Video Streaming
Seminar: Advanced Topis in iOS 2012

Peter Grüner

13.08.2012

Abstract

As Mobile Devices become more powerful and fast network access is abundant, streaming of media becomes more important. Software to achieve such streaming is available from multiple sources but this software is often designed for use across the internet and over network boundaries. The use case mostly looked at is streaming your movie collection from your home network to your mobile device. But if you want to stream a live video feed there are less choices available. These choices are further reduced if you have to provide a stream to multiple devices using minimal bandwidth.

1 Aim

The aim of this paper is to gather all necessary information and actually implement a media streaming library for iOS mobile devices. This library should be published under a open source license and be made available for public download. One thing that is not considered in most streaming libraries that target mobile devices is the ability to stream to multiple devices at the same time. Most of the streaming done today is based on a connection for each client This can be achieved by using the UDP Broadcast or Multicast as a network transport.

Requirements:

- The streaming server and client are within the same middle to high bandwidth wireless network.
- Latency from the server to the client should be lower than 500ms.
- The server should be able to send the media to multiple devices in the same network.

2 Definitions

Before we begin with the main content of this paper we need to define some terms used from here on.

Chunk A chunk is an amount of data or a data packet which has to be processed as a whole. In the context of this paper a chunk represents a length of video and/or audio data that can be play back. The length of a chunk can vary to influence different properties of streams.

Stream A stream consists of chunks which are send from a source to a destination in a serial way. In this paper it will refer to audio/video data that is send to a client in a linear way because either the data is received from cameras or other streams or it is not desirable to store all of the data on the device.

Codec In the context of this paper a codec is a piece of software capable of encoding or decoding a digital data stream or signal. The word codec is a portmanteau of "*co*der-*dec*oder".

Transcode Transcoding is the process of translating a digital data stream from one codec into another. This is often done to make a stream compatible with a client that does not support the original codec or to change the quality of the video data so that it uses less space or bandwidth. This process is often done on a stream to convert a incoming stream to a lower quality on the fly to be able to send the stream on a slower network.

3 Research

3.1 Protocols and Standards for streaming

There exist many different ways to send a stream over a network. Each protocol has it's own use cases and different history which lead to very different properties and advantages.

HTTP The Hyper Text Transport Protocol is a basic building block of the internet. It is very versatile and was extended over the years to allow to send files over a network. This protocol is often used to stream media by splitting a file into chunks and sending them one by one.

HTTP Live Streaming[2] This Protocol builds on top of HTTP but adds a new step to allow a more dynamic way of streaming multimedia over HTTP. A client is pointed to a special playlist file which contains information on streams that are available from the server and contains the urls to chunks that contain the actual media. The playlist file is created dynamically by the server and is redownloaded by the client in a time interval set in the playlist. This way a HTTP Live Streaming server can create new chunks and let a client know that they are available. Due to the necessary chunking of the media this protocol is not usable for live streaming with a latency under a couple of seconds.

RTP The Real-time Transport Protocol defines a standardized packet format for delivering audio and video over IP networks. It uses UDP as the underlying tranport protocol and is due to it's low overhead used in many streaming applications such as telephony, video teleconference and live video streaming. It is important to note that RTP is strictly a transport protocol and does not provide any method or information to manipulate the media stream.

RTCP The RTP Control Protocol provides out-of-band statistics and control information for an RTP flow. The primary function of RTCP is to provide feedback on the quality of service (QoS) in media distribution by periodically sending statistics information to participants in a streaming multimedia session. Typically RTP will be sent on an even-numbered UDP port, with RTCP messages being sent over the next higher odd-numbered port.

RTSP The Real Time Streaming Protocol is used for establishing and controlling media sessions between a client and a media streaming server. This protocol handles the exchange of control instructions and information to get a media stream started while the real stream is send over another channel. RTSP is most often used in conjunction with RTP and RTCP although it is also possible to use other protocols.

SIP The Session Instantiation Protocol is a signaling protocol widely used for controlling communication sessions such as voice and video calls over Internet Protocol (IP). This protocol is very similar to RTSP in that is provides a means for controlling a media stream while the real stream is send over another channel, the main difference between the protocols lies within the context in which the protocols are used. SIP is used in the context of Voice over IP (VoIP) while RTSP is more often used in media streaming applications.

3.2 What is already supported on iOS?

The standard way of doing streams on iOS platform is doing it with HTTP Live Streaming (see section 3.1). This is also the only streaming protocol that is natively supported by Apple's iOS platform. The tight integrate of HTTP Live Streaming in the iOS sdk together with high level APIs make it an ideal choice for near real-time streaming applications were a couple of seconds of latency do not matter that much. The latency in HTTP Live Streaming comes from the fact that the media file is split into chunks and each chunk is send to the client and buffered until it is time to play it. The standard size for chunk recommended by Apple is around 10 seconds. HTTP Live Streaming can achieve lower latencies by reducing the size of a chunk but that has some disadvantages. Because of the way HTTP Live Streaming works the client has to constantly poll the playlist file to see if new chunks are available this introduces additional network overhead. Also with reduction of the size of the chunks the risk of stutters or complete stops in playback on the client side rises. Since HTTP Live Streaming plays one chunk after the other it has to wait for each chunk to complete and can not simply ignore a chunk that does not load fast enough.

3.3 What Apps do already provide streaming in iOS?

Most of the apps in the App Store that are marketed as streaming apps or live stream players were developed with the goal of streaming movies in your home network from a host computer to the mobile device. That means most of them can tolerate high latencies

3.3.1 Air Video[7]

This app uses a server part that runs on a Mac or Windows host. It converts video files into the right formats and streams them to the mobile device.

To implement the streaming a variant of HTTP Live Streaming is used.

3.3.2 VLC Streamer[4]

This app works similar as Air Video with the difference that the software that runs on the host computer uses VLC as a back end to do the actual work of transcoding and splitting the movie into chunks. The chunks are then send to the device over the network were they are shown to the user and optionally stored for a later playback.

3.3.3 ChannelVision[8]

This app is one of the rare ones that actually use RTP/RTSP to stream video and audio information. It is used to play the video streams of network enabled cameras. Sadly this app only supports proprietary camera streams and can't be used to play arbitrary media streams.

3.3.4 Video conferencing apps

Video Conferencing Apps such as Skype or Apple's own FaceTime are designed to stream video and audio data over networks with a minimal latency and bandwidth consumption. That would make them ideal for live media streaming but there are some big disadvantages that prevent these apps from being useful.

1. These protocols are largely proprietary with much of the inner workings hidden and in the case of Skype even obfuscated.
2. They are closely interconnected with a support network provided by the companies that run them and can generally not be used outside of that network.
3. The user interface is specifically designed for a telephony kind of interaction not for media streaming.
4. These protocols are designed to work across the internet and across routers that means they can not take advantage of multicast or broadcast messages to reach multiple destinations in the same network. To support a video conference with more then two participants the same video data has to be send multiple times which raises the bandwidth requirement and introduces additional latency.

Most of the disadvantages of these video conferencing apps come from the intended use case they were developed for while the underlying software provides some very interesting advantages. The proprietary nature of this software prevents reuse of these "good" parts but there are some open source libraries that try to provide the same functionality used in theses apps while allowing the reuse and also adaption to new use cases.

3.4 Existing software and libraries

In this chapter we are looking at different software and libraries that could be used to achieve our goals.

3.4.1 VLC [6]

The VLC media player is a very versatile piece of software. In fact it should not be called a media player at all. It supports playback as well as conversion and streaming under one user interface. This media player can play almost any media file that is in use today and that from many sources. It can access normal files as well as network streams, digital tv tuners and webcams.

Advantages

- Support for many different codecs and file formats
- Almost all aspects of the streaming/replay are configurable
- Transcoding of input possible without additional software
- Available for all major platforms such as Windows, Linux/Unix, Mac OS

Disadvantages

- Not easily extendable to use new codecs

3.4.2 Live555[3]

Live555 is a collection of open source libraries for standards-based RTP/RTCP/RTSP/SIP multimedia streaming. It is specially designed to be suitable for embedded and/or low-cost streaming applications.

Advantages

- Supports low latency streaming.
- Available for all major platforms such as Windows, Linux/Unix, Mac.

Disadvantages

- Does not support many different formats
- No transcoding supported, that means a stream has to be provided in the right format and quality to avoid overloading of the network.

3.4.3 PJSIP[5]

PJSIP is a open source library that tries to provide everything that is necessary to build a video conferencing app similar to Skype and FaceTime. It implements different multimedia streaming protocols such as SIP, SDP and RTP and also includes a rich multimedia framework and NAT traversal functionality. PJSIP provides a high level API to make it easy to implement clients on desktop, embedded systems and mobile devices. Sadly this library currently does not support video streaming on iOS.

Advantages

- Provides everything needed to receive a media stream in one library.
- Available for all major platforms such as Windows, Linux/Unix, Mac, iOS.

Disadvantages

- Currently does not support video streaming on iOS.

3.4.4 FFmpeg[1]

FFmpeg is a open source video library that provides almost anything needed when it comes to video processing.

Advantages

- Support for many different codecs and file formats.
- Has an integrated RTP streaming server.
- Transcoding of input possible without additional software.
- Available for all major platforms such as Windows, Linux/Unix, Mac OS.

Disadvantages

- Not easily extendable to use new codecs.
- Very complex to use.
- Difficult to build for iOS.

4 Possible implementation strategies

Basically we have three different realistic ways of providing a low latency real-time stream to a iOS client.

- Send individual frames
- Using a video conferencing applications
- Stream using VLC and FFmpeg

Individual Frames We could simply send video frame by frame and audio second by second over the network. This would reduce the complexity of our library because we would only need to decode the video data on the server and then send the raw picture information over the networks. Such would have a low latency and could also be build in a way to ignore picture information that is to late. The problem only is a picture with even a relatively low resolution of 640x320 pixels already needs 600kb of space for one single frame. If we want to reach a target of a minimum of 15 frames per seconds that would amount to approximately 8,7mb/s for a video only stream. Most networks can not provide that level of bandwidth.

Using a video conferencing application This would be the simplest possible way of achieve our goals. But the problems described in chapter 3.3.4 and the missing features of opensource libraries as described in chapter 3.4.3 make it unfeasible to implement the streaming server in this way.

Streaming using VLC and FFmpeg VLC already provides a big number of features that we need for our implementation. FFmpeg also has support for a large number of codecs and different platforms.

This makes them a good choice for the implementation of our client and server.

5 Implementation

For the implementation of the library it is best to split the effort into two parts.

The server side which is responsible for reading in the source media files or streams, providing the actual stream and if necessary transcoding the media to a suitable quality and format.

The client side which is responsible for receiving the stream and providing a smooth playback experience on the mobile device.

5.1 Server Side

VLC was the first choice for implementing the needed functionality on the server because it provides everything the server needs in one package with a interface that allows many settings to be tweaked into the right shape. Additionally the use of VLC also allows us to reduce the complexity of our library because it does not have to be compiled for the user but is readily available[6].

VLC supports a host of command line parameter that enable a high degree of manipulation on the stream itself and the quality of the media. We can use scripts to run a VLC server that streams different kinds of content.

5.2 Client Side

On the client side the choice was more difficult. iOS does only support the HTTP Live Streaming protocol natively which is no option for low latency streaming. However the need to decode video data ourselves on the device reduces the available choices of libraries drastically. FFmpeg is the obvious choice for any work on video files on Linux/Unix but it is not easy to get it working in iOS.

6 Testing

Since I was not able to get a working app on iOS I could not test the whole library. But it was possible to test parts of the app specifically the streaming server that was already running.

6.1 How to measure latency?

To test if we reached our latency goal we first need to find a way measure to latency introduced by the different phases in the transportation of the video from the server to the client.

The latency that we experience in this streaming application comes from different parts in the process:

- The Network itself. Although the latency experienced in a single Wi-Fi network is typically very low it still can have some effect on the playback performance of a stream.
- Input Caching on the Server.
- Transcoding.
- Input Caching on the Client.

6.2 Some actual measurements

Network Latency We are measuring the network latency in a typical home network setup with a router that supports the 802.11g standard. To measure the latency on the network we ping another machine on the same network and take the average of the resulting round trip time for a package.

The result we got was a round trip time of around 6ms. That means that one package can reach a client from the server in 3ms. This makes the network latency so small that it is effectively not significant for our results.

Input Caching on the Server VLC's default settings for the streaming server include a 300ms input caching. This can be reduced down to 0ms when no transcoding is used, if transcoding is activated it depends on the transcoder on how much input caching is needed.

Transcoding Depending on the format and codec of the input and the output the transcoding process introduces a different amount of latency. It can range from 0ms to 100ms.

Input Caching on the Client The default value for the input caching on the client for network streams is 1000ms. It can be reduced to 0ms without much reduction of the playback performance.

6.2.1 Testing the latency in different configurations

To test where the latency in the playback comes from a video was used that simply showed a count down from three minutes to 0:00:000 with millisecond resolution. In the following screenshots the upper player is always the client while the other player is the synchronized output of the streaming server. Each screenshot from figure 1 to figure 5 was taken with a different server and client configuration.

6.3 Results of testing

With all settings left on the default value streaming from a VLC server to a VLC client on a local machine results in a latency of around 1900ms (seen in figure 1). After optimizing every possible value we get down to around 900ms of latency (seen in figure 3 and 4). Interesting here is that whether or not transcoding is used does not seem to make a difference. In fact turning transcoding off results in slightly higher latencies, but this is most probable a result of noise in

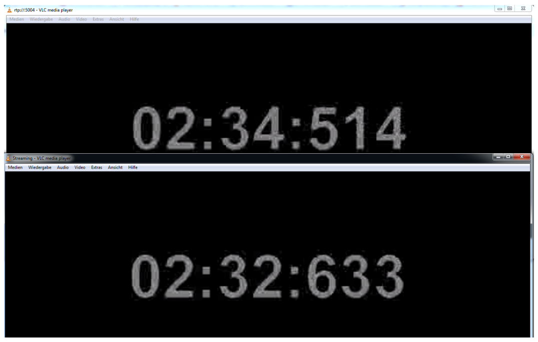

Figure 1: All settings on the client and the server left on the default values we reach a latency of 1881ms

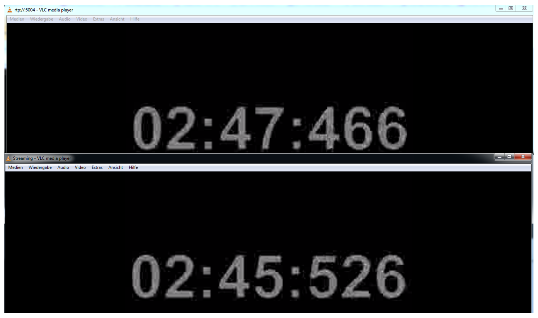

Figure 2: In this picture the input caching on the server was reduced to 100ms. The overall latency does not change.

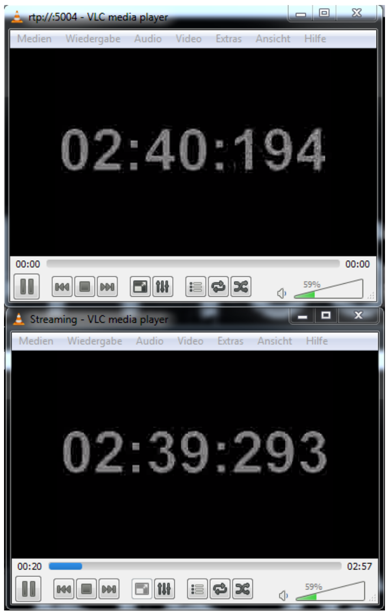
rtp://:5004 - VLC media player
Medien Wiedergabe Audio Video Extras Ansicht Hilfe
02:40:194
00:00
00:00
59%
Streaming - VLC media player
Medien Wiedergabe Audio Video Extras Ansicht Hilfe
02:39:293
00:20
02:57
59%

Figure 3: Here the server and the client side caching was reduced to 0ms. Transcoding was turned on. A Latency of 901ms was reached.

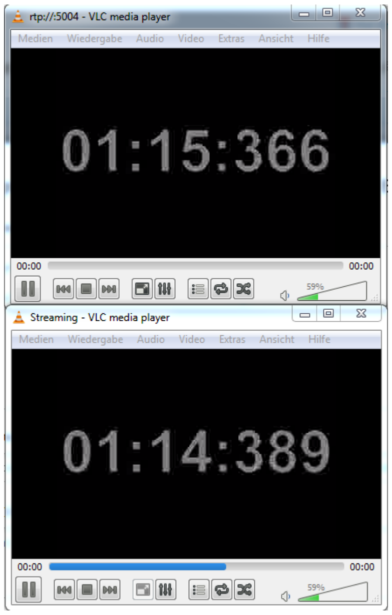
rtp://:5004 - VLC media player
Medien Wiedergabe Audio Video Extras Ansicht Hilfe
01:15:366
00:00
00:00
59%
Streaming - VLC media player
Medien Wiedergabe Audio Video Extras Ansicht Hilfe
01:14:389
00:00
00:00
59%

Figure 4: Server and client caching set to 0ms and transcoding switched of. The latency was 977ms.

the measurements. Another interesting observation that can be made is that changing the value of the input caching on the server side seems to no effect on the overall latency of the stream.

7 Conclusion

In retrospect it may have been a mistake to implement the functionality asked for using VLC and FFmpeg. VLC introduces to much latency that could not, even after much effort and time invested in it, be removed. Using FFmpeg on the client side might have been the right choice if given more resources but in the short time frame of a seminar it is a too large and complicated project to be used in an efficient way.

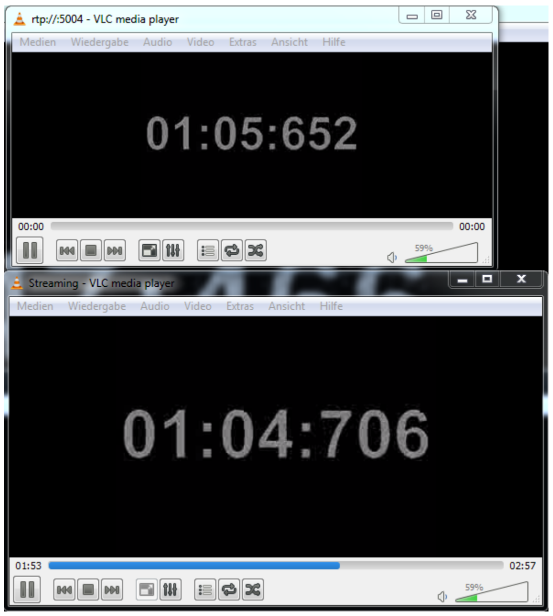
rtp://:5004 - VLC media player
Medien Wiedergabe Audio Video Extras Ansicht Hilfe
01:05:652
00:00
00:00
Streaming - VLC media player
Medien Wiedergabe Audio Video Extras Ansicht Hilfe
01:04:706
01:53
02:57

Figure 5: In this picture we see server caching at 100ms and client caching at 0ms with transcoding turned on. The latency was measured as 946ms.

References

[1] Ffmpeg. http://ffmpeg.org/, August 2012.

[2] *HTTP Live Streaming.* http://tools.ietf.org/html/draft-pantos-http-live-streaming-08, 2012.

[3] Ross Finlayson. Live555. http://www.live555.com/, August 2012.

[4] Hobbyist Software Limited. Vlc Streamer. http://itunes.apple.com/us/app/vlc-streamer/id410031728?mt=8, August 2012.

[5] Teluu Ltd. Pjsip. http://www.pjsip.org/, August 2012.

[6] VideoLAN organization. Vlc media player. http://www.videolan.org/, August 2012.

[7] InMethod s.r.o. Air Video. http://itunes.apple.com/us/app/air-video-watch-your-videos/id306550020?mt=8, August 2012.

[8] Channel Vision. Channelvision. http://itunes.apple.com/de/app/channelvision/id518983315?mt=8, August 2012.