DoubleLi
517712484

< 2018年2月 >

日	一	二	三	四	五	六
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	1	2	3
4	5	6	7	8	9	10

公告

昵称：DoubleLi
园龄：8年1个月
粉丝：1183
关注：29
+加关注

搜索

找找看
谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

随笔分类(4352)

ASP.NET(30)
ASP.NET MVC(11)
Boost(105)
c#(8)
C++/C(700)
c++11(12)
cmake(25)
com/ATL/Activex(75)
Css(16)
CxImage(12)
DataBase(29)
DirectX(16)
Extjs(13)
ffmpeg、ffplay(92)
FREESWITCH
gcc/g++/gdb(29)
gis(1)
h264/h265/mpeg(21)
hisi(11)
hls(2)
html5(3)
http(10)
ICE(4)
java(16)
javascript(66)
jpeglib(3)
Jquery(9)
json(6)
JUCE(2)
libcurl(26)
LINPHONE

H264码流打包分析(精华)

H264码流打包分析

SODB　数据比特串 - - > 最原始的编码数据

RBSP　原始字节序列载荷 - - > 在SODB的后面填加了结尾比特（RBSP trailing bits　一个bit“1”）若干比特“0”,以便字节对齐。

EBSP　扩展字节序列载荷-- >在RBSP基础上填加了仿校验字节（0X03）它的原因是:　在NALU加到Annexb上时，需要填加每组NALU之前的开始码 StartCodePrefix,如果该NALU对应的slice为一帧的开始则用4位字节表示，ox00000001,否则用3位字节表示 ox000001.为了使NALU主体中不包括与开始码相冲突的，在编码时，每遇到两个字节连续为0，就插入一个字节的0x03。解码时将0x03去掉。 也称为脱壳操作。

h264的功能分为两层，视频编码层（VCL）和网络提取层（NAL）

VCL数据即被压缩编码后的视频数据序列。在VCL数据要封装到NAL单元中之后，才可以用来传输或存储。NAL单元格式如下图:

Nal头	EBSP	Nal头	EBSP	Nal头	EBSP

NAL单元
每个NAL单元是一个一定语法元素的可变长字节字符串，包括包含一个字节的头信息（用来表示数据类型），以及若干整数字节的负荷数据。一个NAL单元可以携带一个编码片、A/B/C型数据分割或一个序列或图像参数集。

NAL单元按RTP序列号按序传送。其中，T为负荷数据类型，占5bit；R为重要性指示位，占2个bit；最后的F为禁止位，占1bit。具体如下:

（1）NALU类型位

可以表示NALU的32种不同类型特征，类型1～12是H.264定义的，类型24～31是用于H.264以外的，RTP负荷规范使用这其中的一些值来定义包聚合和分裂，其他值为H.264保留。

（2）重要性指示位

用于在重构过程中标记一个NAL单元的重要性，值越大，越重要。值为0表示这个NAL单元没有用于预测，因此可被解码器抛弃而不会有错误扩散；值高于0表示此NAL单元要用于无漂移重构，且值越高，对此NAL单元丢失的影响越大。

（3）禁止位

编码中默认值为0，当网络识别此单元中存在比特错误时，可将其设为1，以便接收方丢掉该单元，主要 用于适应不同种类的网络环境（比如有线无线相结合的环境）。例如对于从无线到有线的网关，一边是无线的非IP环境，一边是有线网络的无比特错误的环境。假 设一个NAL单元到达无线那边时，校验和检测失败，网关可以选择从NAL流中去掉这个NAL单元，也可以把已知被破坏的NAL单元前传给接收端。在这种情 况下，智能的解码器将尝试重构这个NAL单元（已知它可能包含比特错误）。而非智能的解码器将简单地抛弃这个NAL单元。NAL单元结构规定了用于面向分 组或用于流的传输子系统的通用格式。在H.320和MPEG-2系统中，NAL单元的流应该在NAL单元边界内，每个NAL单元前加一个3字节的起始前缀 码。在分组传输系统中，NAL单元由系统的传输规程确定帧界，因此不需要上述的起始前缀码。一组NAL单元被称为一个接入单元，定界后加上定时信息 （SEI），形成基本编码图像。该基本编码图像（PCP）由一组已编码的NAL单元组成，其后是冗余编码图像（RCP），它是PCP同一视频图像的冗余表 示，用于解码中PCP丢失情况下恢复信息。如果该编码视频图像是编码视频序列的最后一幅图像，应出现序列NAL单元的end，表示该序列结束。一个图像序 列只有一个序列参

数组，并被独立解码。如果该编码图像是整个NAL单元流的最后一幅图像，则应出现流的end。

H.264采用上述严格的接入单元，不仅使H.264可自适应于多种网络，而且进一步提高其抗误码能力。序列号的设置可发现丢的是哪一个VCL单元，冗余编码图像使得即使基本编码图像丢失，仍可得到较"粗糙"的图像。

实现RTP协议的H.264视频传输系统

1. 引言

随着信息产业的发展，人们对信息资源的要求已经逐渐由文字和图片过渡到音频和视频，并越来越强调获取资源的实时性和互动性。但人们又面临着另外一种不可避免的尴尬，就是在网络上看到生动清晰的媒体演示的同时，不得不为等待传输文件而花费大量时间。为了解决这个矛盾，一种新的媒体技术应运而生，这就是流媒体技术。流媒体由于具有启动时延小、节省客户端存储空间等优势，逐渐成为人们的首选，流媒体网络应用也在全球范围内得到不断的发展。其中实时流传输协议 RTP 详细说明了在互联网上传递音频和视频的标准数据包格式，它与传输控制协议 RTCP 配合使用，成为流媒体技术最普遍采用的协议之一。

H.264/AVC 是ITU-T 视频编码专家组（VCEG）和ISO/IEC 动态图像专家组（MPEG）联合组成的联合视频组（JVT）共同努力制订的新一代视频编码标准，它最大的优势是具有很高的数据压缩比率，在同等图像质量的条件下，H.264 的压缩比是MPEG-2 的2 倍以上,是MPEG-4的1.5～2 倍。同时，采用视频编码层（VCL）和网络提取层（NAL ）的分层设计，非常适用于流媒体技术进行实时传输。本文就是基于 RTP 协议，对 H.264 视频进行流式打包传输，实现了一个基本的流媒体服务器功能，同时利用开源播放器VLC 作为接收端，构成一个完整的H.264 视频传输系统。

2. RTP 协议关键参数的设置

RTP 协议是 IETF 在 1996 年提出的适合实时数据传输的新型协议。RTP 协议实际上是由实时传输协议RTP（Real-time Transport Protocol）和实时传输控制协议RTCP（Real-time Transport Control Protocol）两部分组成。RTP 协议基于多播或单播网络为用户提供连续媒体数据的实时传输服务；RTCP 协议是 RTP 协议的控制部分，用于实时监控数据传输质量，为系统提供拥塞控制和流控制。RTP 协议在RFC3550 中有详细介绍。每一个 RTP 数据包都由固定包头（Header ）和载荷（Payload）两个部分组成，其中包头前12个字节的含义是固定的，而载荷则可以是音频或视频数据。RTP 固定包头的格式如图1所示:

图1 RTP固定包头格式

Fig. 1 the RTP fixed header format

其中比较关键的参数设置解释如下:

（1）标示位（M ）：1 位，该标示位的含义一般由具体的媒体应用框架（profile ）定义， 目的在于标记处RTP 流中的重要事件。

（2）载荷类型（PT）：7 位，用来指出RTP负载的具体格式。在RFC3551中，对常用的音视频格式的RTP 传输载荷类型做了默认的取值规定，例如，类型2 表明该RTP数据包中承载的是用ITU G.721 算法编码的语音数据，采用频率为 8000HZ，并且采用单声道。

（3）序号:16 位，每发送一个 RTP 数据包，序号加 1。接受者可以用它来检测分组丢失和恢复分组顺序。

（4）时间戳：32 位，时间戳表示了 RTP 数据分组中第一个字节的采样时间，反映出各 RTP 包相对于时间戳初始值的偏差。对于RTP 发送端而言，采样时间必须来源于一个线性单调递增的时钟。

从 RTP 数据包的格式不难看出，它包含了传输媒体的类型、格式、序列号、时间戳以及

是否有附加数据等信息。这些都为实时的流媒体传输提供了相应的基础。而传输控制协议RTCP为 RTP传输提供了拥塞控制和流控制，它的具体包结构和各字段的含义可参考RFC3550，此处不再赘述。

3. H.264 基本流结构及其传输机制

3.1 H.264 基本流的结构

H.264 的基本流（elementary stream,ES）的结构分为两层，包括视频编码层（VCL）和网络适配层（NAL）。视频编码层负责高效的视频内容表示，而网络适配层负责以网络所要求的恰当的方式对数据进行打包和传送。引入NAL并使之与VCL分离带来的好处包括两方面：其一、使信号处理和网络传输分离，VCL 和NAL 可以在不同的处理平台上实现；其二、VCL 和NAL 分离设计，使得在不同的网络环境内，网关不需要因为网络环境不同而对VCL比特流进行重构和重编码。

H.264 的基本流由一系列NALU （Network Abstraction Layer Unit ）组成，不同的NALU数据量各不相同。H.264 草案指出[2]，当数据流是储存在介质上时，在每个NALU 前添加起始码：0x000001，用来指示一个 NALU的起始和终止位置。在这样的机制下，*在码流中检测起始码，作为一个NALU得起始标识，当检测到下一个起始码时，当前NALU结束。每个NALU单元由一个字节的 NALU头（NALU Header）和若干个字节的载荷数据（RBSP）组成。其中NALU 头的格式如图2 所示：

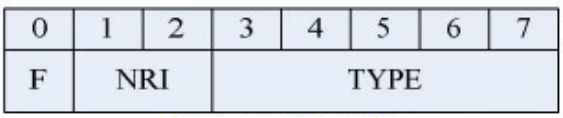

图2 NALU单元头格式

Fig. 2 the NAL Unit header format

F：forbidden_zero_bit.1 位，如果有语法冲突，则为 1。当网络识别此单元存在比特错误时，可将其设为 1，以便接收方丢掉该单元。

NRI：nal_ref_idc.2 位，用来指示该NALU 的重要性等级。值越大，表示当前NALU越重要。具体大于0 时取何值，没有具体规定。

Type：5 位，指出NALU 的类型。具体如表1 所示：

表1 H.264建议规定的NALU类型

Table. 1 the NAL Unit format suggested by H.264 standard

nal_unit_type	NALU 内容	RBSP 语法结构	C
0	未指定		
1	非 IDR 图像的编码 slice	slice_layer_without_partitioning_rbsp()	2,3,4
2	编码 slice 数据划分 A	slice_data_partition_a_layer_rbsp( )	2
3	编码 slice 数据划分 B	slice_data_partition_b_layer_rbsp( )	3
4	编码 slice 数据划分 C	slice_data_partition_c_layer_rbsp( )	4
5	IDR 图像中的编码 slice	slice_layer_without_partitioning_rbsp( )	2,3
6	SEI（补充增强信息）	sei_rbsp( )	5
7	sps（序列参数集）	seq_parameter_set_rbsp( )	0
8	pps（图像参数集）	pic_parameter_set_rbsp( )	1
9	接入单元定界符	access_unit_delimiter_rbsp( )	6
10	序列结束	end_of_seq_rbsp( )	7
11	码流结束	end_of_stream_rbsp( )	8
12	填充数据	filler_data_rbsp( )	9
13~23	保留		
24~31	未指定		

需要特别指出的是，NRI 值为 7 和 8 的NALU 分别为序列参数集（sps）和图像参数集（pps）。参数集是一组很少改变的，为大量VCL NALU 提供解码信息的数据。其中序列参数集作用于一系列连续的编码图像，而图像参数集作用于编码视频序列中一个或多个独立的图像。如果*没能正确接收到这两个参数集，那么其他NALU 也是无法解码的。因此它们一般在发送其它 NALU 之前发送，并且使用不同的信道或者更加可靠的传输协议（如TCP）进行传输，也可以重复传输。

3.2 适用于 H.264 视频的传输机制

前面分别讨论了RTP 协议及H.264基本流的结构，那么如何使用RTP协议来传输H.264视频了?一个有效的办法就是从H.264视频中剥离出每个NALU，在每个NALU前添加相应的RTP包头，然后将包含RTP 包头和NALU 的数据包发送出去。下面就从RTP包头和NALU两方面分别阐述。

完整的 RTP 固定包头的格式在前面图 1 中已经指出，根据RFC3984[3]，这里详细给出

2011年5月 (1)

文章分类(2)
SilverLight(1)
sql server(1)

参考博客
boost使用
boost使用
chenyujing1234
com
Dean Chen的专栏
boost 等
ffmpeg参考
ffmpeg参考
linux驱动
morewindows
Nginx模块开发与原理剖析
Nginx模块开发与原理剖析
opencv教程
Sloan
webrtc参考一
webrtc参考一
大坡3D软件开发
个人开发历程知识库
关注DirectX
关注DirectX
回忆未来-向东
http://justwinit.cn/index.php
雷霄骅(leixiaohua1020)的专栏
雷霄骅(leixiaohua1020)的专栏
音视频FFmpeg等
音视频FFmpeg等

最新评论
1. Re:Websocket协议的学习、调研和实现
你好 要是客户端发来的数据不全是字符串的话 服务端该怎么解析呢? 比如客户端发来的数据是 var buffer = new ArrayBuffer(2); var int16 = new I.....
--kongfu
2. Re:Websocket协议的学习、调研和实现
解析得很明白 谢谢
--kongfu
3. Re:非IE内核浏览器支持activex插件
大神您好，能不能分享下插件 18611145647@163.com
--追风逐雨
4. Re:用Eclipse和GDB构建ARM交叉编译和在线调试环境
\(^o^)/~
--秋风扫落木
5. Re:WebRTC学习与DEMO资源一览
我用java的，用Netty做了个简单im，现在想做个视频通讯，不想用Nodejs，应该用那个demo
--老牛在路上

阅读排行榜
1. Nginx之location 匹配规则详解(141027)
2. SVN安装配置与使用(84792)
3. C++的Json解析库：jsoncpp和boost .(78294)

各个位的具体设置。

V：版本号，2 位。根据RFC3984，目前使用的RTP 版本号应设为0x10。

P：填充位，1 位。当前不使用特殊的加密算法，因此该位设为 0。

X：扩展位，1 位。当前固定头后面不跟随头扩展，因此该位也为 0。

CC：CSRC 计数，4 位。表示跟在 RTP 固定包头后面CSRC 的数目，对于本文所要实现的基本的流媒体服务器来说，没有用到混合器，该位也设为 0x0。

M：标示位，1 位。如果当前 NALU为一个接入单元最后的那个NALU，那么将M位置 1；或者当前RTP 数据包为一个NALU 的最后的那个分片时（NALU 的分片在后面讲述），M 位置 1。其余情况下M 位保持为 0。

PT：载荷类型，7 位。对于H.264 视频格式，当前并没有规定一个默认的PT 值。因此选用大于 95 的值可以。此处设为0x60（十进制96）。

SQ：序号，16 位。序号的起始值为随机值，此处设为 0，每发送一个RTP 数据包，序号值加 1。

TS：时间戳，32 位。同序号一样，时间戳的起始值也为随机值，此处设为0。根据RFC3984， 与时间戳相应的时钟频率必须为90000HZ。

SSRC：同步源标示，32 位。SSRC应该被随机生成，以使在同一个RTP会话期中没有任何两个同步源具有相同的SSRC 识别符。此处仅有一个同步源，因此将其设为0x12345678。

对于每一个NALU，根据其包含的数据量的不同，其大小也有差异。在IP网络中，当要传输的IP 报文大小超过最大传输单元MTU（Maximum Transmission Unit ）时就会产生IP分片情况。在以太网环境中可传输的最大 IP 报文（MTU）的大小为 1500 字节。如果发送的IP数据包大于MTU，数据包就会被拆开来传送，这样就会产生很多数据包碎片，增加丢包率，降低网络速度。对于视频传输而言，若RTP 包大于MTU 而由底层协议任意拆包，可能会导致接收端播放器的延时播放甚至无法正常播放。因此对于大于MTU 的NALU 单元，必须进行拆包处理。

RFC3984 给出了3 中不同的RTP 打包方案：

（1）Single NALU Packet:在一个RTP 包中只封装一个NALU，在本文中对于小于 1400字节的NALU 便采用这种打包方案。

（2）Aggregation Packet:在一个RTP 包中封装多个NALU，对于较小的NALU 可以采用这种打包方案，从而提高传输效率。

（3）Fragmentation Unit:一个NALU 封装在多个RTP包中，在本文中，对于大于1400字节的NALU 便采用这种方案进行拆包处理。

4. H.264 流媒体传输系统的实现

一个完整的流媒体传输系统包含服务器端和客户端两个部分[5][6]。对于服务器端，其主要任务是读取H.264 视频，从码流中分离出每个NALU 单元，分析NALU 的类型，设置相应的 RTP 包头，封装 RTP 数据包并发送。而对于客户端来说，其主要任务则是接收 RTP数据包，从RTP 包中解析出NALU 单元，然后送至*进行解码播放。该流媒体传输系统的框架如图3 所示。

4. HTTP 错误 404 - 文件或目录未找到的最终解决方法(49093)
5. C++中的头文件和源文件(48951)

评论排行榜
1. 非IE内核浏览器支持activex插件(31)
2. Nginx之location 匹配规则详解(15)
3. Javascript中定义类(15)
4. C++中的头文件和源文件(9)
5. boost::asio设置同步连接超时(6)

推荐排行榜
1. C++中的头文件和源文件(22)
2. Nginx之location 匹配规则详解(15)
3. Javascript中定义类(12)
4. JavaScript中typeof知多少?(9)
5. SVN安装配置与使用(6)

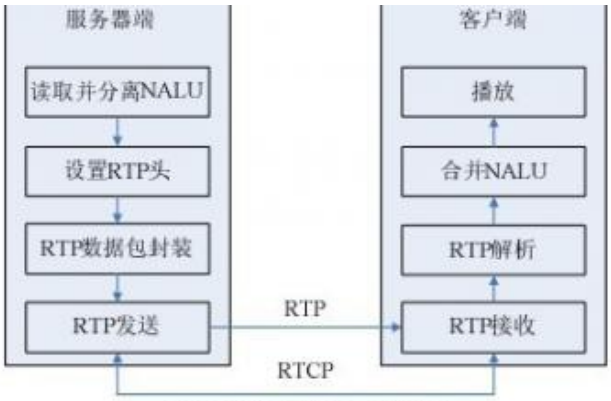

图3 H.264流媒体传输系统框架
Fig. 3 the framework of H.264 video transfer system

其中，服务器端的算法流程如图 4 所示：

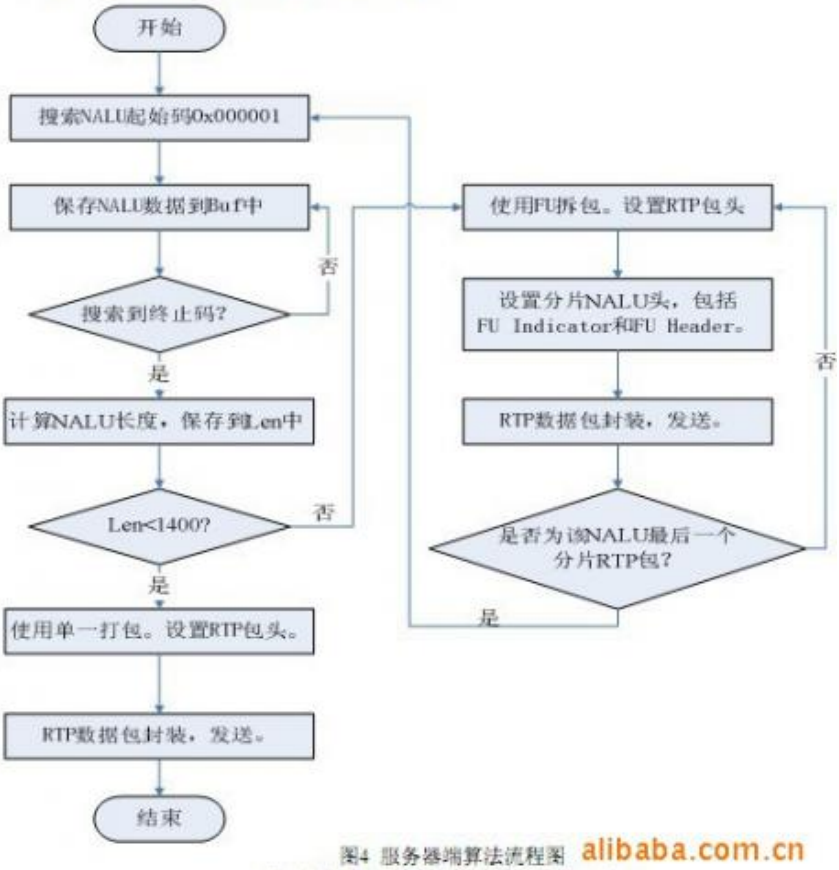

图4 服务器端算法流程图 alibaba.com.cn
Fig. 4 the flow chart of Server-side algorithm

5. 结论

本文所设计的流媒体传输系统服务器端运行在Windows XP 系统，用VLC 播放器作为客户端接收H.264 视频RTP 数据包。经测试，客户端在经过2 秒的缓冲过后即能流畅播放,传输速度设为 30 帧每秒的情况下，未出现丢包拖影等现象，视频主观质量良好，与本地播放该H.264视频无明显区别。

AnyChat采用国际领先的视频编码标准H.264（MPEG-4 part 10 AVC /H.264）编码，H.264/AVC 在压缩效率方面有着特殊的表现，一般情况下达到 MPEG-2 及 MPEG-4 简化类压缩效率的大约 2 倍。H.264具有许多与旧标准不同的新功能，它们一起实现了编码效率的提高。特别是在帧内预测与编码、帧间预测与编码、可变矢量块大小、四分之一像素运动估计、多参考帧预测、自适应环路去块滤波器、整数变换、量化与变换系数扫描、熵编码、加权预测等实现上都有其独特的考虑。

佰锐科技采用先进去马赛克技术，保障在视频通讯过程中不出现花屏、马赛克等现象。免费测试下载地址：
http://www2.bairuitech.com/downloads/bairuisoft/AnyChatCoreSDK_V3.0.rar

H264 视频文件 帧格式 传输封装等 杂碎

rfc3984
Standards Track [Page 2] RFC 3984 RTP Payload Format for H.264 Video February 2005 1.
按照RFC3984协议实现H264视频流媒体

nalu单元 包起始 0x 00 00 00 01

H．264 NAL格式及分析器
http://hi.baidu.com/zsw%5Fdavy/b ... c409cc7cd92ace.html
http://hi.baidu.com/zsw_davy/blo ... 081312c8fc7acc.html

----------比特流信息----------

①NALU(Network Abstract Layer Unit)：两标准中的比特流都是以NAL为单位，每个NAL单元包含一个RBSP，NALU的头信息定义了RBSP所属类型。类型一般包括序列参数集（SPS）、图像参数集（PPS）、增强信息（SEI）、条带（Slice）等，其中，SPS和PPS属于参数集，两标准采用参数集机制是为了将一些主要的序列、图像参数（解码图像尺寸、片组数、参考帧数、量化和滤波参数标记等）与其他参数分离，通过解码器先解码出来。此外，为了增强图像的清晰度，AVS-M添加了图像头（Picture head）信息。读取NALU流程中，每个NALU前有一个起始码0x000001，为防止 内部0x000001序列竞争，H.264编码器在最后一字节前插入一个新的字节——0x03，所以解码器检测到该序列时，需将0x03删掉，而AVS-M只需识别出起始码0x000001。

②读取宏块类型（mb type）和宏块编码模板（cbp）：编解码图像以宏块划分，一个宏块由一个16*16亮度块和相应的一个8*8cb和一个8*8cr色度块组成。

(a) 两标准的帧内、帧间预测时宏块的划分是有区别的。H.264中，I_slice亮度块有Intra_4*4和Intra_16*16两种模式，色度块只有8*8模式；P_slice宏块分为16*16、16*8、8*16、8*8、8*4、4*8、4*4共7种模式。而AVS-M中，I_slice亮度块有I_4*4和I_Direct两模式，P_slice时宏块的划分和H.264中的划分一致。

(b) 两标准的宏块cbp值计算也不相同。H.264中，Intra_16*16宏块的亮度（色度）cbp直接通过读mb type得到；非Intra_16*16宏块的亮度cbp=coded_block_pattern%16，色度cbp=coded_block_pattern/16 。其中，亮度cbp最低4位有效，每位决定对应宏块的残差系数能不能为0；色度cbp为0时，对应残差系数为0，cbp为1时，DC残差系数不为0，AC系数为0，cbp为2时，DC、AC残差系数都不为0。AVS-M中，当宏块类型不是P_skip时，直接从码流中得到cbp的索引值，并以此索引值查表得到codenum值，再以codenum查表分别得到帧内 / 帧间cbp。此cbp为6位，每位代表宏块按8*8划分时能不能包含非零系数，当变换系数不为0时，需进一步读cbp_4*4中每位值来判断一个8*8块中4个4*4块的系数能不能为0。

--
总的来说H264的码流的打包方式有两种,一种为**annex-b byte stream format**的格式，这个是绝大部分编码器的默认输出格式，就是每个帧的开头的3~4个字节是H264的start_code,0x00000001或者0x000001。
另一种是原始的NAL打包格式，就是开始的若干字节（1，2，4字节）是NAL的长度，而不是start_code,此时必须借助某个全局的数据来获得编码器的profile,level,PPS,SPS等信息才可以解码。

AVC vs. H.264
AVC and H.264 are synonymous. The standard is known by the full names "ISO/IEC 14496-10" and "ITU-T Recommendation H.264". In addition, a number of alternate names are used (or have been) in reference to this standard. These include:

- MPEG-4 part 10

- MPEG-4 AVC
- AVC
- MPEG-4 (in the broadcasting world MPEG4 part 2 is ignored)
- H.264
- JVT (Joint Video Team, nowadays rarely used referring to actual spec)
- H.26L (early drafts went by this name)

All of the above (and those I've missed) include the **Annex B byte-stream format**. Unlike earlier MPEG1/2/4 and H.26x codecs, the H.264 specification proper does not define a full bit-stream syntax. It describes a number of NAL (Network Abstraction Layer) units, a sequence of which can be decoded into video frames. These NAL units have no boundary markers, and rely on some unspecified format to provide framing.

Annex B of of the document specifies one such format, which wraps NAL units in a format resembling a traditional MPEG video elementary stream, thus making it suitable for use with containers like MPEG PS/TS unable to provide the required framing. Other formats, such as ISO base media based formats, are able to properly separate the NAL units and do not need the Annex B wrapping.

The H.264 spec suffers from a deficiency. It defines several header-type NAL units (SPS and PPS) without specifying how to pack them into the single codec data field available in most containers. Fortunately, most containers seem to have adopted the packing used by the ISO format known as MP4.
1. H.264起始码
　在网络传输h264数据时，一个UDP包就是一个NALU,解码器可以很方便的检测出NAL分界和解码。但是如果编码数据存储为一个文件，原来的解码器将无法从数据流中分别出每个NAL的起始位置和终止位置，为此h.264用起始码来解决这一问题。

　H.264编码时，在每个NAL前添加起始码 0x000001，解码器在码流中检测到起始码，当前NAL结束。为了防止NAL内部出现0x000001的数据，h.264又提出'防止竞争 emulation prevention"机制，在编码完一个NAL时，如果检测出有连续两个0x00字节，就在后面插入一个0x03。当解码器在NAL内部检测到0x000003的数据，就把0x03抛弃，恢复原始数据。
0x000000　>>>>>>　0x00000300
0x000001　>>>>>>　0x00000301
0x000002　>>>>>>　0x00000302
0x000003　>>>>>>　0x00000303

附上h.264解码nalu中检测起始码的算法流程

```
for(;;)
{
if next 24 bits are 0x000001
{
     startCodeFound = true
     break;
}
else
{
     flush 8 bits
}
}// for(;;)
if(true == startCodeFound)
{
   //startcode found
   // Flush the start code found
   flush 24 bits
   //Now navigate up to next start code and put the in between stuff
   // in the nal structure.
   for(;;)
```

```
  {
    get next 24 bits & check if it equals to 0x000001
    if(false == (next 24 bits == 000001))
    {
      // search for pattern 0x000000
      check if next 24 bits are 0x000000
      if(false == result)
      {
            // copy the byte into the buffer
            copy one byte to the Nal unit
      }
      else
      {
            break;
      }
    }
    else
    {
         break;
    }
  }//for(;;)
}
```

2. MPEG4起始码

MPEG4的特色是VOP，没有NALU的概念，仍使用startcode对每帧进行分界。MPEG4的起始码是0x000001. 另外MPEG4中很多起始码也很有用，比如video_object_sequence_start_code 0x000001B0 表示一个视频对象序列的开始,VO_start_code 0x000001B6 表示一个VOP的开始. 0x000001B6之后的两位，是00表示 I frame， 01 表示 P frame， 10 表示 B frame.

1．引言

H.264的主要目标：

1．高的视频压缩比

2．良好的网络亲和性

解决方案：

VCL　video coding layer　视频编码层

NAL　network abstraction layer　网络提取层

VCL：核心算法引擎，块，宏块及片的语法级别的定义

NAL：片级以上的语法级别（如序列参数集和图像参数集），同时支持以下功能：独立片解码，起始码唯一保证，SEI以及流格式编码数据传送

VCL设计目标：尽可能地独立于网络的情况下进行高效的编解码

NAL设计目标：根据不同的网络把数据打包成相应的格式，将VCL产生的比特字符串适配到各种各样的网络和多元环境中。

NALU头结构：NALU类型(5bit)、重要性指示位(2bit)、禁止位(1bit)。

NALU类型：1～12由H.264使用，24～31由H.264以外的应用使用。

重要性指示：标志该NAL单元用于重建时的重要性，值越大，越重要。

禁止位：网络发现NAL单元有比特错误时可设置该比特为1，以便接收方丢掉该单元。

2．NAL语法语义

NAL层句法：

在编码器输出的码流中，数据的基本单元是句法元素。

句法表征句法元素的组织结构。

语义阐述句法元素的具体含义。

分组都有头部，解码器可以很方便的检测出NAL的分界，依次取出NAL进行解码。

但为了节省码流，H.264没有另外在NAL的头部设立表示起始位置的句法元素。

如果编码数据是存储在介质上的，由于NAL是依次紧密相连的，解码器就无法在数据流中分辨出每个NAL的起始位置和终止位置。

解决方案：在每个NAL前添加起始码：0X000001

在某些类型的介质上，为了寻址的方便，要求数据流在长度上对齐，或某个常数的整数倍。所以在起始码前添加若干字节的0来填充。

检测NAL的开始：

0X000001和0X000000

我们必须考虑当NAL内部出现了0X000001和0X000000

解决方案：

H.264提出了“防止竞争”机制：

0X000000——0X00000300

0X000001——0X00000301

0X000002——0X00000302

0X000003——0X00000303

为此，我们可以知道：

在NAL单元中，下面的三字节序列不应在任何字节对齐的位置出现

0X000000

0X000001

0X000002

Forbidden_zero_bit =0;

Nal_ref_idc：表示NAL的优先级。0～3，取值越大，表示当前NAL越重要，需要优先受到保护。如果当前NAL是属于参考帧的片，或是序列参数集，或是图像参数集这些重要的单位时，本句法元素必需大于0。

Nal_unit_type：当前NAL 单元的类型

3．H.264的NAL层处理

结构示意图：

NAL以NALU（NAL unit）为单元来支持编码数据在基于分组交换技术网络中传输。

它定义了符合传输层或存储介质要求的数据格式，同时给出头信息，从而提供了视频编码和外部世界的接口。

NALU：定义了可用于基于分组和基于比特流系统的基本格式

RTP封装：只针对基于NAL单元的本地NAL接口。

三种不同的数据形式：

SODB 数据比特串 - - > 最原始的编码数据

RBSP 原始字节序列载荷 - - > 在SODB的后面填加了结尾比特（RBSP trailing bits 一个bit“1”）若干比特“0”,以便字节对齐

EBSP 扩展字节序列载荷-->在RBSP基础上填加了仿校验字节（0X03）它的原因是： 在NALU加到Annexb上时，需要添加每组NALU之前的开始码StartCodePrefix,如果该NALU对应的slice为一帧的开始则用4位字节表示，ox00000001,否则用3位字节表示ox000001.为了使NALU主体中不包括与开始码相冲突的，在编码时，每遇到两个字节连续为0，就插入一个字节的0x03。解码时将0x03去掉。也称为脱壳操作

处理过程：

1． 将VCL层输出的SODB封装成nal_unit， Nal_unit是一个通用封装格式，可以适用于有序字节流方式和IP包交换方式。

2． 针对不同的传送网络（电路交换|包交换），将nal_unit 封装成针对不同网络的封装格式。

第一步的具体过程：

VCL层输出的比特流SODB（String Of Data Bits），到nal_unit之间，经过了以下三步处理：

1.SODB字节对齐处理后封装成RBSP（Raw Byte Sequence Payload）。

2.为防止RBSP的字节流与有序字节流传送方式下的
SCP（start_code_prefix_one_3bytes，0x000001）出现字节竞争情形，循环检测RBSP前三个字节，在出现字节竞争时在第三字节前加入emulation_prevention_three_byte（0x03），具体方法：

nal_unit(NumBytesInNALunit) {

forbidden_zero_bit

nal_ref_idc

nal_unit_type

NumBytesInRBSP = 0

for(i = 1; i < NumBytesInNALunit; i++) {

if(i + 2 < NumBytesInNALunit && next_bits(24) = = 0x000003) {

rbsp_byte[NumBytesInRBSP++]

rbsp_byte[NumBytesInRBSP++]

i += 2

emulation_prevention_three_byte /* equal to 0x03 */

} else

rbsp_byte[NumBytesInRBSP++]

}

}

3. 防字节竞争处理后的RBSP再加一个字节的header(forbidden_zero_bit+ nal_ref_idc+ nal_unit_type)，封装成nal_unit.

第二步的具体过程：

case1：有序字节流的封装

byte_stream_nal_unit(NumBytesInNALunit) {

while(next_bits(24) != 0x000001)

zero_byte /* equal to 0x00 */

if(more_data_in_byte_stream( )) {

start_code_prefix_one_3bytes /* equal to 0x000001 */ nal_unit(NumBytesInNALunit)

}

}

类似H.320和MPEG-2/H.222.0等传输系统，传输NAL作为有序连续字节或比特流，同时要依靠数据本身识别NAL单元边界。在这样的应用系统中，H.264/AVC规范定义了字节流格式，每个NAL单元前面增加3个字节的前缀，即同步字节。在比特流应用中，每个图像需要增加一个附加字节作为边界定位。还有一种可选特性，在字节流中增加附加数据，用做扩充发送数据量，能实现快速边界定位，恢复同步

Case2：IP网络的RTP打包封装

分组打包的规则

(1)额外开销要少，使MTU尺寸在100～64k字节范围都可以；

(2)不用对分组内的数据解码就可以判别该分组的重要性；

(3)载荷规范应当保证不用解码就可识别由于其他的比特丢失而造成的分组不可解码；

(4)支持将NALU分割成多个RTP分组；

(5)支持将多个NALU汇集在一个RTP分组中。

RTP的头标可以是NALU的头标，并可以实现以上的打包规则。

一个RTP分组里放入一个NALU，将NALU(包括同时作为载荷头标的NALU头)放入RTP的载荷中，设置RTP头标值。为了避免IP层对大分组的再一次分割，片分组的大小一般都要小于MTU尺寸。由于包传送的路径不同，解码端要重新对片分组排序，RTP包含的次序信息可以用来解决这一问题。

NALU分割

对于预先已经编码的内容，NALU可能大于MTU尺寸的限制。虽然IP层的分割可以使数据块小于64千字节，但无法在应用层实现保护，从而降低了非等重保护方案的效果。由于UDP数据包小于64千字节，而且一个片的长度对某些应用场合来说太小，所以应用层打包是RTP打包方案的一部分。

新的讨论方案(IETF)应当符合以下特征：

(1)NALU的分块以按RTP次序号升序传输；

(2)能够标记第一个和最后一个NALU分块；

(3)可以检测丢失的分块。

NALU合并

一些NALU如SEI、参数集等非常小，将它们合并在一起有利于减少头标开销。已有两种集合分组：

(1)单一时间集合分组(STAP)，按时间戳进行组合；

(2)多时间集合分组(MTAP)，不同时间戳也可以组合。

NAL规范视频数据的格式，主要是提供头部信息，以适合各种媒体的传输和存储。NAL支持各种网络，包括：

1．任何使用RTP/IP协议的实时有线和无线Internet 服务

2．作为MP4文件存储和多媒体信息文件服务

3．MPEG-2系统

4．其它网

NAL规定一种通用的格式，既适合面向包传输，也适合流传送。实际上，包传输和流传输的方式是相同的，不同之处是传输前面增加了一个起始码前缀

在类似Internet/RTP面向包传送协议系统中，包结构中包含包边界识别字节，在这种情况下，不需要同步字节。

NAL单元分为VCL和非VCL两种

VCL NAL单元包含视频图像采样信息，

非VCL包含各种有关的附加信息，例如参数集（头部信息，应用到大量的VCL NAL单元）、提高性能的附加信息、定时信息等

参数集：

参数集是很少变化的信息，用于大量VCL NAL单元的解码，分为两种类型：

1．序列参数集，作用于一串连续的视频图像，即视频序列。

两个IDR图像之间为序列参数集。IDR和I帧的区别见下面。

2． 图像参数集，作用于视频序列中的一个或多个个别的图像

序列和图像参数集机制，减少了重复参数的传送，每个VCL NAL单元包含一个标识，指

向有关的图像参数集，每个图像参数集包含一个标识，指向有关的序列参数集的内容

因此，只用少数的指针信息，引用大量的参数，大大减少每个VCL NAL单元重复传送的信息。

序列和图像参数集可以在发送VCL NAL单元以前发送，并且重复传送，大大提高纠错能力。序列和图像参数集可以在"带内"，也可以用更为可靠的其他"带外"通道传送。

FROM:http://blog.csdn.net/cosmoslife/article/details/7619077

分类: 编解码,音视频、流媒体

DoubleLi
关注 - 29
粉丝 - 1183
+加关注

0 0

« 上一篇: 264分析两大利器：264VISA和Elecard StreamEye Tools
» 下一篇: malloc、calloc、realloc的区别

posted on 2015-06-26 15:43 DoubleLi 阅读(7838) 评论(0) 编辑 收藏

刷新评论 刷新页面 返回顶部

最新知识库文章:

· 作为一个程序员，数学对你到底有多重要
· 领域驱动设计在互联网业务开发中的实践
· 步入云计算
· 以操作系统的角度述说线程与进程
· 软件测试转型之路
» 更多知识库文章...

历史上的今天:

2014-06-26 Linux 静态库&动态库调用
2012-06-26 WM_CREATE和WM_INITDIALOG
2012-06-26 MFC应用程序中处理消息的顺序
2012-06-26 MFC浅析(7) CWnd类虚函数的调用时机、缺省实现 .