JT同学
码龄4年 暂无认证
105 原创 2万+ 周排名 91万+ 总排名 46万+ 访问 等级
5440 积分 1861 粉丝 1016 获赞 346 评论 3945 收藏
私信 关注

搜博主文章

热门文章
从零开始写一个RTSP服务器（一）RTSP协议讲解 57178
C++ deque的用法与示例 38014
Linux I2C驱动框架（超详细） 35491
我的开源项目-RtspServer 25563
从零开始写一个RTSP服务器（二）RTSP协议的实现 21864

最新评论
我的开源项目-RtspServer
kaifa3k: 我测试 ./h264_rtsp_server test.h264 也会这样的问题？请问解决了吗？m...
使用mp4v2封装H.264成mp4最简单示例
热心兔斯基: 应该是播放器的原因。换个播放器试试
使用mp4v2封装H.264成mp4最简单示例
ai604233436: h264的标准中，好像默认的设置的90KHZ。我所以我理解timeScale...
使用mp4v2封装H.264成mp4最简单示例
ai604233436: 有遇到，不知道原因呢。
深入学习Linux摄像头（二）v4l2驱动框架
qq_43745897: 请问往缓存区里面采集图像数据的帧率是多少呢？请问如果缓存区...

您愿意向朋友推荐“博客详情页”吗？
强烈不推荐 不推荐 一般般 推荐 强烈推荐

最新文章
深入浅出MySQL事务（二）MVCC的实现原理
深入浅出MySQL事务（一）事务隔离
深入浅出MySQL索引（二）InnoDB存储引擎的索引
2020年 4篇 2019年 95篇
2018年 7篇

从零开始写一个RTSP服务器（七）多播传输RTP包

原创 JT同学 于 2019-08-12 19:42:28 发布 5574 收藏 18 版权
分类专栏： 从零开始写一个RTSP服务器 文章标签： rtsp rtp 多播
从零开始写一个RTS... 专栏收录该内容 187 订阅 10 篇文章 订阅专栏

从零开始写一个RTSP服务器系列

★我的开源项目-RtspServer

从零开始写一个RTSP服务器（一）RTSP协议讲解

从零开始写一个RTSP服务器（二）RTSP协议的实现

从零开始写一个RTSP服务器（三）RTP传输H.264

从零开始写一个RTSP服务器（四）一个传输H.264的RTSP服务器

从零开始写一个RTSP服务器（五）RTP传输AAC

从零开始写一个RTSP服务器（六）一个传输AAC的RTSP服务器

从零开始写一个RTSP服务器（七）多播传输RTP包

从零开始写一个RTSP服务器（八）一个多播的RTSP服务器

从零开始写一个RTSP服务器（九）一个RTP OVER RTSP/TCP的RTSP服务器

从零开始写一个RTSP服务器（七）多播传输RTP包

文章目录

从零开始写一个RTSP服务器（七）多播传输RTP包
一、多播
1.1 多播简介
1.2 多播示例
二、多播的sdp描述文件
三、多播传输RTP包实现
3.1 实现
3.2 源码
multicast.c
rtp.h
rtp.c
四、测试

一、多播

1.1 多播简介

单播地址标识当个IP接口，广播地址标识某个子网的所有IP接口，多播地址标识一组IP接口

单播和多播是两个极端，多播则在这两者之间折衷

多播数据报只由加入多播组的应用的主机的接口接收

IPV4的D类地址是IPV4的多播地址，范围是（224.0.0.0-239.255.255.255）

下面这张图对多播地址进行细分

范　围	IPv6范围	IPv4	
		TTL范围	可管理范围
接口局部	1	0	
链路局部	2	1	224.0.0.0到224.0.0.255
网点局部	5	<32	239.255.0.0到239.255.255.255
组织机构局部	8		239.192.0.0到239.195.255.255
全球	14	≤255	224.0.1.0到238.255.255.255

图21-3　IPv4和IPv6多播地址范围

1.2 多播示例

下面讲解一下多播的客户端与服务端编程

服务端

多播服务端并不需要什么特殊的操作，只需要创建udp套接字，然后向多播地址指定端口发送数据就行

客户端

多播客户端需要做的工作是，创建udp套接字，绑定多播端口，加入多播组

下面是示例代码

服务端

```c
/*
*broadcast_server.c - 多播服务程序
*/
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <time.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

#define MCAST_PORT 8888
#define MCAST_ADDR "239.255.255.11"
#define MCAST_DATA "BROADCAST TEST DATA"            /*多播发送的数据*/
#define MCAST_INTERVAL 1                            /*发送间隔时间*/

int main(int argc, char*argv)
{
    struct sockaddr_in mcast_addr;
    int fd = socket(AF_INET, SOCK_DGRAM, 0);        /*建立套接字*/
    if (fd == -1)
    {
        perror("socket()");
        exit(1);
    }

    memset(&mcast_addr, 0, sizeof(mcast_addr));/*初始化IP多播地址为0*/
    mcast_addr.sin_family = AF_INET;                /*设置协议族类行为AF*/
    mcast_addr.sin_addr.s_addr = inet_addr(MCAST_ADDR);/*设置多播IP地址*/
    mcast_addr.sin_port = htons(MCAST_PORT);        /*设置多播端口*/

    /*向多播地址发送数据*/
    while(1)
    {
        int n = sendto(fd,MCAST_DATA,sizeof(MCAST_DATA),0,(struct sockaddr*)&mcast_addr,sizeof(mcast_addr));
        if( n < 0)
        {
            perror("sendto()");
            exit(1);
        }
        sleep(MCAST_INTERVAL);                      /*等待一段时间*/
    }

    return 0;
}
```

客户端

```c
/*
*broadcast_client.c - 多播的客户端
*/
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <time.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

#define MCAST_PORT 8888
#define MCAST_ADDR "239.255.255.11"
```

目录
从零开始写一个RTSP服务器（七）多播...
文章目录
一、多播
1.1 多播简介
1.2 多播示例
二、多播的sdp描述文件
三、多播传输RTP包实现
3.1 实现
3.2 源码
四、测试

分类专栏
C/C++ 1篇
STL源码剖析 18篇
MySQL 4篇
Linux内核 14篇
分布式
nginx
从零开始写一个RTSP服... 10篇
live555源码分析与应用 9篇
Linux驱动 19篇

```c
#define MCAST_INTERVAL 1                /*发送间隔时间*/
#define BUFF_SIZE 256                   /*接收缓冲区大小*/

int main(int argc, char*argv[])
{
    struct sockaddr_in local_addr;              /*本地地址*/

    int fd = socket(AF_INET, SOCK_DGRAM, 0);     /*建立套接字*/
    if (fd == -1)
    {
        perror("socket()");
        exit(1);
    }

    int yes = 1;
    if (setsockopt(fd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(yes)) < 0)
    {
        perror("Reusing ADDR failed");
        exit(1);
    }

    /*初始化本地地址*/
    memset(&local_addr, 0, sizeof(local_addr));
    local_addr.sin_family = AF_INET;
    local_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    local_addr.sin_port = htons(MCAST_PORT);

    /*绑定socket*/
    int err = bind(fd,(struct sockaddr*)&local_addr, sizeof(local_addr)) ;
    if(err < 0)
    {
        perror("bind()");
        exit(1);
    }

    /*设置回环许可*/
    int loop = 1;
    err = setsockopt(fd,IPPROTO_IP, IP_MULTICAST_LOOP,&loop, sizeof(loop));
    if(err < 0)
    {
        perror("setsockopt():IP_MULTICAST_LOOP");
        exit(1);
    }

    /*加入多播组*/
    struct ip_mreq mreq;
    mreq.imr_multiaddr.s_addr = inet_addr(MCAST_ADDR); /*多播地址*/
    mreq.imr_interface.s_addr = htonl(INADDR_ANY); /*本地网络接口为默认*/

    /*将本机加入多播组*/
    err = setsockopt(fd, IPPROTO_IP, IP_ADD_MEMBERSHIP,&mreq, sizeof(mreq));
    if (err < 0)
    {
        perror("setsockopt():IP_ADD_MEMBERSHIP");
        exit(1);
    }

    int times = 0;
    int addr_len = sizeof(local_addr);
    char buff[BUFF_SIZE];
    int n = 0;

    /*循环接收多播组的消息，5次后退出*/
    for(times = 0;times < 5;times++)
    {
        memset(buff, 0, BUFF_SIZE);                 /*清空接收缓冲区*/

        /*接收数据*/
        n = recvfrom(fd, buff, BUFF_SIZE, 0,(struct sockaddr*)&local_addr,&addr_len);
        if( n== -1)
        {
            perror("recvfrom()");
        }
                                                    /*打印信息*/
        printf("Recv %dst message from server:%s\n", times, buff);
        sleep(MCAST_INTERVAL);
    }

    /*退出多播组*/
    err = setsockopt(fd, IPPROTO_IP, IP_DROP_MEMBERSHIP,&mreq, sizeof(mreq));

    close(fd);
    return 0;
}
```

需要好好看一看这个示例，了解多播编程的流程

二、多播的sdp描述文件

```
m=video 9832 RTP/AVP 96
c=IN IP4 239.255.255.11/255
a=rtpmap:96 H264/90000
a=framerate:25
```

这是一个多播H.264的sdp文件

相关的讲解前面已经讲了很多了（详情看：从零开始写一个RTSP服务器（一）不一样的RTSP协议讲解）

这里需要注意的是目的端口号为`9832`，多播IP为`239.255.255.11`

三、多播传输RTP包实现

3.1 实现

对于我们来说，我们实现的是服务端，所以非常简单，无需过多操作，只需要把从零开始写一个RTSP服务器（三）RTP传输H.264中的目的IP和目的端口改为目的多播IP和目的多播端口就行

关于RTP打包这里不再重复，前面已经讲解得非常详细了

3.2 源码

multicast.c

```c
/*
 * 作者: _JT_
 * 博客: https://blog.csdn.net/weixin_42462202
 */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>

#include "rtp.h"

#define H264_FILE_NAME      "test.h264"
#define MULTICAST_IP        "239.255.255.11"
#define MULTICAST_PORT      9832
#define FPS                 25

static inline int startCode3(char* buf)
{
    if(buf[0] == 0 && buf[1] == 0 && buf[2] == 1)
        return 1;
    else
        return 0;
}

static inline int startCode4(char* buf)
{
    if(buf[0] == 0 && buf[1] == 0 && buf[2] == 0 && buf[3] == 1)
        return 1;
    else
        return 0;
}

static char* findNextStartCode(char* buf, int len)
{
    int i;

    if(len < 3)
        return NULL;

    for(i = 0; i < len-3; ++i)
    {
        if(startCode3(buf) || startCode4(buf))
            return buf;

```

```
        ++buf;
    }

    if(startCode3(buf))
        return buf;

    return NULL;
}

static int getFrameFromH264File(int fd, char* frame, int size)
{
    int rSize, frameSize;
    char* nextStartCode;

    if(fd < 0)
        return fd;

    rSize = read(fd, frame, size);
    if(!startCode3(frame) && !startCode4(frame))
        return -1;

    nextStartCode = findNextStartCode(frame+3, rSize-3);
    if(!nextStartCode)
    {
        lseek(fd, 0, SEEK_SET);
        frameSize = rSize;
    }
    else
    {
        frameSize = (nextStartCode-frame);
        lseek(fd, frameSize-rSize, SEEK_CUR);
    }

    return frameSize;
}

static int createUdpSocket()
{
    int fd;
    int on = 1;

    fd = socket(AF_INET, SOCK_DGRAM, 0);
    if(fd < 0)
        return -1;

    setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, (const char*)&on, sizeof(on));

    return fd;
}

static int rtpSendH264Frame(int socket, char* ip, int16_t port,
                            struct RtpPacket* rtpPacket, uint8_t* frame, uint32_t frameSize)
{
    uint8_t naluType; // nalu第一个字节
    int sendBytes = 0;
    int ret;

    naluType = frame[0];

    if (frameSize <= RTP_MAX_PKT_SIZE) // nalu长度小于最大包场：单一NALU单元模式
    {
        /*
         *   0 1 2 3 4 5 6 7 8 9
         *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
         *  |F|NRI|  Type   | a single NAL unit ... |
         *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
         */
        memcpy(rtpPacket->payload, frame, frameSize);
        ret = rtpSendPacket(socket, ip, port, rtpPacket, frameSize);
        if(ret < 0)
            return -1;

        rtpPacket->rtpHeader.seq++;
        sendBytes += ret;
        if ((naluType & 0x1F) == 7 || (naluType & 0x1F) == 8) // 如果是SPS、PPS就不需要加时间戳
            goto out;
    }
    else // nalu长度小于最大包场：分片模式
    {
        /*
         *  0                   1                   2
         *  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
         * +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
         * | FU indicator  |   FU header   |   FU payload   ... |
         * +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
         */

        /*
         *     FU Indicator
         *    0 1 2 3 4 5 6 7
         *   +-+-+-+-+-+-+-+-+
         *   |F|NRI|  Type   |
         *   +---------------+
         */

        /*
         *      FU Header
         *    0 1 2 3 4 5 6 7
         *   +-+-+-+-+-+-+-+-+
         *   |S|E|R|  Type   |
         *   +---------------+
         */

        int pktNum = frameSize / RTP_MAX_PKT_SIZE;       // 有几个完整的包
        int remainPktSize = frameSize % RTP_MAX_PKT_SIZE; // 剩余不完整包的大小
        int i, pos = 1;

        /* 发送完整的包 */
        for (i = 0; i < pktNum; i++)
        {
            rtpPacket->payload[0] = (naluType & 0x60) | 28;
            rtpPacket->payload[1] = naluType & 0x1F;

            if (i == 0) //第一包数据
                rtpPacket->payload[1] |= 0x80; // start
            else if (remainPktSize == 0 && i == pktNum - 1) //最后一包数据
                rtpPacket->payload[1] |= 0x40; // end

            memcpy(rtpPacket->payload+2, frame+pos, RTP_MAX_PKT_SIZE);
            ret = rtpSendPacket(socket, ip, port, rtpPacket, RTP_MAX_PKT_SIZE+2);
            if(ret < 0)
                return -1;

            rtpPacket->rtpHeader.seq++;
            sendBytes += ret;
            pos += RTP_MAX_PKT_SIZE;
        }

        /* 发送剩余的数据 */
        if (remainPktSize > 0)
        {
            rtpPacket->payload[0] = (naluType & 0x60) | 28;
            rtpPacket->payload[1] = naluType & 0x1F;
            rtpPacket->payload[1] |= 0x40; //end

            memcpy(rtpPacket->payload+2, frame+pos, remainPktSize+2);
            ret = rtpSendPacket(socket, ip, port, rtpPacket, remainPktSize+2);
            if(ret < 0)
                return -1;

            rtpPacket->rtpHeader.seq++;
            sendBytes += ret;
        }
    }

out:

    return sendBytes;
}

int main(int argc, char* argv[])
{
    int socket;
    int fd;
    int fps = 25;
    int startCode;
    struct RtpPacket* rtpPacket;
    uint8_t* frame;
    uint32_t frameSize;

    fd = open(H264_FILE_NAME, O_RDONLY);
    if(fd < 0)
    {
        printf("failed to open %s\n", H264_FILE_NAME);
```

```
        return -1;
    }

    socket = createUdpSocket();
    if(socket < 0)
    {
        printf("failed to create socket\n");
        return -1;
    }

    rtpPacket = (struct RtpPacket*)malloc(500000);
    frame = (uint8_t*)malloc(500000);

    rtpHeaderInit(rtpPacket, 0, 0, 0, RTP_VESION, RTP_PAYLOAD_TYPE_H264, 0,
                  0, 0, 0x88923423);

    while(1)
    {
        frameSize = getFrameFromH264File(fd, frame, 500000);
        if(frameSize < 0)
        {
            printf("read err\n");
            continue;
        }

        if(startCode3(frame))
            startCode = 3;
        else
            startCode = 4;

        frameSize -= startCode;
        rtpSendH264Frame(socket, MULTICAST_IP, MULTICAST_PORT,
                         rtpPacket, frame+startCode, frameSize);
        rtpPacket->rtpHeader.timestamp += 90000/FPS;

        usleep(1000*1000/fps);
    }

    free(rtpPacket);
    free(frame);

    return 0;
}
```

rtp.h

```
/*
 * 作者: _JT_
 * 博客: https://blog.csdn.net/weixin_42462202
 */

#ifndef _RTP_H_
#define _RTP_H_
#include <stdint.h>

#define RTP_VESION              2

#define RTP_PAYLOAD_TYPE_H264   96
#define RTP_PAYLOAD_TYPE_AAC    97

#define RTP_HEADER_SIZE         12
#define RTP_MAX_PKT_SIZE        1400

/*
 *
 *    0                   1                   2                   3
 *    7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0
 *   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 *   |V=2|P|X|  CC   |M|     PT      |       sequence number         |
 *   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 *   |                           timestamp                           |
 *   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 *   |           synchronization source (SSRC) identifier            |
 *   +=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
 *   |            contributing source (CSRC) identifiers             |
 *   :                             ....                              :
 *   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 *
 */
struct RtpHeader
{
    /* byte 0 */
    uint8_t csrcLen:4;
    uint8_t extension:1;
    uint8_t padding:1;
    uint8_t version:2;

    /* byte 1 */
    uint8_t payloadType:7;
    uint8_t marker:1;

    /* bytes 2,3 */
    uint16_t seq;

    /* bytes 4-7 */
    uint32_t timestamp;

    /* bytes 8-11 */
    uint32_t ssrc;
};

struct RtpPacket
{
    struct RtpHeader rtpHeader;
    uint8_t payload[0];
};

void rtpHeaderInit(struct RtpPacket* rtpPacket, uint8_t csrcLen, uint8_t extension,
                   uint8_t padding, uint8_t version, uint8_t payloadType, uint8_t marker,
                   uint16_t seq, uint32_t timestamp, uint32_t ssrc);
int rtpSendPacket(int socket, const char* ip, int16_t port, struct RtpPacket* rtpPacket, uint32_t dataSize);

#endif //_RTP_H_
```

rtp.c

```
/*
 * 作者: _JT_
 * 博客: https://blog.csdn.net/weixin_42462202
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#include "rtp.h"

void rtpHeaderInit(struct RtpPacket* rtpPacket, uint8_t csrcLen, uint8_t extension,
                   uint8_t padding, uint8_t version, uint8_t payloadType, uint8_t marker,
                   uint16_t seq, uint32_t timestamp, uint32_t ssrc)
{
    rtpPacket->rtpHeader.csrcLen = csrcLen;
    rtpPacket->rtpHeader.extension = extension;
    rtpPacket->rtpHeader.padding = padding;
    rtpPacket->rtpHeader.version = version;
    rtpPacket->rtpHeader.payloadType =  payloadType;
    rtpPacket->rtpHeader.marker = marker;
    rtpPacket->rtpHeader.seq = seq;
    rtpPacket->rtpHeader.timestamp = timestamp;
    rtpPacket->rtpHeader.ssrc = ssrc;
}

int rtpSendPacket(int socket, const char* ip, int16_t port, struct RtpPacket* rtpPacket, uint32_t dataSize)
{
    struct sockaddr_in addr;
    int ret;

    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    addr.sin_addr.s_addr = inet_addr(ip);

    rtpPacket->rtpHeader.seq = htons(rtpPacket->rtpHeader.seq);
    rtpPacket->rtpHeader.timestamp = htonl(rtpPacket->rtpHeader.timestamp);
    rtpPacket->rtpHeader.ssrc = htonl(rtpPacket->rtpHeader.ssrc);

    ret = sendto(socket, (void*)rtpPacket, dataSize+RTP_HEADER_SIZE, 0,
                 (struct sockaddr*)&addr, sizeof(addr));

    rtpPacket->rtpHeader.seq = ntohs(rtpPacket->rtpHeader.seq);
    rtpPacket->rtpHeader.timestamp = ntohl(rtpPacket->rtpHeader.timestamp);
    rtpPacket->rtpHeader.ssrc = ntohl(rtpPacket->rtpHeader.ssrc);
```

```
48
49      return ret;
50  }
```

四、测试

将 `multicast.c`、`rtp.h`、`rtp.c` 保存下来

编译运行，程序默认会打开 `test.h264` 的视频文件，如果你没有视频源的话，可以从RtspServer的example目录下获取

```
1  # gcc multicast.c rtp.c
2  # ./a.out
```

将sdp保存为multicast.sdp

```
1  a=type:broadcast
2  a=rtcp-unicast: reflection
3  m=video 9832 RTP/AVP 96
4  c=IN IP4 239.255.255.11/255
5  a=rtpmap:96 H264/90000
6  a=framerate:25
```

使用vlc打开sdp文件

```
1  # vlc multicast.sdp
```

运行效果