转 live555学习笔记－RTP打包与发送

2013年09月20日 17:16:58　阅读数：8505

RTP打包与发送

rtp传送开始于函数：MediaSink::startPlaying()。想想也有道理，应是sink跟source要数据，所以从sink上调用startplaying（嘿嘿，相当于directshow的拉模式）。

看一下这个函数：

```cpp
Boolean MediaSink::startPlaying(MediaSource& source,
afterPlayingFunc* afterFunc, void * afterClientData)
{
//参数afterFunc是在播放结束时才被调用。
// Make sure we're not already being played:
if (fSource != NULL) {
envir().setResultMsg( "This sink is already being played" );
return False;
}


// Make sure our source is compatible:
if (!sourceIsCompatibleWithUs(source)) {
envir().setResultMsg(
"MediaSink::startPlaying(): source is not compatible!" );
return False;
}
//记下一些要使用的对象
fSource = (FramedSource*) &source;


fAfterFunc = afterFunc;
fAfterClientData = afterClientData;
return continuePlaying();
}
```

为了进一步封装（让继承类少写一些代码），搞出了一个虚函数continuePlaying()。让我们来看一下：

```cpp
Boolean MultiFramedRTPSink::continuePlaying() {
// Send the first packet.
// (This will also schedule any future sends.)
buildAndSendPacket(True);
return True;
}
```

MultiFramedRTPSink是与帧有关的类，其实它要求每次必须从source获得一个帧的数据，所以才叫这个name。可以看到continuePlaying()完全被buildAndSendPacket()代替。看一下buildAndSendPacket():

```cpp
void MultiFramedRTPSink::buildAndSendPacket(Boolean isFirstPacket)
{
//此函数中主要是准备rtp包的头，为一些需要跟据实际数据改变的字段留出位置。
fIsFirstPacket = isFirstPacket;


// Set up the RTP header:
unsigned rtpHdr = 0x80000000; // RTP version 2; marker ('M') bit not set (by default; it can be set later)
rtpHdr |= (fRTPPayloadType << 16);
rtpHdr |= fSeqNo; // sequence number
fOutBuf->enqueueWord(rtpHdr); //向包中加入一个字


// Note where the RTP timestamp will go.
// (We can't fill this in until we start packing payload frames.)
fTimestampPosition = fOutBuf->curPacketSize();
fOutBuf->skipBytes(4); // leave a hole for the timestamp  在缓冲中空出时间戳的位置


fOutBuf->enqueueWord(SSRC());


// Allow for a special, payload-format-specific header following the
// RTP header:
fSpecialHeaderPosition = fOutBuf->curPacketSize();
fSpecialHeaderSize = specialHeaderSize();
fOutBuf->skipBytes(fSpecialHeaderSize);


// Begin packing as many (complete) frames into the packet as we can:
fTotalFrameSpecificHeaderSizes = 0;
fNoFramesLeft = False;
fNumFramesUsedSoFar = 0; // 一个包中已打入的帧数。
//头准备好了，再打包帧数据
packFrame();
}
```

继续看packFrame()：

```cpp
void MultiFramedRTPSink::packFrame()
{
// First, see if we have an overflow frame that was too big for the last pkt
if (fOutBuf->haveOverflowData()) {
//如果有帧数据，则使用之。OverflowData是指上次打包时剩下的帧数据，因为一个包可能容纳不了一个帧。
// Use this frame before reading a new one from the source
unsigned frameSize = fOutBuf->overflowDataSize();
struct timeval presentationTime = fOutBuf->overflowPresentationTime();
unsigned durationInMicroseconds =fOutBuf->overflowDurationInMicroseconds();
fOutBuf->useOverflowData();


afterGettingFrame1(frameSize, 0, presentationTime,durationInMicroseconds);
} else {
//一点帧数据都没有，跟source要吧。
// Normal case: we need to read a new frame from the source
if (fSource == NULL)
return ;


//更新缓冲中的一些位置
fCurFrameSpecificHeaderPosition = fOutBuf->curPacketSize();
fCurFrameSpecificHeaderSize = frameSpecificHeaderSize();
fOutBuf->skipBytes(fCurFrameSpecificHeaderSize);
fTotalFrameSpecificHeaderSizes += fCurFrameSpecificHeaderSize;


//从source获取下一帧
fSource->getNextFrame(fOutBuf->curPtr(), //新数据存放开始的位置
fOutBuf->totalBytesAvailable(), //缓冲中空余的空间大小
afterGettingFrame, //因为可能source中的读数据函数会被放在任务调度中，所以把获取帧后应调用的函数传给source
this ,
ourHandleClosure, //这个是source结束时(比如文件读完了)要调用的函数。
this );
}
}
```

可以想像下面就是source从文件（或某个设备）中读取一帧数据，读完后返回给sink，当然不是从函数返回了，而是以调用afterGettingFrame这个回调函数的方式。所以下面看一下afterGettingFrame():

```cpp
void MultiFramedRTPSink::afterGettingFrame( void * clientData,
unsigned numBytesRead, unsigned numTruncatedBytes,
struct timeval presentationTime, unsigned durationInMicroseconds)
{
MultiFramedRTPSink* sink = (MultiFramedRTPSink*) clientData;
sink->afterGettingFrame1(numBytesRead, numTruncatedBytes, presentationTime,
durationInMicroseconds);
}
```

没什么可看的，只是过度为调用成员函数，所以afterGettingFrame1()才是重点：

```cpp
void MultiFramedRTPSink::afterGettingFrame1(
unsigned frameSize,
unsigned numTruncatedBytes,
struct timeval presentationTime,
unsigned durationInMicroseconds)
{
if (fIsFirstPacket) {
// Record the fact that we're starting to play now:
gettimeofday(&fNextSendTime, NULL);
}


//如果给予一帧的缓冲不够大，就会发生截断一帧数据的现象。但也只能提示一下用户
if (numTruncatedBytes > 0) {


unsigned const bufferSize = fOutBuf->totalBytesAvailable();
envir()
<< "MultiFramedRTPSink::afterGettingFrame1(): The input frame data was too large for
 our buffer size ("
<< bufferSize
<< ").  "
<< numTruncatedBytes
<< " bytes of trailing data was dropped!  Correct this by increasing \"OutPacketBuff
er::maxSize\" to at least "
<< OutPacketBuffer::maxSize + numTruncatedBytes
<< ", *before* creating this 'RTPSink'.  (Current value is "
<< OutPacketBuffer::maxSize << ".)\n" ;
}
unsigned curFragmentationOffset = fCurFragmentationOffset;
unsigned numFrameBytesToUse = frameSize;
unsigned overflowBytes = 0;


//如果包只已经打入帧数据了，并且不能再向这个包中加数据了，则把新获得的帧数据保存下来。
// If we have already packed one or more frames into this packet,
// check whether this new frame is eligible to be packed after them.
// (This is independent of whether the packet has enough room for this
// new frame; that check comes later.)
if (fNumFramesUsedSoFar > 0) {
//如果包中已有了一个帧，并且不允许再打入新的帧了，则只记录下新的帧。
if ((fPreviousFrameEndedFragmentation && !allowOtherFramesAfterLastFragment())
|| !frameCanAppearAfterPacketStart(fOutBuf->curPtr(), frameSize))
{
// Save away this frame for next time:
numFrameBytesToUse = 0;
fOutBuf->setOverflowData(fOutBuf->curPacketSize(), frameSize,
presentationTime, durationInMicroseconds);
}
}

//表示当前打入的是否是上一个帧的最后一块数据。
fPreviousFrameEndedFragmentation = False;


//下面是计算获取的帧中有多少数据可以打到当前包中，剩下的数据就作为overflow数据保存下来。
if (numFrameBytesToUse > 0) {
// Check whether this frame overflows the packet
if (fOutBuf->wouldOverflow(frameSize)) {
// Don't use this frame now; instead, save it as overflow data, and
// send it in the next packet instead.  However, if the frame is too
// big to fit in a packet by itself, then we need to fragment it (and
// use some of it in this packet, if the payload format permits this.)
if (isTooBigForAPacket(frameSize)
&& (fNumFramesUsedSoFar == 0 || allowFragmentationAfterStart())) {
// We need to fragment this frame, and use some of it now:
overflowBytes = computeOverflowForNewFrame(frameSize);
numFrameBytesToUse -= overflowBytes;
fCurFragmentationOffset += numFrameBytesToUse;
} else {
// We don't use any of this frame now:
overflowBytes = frameSize;
numFrameBytesToUse = 0;
}
fOutBuf->setOverflowData(fOutBuf->curPacketSize() + numFrameBytesToUse,
overflowBytes, presentationTime, durationInMicroseconds);
```

```
} else if (fCurFragmentationOffset > 0) {
// This is the last fragment of a frame that was fragmented over
// more than one packet.  Do any special handling for this case:
fCurFragmentationOffset = 0;
fPreviousFrameEndedFragmentation = True;
}
}


if (numFrameBytesToUse == 0 && frameSize > 0) {
//如果包中有数据并且没有新数据了，则发送之。（这种情况好像很难发生啊！）
// Send our packet now, because we have filled it up:
sendPacketIfNecessary();
} else {
//需要向包中打入数据。

// Use this frame in our outgoing packet:
unsigned char * frameStart = fOutBuf->curPtr();
fOutBuf->increment(numFrameBytesToUse);
// do this now, in case "doSpecialFrameHandling()" calls "setFramePadding()" to appe
nd padding bytes


// Here's where any payload format specific processing gets done:
doSpecialFrameHandling(curFragmentationOffset, frameStart,
numFrameBytesToUse, presentationTime, overflowBytes);


++fNumFramesUsedSoFar;


// Update the time at which the next packet should be sent, based
// on the duration of the frame that we just packed into it.
// However, if this frame has overflow data remaining, then don't
// count its duration yet.
if (overflowBytes == 0) {
fNextSendTime.tv_usec += durationInMicroseconds;
fNextSendTime.tv_sec += fNextSendTime.tv_usec / 1000000;
fNextSendTime.tv_usec %= 1000000;
}


//如果需要，就发出包，否则继续打入数据。
// Send our packet now if (i) it's already at our preferred size, or
// (ii) (heuristic) another frame of the same size as the one we just
//      read would overflow the packet, or
// (iii) it contains the last fragment of a fragmented frame, and we
//      don't allow anything else to follow this or
// (iv) one frame per packet is allowed:
if (fOutBuf->isPreferredSize()
|| fOutBuf->wouldOverflow(numFrameBytesToUse)
|| (fPreviousFrameEndedFragmentation
&& !allowOtherFramesAfterLastFragment())
|| !frameCanAppearAfterPacketStart(
fOutBuf->curPtr() - frameSize, frameSize)) {
// The packet is ready to be sent now
sendPacketIfNecessary();
} else {
// There's room for more frames; try getting another:
packFrame();
}
}
}
```

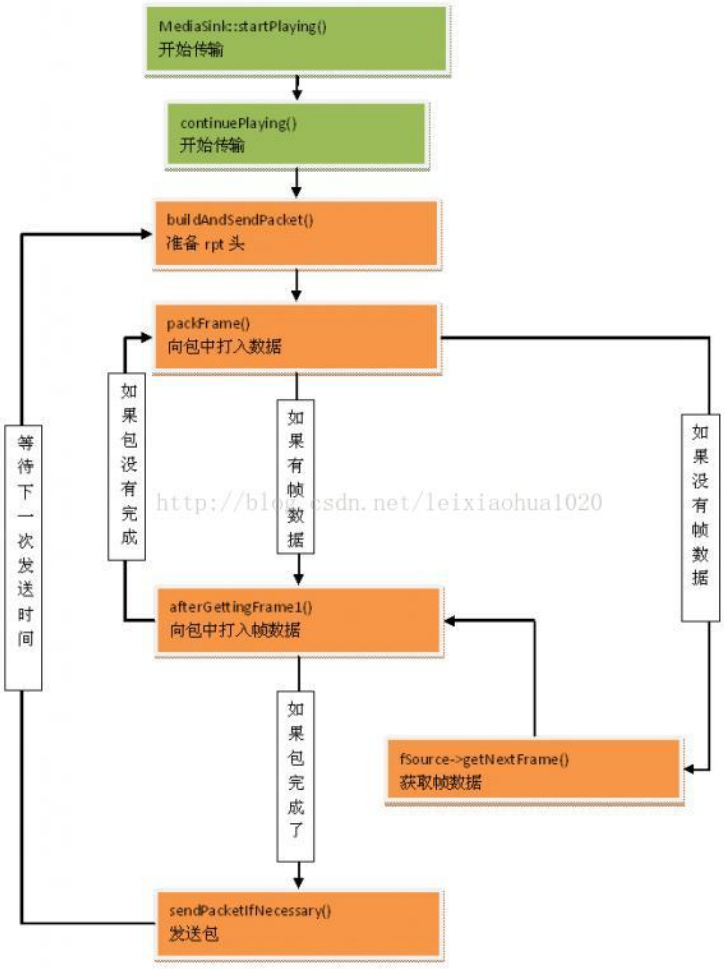
看一下发送数据的函数：

```cpp
void MultiFramedRTPSink::sendPacketIfNecessary()
{
//发送包
if (fNumFramesUsedSoFar > 0) {
// Send the packet:
#ifdef TEST_LOSS
if ((our_random()%10) != 0) // simulate 10% packet loss #####
#endif
if (!fRTPInterface.sendPacket(fOutBuf->packet(),fOutBuf->curPacketSize())) {
// if failure handler has been specified, call it
if (fOnSendErrorFunc != NULL)
(*fOnSendErrorFunc)(fOnSendErrorData);
}
++fPacketCount;
fTotalOctetCount += fOutBuf->curPacketSize();
fOctetCount += fOutBuf->curPacketSize() - rtpHeaderSize
- fSpecialHeaderSize - fTotalFrameSpecificHeaderSizes;


++fSeqNo; // for next time
}


//如果还有剩余数据，则调整缓冲区
if (fOutBuf->haveOverflowData()
&& fOutBuf->totalBytesAvailable() > fOutBuf->totalBufferSize() / 2) {
// Efficiency hack: Reset the packet start pointer to just in front of
// the overflow data (allowing for the RTP header and special headers),
// so that we probably don't have to "memmove()" the overflow data
// into place when building the next packet:
unsigned newPacketStart = fOutBuf->curPacketSize()-
(rtpHeaderSize + fSpecialHeaderSize + frameSpecificHeaderSize());
fOutBuf->adjustPacketStart(newPacketStart);
} else {
// Normal case: Reset the packet start pointer back to the start:
fOutBuf->resetPacketStart();
}
fOutBuf->resetOffset();
fNumFramesUsedSoFar = 0;


if (fNoFramesLeft) {
//如果再没有数据了，则结束之
// We're done:
onSourceClosure( this );
} else {
//如果还有数据，则在下一次需要发送的时间再次打包发送。
// We have more frames left to send.  Figure out when the next frame
// is due to start playing, then make sure that we wait this long before
// sending the next packet.
struct timeval timeNow;
gettimeofday(&timeNow, NULL);
int secsDiff = fNextSendTime.tv_sec - timeNow.tv_sec;
int64_t uSecondsToGo = secsDiff * 1000000
+ (fNextSendTime.tv_usec - timeNow.tv_usec);
if (uSecondsToGo < 0 || secsDiff < 0) { // sanity check: Make sure that the time-to-
delay is non-negative:
uSecondsToGo = 0;
}


// Delay this amount of time:
nextTask() = envir().taskScheduler().scheduleDelayedTask(uSecondsToGo,
(TaskFunc*) sendNext, this );
}
}
```

可以看到为了延迟包的发送，使用了delay task来执行下次打包发送任务。

sendNext()中又调用了buildAndSendPacket()函数，呵呵，又是一个圈圈。

总结一下调用过程：



最后，再说明一下包缓冲区的使用：

MultiFramedRTPSink中的帧数据和包缓冲区共用一个，只是用一些额外的变量指明缓冲区中属于包的部分以及属于帧数据的部分（包以外的数据叫做overflow data）。它有时会把overflow data以mem move的方式移到包开始的位置，有时把包的开始位置直接设置到overflow data开始的地方。那么这个缓冲的大小是怎样确定的呢？是跟据调用者指定的的一个最大的包的大小+60000算出的。这个地方把我搞胡涂了：如果一次从source获取一个帧的话，那这个缓冲应设为不小于最大的一个帧的大小才是，为何是按包的大小设置呢？可以看到，当缓冲不够时只是提示一下：

```cpp
if (numTruncatedBytes > 0) {


unsigned const bufferSize = fOutBuf->totalBytesAvailable();
envir()
<< "MultiFramedRTPSink::afterGettingFrame1(): The input frame data was too large for our buffer size ("
<< bufferSize
<< ").  "
<< numTruncatedBytes
<< " bytes of trailing data was dropped!  Correct this by increasing \"OutPacketBuffer::maxSize\" to at least "
<< OutPacketBuffer::maxSize + numTruncatedBytes
<< ", *before* creating this 'RTPSink'.  (Current value is "
<< OutPacketBuffer::maxSize << ".)\n" ;
}
```

当然此时不会出错，但有可能导致时间戳计算不准，或增加时间戳计算与source端处理的复杂性(因为一次取一帧时间戳是很好计算的)。

原文地址： http://blog.csdn.net/niu_gao/article/details/6921145

live555源代码（VC6）： http://download.csdn.net/detail/leixiaohua1020/6374387

文章标签： live555 rtp 源代码

个人分类： Live555

所属专栏： 开源多媒体项目源代码分析