creative commons
COMMONS DEED

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.

변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

Master's Thesis

Power-efficient multimedia streaming for mobile devices reducing overload of media servers

May, 2013

Seoul National University
Electrical Engineering and Computer Science
Mikhail Oparin

Contents

Introduction

Lately, drastic increase in smartphones popularity spread all around the world. According to Strategy Analytics, a market research firm, specializing in digital products and electronics, end users had bought around 1 billion smartphones in the third quarter of 2012 compared to 700 million smartphones been sold in 2011. Such a phenomenal growth is a result of fast expansion of a mobile device's functionality. Today's handheld devices are used mostly for video games, social networks, various online applications, and, of course, multimedia services. There are loads of ways how films can be viewed these days, among which are: Live Television, Recorded Television, Streaming Internet Video, Downloaded Internet Video, DVD or Blu-rays and others. According to CISCO statistics even today mutual percentage of time people spend watching video through the internet surpass that of DVDs/Blu-ray discs, VOD content, or Live Premium Cable (see Figure 1). Also, paying attention to different types of online video we can notice that the time consumers spend for streaming videos is about four times greater than the time they commit to downloading Internet video [1].

Moreover, future forecasts of changes in watching Internet videos predict the increase of Internet video usage on all devices. Even this is true, still Laptop Computers, Tablets, and Smartphones play a leading role in this matter with forty-two, forty, and thirty-nine percent of holders intended to spend more time watching video in future (see Figure 2). Taking into account that Smartphones will become more powerful in the future, which will make video viewing process even easier for the consumers, this category of portable devices is considered to lead among all other devices.

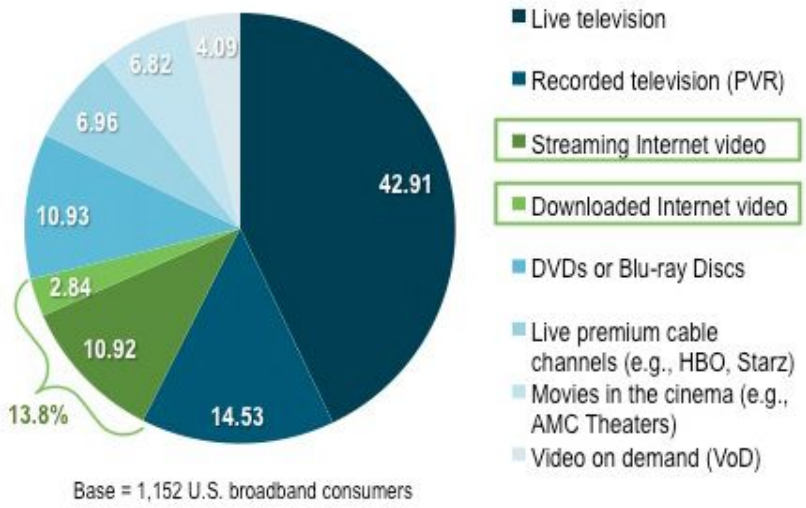

Figure 1. Percentage of time spent watching categories of Professionally Produced Video

Under the growth of Internet video streaming several issues may become apparent. One of them is the problem of centralized media servers overload. The increasing number of queries and outgoing requests along with the amount of data a media server has to process can become for it an insurmountable burden. Another factor impeding even faster development of multimedia content usage is a power consumption required for viewing multimedia data. Scanty of energy resources of portable devices make the problem even more critical.

This Master's thesis consists of four parts, Experimental Evaluation and Conclusions. First part is devoted to various video streaming methods. It introduces the concept of video streaming, explains how video streming methods are implemented and presents benefits of each method being applied to specific problem scope. Second part explains the basic ideas of Aspect-Oriented programming, shows AOP benefits compared to Object-Oriented programming and how

gives suggestion how AOP programming can be used for streaming. Third part describes SorMob offloading framework. In forth part, SorTube application structure and implemetation method is introduced. Finally, in chapter five introduces the experiments performed with SorMob-SorTube system and shown the results of server overload reduction. After that, the author summarizes conclusions and proposes plan for future work.

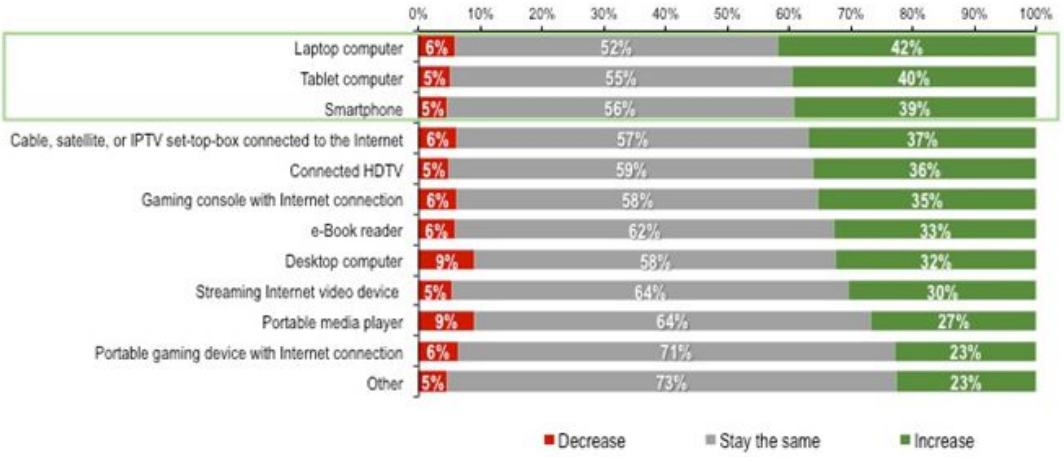

Figure 2. Expected change in watching Internet video over next two years by device types

1. Video Streaming

1.1 Digital streaming

First occurrence of what we might recognize now as ‘streaming media’ happened in 1922, when George Owen Squier patented Wired Radio, a service that piped music to businesses and subscribers over wires. This was the first and the last successful attempt until the age of personal computers. Hardware and software of the last gave an opportunity to play audio and display video. The problem remained that time was the lack of power required for video rendering and the capacity of transmission channels big enough to transmit video data. The only solution available at that time was to download a video file first and then play it once the file is fully downloaded. This solution also involved additional inconveniences such as Extended Graphics Array (XGA) monitor with a resolution of 640 x 480 pixels at 16 bits per pixel. Average video parameters were following: 320 x 240 pixels resolution, 24 frames per second a video refresh rate. Thus, processed data per second was:

$$D\_s=W\times H\times nbytes\_pix\times nbytes\_sec=320\times 240\times 2\times 24\approx 3.5MB/sec$$

where W – frame width; H – frame height; nbytes_pix – number of bytes per pixel; nbytes_sec – number of bytes per second.

Thus, one of the major thing had to be done before streaming media could happen was video compression. With such a huge data as 3.5MB per second one minute video would take about 200MB. At that time this amount of space was not even available on the hard

drive. Another obstruction was the requisite of CPU to be able to decompress the video data in real time and render frames at the correct frame rate. Also, the data bus able to handle transferring of the required amount of data to the video sub-system was needed. Answer for all those problems appeared by the mid-1990s.

1.2 Multicasting

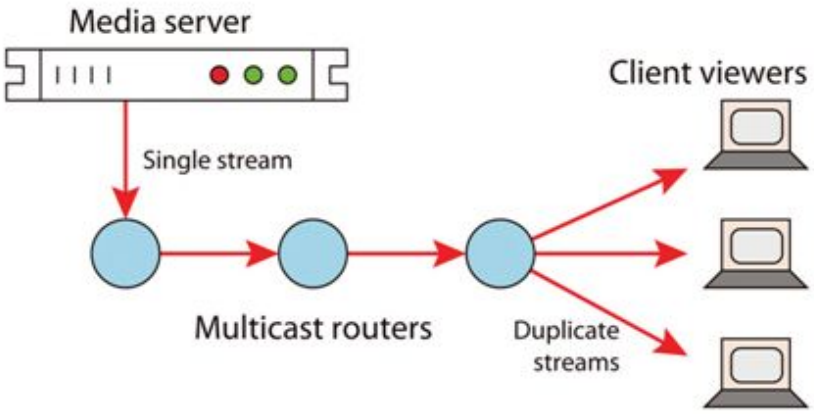

Figure 3. A multicast network

Multicast is the delivery of a message or information to a group of destination computers simultaneously in a single transmission from the source. Thus it allowed video data to be streamed efficiently from one server to several receivers concurrently. A good example of multicast system is an internet radio, where all users hear the same stream. In multicasting, media server usually has only a single low-bandwidth connection to a multicast node, where the rest of the transmission and eventual data streams duplication is done by the nodes in the internet. It results in independence of the server load from the number of consumers. Such a technology is called 'unicast'

and this is what we encounter when we watch a YouTube video or any movie online.

However, there are multiple issues with multicasting. First, considering 'unicast' implementation, the media server does not benefit with number of users growth. Anyway, the server has to transmit a data to every consumer. Therefore, while the number of users will increase, the media server and network capacities must grow too. Also, special routers, which would be able to pass single data stream on, are required. A requisite ability to program those routers so that only a single data stream is passed between brings on the necessity to build a network of multicast routers. This evokes a 'tunneling idea', which is basically a connection of the special routers, which tunnel the multicast data between them over the normal Internet. When a consumer needs the multicast data it identifies the nearest multicast router so that it can receive a 'unicast' of the data stream from that router (see Figure 3). Another factor impeding multicast networks from getting caught on is the payment issue. Because the router issuing the data and the router duplicating the data can be far away from each other and may be controlled by different ISPs the cost for those ISPs will be different.

Thus, to make streaming viable following demands should be met:

1) Special compressed video format to facilitate play while downloading

2) Buffering of data

3) The protocol between viewer and remote media server allowing for renegotiating the resolution in case of network latency or bandwidth changes

These requirements have been fulfilled by Macromedia Flash, which provided a multi-platform, multi-browser streaming viewer free of

charge. Flash was available on the vast majority of PCs, and formed the basis of streaming sites, such as YouTube, Vimeo and others. However, Flash is not that popular as a streaming viewer anymore. The reason is it requires intensive CPU resources and as a result compromises the battery life of handheld devices such as smartphones and tablets.

1.3 Progressive Download

Progressive download is a basic method which most of web servers support. The idea of this method is the user downloads an entire video file and plays it. In case of long videos, the viewer can start playing video from the buffer while keeping downloading the rest of the video file (see Figure 4). Major advantages here are the simplicity to organize progressive download for publisher and the ability to stop and buffer video for users.

Nevertheless, progressive download method has disadvantages. The main is that the full video is downloaded to the client. Suppose, for example, a viewer does not need to watch all two and half hours of the movie, but watches only first ten minutes. Then, if the viewer's Internet is fast enough the whole video will be downloaded before he finishes watching a short part of it. This means, it will take up bandwidth resources which could be used for other purposes. Also, it will add up more traffic for which user will need to pay.

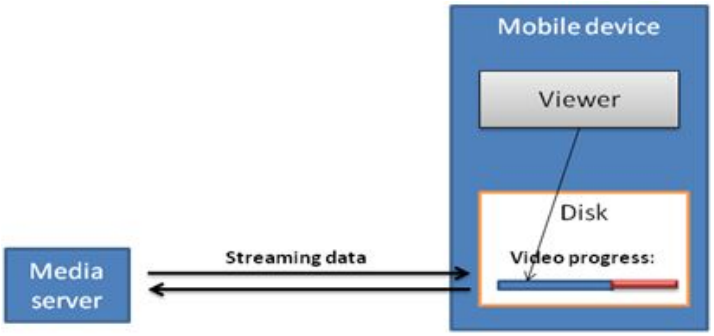

Figure 4. Progressive download

1.4 Pseudo streaming

Pseudo-streaming main feature is the ability to download an actual file and to play that file as it is being downloaded. Nowadays YouTube tends to use this technology. Because the video file is fully downloaded, one can replay YouTube video very quickly, no more data needed to be downloaded. However, already downloaded video file will be deleted once you move to another video.

Pseudo streaming is based on progressive download but mimics Video on Demand (VOD) streaming. Usually it is not supported by web servers however, additional extensions or plug-ins can be added to Apache, Tomcat, IIS and others. Besides the advantages pseudo streaming method inherits from progressive download, it has its own peculiarity. For example, it adds an ability to seek through the video. As a result, some parts of the video can be skipped and not downloaded by user, which provides a solution for the problems existing in progressive download method.

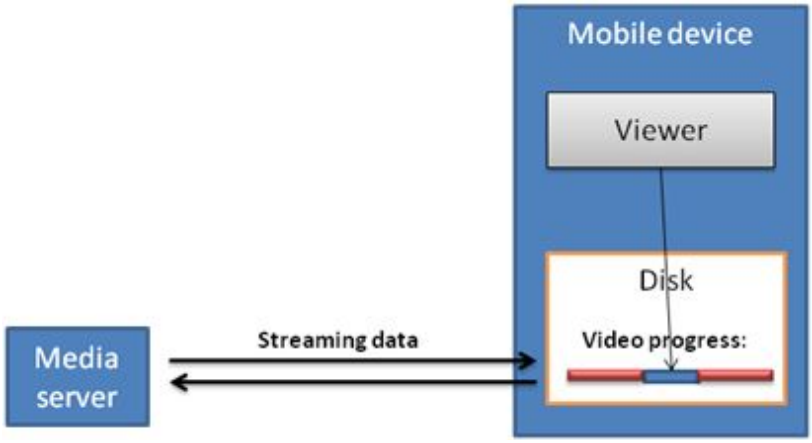

Figure 5. Pseudo streaming

1.5 Dynamic Adaptive Streaming over HTTP

Dynamic Adaptive Streaming over HTTP (DASH) is one of the latest technologies. The key idea here is to split a video into multiple parts and deliver them over HTTP. Later, those parts are merged together at the client side. VOD and live streaming can be implemented by this method. This method is already used by big companies such as Apple (HTTP Live Streaming), Microsoft (Smooth Streaming), Adobe (HTTP Dynamic Streaming), Octoshape (Adaptive Bitrate) and others.

The advantages of DASH are independence from specific streaming servers or protocols along with better content protection, since video is delivered in chunks. It also has some disadvantages, among which are following:

-Requires client support (can be supported only after some version of client player)
- Not full compatibility of different implementations
- Longer delay compared to traditional live streaming (RTP, RTMP)

- Quality adaption is slower compared to traditional live streaming

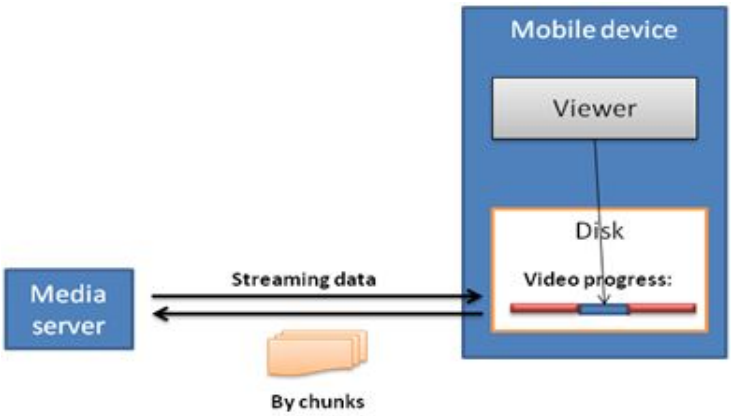

Figure 6. Dynamic Adaptive Streaming over HTTP

1.6 Real streaming

Real streaming is characterized by data-buffering without saving file on disk. It also allows automatic resolution changes once the network throughput or latency changes happen. In comparison with the service we see on YouTube, resolution changes we can make there are one time made option, which is if you choose one out of several quality option all the video will be downloaded in this quality. For YouTube-like service been able to work, multiple resolution versions of the video must be stored at the server.

Also, another type of protocol and port is used by media servers providing real time video and audio streaming. A common protocol used in real streaming is Real-Time Message Protocol (RTMP) together with 1935 port. Another, protocol used for real streaming is Real-Time Streaming Protocol (RTSP), which contains Real-time Transport Protocol (RTP) and Real-Time Control Protocol (RTCP). These protocols divide the stream into very small packets and send the packets to the client viewer.

Considering all above mentioned, streaming has a big history and it keeps developing nowadays. The trade-offs between quality of service provided and the quickness of data delivery from server to client engendered various types of data streaming. With today's growing popularity of media services, especially online video services, the amount of data needed to be streamed grows rapidly. To contend the arising issue, more efficient and sophisticated methods of data delivery should be proposed.

2. Aspect-Oriented Programming

Object-Oriented (OO) programming can be considered as a predecessor of Aspect-oriented programming (AOP). When OO programming appeared first time, it caused a furor and quickly became a mainstream of software development. Programmers could simplify complex and intricate problems by interpreting big systems as an assortment of objects and setting the rules by which these objects interact. The only issue characterizing OO programming is that it is static, which means that a small change in requirements may prolong development timeline.

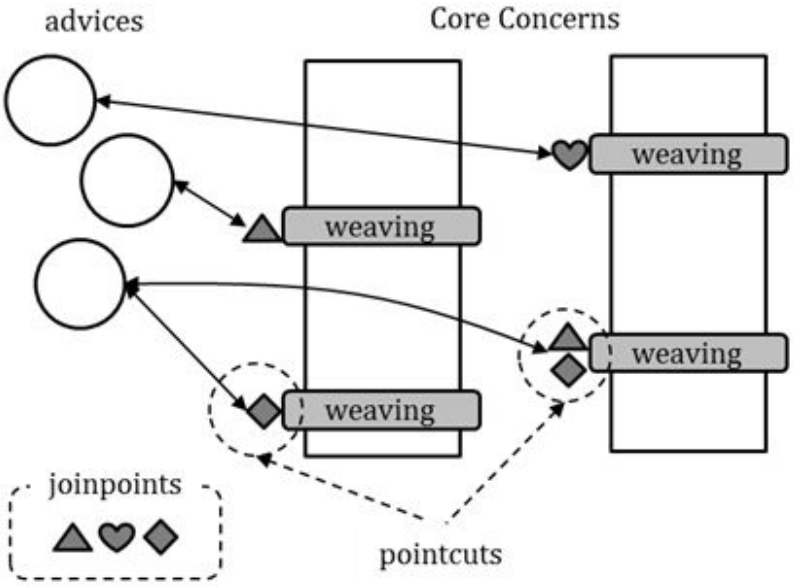

Figure 7. AOP components

AOP, in contrast to OO programming, allows the developers to dynamically modify the static OO model. Developers can include the code required to fulfill new additional requirements without need to modify the original static model. Moreover, presence of original

source code is not necessary. And with AOP we do not need to scatter our new code across the previously written code, as we would have to do using only OO programming. Now we can store our new code in a stand-alone module.

2.1 AOP terminology

Cross-cutting concerns: In OO programming we can have several classes performing given distinct functions. Each class is in order for its own tasks and therefore does not overlap with the rest of classes. However, if, for example, we need to add 'grades' property to 'undergraduate students' class and to 'graduate students' class it will be the same for both of them. In given example 'grades' represents cross-cutting concern.

Advice: This is an additional code that you want to apply to existing model. Considering the example above advice is the implementation of 'grades' property.

Point-cut: This is the point of execution in the application at which cross-cutting concern starts its execution. In the example above point-cut is reached when the thread enters a method referring to 'grades' and another point-cut is reached when the thread exits the method.

Aspect: This is a combination of a point-cut and corresponding to this point-cut advice.

Introduction: This is an inter-type declaration, which is declaring of additional methods or fields on behalf of a type. As an example of introduction a new member (field, method, or constructor) declared by aspect class but owned by other types can be regarded. This member is called inter-type member. In Figure 8 PointAssertions aspect class

privately declares the assertion methods ‘assertX’ and ‘assertY’ of Point class.

Target object: The object being advised by one or more aspects

AOP proxy: An object created by the AOP framework in order to implement the aspect contracts (it can be JDK dynamic proxy or a CGLIB proxy).

Weaving: linking aspects with other application types or objects to create an advised object.

Joint point: A point duringthe execution of a method or the handling of an exception.

```
class Point  {
      int x, y;

      public void setX(int x) { this.x = x; }
      public void setY(int y) { this.y = y; }

      public static void main(String[] args) {
           Point p = new Point();
           p.setX(3); p.setY(333);
      }
}

aspect PointAssertions {

      private boolean Point.assertX(int x) {
           return (x <= 100 && x >= 0);
      }
      private boolean Point.assertY(int y) {
           return (y <= 100 && y >= 0);
      }

      before(Point p, int x): target(p) && args(x) && call(void setX(int)) {
           if (!p.assertX(x)) {
                System.out.println("Illegal value for x"); return;
           }
      }
      before(Point p, int y): target(p) && args(y) && call(void setY(int)) {
           if (!p.assertY(y)) {
                System.out.println("Illegal value for y"); return;
```

```
        }
    }
}
```

Figure 8. Inter-type declaration

2.2 AOP frameworks

There are multiple AOP frameworks out of which the most known are:

JBossAOP: This framework is pure Java oriented and therefore, can be used in any programming environment. It provides cleaner separation from application logic system code. It also contains a prepackaged set of aspects that can be applied via annotations, point-cut, expressions, or dynamically at runtime. Moreover, JBossAOP framework allows to apply interceptor technology and patterns to plain Java classes and Dynamic Proxies.

AspectWerkz: This is a dynamic, taking little space, and high-performant AOP framework for Java. It utilizes runtime bytecode modification to weave the classes at runtime, which makes it the quickest and fully-featured available framework. Though, it has less features than AspectJ. Advices here can be added, removed, and re-structured at runtime. Also, the aspects in AspectWerkz can be defined through XML file or using runtime attributes.

Probably the most notable feature of AspectWerkz is its ability to run in two different modes: online and offline. In online mode, it uses low-level class-loading mechanism, which is a part of Java Virtual Machine (JVM). It results in the ability to intercept all class-loading calls and transform bytecode on the fly. This method's advantage is that it can use any class-loader and weave classes at class-load time, which implies that classes are not modified but deployed as

usual. The drawback here is that you need to configure the application server, and it may not be possible.

In offline mode, to generate the classes you need to go through two stages. The fist stage is the standard 'javac' compilation, after which AcpectWerkz compiler need to be run in offline mode, being pointed to the newly created class files. After that, the compiler will modify the bytecode of the classes to include new advices at the correct pont-cuts. The advantage of this method is that newly created classes will be able to run in any JVM starting from version 1.3.

AspectJ: Currently is one of the most powerful and fully featured frameworks. The architects of AspectJ from PARC added new keywords to the Java language, which makes the users of their implementation to use special compiler and potentially reconfigure the editor. Nowadays, AspectJ provides plug-in for Eclipse IDE tool, which makes the use of the framework more comfortable.

With AspectJ users can define their own aspects, and once the aspect is defined, the programmer can perform corresponding advice in following modes using annotations:

'@Before' - advice starts execution before the method

'@After' - advice starts execution after method returns its result (see Figure 7)

'@AfterReturning' - advice starts after method returns its result and it can intercept method's result

'@AfterThrowing' - advice starts after method throws an exception

'@Around' - combines all mentioned above modes

```
package ru.javaxblog.aspect;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.After;
```

```
@Aspect
public class LoggingAspect {

    @After("execution(*
ru.javaxblog.customer.services.CustomerService.addCustomer(..))
")
    public void logAfter(JoinPoint joinPoint) {

        System out println("logAfter() is running!");
        System out println "перехват :
" + joinPoint.getSignature().getName());
        System out println("******");

    }

}
```

Figure 9. Example of “@After” annotation use

In Figure 9 is shown an example of “@After” annotation use. There the ‘logAfter’ method of aspect class will be executed every time after the ‘addCustomer’ method of the ‘CustomerService’ class call.

2.3 AOP for Android

Android is a Linux-based operating system (OS) for majority of touchscreen mobile devices such as smartphones and tablets. Android OS consists of a kernel based on Linux kernel version 2.6, Android 4.0 Ice Cream Sandwich with middleware, libraries, and APIs written in C, and application software running on an application framework with Java-compatible libraries based on Apache Harmony. Android uses Dalvik virtual machine (DVM) with just-in-time compilation to run Dalvik ‘.dex code’, which is obtained from Java bytecode.

Unfortunately AOP advices cannot be applied directly to .dex files,

because Android does not have a byte code generation scheme. At the same time, Java supports several forms of advice introduction:

- Compile time bytecode manipulation
- Load time bytecode manipulation
- Run time advices using auto-proxies or jdk interceptors

Thus in case of Android application, to apply an advice to it we need to use first method. We should compile aspects directly into Java code and convert the resulting code to .dex format to run on Android device. Also, Android does not support AOP by itself, that is one of AOP frameworks, examples of which were mentioned above, should be used. Thus, we can swap the Java compiler with AspectJ, or any other AOP framework, compiler, and new compilation will be able to compile .aj aspects without destroying pre-processed resources. And because the end file will be of .dex format, the compiled applications will run seamlessly on all Android phones without need for any modifications.

3. SorMob framework

3.1 Offloading

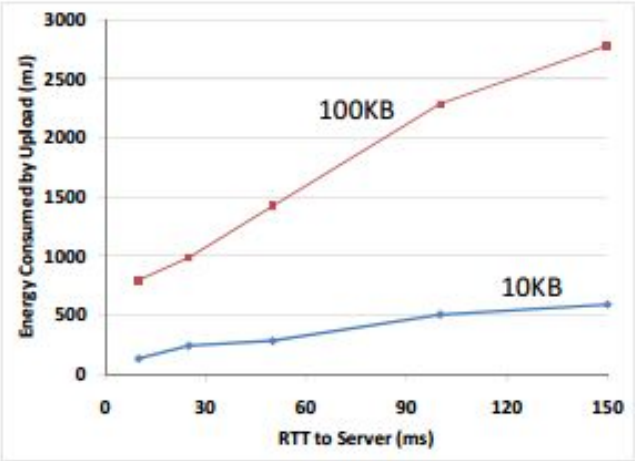

Figure 10. Offloading energy consumption depending on RTT

Today's mobile device energy lack problem involves due to two main factors: limited battery capacity and increasing energy demand from high-performance applications. Nowadays, there are three different types of applications, which facilitate the growth. Those are: video games, streaming videos, and a mobile device sensors. Moreover, current tendencies in battery technology does not give a hope that the energy problem will disappear in near future. Recent researches about using newer chemicals or fuel cells do not seem able to increase the capacity of batteries significantly. One of the solutions, which can be proposed to contend current energy problem is offloading technique. So far, two ways exist to apply offloading to mobile device computations. These are either offload through 3G or over Wi-Fi. 3G has its benefits because of almost-ubiquitous

coverage, however, sending data over 3G takes time and bandwidth is limited. Also, 3G has very high energy consumption, and poor performance which definitely makes it not-a-choice in my research. In contrast to 3G network, Wi-Fi consumes not that much energy and has much better performance.

Previous researches [1] show that Wi-Fi offloading can benefit even more if specific demands will be met. First, the latency to the cloud server should be minimum for mobile device users. It makes shorter round trip time (RTT), which, in turn, results in substantial decrease of energy consumption. Additionally, remote execution shows best result when the remote server is on the same LAN as the Wi-Fi access point. Dependence of energy consumption from the length of RTT is represented in Figure 10.

3.2 SorMob architecture

SorMob is a framework serving for offloading computations from a mobile device to cloud server based on AOP. Current scope of SorMob is Java applications running on Android. SorMob's specificity is that it operates at the source level making offloading process not that lucid to application developers and users. It makes the whole working process more secure compared to other binary level approaches. No binary or source code is needed to be modified. Alternatively, user can choose which target objects or methods to upload. Sormob abstract architecture and processing flow are shown in Figure 11.

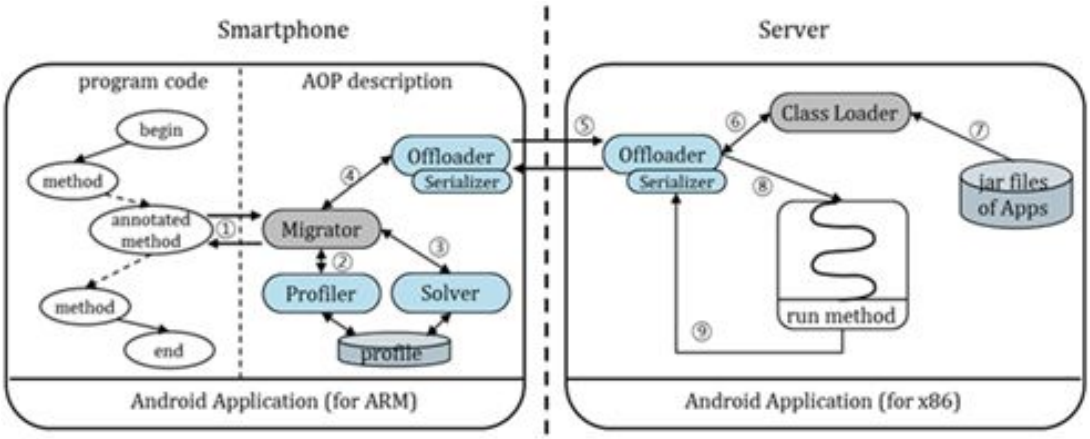

Figure 11. SorMob framework and processing flow

3.3 SorMob workflow

The goal of SorMob framework is to provide effective offloading of heavy and therefore power consuming computations. For that purpose, we need to go through three stages

- making a decision which code is worthy of offloading
- weaving of necessary aspects into the source code
- code offloading

Currently, decision making scheme is trivial and it always offloads transcoding part if it has an access to cloud server. In future, more sophisticated decision making scheme will be developed, so that it can offload code considering various factors such as connection bandwidth, computational abilities and workload of the mobile device and cloud server.

After code offloading decision is made, AOP starts its work. It weaves the code which was chosen for offloading with preamble and coda parts, where preamble is in order for serialization, and offloading, while coda part receives the result of computations done by cloud server. Also, as part of future work, preambles will contain

source code allowing to make dynamic decisions on offloading.

Finally, after SorMob framework linked all aspects and, all decision about offloading are made, the framework simply sends these functions' requisites to SorMob at the offloading server. At server side SorMob compares received requisites with its version of SorTube application and executes corresponding functions.

4. SorTube

4.1 Overview

SorTube is an application which's main goal is to make video playback even faster and more accessible for users. Two techniques in SorTube help it to achieve the goal, those are: the use of the offloading server and precise adjusting parameters of video, making it perfectly fit characteristics of viewer's device. The offloading server may undertake part of media server workload, making it possible for more devices to access the same server. Also, precise quality adjusting will provide more adequate video picture and will reduce useless power consumption of devices which's playback characteristics are lower than the quality of the video viewed on the device.

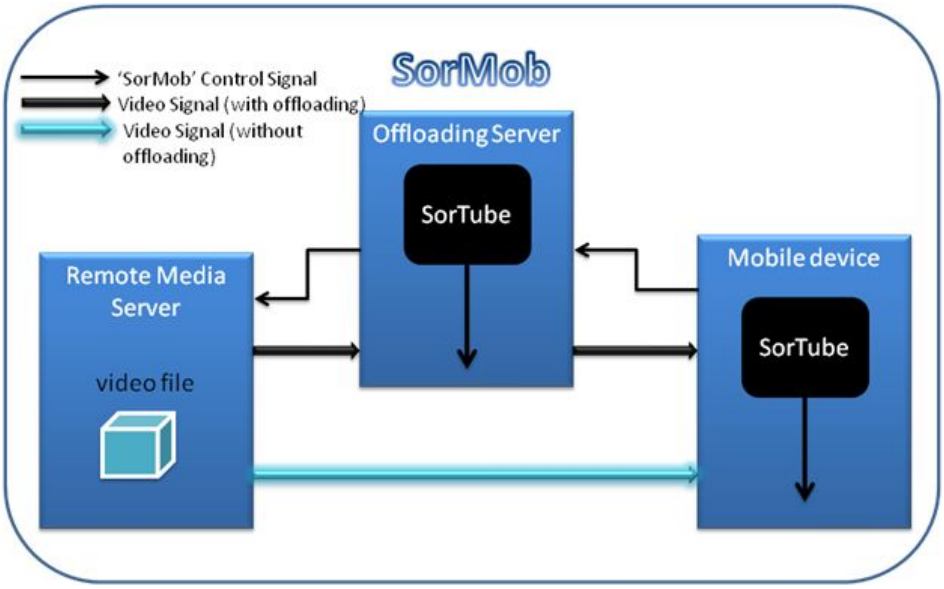

Figure 12. 'SorMob-SorTube' abstract scheme

Primary functionality of the 'SorTube' application consists of following services: dynamic adaptive streaming over HTTP, video

transcoding, and video playback. DASH, being based on 'progressive download', is not a real streaming, but rather a bulk download of a video. In SorTube application it solves a problem of video transferring between a mobile device, offloading server, and remote media server. Video transcoding makes original high-quality video, needed to be transferred from remote media server to the offloading server, less in size and exactly fitting the requirements of the mobile device. Finally, video player allows a user to watch the requested video without need to install any additional software.

In my research I used SorTube application in collaboration with SorMob offloading framework. From the whole process prospect, while SorTube is responsible for all video related work, SorMob coordinates mutual work of two SorTube applications running at server and mobile device side and their communication with the remote media server. Therefore, we can treat SorTube and SorMob software as a resource and manager elements correspondingly, where SorMob element sets an optimal control-flow for the whole 'SorMob-SorTube' system by effective manipulation of its under-controlled resources. A generalized scheme of the system is represented in figure 12. It consists of a mobile device, offloading cloud server, remote media server, and two specimens of SorTube application installed on a mobile device and cloud server. Now, let us see in more detail how the process of adaptive video viewing works.

4.2 Workflow

When user starts SorTube application first time on a mobile device, he or she needs to provide SorTube application with the ip address of the offloading server and the remote media server storing a required video file. Next, version of SorTube application stored at the mobile device connects to the machine serving as an offloading

server and sends to it video request. Because the offloading server downloads original video file, we can save extra energy by exempting the mobile device from necessity to work with a video of deliberately too high quality. Thus, we can reduce a mobile device energy expenditure and watch a video of the quality we need.

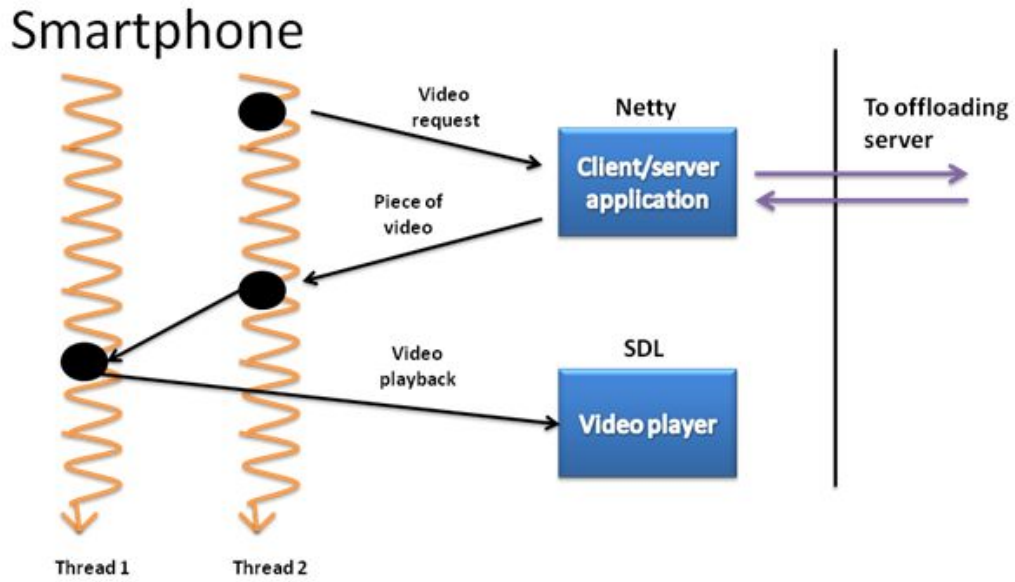

Figure 13. SorTube workflow at a mobile device side

After the specimen of SorTube application on the offloading server receives the request it transfers the request to the media server. In turn, media server sends the original video file to the offloading server by DASH. At this stage offloading server obtains the packets of high quality video, which now should be processed. Video processing is done by a transcoder at the offloading server side. Transcoding is done in such a way that the transcoder receives video data by packets, converts video frames into the ones of required format and sends them to a mobile device by chunks of 5-seconds

length. The media standard of 5-seconds chunks, and therefore of the end video, is H.264 Part 14, also known as MP4. It perfectly combines high quality of a video along with the compression that gives great file size. In the end, when the mobile device receives a chunk of video it starts playing it in a 'near real-time' way, where 'near real-time' implies a delay needed for video transcoding. Abstract schemes of the data and control flows on offloading server and mobile device sides are represented in figure 13 and 14 respectively.

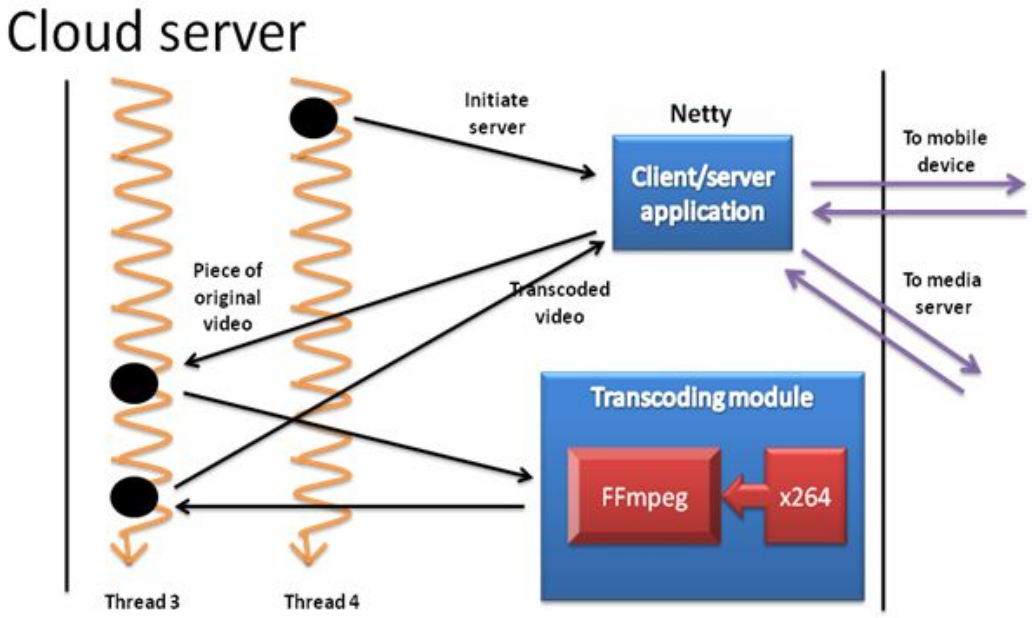

Figure 14. 'SorTube' workflow at the offloading server side

4.3 Implementation

Sortube contains three major parts: client-server framework, multimedia data transcoder, and multimedia viewer. To implement all the functionality multiple APIs, libraries, and plug-ins were used. Realization of all these parts as well as SorTube application in general is deliniated lower.

Client-server: This part is done with the use of Netty, which is an asynchronous event-driven network application framework. Asynchronous in this context means that there is no return value and invocation is usually instantaneous. The results, if there are any, will be delivered back in another thread. Netty greatly simplifies network programming and helps to implement client-server network through TCP, UDP, or HTTP. In my research I used HTTP as a transferring protocol for video data.

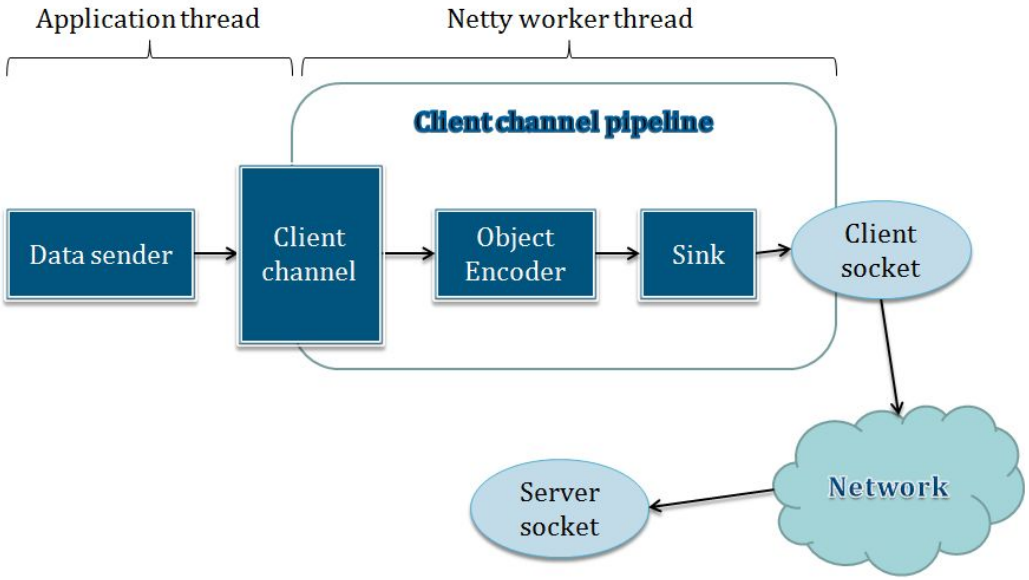

Figure 15. Netty client side representation

Netty work is organized in a following manner. It has the basic currency, which is a ChannelBuffer - sequential accessible sequence os zero or more bytes. Through ChannelBuffers data is passed around in Netty. Thus, to send an object or a request to the server, your data need to be wrapped into a request that HTTP server would understand after the data is transferred. The abstraction of the network through which ChannelBuffer is transported is the Netty Channel. It provides the interface to connect and to write to the

destination (no read, since connection is asynchronous). Another abstraction in Netty framework is the ChannelFuture, which performs a role of container for methods' results which are not known yet. But as soon as they become available ChannelFuture will deliver them.

Channels are created by ChannelFactories of two types, for client channels and for server channels. Both types contain variations for different transporting protocols, one of which is HTTP. However, having only NettyChannel is not enough to transfer data because, as was mentioned above, pure data bytes need to be adjusted to form a request or any representation of structured data. For that purpose Netty deploy ChannelPipelines, which is a stack of interceptors that can manipulate or transform the values that are passed to them (see Figure 15).

Transcoding: SorTube transcoder is based on FFmpeg native libraries. Description of transcoding process is shown in Figure 16. First, demuxer obtains encoded data packets out of input multimedia file. Next, decoder is applied retrieving decoded frames. Later, decoded frames can be filtered and their properties, but not the content, can be changed. As a result, newly quality-adjusted video frames appear, which should be re-encoded and multiplexed to receive end video file.

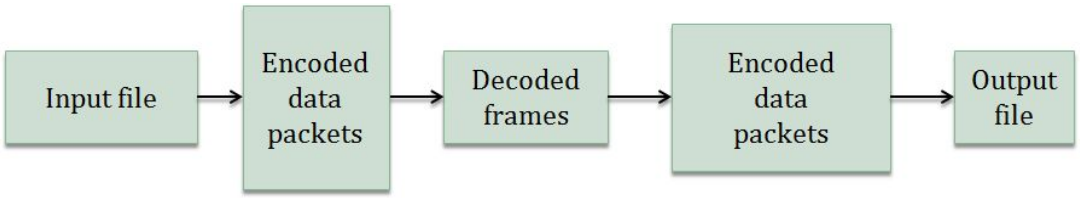

Figure 16. FFmpeg transcoding process

Also for my research I needed H.264 encoder which is not a part of FFmpeg. Therefore, to allot to SorTube an ability to convert video data into H.264 FFmpeg was build together with x264 encoder.

Finally, to let transcoding work on a mobile device when there is no offloading server around, a sheaf FFmpeg-x264 was build for ARM as well as for x86 architectures. Thus, SorTube transcoder supports the vast majority of common standards with additional ability to encode in H.264.

Viewer: SorTube viewer is based on Simple Directmedia Layer (SDL), which is a cross-platform multimedia library providing low level access to audio and video via OpenGL. SDL is used in majority of video playback software and supports most common operating systems. SDL support by FFmpeg makes 'FFmpeg transcoding - SDL viewing' work fluently.

5. Experimental Evaluation

5.1 Media server overload because of transcoding

This experiment shows the dependance of server overload because of heavy video transconding computations from number of clients needed to be served by the server. It can be seen that while transcoding process of the tested video file takes about 25 seconds, when we increase the number of mobile devices accessing the server up to 10, the time required for transoding of the same video file increases up to 100s. Considering the real number of mobile devices trying to access a media server is estimated at hundreds of thousands, server overload problem may become a serious obstruction for real-time streaming in future.

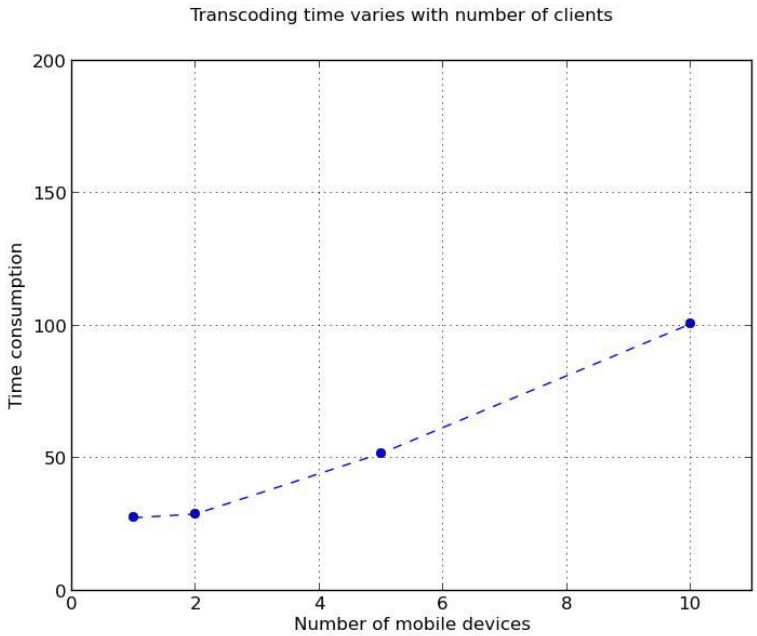

Figure. 17 Dependence of time required for transcoding from number of mobile devices

5.2 Burden of transcoding performed at a mobile device side

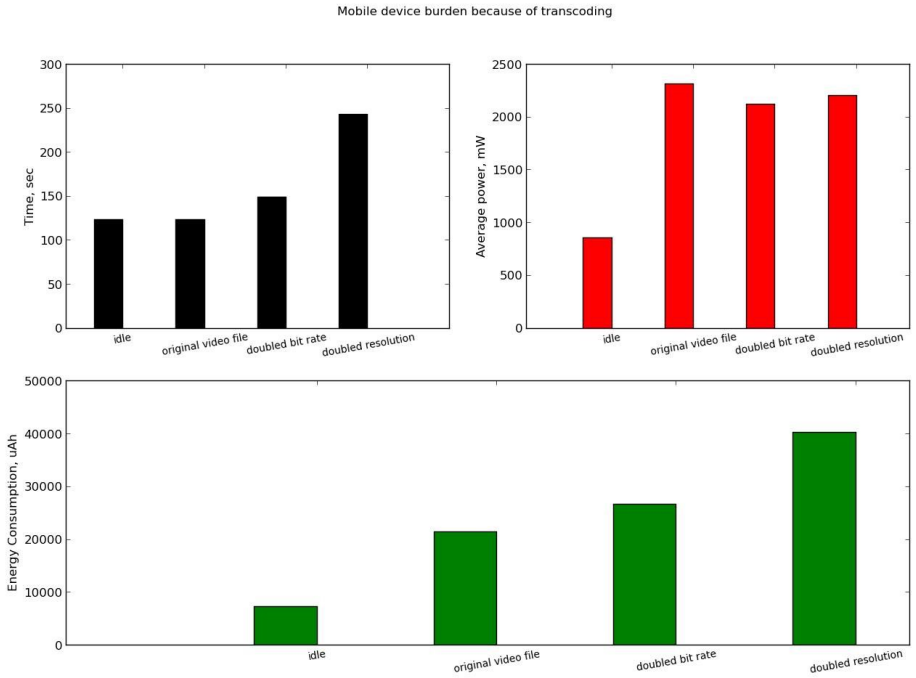

Fig. 18 Time, Energym and Power consumption by transcoding performed on a mobile device

Video transcoding contains a lot of computations which makes the process a hard work for CPU and consequently consumes a lot of energy. Because of the limits of energy of mobile devices, such a heavy data processing as transcoding should not be performed on them. In figure 18, given the comparison of time, average power consumption and total energy required for transcoding process performed on a mobile device. The experiment was conducted for four different cases, which are: idle, no transcoding is performed; for original video file, video file with double bitrate, and video file with

double resolution. Proceeding from the results, it can be inferred that video with double resolution is processed about twice longer than the normal one. At the same time, average power consumption of all video files is about the same. Hence, the results for energy consumption, according to which the amount of energy consumed for transcoding the video with double resolution is about two times greater than the energy consumed for normal video and about four times greater than the energy spent in idle state, are justified.

5.3 Video playback power consumption with and without offloading

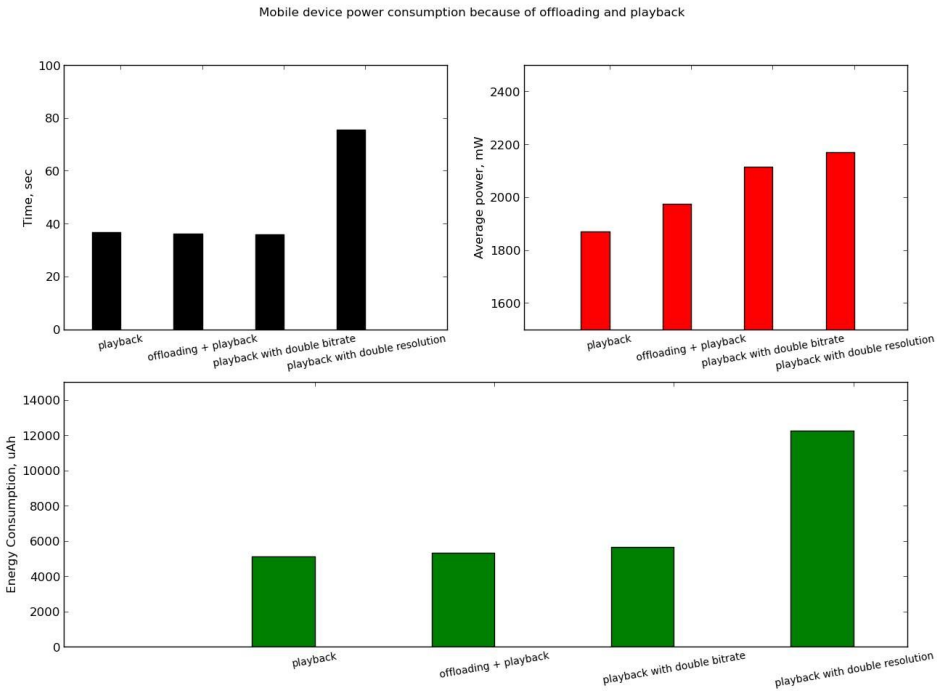

Figure 19. Dependence of a mobile device energy consumption from different ways of video playback

Third experiment was conducted to check the amount of energy which can be saved due to offloading compared to usual video download and playback. In figure 19, one can see that total time required for offloading and playback of the video is about the same as the time required just for playback. It shows that extra time required by offloading technique is insignificant and does not spoil the advantages. Also, it is apparent that major increase in energy consumption happens because of the increase in quality of the video, exactly resolution. Thus, the contribution of transcoding performed at the offloading server is substantial in terms of reducing a mobile device's energy usage.

Conclusions

To conclude, above stated research addressed an idea of real-time mass streaming for heterogeneous assortment of mobile devices. The results of experiment show that the main demand to reduce the overload of centralized media servers is met successfully, which makes the idea viable for future ubiquitous adoption. Besides the reduction of media servers overload, proposed method allows to reduce a mobile device energy consumption of video viewing significantly. The importance of this achievement can be understood the best while thinking about an average battery life of current smartphones, which is approximately 6 hours.

References

[1] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, Paramvir Bahl, “MAUI: Making Smartphones Last Longer with Code Offload”, in Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services. ACM, 2010, pp. 49-62

[2] S. Ou, K. Yang, and J. Zhang, “An effective offloading middleware for pervasive services on mobile devices”, Pervasive and Mobile Computing, vol. 3, no. 4, pp. 362-385, 2007

[3] Hsing-Yu Chen, Yue-Hsun Lin, Chen-Mou Cheng, “COCA: Computation Offload to Clouds using AOP”, 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing.

[4] AspectJ crosscutting objects for better modularity http://www.eclipse.org/aspectj/.

[5] Android http://www.android.com/.

[6] R. E. Filman, T. Elrad, S.Clarke, and M. Aksit, “Aspect-Oriented software development”, Addison-Wesley Professional, 2004.

[7] Jeonghun Noh, Mina Makar, Bernd Girod, “Streaming to Mobile Users In A Peer-to-Peer Network”, MOBIMEDIA ‘09, September 7-9, 2009, London, UK.

[8] FFmpeg http://www.ffmpeg.org/.

[9] SDL

http://www.libsdl.org/.

[10] x264

http://www.videolan.org/developers/x264.html

[11] Netty

http://netty.io/wiki/index.html

[12] Mikhail Oparin, Yeongpil Cho, Yongin Kwon, Kwangman Ko, Yunheung Paek, “Cross-Platform Application for Multimedia Data Playback Optimization”, Korea Information Processing Society Conference, 2013

Abstract

In these days, video games and Internet video streaming becomes more and more popular among mobile device users. This tendency can be an obstacle for media servers needing to convert video data for heterogenous range of mobile devices used nowadays. Constant need to convert video data to fit each type of user's device may cause media server overload. The picture becomes even worse after observing the forecast of video streaming demand increase by 42% of laptop users, 40% of tablet computer users, and 39% of smartphone users. To contend the impending issue of possible media server overload a concept of offloading technique based on mutual work of SorMob offloading framework and SorTube application is introduced. It allows to shift the burden of transcoding from centralized media servers to local cloud servers and, therefor, to unload media servers for other tasks. Additionally, it decreases a mobile device energy expenditure because of need to download a video of quality higher than the device actually can view. Using this approach we show in the experiments that it is viable and indeed can be a solution for the problem.

Keywords: Cloud Computing, Android, AOP, Video Transcoding, FFmpeg, Netty, Offloading, x264, SDL