端可通过这个域来通知接收端。

RTP所能支持的声音有效载荷类型如表1所示



表1

RTP所能支持的视频有效载荷类型如表2所示



表2

* SSRC：同步源标识，32位，表示RTP数据包流的来源，RTP会话的每个数据包流都有一个清楚的SSRC
* CSRC：贡献源标识，32位，确认多个包中的同步源。该域最多可有15项，每项长度位32位，列举出产生该RTP数据包的所有源。

RTP协议的数据分组和控制分组使用相邻的不同端口，这样大大提高了协议的灵活性和处理的简便性。RTP协议本身包括两部分：RTP数据传输协议和RTCP传输控制协议。为了可靠、高效地传送实时数据，RTP和RTCP必须配合使用，通常RTCP包的数量占所有传输量的5%。RTCP协议作为RTP协议的一个重要的控制补充协议，以它的反馈机制实现对流媒体服务的QoS控制，配合传输层协议，保证了流媒体的实时性特征，满足了在IP网上对QoS的需求。

2.3.2 RTCP实时传输控制协议

RTCP协议是RTP协议的补充协议，是一个控制协议，其与RTP协议共同合作，为顺序传输数据包提供可靠的传送机制，并对网络流量和阻塞进行相关控制。在RTP会话期间，每个参与者周期性地彼此发送RTCP控制包，包中封装了已发送的数据包的数量、丢失的数据包的数量、包抖动等发送端或接收端的统计信息。RTCP在流媒体传输中的主要作用为QoS管理与控制、媒体同步以及附加信息的传递。

RTCP数据包是控制包，由固定头和可变长结构元素组成，以一个32位边界结束。RTCP包可堆叠，不需要任何分隔符可将多个RTCP包连接起来形成一个RTCP组合包，以低层协议用单一包发送出去。由于需要低层协议提供整体长度决定组合包的结尾，在组合包中没有单个RTCP显式计数[19]。RTCP报头如图6所示。



图6. RTCP报头格式

Version：识别RTCP协议版本号。RTP数据包中的该值与RTCP数据包中的一致。当前规定值为2；

P：间隙，填充位（Padding）；

RC：接收方报告计数。接受放报告块的编号包含在该数据包中。

Packet Type：包括常量200，识别一个RTCP SR数据包。

在RTCP通信控制中，RTCP协议的功能是通过不同的RTCP数据报来实现的，主要有以下几种类型：

1.SR：发送端报告，发送端是指发出RTP数据报的应用程序或者终端，送端同时也可以是接收端。

2.RR：接收端报告，接收端是指仅接收但不发送RTP数据报的应用程序或者终端。

3.SDES：源描述，主要功能是作为会话成员有关信息的载体，如用户名、邮箱地址、电话号码等，此外还具有向会话成员传达会话控制信息的功能。

4.BYE：通知离开，主要功能是指示某一个或者几个源不再有效，即通知会话中的其他成员自己将退出会话。

5.APP：由应用程序自己定义，解决了RTCP的扩展性问题，并且为协议的实现者提供了很大的灵活性。

2.3.3 RTSP实时流协议

RTSP实时流协议，是一种应用层协议，用于实时地控制数据的传输。RTSP提供一个可扩展的架构来实现实时媒体资源的在线点播，如视频或音频内容。数据源可以是直播信号也可以是制作好的音视频文件。RTSP能够同时控制多个RTP的会话过程。

RTSP实时流媒体协议本身并不进行数据流的传输，而只是对数据流的传输进行控制，它包括了对RTSP连接进行建立以及类似于VCR（什么VCR）的会话控制功能，包括播放、暂停、快进、快退等。RTSP可以选择使用TCP或者UDP进行连接，因为RTSP只是发送控制信息，一般使用TCP的方式。RTSP与HTTP类似，但HTTP是一种无状态的协议，而RTSP是一种有状态的协议，客户端和服务器都可以互相发送RTSP请求。

RTSP请求报文的方法包括：OPTIONS、ANNOUNCE、DESCRIBE、SETUP、TEARDOWN、PLAY、RECORD、PAUSE、GET\_PARAMETER和SET\_PARAMETER等[8]。

2.3.4 H.264数据的RTP封装

根据前文所述，H.264分为两层，分别是网络抽象层（NAL）以及视频编码层（VCL）。当对H.264数据进行RTP封装时，RTP负载的第一个字节图7所示。

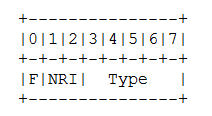


图7.RTP负载第一个字节

字节中各域的含义如下说明：

* F：1个比特

forbidden\_zero\_bit,在H.264标准中明确规定了这一位必须为0。

* NRI：2个比特

nal\_ref\_idc，取00~11，用于指示当前NAL单元的重要性，如00表明该NAL单元可以丢弃而不影响图像的回放。

* Type：5个比特

nal\_unit\_type，用于指示当前NAL单元的类型，其类型如表3所示。

|  |  |  |
| --- | --- | --- |
| 类型 | 包(Packet) | 类型名 |
| 0 | 未定义 |  |
| 1~23 | SINGLE | 单一NAL单元模式 |
| 24 | STAP-A | 单一时间的组合包 |
| 25 | STAP-B | 单一时间的组合包 |
| 26 | MTAP16 | 多个时间的组合包 |
| 27 | MTAP24 | 多个时间的组合包 |
| 28 | FU-A | 分片的单元 |
| 29 | FU-B | 分片的单元 |

表3

单一NAL单元模式

对于NAL单元的长度小于MTU大小的包，一般采用单一NAL单元模式。对于一个原始的H.264 NAL单元常由[Start Code][NALU Header][NALU Payload]三部分组成，起始码（Start Code）用于表示一个NAL单元的开始，必须是“00 00 00 01”或者“00 00 01”，NAL单元的头部仅一个字节长，其后为NAL单元的负载。图8所示为单一NAL单元模式的数据格式。

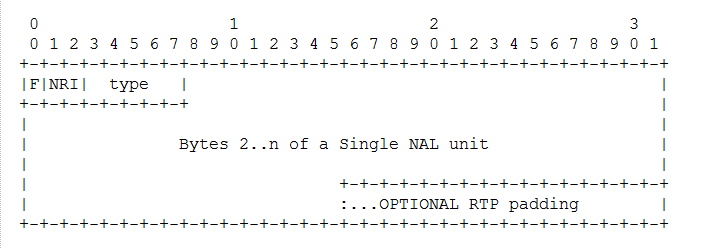


图8. 单一封包

组合封包格式

当NAL单元的数据长度非常小时，以至于可以在RTP包中封装多个NAL单元时，可以使用组合封包格式来封装H.264视频数据。组合封包格的数据格式如图9所示。

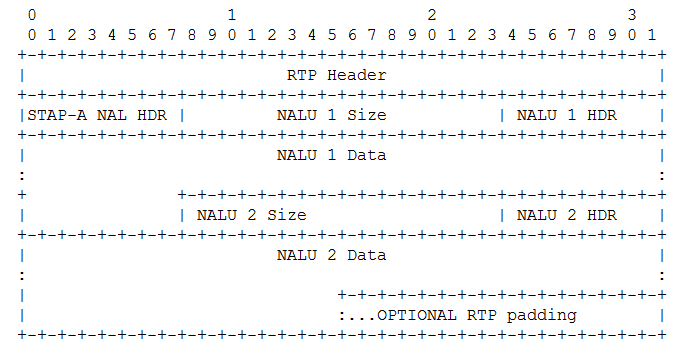


图9. 组合封包格式

分片单元格式

当NAL单元的长度超过MTU的限制时，就必须对NAL单元进行分片封包，也称为Fraamentation Units（FUs）。简单的说就是当NAL单元过长时，需要将该NAL单元拆分成多个部分，通过多个RTP数据包进行发送。分片单元的数据格式图10所示。

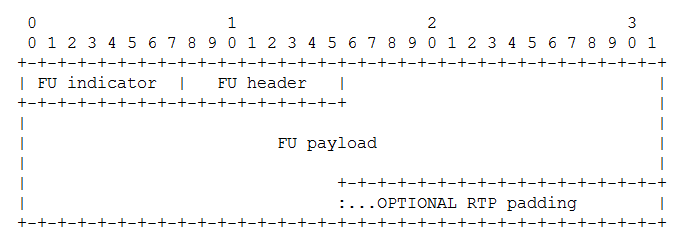


图10. 分片单元格式

从而，完整的H.264数据的封包过程如图11所示。

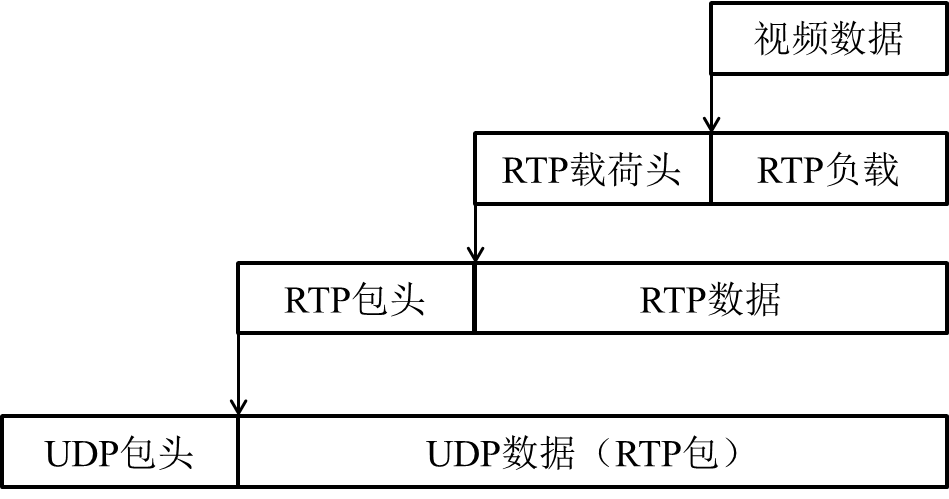


图11. H264数据封包流程图

2.3.5 视频发送子系统的实现

利用JRTPLIB库来实现RTP载荷包的数据传输，设计一个串行与视频编码压缩的进程来处理。程序设计关键点如下：

（1）一个编码后的视频帧包含若干个NAL单元，每个NAL单元都由一个起始码开头，对于H.264压缩编码算法起始码有0x000001或者0x000001构成。

（2）因特网中，数据长度大于1400字节的数据包被丢弃的概率很大，所以RTP包的数据长度不应该超过这个数值。而RTP数据包的头部需要占用12个字节，因此RTP数据包的有效负载长度应该小与1388个字节。

（3）参数 M， PT， timestamp 设置。 M，对于 NALunit 包类型，值为 1。 FU-A 类型除该 NAL 单元的最后一个分片设置为 1 外，其它为 0；载荷类型 PT 根据标准设置为 96；时间戳在 SendPacket 函数中传入的是其增量 timestamp\_inc，对于 NAL unit，由于视频采用的是 PAL 制式，故值设为 9000/25。对于同一 NAL 单元的所有 FU-A 分片包， timestamp 值应相同。因此，除第一个包设置值以外，其它为 0。

因此，视频传输子系统的程序流程图如图12所示。

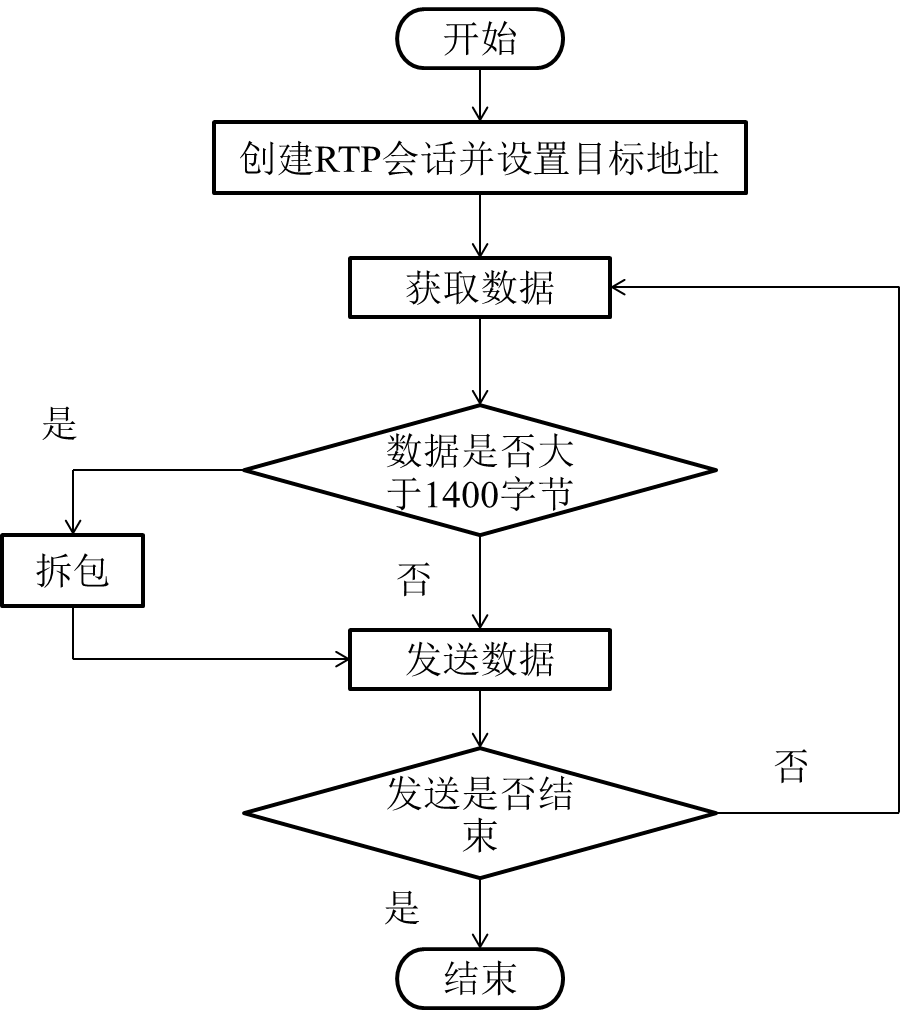


图12. JRTPLIB发送流程图

第三章、流媒体视频客户端的设计与实现

本文第二章通过研究视频采集接口、H.264视频编码压缩标准以及RTP/RTCP实时流媒体协议，分别完成了视频采集子系统、视频编码子系统以及视频传输子系统，在此基础之上，设计并实现了视频流媒体服务器，得到了视频码流。为了接收视频码流，本章提出一种基于基于Android操作系统的流媒体的客户端的设计方案，该方案主要包含三个子系统，分别是：视频码流接收子系统、视频解码子系统以及视频流回放子系统。系统架构如图13所示：

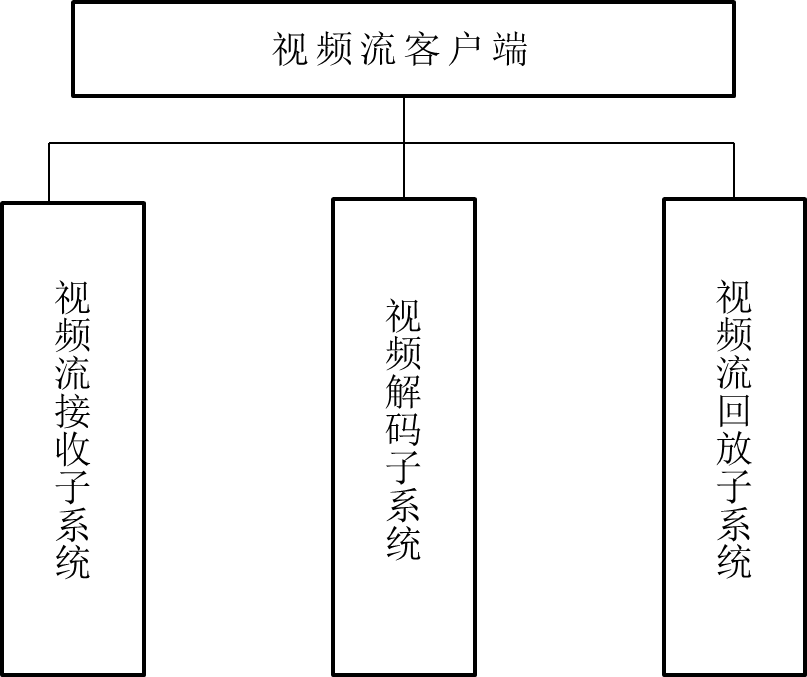


图13. 流媒体客户端架构图

视频流接收子系统利用前文所述的JRTPLIB库实现对于视频码流的接收，所不同的是，在流媒体客户端所使用的库为针对Android操作系统环境进行了适配的版本。

视频解码子系统利用FFmpeg视频解码库对于接收到的视频码流进行解码，将视频数据由H.264压缩码还原为YUV格式或者转码为RGB格式的原始视频数据。

视频回放子系统利用Android操作系统提供的ImageView控件将视频解码子系统输出的原始视频数据显示在移动终端的显示设备上，以供终端用户观看。

3.1 Android操作系统介绍

3.1.1 概述

Android（中文译名安卓）是谷歌（Google）公司在2007年11月5日推出的第一个完整、开发免费的手机操作系统，是基于Linux开放性内核的手机开发平台，在此平台上可以开发出涉及工具、管理、互联网、游戏等丰富多彩的应用[9]。安卓平台有以下几大优点：

* 开发性。安卓平台的免费与开源，大大降低了新型手机设备的研发成本。
* 开发人员自己决定应用程序的权限。
* 充分体现个性。
* 应用程序之间的无障碍沟通。在安卓平台至少有四种通信方式。
* 物理键盘和虚拟键盘双管齐下。
* 拥抱Web的时代。基于Webkit内核的Webview控件可以完成一切。
* 舒适的开发环境。

3.1.2 系统框架

安卓的系统框架从架构图上看，从顶层到底层分别是应用程序层（Applications）、应用程序框架层（Application Framwork）、系统运行层（Libraries）、安卓运行时（Android Runtime）以及Linux内核层。如图14所示。

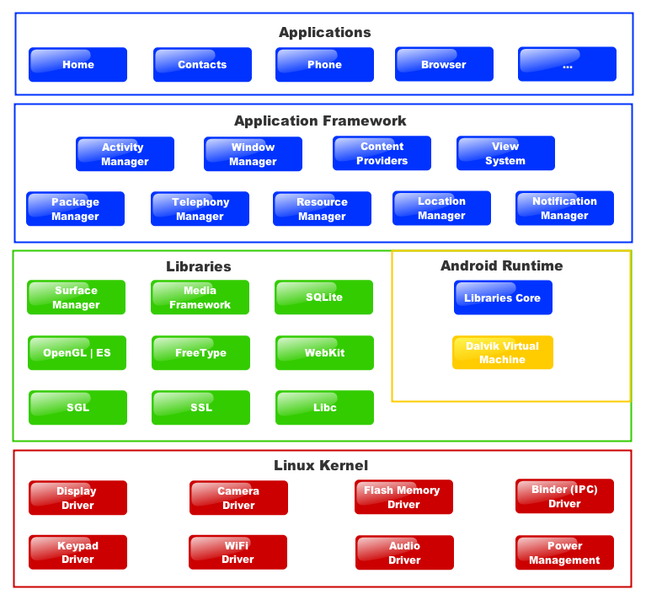


图14. Android操作系统架构图

应用程序层：该层是由运行在Android Dalvik虚拟机上的应用程序构成。该层的所有应用程序都是用Java语言编写的，应用程序包高扩E-mail客户端、SMS短消息程序、日历、地图、浏览器、联系人管理程序等。Android嵌入式平台的应用程序与其他手机系统平台相比更注重灵活性和个性，比如开发人员可以根据自己的爱好设计出个性化的电话簿、地图等应用程序，而不像其他手机操作系统直接固化这些体统软件。

应用程序框架层：该层是Google发布的编写核心应用时所使用的API框架，开发者可以完全使用相同的框架进行应用程序开发。该层主要有消息管理器、资源管理器、视图管理器等开发人员可以直接调用的应用组件组成，这些组件是我们从事Android应用程序开发的基础，同时，我们也可以对这些组件进行继承，从而开发出更具个性化的应用程序[10]。

系统运行层：Android包含一些C/C++库，这些库能够被Android系统中不同的组件使用，它们通过Android应用程序框架为开发者提供服务。

安卓运行时：Android包括一个核心库，该核心库提供了Java变成语言核心库的大多数功能。每一个Android应用程序都在自己的进程中运行，都拥有一个独立的Dalvik虚拟机实例。Dalvik被设计成一个设备，可以同时高效地运行多个虚拟系统。Dalvik虚拟机执行后缀名为.dex文件，该格式是被优化过的占用最小内存的可执行文件。同时Dalvik虚拟机是基于寄存器的，所有类都经由编译器编译，然后转换成.dex格式，有虚拟机执行。Dalvik虚拟机依赖于Linux内核的一些功能，如线程机制和低层内存管理机制。

Linux内核：Android的核心系统服务依赖于Linux2.6内核，如安全性、内存管理、进程管理、网络协议栈和驱动模型。Linux内核也同时作为硬件和软件栈之间的抽象层。

3.1.3应用程序框架

应用程序开发人员在开发应用时通过应用程序框架来与Android底层进行交互。应用程序框架可被看作一个应用程序的额核心，是所有参与开发的人员共同使用和遵守的约定，可在其约定上进行必要的扩展，但程序必须始终保持主体结构的一致性。Android提供4种通用的应用程序组件及功能。

Activity：作为Android平台的核心类，Activity被视作应用程序的入口，负责界面的显示、事件处理、控制界面跳转等操作，是Android当中最基本的模块之一。一个提供用户交互界面的Android应用程序可以由一个或多个Activity组成。Activity的生命周期如图15所示。



图15. Activity生命周期图

* onCreate()：首次创建Activity时调用。您应该在此方法中执行所有正常的静态设置，包括创建视图、将数据绑定到列表等等。系统向此方法传递一个Bundle对象，其中包含有Activity 的上一状态，不过前提是捕获了该状态。该方法后调用后接onStart()。
* onRestart()：在Activity已停止并即将再次启动前调用。该方法调用后接onStart()。
* onStart()：在Activity即将对用户可见之前调用。如果Activity转入前台，则该方法后接onResume()。如果Activity转入隐藏状态，则该方法后接onStop()。
* onResume()：在Activity即将开始与用户进行交互之前调用。此时，Activity处于Activity堆栈的顶层，并具有用户输入焦点。该方法调用后接onPause()。
* onPause()：当系统即将开始继续另一个 Activity 时调用。 此方法通常用于确认对持久性数据的未保存更改、停止动画以及其他可能消耗 CPU 的内容，诸如此类。 它应该非常迅速地执行所需操作，因为它返回后，下一个 Activity 才能继续执行。如果 Activity 返回前台，则后接 onResume()，如果 Activity 转入对用户不可见状态，则后接 onStop()。
* onStop()：Activity 对用户不再可见时调用。如果 Activity 被销毁，或另一个 Activity（一个现有 Activity 或新 Activity）继续执行并将其覆盖，就可能发生这种情况。如果 Activity 恢复与用户的交互，则后接 onRestart()，如果 Activity 被销毁，则后接 onDestroy()。
* onDestroy()：在 Activity 被销毁前调用。这是 Activity 将收到的最后调用。 当 Activity 结束，或系统为节省空间而暂时销毁该 Activity 实例时，可能会调用它。

Service：作为一个独立的进程，Service没有可视化的用户交互界面，而是一直在后台运行。可以与其他组件（如Activity、Service、ContentProvider）通过远程连接绑定（Bind）的方式进行交互。可对外（其他应用程序）提供服务交互接口。生命周期不依赖与其他组件。

Broadcast Receiver：该组件用于注册和接收广播事件，可自定义广播。大部分时间广播消息是由系统发出的，如时区变化、电池电量不足、收到短信等。除此之外应用程序也可以发送广播消息。

Content Provider：该组件提供数据库查询接口，可对外（其他应用程序）公开数据查询服务。

3.1.4 NDK简介

Android NDK（Native Development Kit）是一套Android应用程序的开发包，允许开发人员将本地代码加入到Android应用程序中。众所周知，Android应用程序运行在Dalvik虚拟机上，而NDK允许开发人员将Android应用程序中的部分功能用C/C++语言实现，并将这部分代码编译成可直接在Android平台上运行的本地代码。

Android NDK提供了大量稳定的C库头文件、JNI接口、OpenGL ES及其他的库。NDK还集成了交叉编译器，并提供了相应的mk文件隔离CPU、平台、ABI等差异，帮助开发者快速开发C/C++的动态库。开发人员只需要简单修改mk文件，就可以创建出so库文件，并能自动将so库文件和Java应用一起打包成apk文件，极大地减轻了开发人员的打包工作。NDK提供的开发工具集合，使开发人员可以便捷地开发、发布C组件。

对性能要求较高的Android应用程序应用逻辑使用NDK进行C/C++开发，从而提高应用程序的运行效率和速度。对需要保密的应用逻辑，使用NDK进行C/C开发，因为Java包都是通过反编译获取内部实现的代码。Android NDK促使专业的so组件商的出现，NDK是Android平台支持C/C开发的开端。

虽然NDK可以大大提高运行的速度，但是用NDK也会带来很多副作用。如使用NDK并不是总会提高应用程序的性能，但却一定会增加应用程序的复杂度，而且使用NDK必须要自己控制内存的分配和释放，这样将无法利用Dalvik虚拟机来管理内存，也会给应用程序带来很大的风险。因此，NDK适合用于独立的、占用内存少、占用较多CPU资源的运算过程，如信号处理、物理仿真等。

3.1.5 JNI介绍

前述NDK组件主要用于从源生C/C++代码生成在Android操作系统下可用的第三方静态库，而如果在Android应用程序通过调用C/C++代码来实现Java与C/C++的混合编程，则需要用到JNI（Java Native Interface）应用程序框架[22]。

Java本地程序接口（Java Native Interface）是JDK（Java Development Kit）的一部分，通过使用Java本地程序接口书写程序，可以确保代码在不同的平台上方便移植。JNI使得运行与Java虚拟机的代码可以操作用C/C++、汇编等语言书写的应用程序或库。同时也可以通过调用相应的接口函数将Java虚拟机内嵌到本地应用程序中[23]。

本文中所采用的第三库均由C/C++语言写就，因此仅仅使用Java语言无法满足编程要求。而使用JNI编写Java本地方法，则可以很好地解决这个问题。通常在以下几种情况下可以考虑使用JNI：

（1）平台相关。Java语言是平台无关的编程语言。若应用程序需要访问平台相关的特性，而标准Java语言类库并不支持该特性。

（2）注重速度。Java代码通常比本地（C/C++）代码在执行速度上要慢一些。如果在某段应用程序的执行速度上有较高要求，可使用C/C++编写代码，然后在Java通过JNI调用基于C/C++编写的代码，常常能获得更快的运行速度。

（3）硬件控制。为了更好地控制硬件，硬件控制代码通常使用C语言编写，通过JNI将其与Java层代码连接起来，从而实现对硬件的控制。

（4）代码复用。在程序编写过程中，常常会使用一些已经编写好的第三方C/C++代码，既提高了编程效率，又确保了程序的安全性与健壮性。复用这些C/C++代码，就可通过JNI来实现。

应用JNI技术开发的应用技术框架如图16所示。

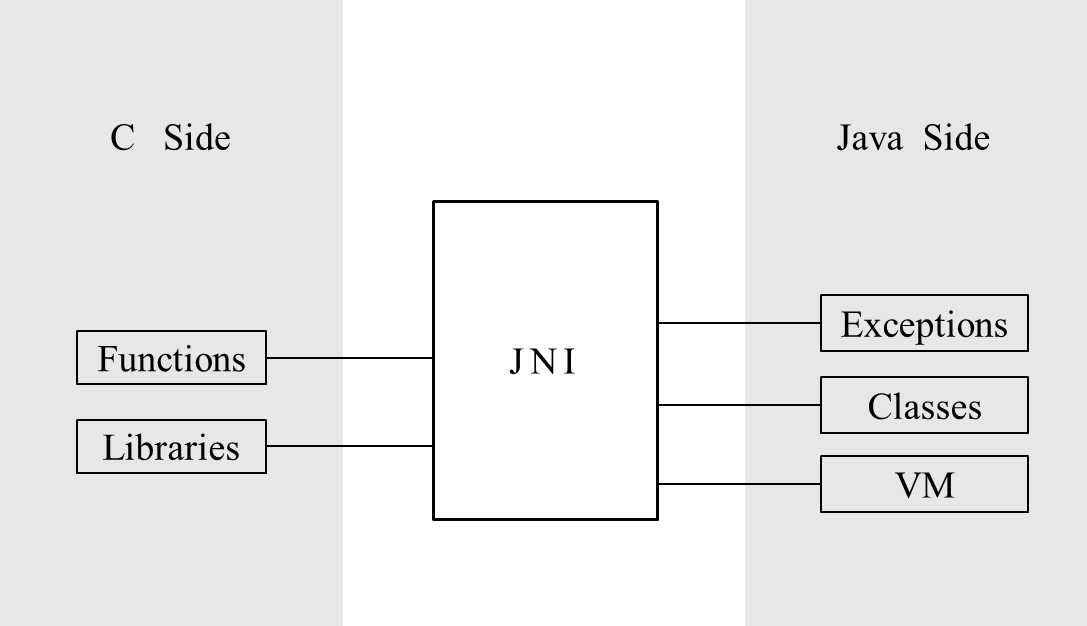


图16. JNI应用技术框架

从图jni可以看出，JNI是一种包容极广的编程接口。JNI提供了完整的应用程序接口，允许将Java虚拟机直接嵌入到本地的应用程序中。

在JNI技术中，由于是两种语言环境的信息交换，所以存在着数据类型的对应问题，这也是JNI技术实现不同语言之间相互通信的基础。JNI处理基本数据类型和引用数据类型的方法不尽相同。对基本数据类型的映射是直接的，如Java语言中的int被映射为C++语言中的jint[24]。具体的映射关系如表4所示。

|  |  |  |
| --- | --- | --- |
| Java类型 | 本地类型 | 物理存储 |
| boolean | jboolean | unsigned,8bit |
| byte | jbyte | signed,8bit |
| char | jchar | unsigned,16bit |
| short | jshort | signed,16bit |
| int | jint | signed,32bit |
| long | jlong | signed,64bit |
| float | jfloat | 32float |
| double | jdouble | 64bit |
| oid | void | n/a |

表4. Java类型与本地类型对照

3.2 JRTPLIB库的研究

JRTPLIB是Jori Liesenborgs开发的基于Windows、Linux等操作系统下，用C++实现的一个RTP/RTCP开源协议库，其完全遵守RFC1889设计，适合网络多媒体的开发使用。JRTPLIB库用于发送和接收RTP数据，自动发送RTCP数据[11,12,13]。在视频接收子系统的设计中涉及到的类有以下几类：

* RTPSession Class：该类为RTP高层类，内部实现了数据接收和发送相关的大部分函数。如会话创建函数Create()，RTP分组发送函数SendPacket()以及游标定位函数GoToNextSourceWithData()等。它内置的处理了RTCP部分，不需要额外的进行设计。
* RTPSessionParams Class：该类主要用来描述RTPSession实例将要使用的参数。相关的成员函数有，设置时间戳函数SetOwnTimeStampUnit，设置RTP分组最大长度函数SetMaximumPacketSize等。
* RTPUDPv4TransmissionParams Class：主要用于RTP发送端口设置，如函数SetPortbase。
* RTCPPacket Class：该类是各种RTCP包的基类，其中两个子类RTCPRRPacket和RTCPSRPacket分别用于统计RTCP发送报告和接收报告。
* RTCPSourceData Class：主要负责维护会话参与者的相关信息，如SSRC，RTCP统计信息等。

调用JRTPLIB库接收RTP数据包工作流程如图17所示。

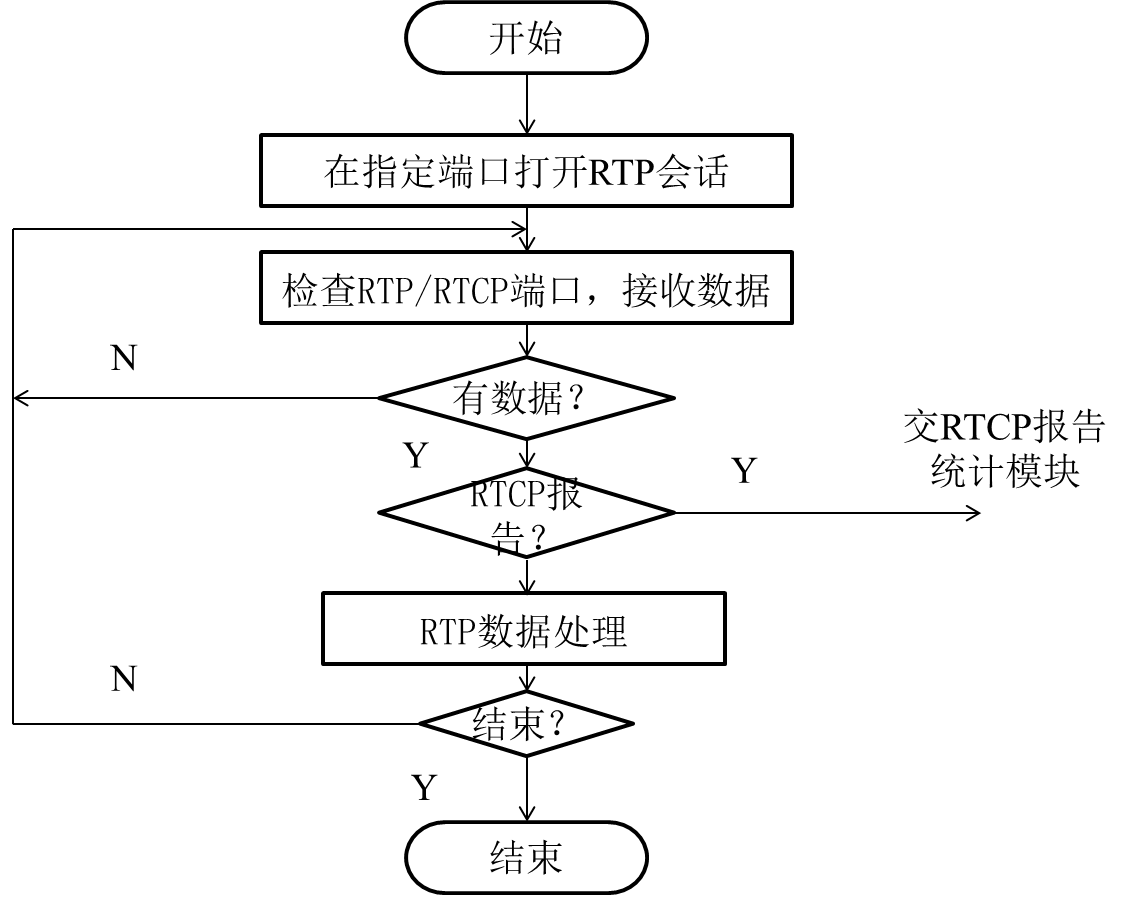


图17. RTP数据包接收流程图

3.3 视频解码子系统的设计与实现

3.3.1 FFmpeg解码库的研究

FFmpeg是一套对音频和视频进行解码录制转换的完整开源库。它能够快速实现音视频格式转换，也能从音视频源获取数据并将其保存。用户可以设置FFmpeg命令行的参数来控制音视频数据。FFmpeg也能转换不同的采样速率，还可以通过高质量的多相滤波器来调整视频大小。FFmpeg可以在命令行界面输入不同的参数来对音视频文件进行转换控制。对视频的设置包括比特率、帧率、帧大小、屏幕高宽比等基本参数以及设置DCT算法、IDCT算法、视频量化器比例、压缩范围、帧间偏移等高级参数。在捕获视频数据的时候可以设置捕获设备，设置视频输入标准包括NTSC、PAL以及SECAM，对于DV1394还可以设置视频捕获通道[20]。FFmpeg使用V4L2架构的视频源以及其他任何开源系统的音频源。在使用FFmpeg 的时候必须先激活正确的视频源设备。它支持多种视频输入格式，如YUV文件、YUV420P等。FFmpeg能够同时设置输入与输出文件，同时对音视频进行转换。它所支持的视频输入格式包括MPEG audio、MPEG-1 systems、MPEG-2 PS、MPEG-2 TS、ASF、AVI、WAV、FLV、MPEG-4等多种格式，同它所支持的视频编解码器也相当丰富。FFmpeg能够在多种平台下运行，包括Linxu、Free BSD、Windows、Mac OS X、BeOS等操作系统[14,15]。

3.3.2 常用FFmpeg的结构体

在使用FFmpeg解码视频过程会涉及到很多的结构体，同时FFmpeg当中提供有非常多的结构体用于视频的编解码，这其中，关键的结构体可以分为如下几类：

（1）解协议类

此处的协议主要是指实时流媒体协议，如RTSP、RTP/RTCP、MMS等。此类结构体有AVIOContext，URLProtocol，URLContext等，主要存储音视频流使用的协议类型以及状态。每种协议都对应一个URLProtocol结构。

（2）解封装类

此处的封装是指音视频文件的容器类型，主要类型有flv、avi、rmvb、mp4、mkv等。此类结构体主要有AVFormatContext和AVInputFormat。AVFormatContext主要存储音视频封装格式中包含的信息；AVInputFormat存储输入音视频使用的封装格式。每种音视频封装格式都对应一个AVInputFormat结构

（3）解码类

此类结构体主要有AVStream、AVCodecContext以及AVCodec等。每个AVStream存储一个音频/视频流的相关数据；每个AVStream对应一个AVCodecContext，存储该音频/视频流使用解码方式的相关数据；每个AVCodecContext对应一个AVCodec，包含该音频/视频对应的解码器；每种解码器对应一个AVCodec结构。

（4）缓存数据类

对于视频流，每个结构体通常会存储一帧数据；对于音频流，每个结构会存储多帧数据。解码前的压缩数据缓存使用AVPacket结构，解压缩后的数据缓存使用AVFrame结构。

AVFrame：该结构体一般用于存储原始数据，即非压缩数据，例如对视频数据来说是YUV，RGB等格式，对于音频数据来说是PCM，此外还包含了一些相关的信息。比如，在解码的时候存储了宏块类型表，运动矢量表等数据。

AVPacket：该结构体是存储压缩编码数据相关信息的结构体。其中比较重要的数据域有显示时间戳pts、解码时间戳dts以及压缩编码的数据数组data。

AVFormatContext：在使用FFmpeg进行视频解码的时候，AVFormatContext是一个贯穿始终的数据结构，是FFmpeg解封装功能的结构体。其中比较重要的数据域有输入数据的缓存指针AVIOContext \*pb，时长duration，比特率bit\_rate以及文件名数组filename。

AVCodecContext：该结构体主要用于在解码的过程中配置编解码上下文信息的设置。其中很多参数是编码的时候使用，而不是在解码的时候使用。这其中比较重要的数据域有视频帧的宽width，视频帧的高height，采样格式sample\_fmt编解码器的类型codec\_type，运动估计参考帧的个数refs等。

AVCodec：该结构体用于存储编解码器的信息。

3.3.2 视频解码的实现

视频解码的输入文件来源于接收子系统获得到的实时视频流，该视频码流的编码标准所采用的YUV420P的格式，满足FFmpeg开源的格式要求，故而无需进行格式转换即可进行视频的解码。在使用FFmpeg解码之前，首先需要对FFmpeg库进行初始化，注册所有的编解码器以及文件格式，设置解码器码率、帧速率、编码像素格式等参数，然后寻找解码器并打开，之后才可以进行解码。其解码流程如图18所示。

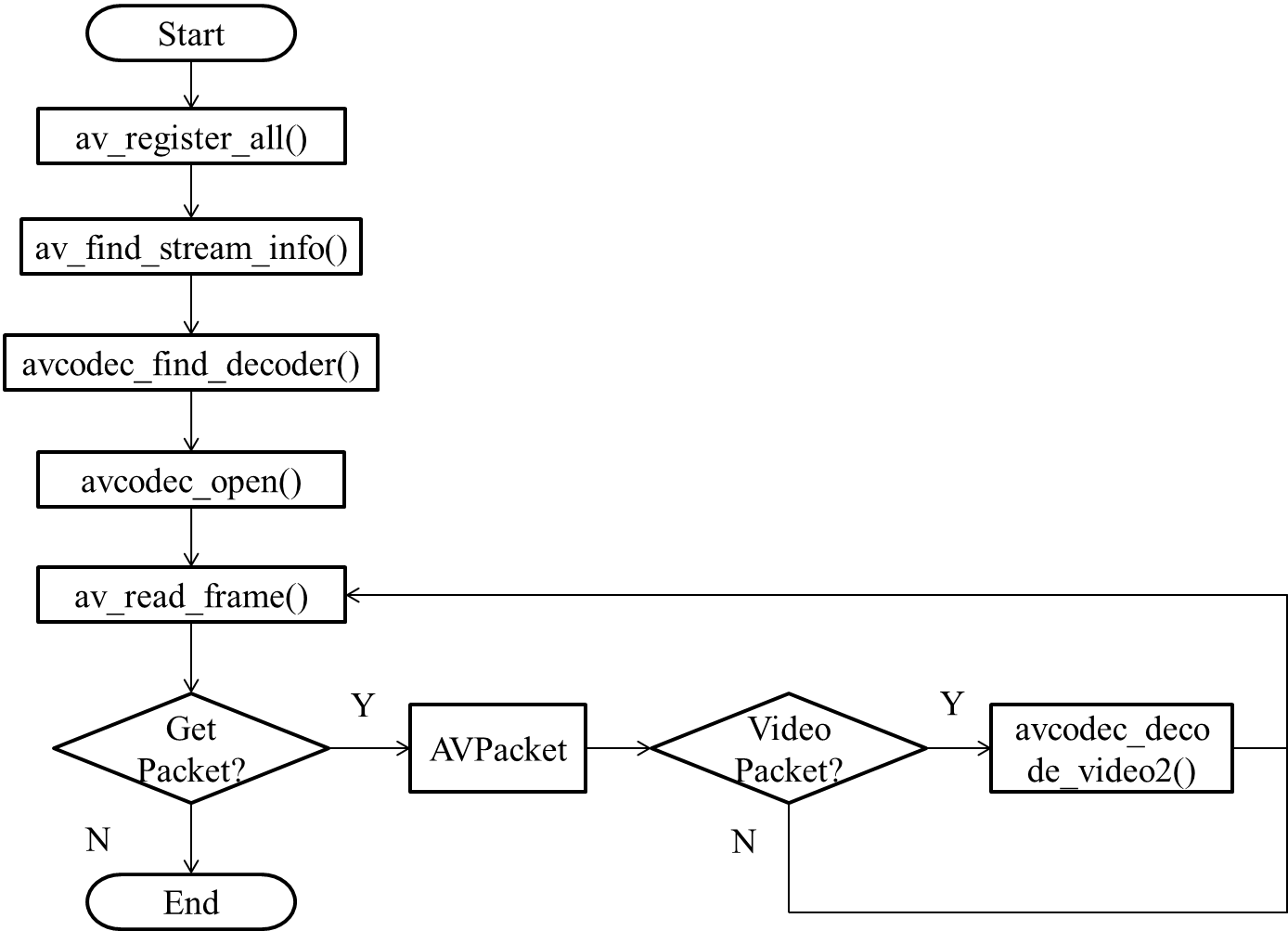


图18. FFmpeg解码流程图

av\_register\_all()：注册库中含有所有文件格式和编解码器，没有这一步无法打开编解码器。

av\_find\_decoder()：寻找解码器，解码器参数需在pCodec中初始化，参数的初始化很重要，对解码的图像质量影响较大。

avcodec\_open()：打开解码器。

avcodec\_alloc\_frame()：为解码帧分配内存。

av\_read\_frame()：从视频流中读取一帧视频数据。

avcodec\_decode\_video2：解码一帧视频数据。

3.4 视频回放子系统的设计与实现

Android 中的图形系统采用Client/Server 架构。Server (即 Surface Flinger) 用 C++语言编写而成，它主要负责处理 Client 端的生成界面请求，提供法实现各个 Surface界面的层叠显示，然后通过 Open GL ES 显示到 Frame Buffer 上。Client 端（即 Surface）代码分为两部分，与底层驱动等底层实现有关的部分使用 C++语言编写，与 Skia 和Open GL 交互的部分由 Java 编写，它们一起组成了供应用程序使用的 API 函数。Client和 Server 使用 Binder 机制通信，图19概要介绍了 Android 图形系统的架构。



图19. Android图形系统结构

（1）Surface

Surface是Android基本的图形绘制单元，类似于Windows系统的窗口，一个应用程序对应着一个或多个界面，每个界面就是一个Surface。Java程序中的View及其子类，都要绘制在一个Surface上。每个Surface将创建一个Canvas对象，用它来管理View和SurfaceView在Surface上的操作，比如画点和画线。每个Canvas对象又对应一个bitmap，存储画在Surface上的内容。

（2）Layer

每个 Surface 都对应着一个 Layer，Layer 中包含有相应 Surface 的缓存区信息，客户端通过 Binder 机制请求 SurfaceFlinger 分配一个 Layer，同时将 Layer 和 Surface 绑定。通过Layer可以更改 Surface的内容，可以实现对窗口的控制，还可以调用OpenGL或者Skia来加强显示效果。

（3）SurfaceFlinger

SurfaceFlinger在真个图形系统中担任着Server的角色，它接受作为客户端的Surface组件产生的各种要求，完成相应的功能[16]。

本文所采用的视频数据显示方案需要的主要控件为Android操作系统提供的ImageView控件。ImageView继承自android.view.View，用于显示任意图像。ImageView控件可以通过setImageBitmap方法将需要显示的图像设置为一个Bitmap图像。本文通过JNI接口获取来自已解码的YUV视频数据，然后根据视频帧数据生成Bitmap对象，最终通过ImageView将Bitmap对象显示在移动设备的显示屏上。

第四章、总结与展望

本文在完成了流媒体服务器端和流媒体客户端的设计与开发后，将系统的两个部分置于同一个局域网当中进行实验，结果表明在实验室环境下该系统可以完成预期的功能。对于服务质量的控制，本文设计的系统采用的方案是通过RTCP数据包在服务器端和客户端发送应用程序自定义APP包用于分别通知系统的参与者当前系统的工作状况。

而当前系统存在的不足是当网络出现抖动或者丢包率较高时，没有对应的措施，在系统之后得演进中，需要针对这些问题进行改进，同时将视频的分发方式由当前的单播的方式改进为多播方式以节省网络带宽，容纳更多的系统参与者，即用户。

在本文完成之时，国际上出现了下一代的视频编码压缩标准，分别是国际电信联盟制定的H.264/HEVC标准以及由谷歌公司开发的VP9标准。H.265是ITU-T VCEG 继H.264之后所制定的新的视频编码标准。H.265标准围绕着现有的视频编码标准H.264，保留原来的某些技术，同时对一些相关的技术加以改进。新标准使用先进的技术以改善码流、编码质量、延时和算法复杂度之间的关系，达到最优化设置。具体的研究内容包括：提高压缩效率、提高鲁棒性和错误恢复能力、减少实时的时延、减少信道获取时间和随机接入时延、降低复杂度等。H264由于算法优化，可以低于1Mbps的速度实现标清数字图像传送；H265则可以实现利用1~2Mbps的传输速度传送720P（分辨率1280\*720）普通高清音视频传送。

反复的质量比较测试已经表明，在相同的图象质量下，相比于H.264，通过H.265编码的视频大小将减少大约39-44%。通过主观视觉测试得出的数据显示，在码率减少51-74%的情况下，H.265编码视频的质量还能与H.264编码视频近似甚至更好，其本质上说是比预期的信噪比(PSNR)要好。

故本系统在以后的改进中，可以将视频编码压缩标准由H.264升级为H.265，以适应高清视频的传输。

致 谢

短短的三年研究生生活即将结束，马上要走上新的工作岗位。在此首先我要感谢我的导师赵磊教授。导师渊博的知识、踏实严谨的工作作风、精益求精

的实干精神、清晰的思路及敏锐的洞察力，使我深受感染、受益匪浅。从论文课题的选题、开题、实验到论文的撰写，导师都倾注了大量的心血。无论是在学术上还是在生活上，导师都给予了我极大的帮助。在今后的道路上，导师正直、认真、踏实、热情、无私的优秀个人品质将不断激励我。在此，我要向尊敬的赵磊老师表达最崇高的敬意和衷心的感谢！

感谢王雷老师和刘程鹏、刘帅同学在实验工作和论文写作过程中给予的帮助与支持。

感谢在这三年的研究生学习过程中，给予我知识和帮助的各位老师和同学。特别要感谢的是我的舍友刘程鹏、刘帅、安建瑞，他们对知识的渴望和对学习的认真态度，让我有莫大的收获。

感谢我的家人，是父母的鼓励和支持，使我在求学的道路上一步步走到现在，你们永远是我努力求学的坚强后盾。

最后，感谢各位专家和老师在百忙之中对本文的审阅和指导！