Bachelor Thesis

CROSS-PLATFORM REMOTE CONTROL OF WEB-BASED MEDIA

Author:

TIM BENEDICT JAGLA

19th June 2014

Advisors:

Prof. Dr.-Ing. Frank Ortmeier *
M.Sc. Marco Filax *

* Department of Distributed Systems
Jagla, Tim Benedict:
Cross-Platform Remote Control of Web-Based Media
Bachelor Thesis, Otto-von-Guericke-University Magdeburg, © 2014
ABSTRACT

A multitude of network and media related topics will be discussed in this work. From the basics of networking and different web communication techniques as well as media preparation and provision a concept will be derived to solve the problem of the general lack in web standard compliant browsers for mobile devices within the domain of web applications. A prototypical implementation will be presented and critically examined.
ACKNOWLEDGEMENTS

I wish first of all to thank my advisor Prof. Dr.-Ing. Frank Ortmeier for the idea and input to write a thesis about remote browsing and therefore home entertainment appliances, as well as his support and trust. Furthermore my second advisor M.Sc. Marco Filax for his valuable aid and comments during the writing of this work. My parents, for their continuous support and patience. Finally, thanks to my loving girlfriend Sarah, for her wonderful support in every possible aspect.
LIST OF FIGURES

Figure 1  Usage Scenario ................................. 3
Figure 2  Web Communication Techniques ............... 13
Figure 3  WebRTC Architectural Models .................. 14
Figure 4  Adaptive Streaming - Transcoding .............. 17
Figure 5  Adaptive Streaming - Scalable Encoding .......... 17
Figure 6  Adaptive Streaming - Stream Switching .......... 18
Figure 7  Video Frame Types ............................... 19
Figure 8  Android Version Distribution ..................... 24
Figure 9  YUV Colour Model ................................. 27
Figure 10  Experimental Environment ....................... 45
Figure 11  MPEG-1 Compression Artefacts .................. 47
Figure 12  Test Scenarios .................................. 48
Figure 13  Quality Evaluation ............................... 49
LIST OF TABLES

Table 1     Transport Layer Protocols Comparison . . . . . . . . . . 9
Table 2     H.264/AVC Profile Comparison . . . . . . . . . . . . . . . 20
Table 3     Media Container Comparison . . . . . . . . . . . . . . . 23
Table 4     Android Media Format Comparison . . . . . . . . . . . . 24
Table 5     Mirroring Approaches . . . . . . . . . . . . . . . . . . . 31
Table 6     Decoding & Presentation Timestamp . . . . . . . . . . . 36
Table 7     Experimental Devices . . . . . . . . . . . . . . . . . . . 46
# Table of Contents

**Abstract** iii

**Acknowledgements** v

**List of Figures** vii

**List of Tables** ix

1 **Introduction** 1
   1.1 Motivation 1
   1.2 Goals 3
   1.3 Scope & Limitations 4
   1.4 Audience 4
   1.5 Outline 5

2 **Background** 7
   2.1 Internet Protocol Suite 7
      2.1.1 Transport Layer 7
      2.1.2 Application Layer 9
   2.2 Web Communication 11
      2.2.1 Brief History of Web Communication 12
      2.2.2 Web Real-Time Communication 13
   2.3 Streaming 14
      2.3.1 Traditional Streaming 15
      2.3.2 Progressive Download 15
      2.3.3 Adaptive Streaming 16
   2.4 Encoding & Decoding 18
      2.4.1 Video 18
      2.4.2 Audio 20
      2.4.3 Container 21
   2.5 Android Operating System 23
      2.5.1 Android Media Support 23

3 **Related Work** 25
   3.1 Original Object Transfer 25
   3.2 Direct Mirroring 26
      3.2.1 Pixmap Transfer 26
      3.2.2 Streaming 27
   3.3 Indirect Mirroring 28
   3.4 Hybrid Mirroring 29
   3.5 Remote Control 30
   3.6 Remote Execution 30
   3.7 Summary 30

4 **Concept** 33
   4.1 Requirement Analysis 33
   4.2 Design Concept 34
   4.3 Implementation Concept 35
      4.3.1 Challenges 35
# Table of Contents

4.3.2 Client & Server Architecture ........................................ 37
4.3.3 Communication ...................................................... 38
4.4 HTTP Adaptive Streaming ................................................... 38
4.5 Summary ............................................................. 39

5 IMPLEMENTATION ........................................................ 41
5.1 Client ........................................................................ 41
5.1.1 Media Playback ......................................................... 41
5.2 Server ....................................................................... 42
5.2.1 Media Preparation ....................................................... 42
5.2.2 Media Provision ........................................................ 43
5.2.3 Enabling Remote Control ............................................. 44
5.3 Summary ............................................................. 44

6 EVALUATION ................................................................. 45
6.1 Experimental Environment ............................................. 45
6.2 Quality Metrics ........................................................... 46
6.3 Testing ....................................................................... 47
6.3.1 Test Scenarios .......................................................... 47
6.3.2 Benchmarks ............................................................. 48
6.4 Summary ............................................................. 48

7 CONCLUSION ............................................................... 51
7.1 Discussion ............................................................. 51
7.2 Future Work ........................................................... 51

LIST OF ACRONYMS .......................................................... 53

BIBLIOGRAPHY ................................................................. 57
In times where mobile devices get increasingly small, affordable and rich in applications it is unsurprising they are the fastest growing segment in computer industry [GI14b]. Gone are the days where mobile phones fulfil the sole purpose of communication at any time and anywhere. Besides the safety benefit in emergency situations they provide access to internet services over rapidly evolving wireless and broadband connections. In 2013 the average mobile network connection speed more than doubled to 1387 Kbps reinforcing the growth of global mobile data traffic by 81% [VNI14].

A mobile device is characterised as smart if the locus of control and user interface resides within the device, it is multifunctional and features dynamic service discovery [Pos09]. While smart devices are far from the norm on a global scale with a share of about 25% in contrast to non-smart devices, their percentage is forecast to surpass the latter between 2017 and 2018 [VNI14]. Meanwhile the increasing diversity of smart devices and applications enlarge their operative range from basic communication and information acquisition, over entertainment up to remote access of other platforms. In this context a platform will be defined in a general sense by its intrinsic set of constraints on the hardware architecture, toolchain and Operating System (OS) [Keu+00].

PocketWeb is the name of the first mobile web browser developed for a Personal Digital Assistant (PDA) [GK95; LG97]. Ever since its release in 1994, mobile browsers converged towards their desktop equivalent in terms of usability and adherence to web standards. Nevertheless, due to the diversification process in web technology there is no ‘one size fits all’ browser solution for mobile devices. Today’s internet became a hardly dispensable part of workaday life. It offers value in terms of pastime, everyday planning, shopping or work and forms a piece of the daily routine for many people, as the access to the web increases [ICT14]. Primarily due to the amplified human mobility in developed countries [HDR] mobile devices serve growing parts of their population as constant companions. Since always at hand they already play an essential role in the home infrastructure as a convenient and multi-purpose communication and entertainment device.

1.1 MOTIVATION

Generally the modern household distinguishes oneself by the variety of end-user devices. Interconnected in a home network, accompanying its heterogeneity, the major challenge for the end-user is complexity management as well as diverse capabilities. Viable solutions to this problem could be the unification of use patterns and the shift and centralisation of responsibility. Nevertheless, a comprehensive and mutually consistent set of user actions across a variety of devices with different input methodologies and OSs is hardly realisable, necessitating a central control device solution. Therefore
the centralised concept, using a always at hand smart device as the remote control for a standard compliant web browser is elaborated in the course of this thesis, from basic backgrounds and circumstances up to a prototypical implementation.

The compliance to web standards is particularly important. Every website should be conformable to function efficiently and display correctly on access. Of course, the same goes for the type of applications exercising the standards, browsers. Even though multimedia content is often authored for multiple platforms by service providers to generate as much value as possible, full feature support can only be guaranteed on a subset of devices. On top of that, the necessary infrastructure in a so called Content Delivery Networks (CDNs) requires modern hardware to cope with coding and distribution complexity plus massive investments to maintain it. Therefore the reduction of target devices is generally desirable, to lower the cost and increase service quality.

Remote control of web-based media, remote browsing in short, follows the concept of centralising responsibility onto the most standard compliant device to provide the best possible user experience for the end-user. Usually a variety of mobile devices can benefit from outsourcing their browsing responsibility. For the battery it is a thin line between increased communicational complexity and savings due to remote execution of complex computations. Despite its universal application in everyday routine the general topic is an open area for interdisciplinary discussion and research.

As a common denominator one has to conceptualise the notion of view, controller, content provider and a session in terms of remote browsing. Basically a view enables the user a visual representation of the involved web-based media. Controllers also require view capabilities for line of sight independence of the view and a way to remotely control the media content on the provider with their intrinsic input scheme. Lastly a session is tied to each device respectively. It usually spans from the connection to the remote browsing service provider till the disconnection of the device. On the content provider’s side the session is equivalent to a browser session.

Next to the general benefits of remote browsing, two other use cases motivate further research. In education or a business environment the collaboration potential of remote browsing is promising. Usually workplace team meetings as visible in figure 1 share the general concept with lectures or seminars in academia. While the information is commonly conveyed via a projector or large screen the focus rests on a single individual, the presenter. In remote browsing the team members can connect to the main screen and interact, notify others of a problems or communicate their opinion, particularly if the corporations has the creativity and expertise to adopt to modern browser based presentations and infrastructure. Moreover, it is easy to share the experience remotely if anyone is unable to attend. Last but not least, forgetting a note or bookmark made in the browser of ones office PC will be mitigated by the always available remote access.

The home entertainment industry is a large market with a lot of monetisation opportunities. Surfing the web is an essential requirement in the entertainment landscape. Today’s TVs are rarely shipped without a browser anymore, while dedicated High Definition Multimedia Interface (HDMI) sticks expand the feature set of enabled devices with such functionality. Nevertheless, many existing remote browsing solutions either compete with the local network or
are adversely affected by slow screen updates and high input lag. Lastly they drain the battery life of a mobile controller device. Alongside those developments and maturing web technologies, particularly mobile browsers cannot keep up with web standards. For this reason the existence of the browser is plainly not enough and capabilities have to be extended by remote browsing.

1.2 GOALS

Most of the goals of this thesis are motivated by the variety of advantages and use scenarios according to section 1.1. To provide an efficient and utmost barrier free browser to mobile devices is the highest goal. For the purpose of later success assessment of the prototypical implementation there are a few general criteria to meet:

Firstly the user has to be able to display and control modern asynchronous web applications directly on a mobile device, surpassing the limits induced by the constraints and limitations of its respective platform. Furthermore a controller should be detachable to spare its energy and preserve power, without causing an interruption of the browser session by the content provider. Last but not least a controller should benefit from its strengths in terms of input methodology or sensors.
Mobility mandates the substitution of inflexible wired infrastructure for wireless communication in constantly changing network topologies, increasing computational complexity and power consumption [Per+09]. Furthermore advanced applications and permanent background services strain the already limited power supplies. Because modern battery technology diminishes over lifetime, per charge and is continuously outpaced by computational hardware capabilities it is the primary bottleneck for mobile devices [Rav+08]. Although capacity is proportional to size there is nothing to be gained without sacrificing portability because batteries are already the single largest source of weight in a mobile device [FZ94]. Thus power consumption needs to be taken into account on the application layer.

Depending on a wireless infrastructure is both an opportunity and challenge. Because most of the structural elements of a fixed network like cables and switches are virtually obsolete the user regains flexibility. Complexities of long term planning are mitigated and subsequent changes to the network architecture remain simple. Furthermore the low acquisition cost of a wireless infrastructure leads to a higher cost efficiency. Wireless Local Area Networks (WLANs) are an important part of the computer industry and a domain of major research and cooperation. While 802.11n already enables many mobile devices transmission rates of up to 600 Mbps one of its successors 802.11ac promises 1 Gbps, with a periodic upwards trend [Van11].

1.3 Scope & Limitations

Indicating scale and highlighting shortcomings is the general purpose of this section. Due to the sheer scale of the topic remote browsing there are too many areas to touch to go into the specifics of everything.

Although they are important topics regarding smart devices, NAT traversal and service discovery will not be covered. Media is dealt with in a general overview of applicable codecs and formats to a remote browsing solution. State of the art codecs like H.265 are not considered due to the lack of open source implementations with comparable performance to their predecessors. Moreover the implementation falls a bit short in some aspects. It provides no scalability, although the implications of multi-user access are theoretically discussed. Furthermore the prototype achieved a seemingly low quality outcome, cutting the evaluation potential even further. One of the highlights of this thesis are the descriptions of related work. It tries to categorise the various approaches of remote browsing in a comprehensible manner and exposes interesting correlations.

1.4 Audience

The primary target audience of this thesis are software engineers and researchers with a general interest in the topics of remote control and screen sharing. A major part of it is a survey of technology, the current state of the art in remote browsing and their predecessors. Existing and generally possible approaches as well as their respective advantages and disadvantages are made a subject of discussion. A fundamental knowledge of networking
and digital media is desirable but not necessary. Furthermore hobbyists and android developers will be able to cherry-pick the latest issues and practical information utilised in the development process.

1.5 OUTLINE

This section enlists the structure of this thesis to provide the reader with a general overview and find chapters of interest.

CHAPTER 1 (INTRODUCTION) gives an overview of the topic and the motivation for further research.

CHAPTER 2 (BACKGROUND) explains related and important background information for a deeper understanding of the general topic. Fundamental comparisons and descriptions help to comprehend later reasoning and conclusions and give an overview of the broad spectrum of web technology.

CHAPTER 3 (RELATED WORK) examines prior and similar research. Furthermore some existing open source and commercial solutions are described and explored for possible advantages and their respective disadvantages.

CHAPTER 4 (CONCEPT) comments on the design decisions made about the prototypical implementation and justifies their reasons in the context of alternative approaches.

CHAPTER 5 (IMPLEMENTATION) highlights the essential application logic as well as problems in the course of the implementation.

CHAPTER 6 (EVALUATION) explains the test cases and reports on their outcome. Moreover it describes the evaluation metrics and outlines possible problems or measuring inaccuracies critically.

CHAPTER 7 (CONCLUSION) summarises the lessons learned, reviews results and draws a conclusion. Beyond the corollary this chapter covers potential future work.
Chapter 2 | Background

In this chapter the basic knowledge about a variety of topics regarding remote browsing is collected. From low level transport layer protocols in section 2.1 over the various web communication techniques building up on their concepts in section 2.2. Thereafter, in section 2.4, the specifics of media coding and formats are described.

2.1 Internet Protocol Suite

A basic understanding of the Internet Protocol (IP) [Pos81a] suite is mandatory to reason about networking in general as well as web communication techniques. Especially the respective advantages and disadvantages performance wise are sometimes only deductible from the protocols specification and need to be discussed. This sections examines only small parts of the two upper layers of the TCP/IP four layer model in sections 2.1.1 and 2.1.2. The others below are out of the scope of this thesis and not further discussed as their influence not improvable or even influenceable from an end-user application.

2.1.1 Transport Layer

The transport layer of the internet protocol stack provides end-to-end services that are independent of the structure of user data and the logistics of the exchange. Its responsibilities are message transfer, segmentation, error-, flow- and congestion control as well as addressing. Transmission at the transport layer can be categorised as either connection-oriented (see section 2.1.1.1) or connectionless (see section 2.1.1.2) [Bra89].

2.1.1.1 Transmission Control Protocol (TCP)

Introduced in May 1974 by the Institute of Electrical and Electronic Engineers (IEEE) [CK74] and standardised in December by the Internet Engineering Task Force (IETF) [CD874] the Transmission Control Protocol (TCP) [Pos81b] is one of the core protocols of the IP suite. It provides reliable, ordered and error-checked delivery of a stream of octets, grouped into segments, between connected devices. Many firewalls are configured for the common TCP ports [IANA] which contributes to the avoidance of user errors.

For real-time applications with low latency requirements the reliability of TCP becomes a disadvantage. Packets are re-sent until received, meaning that more recent data is inaccessible coming after a lost packet until the retransmitted copy of the lost data is received. Beyond this inherently increased latency the usable packet size, although specified to a maximum
of 1500 B [Hor84], is reduced by the headers of IP and TCP as well as the Point-to-Point Protocol (PPP) [Sim92] for access over the widespread Digital Subscriber Line (DSL). As shown in equation (1) only 1452 B of payload remain available, which corresponds to an efficacy of 96.8%. Using the optional timestamps with a size of 10 B [JBB92] even less: \( \sim 96.1\% \).

\[
1500 \text{ B} - 20 \text{ B}_{IP} - 20 \text{ B}_{TCP} - 8 \text{ B}_{PPP} = 1452 \text{ B}
\] (1)

TCP is limited to unicast transmission, meaning the delivery to a single network destination identified by a unique address.

2.1.1.2 User Datagram Protocol (UDP)

Applications that do not require the reliability of a TCP connection may instead use the connectionless User Datagram Protocol (UDP) [Pos80], which emphasises low-overhead operation and reduced latency rather than error checking and delivery validation. It neither provides virtual circuits, nor reliable communication, delegating these functions to the application program. UDP packets are called datagrams, rather than segments.

In contrast to TCP, UDP supports multicast, which is advantageous for local deployments with a multitude of concurrent devices. Furthermore the minimalistic UDP header of 8 B described in equation (2) enables 1464 B of payload with an efficacy of 97.6% and thus savings of 0.8% per network packet compared to TCP with a header size of 20 B.

\[
16 \text{ b}_{src} + 16 \text{ b}_{dst} + 16 \text{ b}_l + 16 \text{ b}_c = 8 \text{ B}
\] (2)

where the indices indicate the information hold by the respective bits. 16 \text{ b}_{src} and 16 \text{ b}_{dst} contain the source and destination port number while 16 \text{ b}_l encodes the length of the UDP packet including everything from header to data. Finally 16 \text{ b}_c is the checksum for an integrity check over the whole length.

2.1.1.3 Lightweight User Datagram Protocol (UDP Lite)

For the purpose of low latency transmission the Lightweight User Datagram Protocol (UDP Lite) [Lar+04] allows erroneous data to be delivered rather than discarded. Therefore it is well suited for error-prone network environments such as local area network. Often the assumption of accepting a packet with a damaged payload being better than receiving no packet at all is valid, especially in multimedia.

UDP Lite enables the user to specify the checksum coverage, the number of bytes of the packet which should be verified for integrity. For example, a checksum coverage of 24 would force an integrity check for the default 8 B of the header and the first 16 B of the payload. Packets below the size of 24 B would be covered entirely by the checksum. Regardless of the differences, UDP Lite uses the same standardised checksum algorithm as TCP and UDP with equal performance [BBP89].
2.1.1.4 Summary

In summary, it can be said that TCP is the preferable protocol where reliable communication without custom flow control logic within the application layer is required. Hence it still is the most widespread protocol on the internet, despite its induced overhead. UDP is free of this overhead but therefore lacks in convenience functionality. The checksum algorithms of these two protocols are the bottleneck for streaming purposes, which ratifies UDP Lite in the list of important protocols to keep in mind. Table 1 contrasts the results of this section directly.

Table 1: Comparison of the already described transport layer protocols.

<table>
<thead>
<tr>
<th></th>
<th>TCP</th>
<th>UDP</th>
<th>UDP Lite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header Size</td>
<td>20 B</td>
<td>8 B</td>
<td>8 B</td>
</tr>
<tr>
<td>Transfer</td>
<td>stream</td>
<td>packet</td>
<td>packet</td>
</tr>
<tr>
<td>Checksum Coverage</td>
<td>complete</td>
<td>complete</td>
<td>custom</td>
</tr>
<tr>
<td>Connection</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Ordering</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Error Recovery</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Flow Control</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Congestion Control</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

2.1.2 Application Layer

The application layer includes protocols used by most applications for providing user services or exchanging application data over the network connections established by lower layer protocols [Bra89].

Protocols on the transport layer (see section 2.1.1) are largely independent of the specifics of application layer protocols. Notwithstanding it is sometimes necessary for Network Address Translator (NAT) traversal to consider the payload within the application layer.

2.1.2.1 Hypertext Transfer Protocol (HTTP)

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems and is also used as a generic protocol for communication between user agents and proxies or gateways to other Internet systems [Fie+99]. Most notable the standardisation of the HTTP/1.1 standard, a version of HTTP in common use, took place in 1999 and was coordinated by the IETF as well as the World Wide Web Consortium (W3C). Though cookies may be used to make it stateful [Bar11], HTTP is considered a stateless protocol. Additionally HTTP/1.1 may use persistent connections as a performance improvement [Fie+99].

While the specification of HTTP/1.0 [BFF96] defines exclusively the GET, POST and HEAD methods the HTTP/1.1 specification supplements five new methods: OPTIONS, PUT, DELETE, TRACE and CONNECT. Methods are categorised by safety and idempotence:
1. a method is safe if it does not have the significance of taking an action other than retrieval:
   GET, HEAD

2. a method is idempotent if the side-effects of \( n > 0 \) identical requests is the same as for a single request. However it is possible that a sequence of idempotent requests is non-idempotent [Fie+99]:
   GET, HEAD, PUT, DELETE and usually OPTIONS, TRACE, depending on the implementation.

Because of the concrete specification the semantics of HTTP methods can be depended upon. A client may use anyone in the request and the server may support any combination of methods. Unknown methods be treated as an unsafe and non-idempotent. Usually an error message in such a case. Although the specification presumes an underlying reliable transport layer (see section 2.1.1) protocol and TCP is in common use HTTP may take advantage of unreliable protocols like UDP, for example in service discovery.

One HTTP session is defined as a sequence of network request-response transactions. The HTTP client initiates a request by establishing a TCP connection to typically port 80 [IANA] on a server. Afterwards the HTTP server listening on that port waits for a clients request message.

2.1.2.2 Real-Time Transport Protocol (RTP)

The Real-Time Transport Protocol (RTP) defines a packet format for delivering multimedia streams over IP networks and is regarded as the primary standard for this purpose [Per03]. It was standardised by the Audio/Video Transport (AVT) working group within the IETF in 1996 and superseded in 2003.

RTP is an end-to-end, real-time protocol suitable for both unicast and multicast network services. Although TCP is standardised for use with RTP the majority of implementations are built on top of UDP, because real-time multimedia streaming requires timely delivery and is less error prone to packet loss, especially with suitable error concealment algorithms [Per03]. Thus RTP is widely used for distributing media in the case of Internet Protocol Television (IPTV), as the Internet Service Provider (ISP) can control the amount of multicast traffic and gains from the scaling which multicast offers [PJ09].

The specification of RTP [Sch+03] describes two sub-protocols:

1. RTP, the data transfer protocol, provides meta data and the actual real-time data. It describes the format of the payload, sequence numbers for the detection of packet loss or reordering and timestamps for synchronisation [Per03].

2. Real-Time Transport Control Protocol (RTCP) specifies quality of service feedback and synchronisation between media streams. Nevertheless it neither ensures timely delivery nor guarantees quality of service or in-order delivery. More recent and specialised protocols like the Datagram Congestion Control Protocol (DCCP) [KHFo6] exist for this purpose. Because of the more diagnostic nature of the RTCP protocol its traffic bandwidth compared to RTP is around 5% [PD11].
Optionally RTP can be used in conjunction with a signalling protocol like the Session Initiation Protocol (SIP) [Ros+02], to negotiate the ports which form an RTP session, or one for media description such as the Session Description Protocol (SDP) [HJP06].

Commonly a session is composed of nothing but one stream of a particular media type. This ensues separate RTP sessions for audio and video, enabling the receiver to ignore unusable stream types [DJ05]. Typically the size of a RTP packet is 1452 B [Zam09] (compare section 2.1.1.1). For a media stream encoded at one megabits per second (Mbps) each packet carries approximately 11.6 ms of media:

\[
\frac{1 \text{ Mb/s}}{125000 \text{ B/s}} = 86.1 \text{ ms/s}
\]

\[
\frac{1452 \text{ B}}{\frac{86.1 \text{ ms/s}}{\text{s}}} = 0.0116 \text{ s} = 11.6 \text{ ms}
\]

Therefore the loss of a single packet does not influence the visual perception in an exaggerated manner [Ste96], especially if the frames are interleaved to distribute occurring damage over time [Per98].

2.1.2.3 Real-Time Streaming Protocol (RTSP)

The Real-Time Streaming Protocol (RTSP) is a network control protocol which provides an extensible framework to establish and supervise media sessions between end points. It was developed by RealNetworks, Netscape Communications and Columbia University in 1996 [Oss99]. In 1998 the Multi-Party Multimedia Session Control (MMUSIC) working group of the IETF standardised it.

As stated in the specification of RTSP [SRL98] it is not responsible for the transmission of media data and relies on different, out-of-band protocols like RTP for that purpose. While HTTP is stateless, RTSP is stateful and uses TCP to maintain an end-to-end connection where both client and server can issue requests. These requests can be performed in three different ways:

1. persistent connections used for several request/response transactions
2. one connection per request/response transaction
3. connectionless mode

Some popular RTSP implementations are the commercial Helix Universal Server\(^1\), the open-source fork of QuickTime Streaming Server called Darwin Streaming Server\(^2\) or ffserver as part of FFmpeg\(^3\).

2.2 WEB COMMUNICATION

The interchange of information over signals or messages is summarised as web communication. One typical message exchange pattern is the well
known request and response cycle. While a client has the active role of querying for content the server reacts and responds accordingly.

Exchange is distinguished between synchronous and asynchronous. Former waits for a timely message delivery or a time out before closing the connection, while latter supports delivery at an unknown later time besides staying responsive.

2.2.1 Brief History of Web Communication

Elementary web architecture semantics are based on a synchronous client-server paradigm. Clients send a HTTP request for content to the web server, which replies with a response containing the requested information. It will be provided by the Uniform Resource Locator (URL) referencing the network location of an entity. Sampling the state of a server exclusively by request of the client is called polling and is a synchronous activity. Emerging from the need for interactivity and dynamic content in a responsive web a variety of approaches to web communication enabled ongoing client-server transactions since regular HTTP [LR14].

In such a web application scenario, the server can embed some JavaScript (JS) code into the Hypertext Markup Language (HTML) web page response before returning it to the client. The standard JS APIs is of avail for code to browser interaction and users may additionally influence it through their user interface. A JS object known as XMLHttpRequest allows to fetch more data from the server without downloading a whole other web page.

Asynchronous JavaScript and XML (AJAX) is a concept introduced by Garrett in an essay [Gar05] which was rapidly adopted in software and JS frameworks. Plain AJAX is comparable to HTTP polling, because each call initiates a single request and response cycle and the established connection is closed afterwards. More efficient methods based on the concept are explained below and shown in figure 2.

Each of the following dynamic techniques begins its communication by a regular HTTP web page request that will not be mentioned again:

**AJAX polling** (see figure 2(a)) executes the embedded JS on the client side to request a file from the server at regular intervals, for example one time per second. The server processes each request, calculates the response and transfers it to the client as in usual HTTP traffic. Despite the obvious disadvantage this method is widely utilised for compatibility reasons [BMD07].

**AJAX long-polling** (see figure 2(b)) is basically the same as simple polling. The only difference is that the connection stays open until a time out occurs. Meanwhile the server is on standby to respond with information when there is new available. Upon receipt of a response the client immediately makes another request to the server to restart the process [BMD07].

**HTML5 SSE** (see figure 2(c)) transfers events from the server to the client as soon as they happen. For Server Sent Events (SSE) it is recommended to run an asynchronous event loop on the server to stay responsive and real-time. It is not possible to connect to a SSE deployment with a server from another domain [Hic12a].
**HTML5 WebSocket** (see figure 2(d)) also profit from an asynchronous event loop on the server side. WebSockets enable bidirectional traffic from server to client and the other way round. Packets will be exchanged in a proper communication protocol if new data is available on either side. It is possible to connect with a server from another domain [FM11].

**Plug-in Communication** as a general term for a browser extension providing web communication capability has the disadvantage of being proprietary and an installation requirement in itself. Although some implementations are exceptionally fast for a specific purpose, e.g. the raw sockets of Adobe Flash\(^5\), the lack of native support and therefore platform dependencies are a major disadvantage.

**WebRTC** is a modern and also transport-agnostic way to establish a communication path. Clients may exchange data over Peer-To-Peer (P2P), independently from each other and a centralised server. WebRTC will be explained in detail after this section.

### 2.2.2 Web Real-Time Communication

Initiated by Google, Web Real-Time Communication (WebRTC) is an Application Programming Interface (API) specification for real-time multimedia browser communication without proprietary plug-in or installation requirements [Alv14]. Since open source release in May 2011 the W3C and the IETF are jointly defining the JS APIs and the underlying protocols for the establishment and the maintenance of a reliable communication channel across any pair of next-generation web browsers and platforms.

\(^5\) [URL: http://www.adobe.com/products/flashplayer.html]
Goal of this standardisation effort is a common WebRTC API that enables secure access to input peripherals such as microphones and webcams on any device as well as the exchange of real-time media and data with a remote party.

Inspired by the scheme of the SIP trapezoid in Rosenberg et al. [Ros+02], WebRTC essentially extends its client-server principle with P2P communication between browsers. Figure 3(a) shows two clients running a web application, each requested from a different server. This architecture of distributed server infrastructure is very flexible and extensible but less common than the architectural triangle in figure 3(b) where both clients establish a connection with the same server.

Although WebRTC enables direct P2P communication, the exchange of session description information, also referred to as signalling, still requires servers to cope with NAT traversal and firewalls. Foremost a session description specifies transport and Interactive Connectivity Establishment (ICE) information and the media type, format and configuration parameters to successfully establish a media path. WebRTC primarily specifies the media plane while keeping the signalling in the application layer to satisfy different purposes in distinct applications.

Due to the fact that signalling is considered a part of the application it is not standardised within WebRTC but usually transported by HTTP or WebSockets for client-server communication. Servers correspond over proprietary signalling or standard protocols such as SIP or Jingle [LR14].

After establishing the media path with the help of the signalling plane, data may flow directly between browsers via P2P without an additional indirection.

2.3 STREAMING

Streaming is the process of constantly delivering media for immediate representation to an end user. Examples for inherently streaming systems are radio and television. Popular deployments providing live streaming over the internet are based on RTP/RTSP or HTTP.
Three general delivery methods will be distinguished and discussed to get a grasp of potential and problems in the domain of streaming:

1. Traditional Streaming (Section 2.3.1)
2. Progressive Download (Section 2.3.2)
3. Adaptive Streaming (Section 2.3.3)

They differ in computational overhead, complexity and actuality in ascending order.

### 2.3.1 Traditional Streaming

Traditional streaming requires a stateful protocol which establishes a session between the service provider and client [Lee05]. Because of the statefulness of the protocol the server is able to track of the clients state from the first connection until the disconnection.

RTP (see section 2.1.2.2) and RTSP (see section 2.1.2.3) are frequently used to implement a traditional streaming service. On a successfully established session the server begins to send media as a continuous stream of RTP network packets at the bit rate at which the media is encoded. A client may issue commands like the self explanatory PLAY and PAUSE or the less verbose TEARDOWN, to close the session, via RTSP.

Traditional streaming systems use a client buffer to absorb channel variation and to facilitate the utilisation of error-recovery schemes [SZ02]. As the client buffer gets full the server stops the transfer. Thus a client never fetches more data than its buffer size from a paused stream. Though the buffer size depends on implementation specifics of the client a common default is 5 seconds of media which is unsuitable for real-time streaming [Zam09].

### 2.3.2 Progressive Download

Progressive download is a technique to transfer digital media files from a server to a client which is widely adopted due to its simplicity. For a common deployment by using the HTTP protocol, nothing more than a server with HTTP/1.1 [Fie+99] support is required.

On the client side media is downloaded to a physical drive as a temporary file. Since the introduction of progressive download playback it is possible to start the media output before the download is complete, as long as specific requirements are met. Content creators may issue the encoder to embed a specified amount of buffer into the file and additional buffer settings are imposed by the client-side media player.

The key difference between traditional streaming (compare section 2.3.1) and progressive download is in how the digital media data is received and stored by the accessing end user device. Stockhammer summarises the disadvantages of progressive download as follows [Sto11]:

1. already transferred and buffered bandwidth may be wasted if the user decides to stop watching the content after progressive download has started, for example by switching to another content
2. no support for bit rate adaptation, since every client is considered equal in terms of available bandwidth
3. no support for live media sources

While its wasted bandwidth and the lack of bit rate adaptation are secondary, the missing support for live media sources is a definitive criterion for the exclusion of progressive download as the back end of a remote browsing approach.

### 2.3.3 Adaptive Streaming

A modern approach to stream delivery is adaptive streaming. It takes the end users available resources such as bandwidth and Core Processing Unit (CPU) into account in order to continuously adjust the quality of the streaming media [Lee05]. Required is either a specialised encoder able to provide video at multiple bit rates or a multitude of encoders and can be deployed within CDNs to provide improved scalability [Sto11]. That way adaptive streaming ensures to offer the highest possible quality under the given users circumstances without a need for surplus interaction.

Adaptive streaming enables multimedia devices of diverse capabilities and formats to exchange video content on heterogeneous network platforms. One related scenario is delivering a high-quality multimedia source to various mobile devices on wireless networks.

Techniques to adapt the video sources bit rate to variable bandwidth are organised into the categories transcoding (in section 2.3.3.1), scalable encoding (in section 2.3.3.2) and stream switching (in section 2.3.3.3).

#### 2.3.3.1 Transcoding

Transcoding is on-the-fly encoding at a desired bit rate to match the available bandwidth and format of a client. Therefore it might not introduce additional functional requirements in the decoder. Conversion is performed by a transcoder which can generate the appropriate bit stream directly from the original source without having to decode and re-encode.

One main objective of transcoding is reduction of the download delay of media content over low-bandwidth networks [SML98]. Furthermore it provides video format conversion to enable content exchange with computationally constrained or limited-display client devices [Ahm+05].

An important disadvantage of this approach is the decreased scalability since transcoding needs to be performed for each client respectively. Thus it is a procedure which yields high processing and deployment costs. The encoding process is required to be performed on appropriate, dedicated servers in order to be deployed in a CDN [CMP11].

Figure 4 visualises the concept of transcoding. The data fed into the transcoder will be transmitted in a format suitable for adaptive streaming.
2.3.3.2 Scalable Encoding

Scalable encoding takes advantage of temporal, spatial and quality scalability of video material and enables the transmission of partial bit streams while retaining a high decoding quality [SMW07]. As a result the resolution and frame rate of the extracted bit stream stays adaptive without any further transcoding by the provider [ISO12] which leads to a notable decrease in processing costs because of the gain in coding efficiency.

Nevertheless, deployment of a scalable encoding solution into CDNs is complicated because specialised servers are required to implement the adaptation logic [CMP11]. Additionally it is limited to a set of scalable codec standards such as H.264/AVC (see section 2.4.1.2) and thus codec dependent. Figure 5 outlines the general process of stream switching.

2.3.3.3 Stream Switching

Stream switching addresses adaptive streaming by encoding source material at several different bit rates, generating n variants of the same content also known as video levels as shown in figure 6. The scheduler or, in more general terms, controller dynamically chooses the video level matching the end users available bandwidth. Changing conditions cause switching to different video levels for highest possible quality continuous playback.

Once the preprocessing of generating the video levels is done the real-time requirements are low and it is easy to be deployed on a CDN [CMP11]. Unlike scalable encoding this approach is codec agnostic so that it does not depend on specific codec formats. However, because of the video level encodings at different bit rates of the same material the storage cost is considerable.
Another disadvantage of this approach is the coarse granularity since there is only a finite set of video levels.

![Diagram of stream switching approach to adaptive streaming](image)

**Figure 6:** The stream switching approach to adaptive streaming. Adapted from [CMP11].

Figure 6 illustrates an example of the stream switching process over time. It is simplified by assuming that switching operations are only performed after a complete segment, every one of which has the same duration. In practice, segment durations as well as the number of video levels are flexible and purposeful design choices. Examples for existing solutions are HTTP Dynamic Streaming (HDS)[6], IIS Smooth Streaming [Zam09] and MPEG-DASH [ISO14].

### 2.4 Encoding & Decoding

Coding, which encompasses encoding and decoding, translates information from one format into another. Generally for the purposes of standardisation, speed, privacy or compression.

However, varying availability of coding algorithms cause the necessity of multimedia being encoded into a supported format. Specific requirements of coder implementations or capabilities of target devices are major issues in the distribution of multimedia today.

#### 2.4.1 Video

Although the best practice for high quality video coding is to avoid unnecessary encoding steps entirely it is not always possible to access or capture material in the required format. If encoding becomes inevitable the recommended approach is to encode from a high quality source to a lower grade format to avoid encoding degradation. Essential aspects for video encoding in the best possible quality are:

1. the original source and/or the capture method
2. each additional encoding step

---

3. the intended output

Only by analysing and optimizing those aspects while considering all possibilities the goal can be met. Decoding is primarily dependent on the input, decoder support and processing capabilities.

2.4.1.1 Frame Types

In video coding a frame is one of the many still images which compose a moving picture. Because of general spatial and temporal redundancy in video material and a common lack of storage and bandwidth a diversity of compressed video standards emerged. Their general terms and definitions in relation to frames will be explained here.

The complete data of a frame is only encoded for key frames, specified as Intra Frames (I-Frames). They are least compressible, often only by statistical redundancy, but do not require other video frames to decode.

After a key frame the video coding considers only differences with the preceding frame, resulting in a better compression for the subsequent frames. Hence their predictive nature they are called Predicted Frames (P-Frames). Especially fast paced and dynamic video material with rapidly changing frame information requires more key frames than slower scenes. An example of the relationship between several frames is shown in figure 7.

On a final note, bidirectional encoding profits from the highest compression ratio. Bidirectional Frames (B-Frames) take both previous and successive frame differences combined by a weighted average into account to achieve a better overall compression by compromising processing power.

A union of interdependent frames is called Group Of Pictures (GOP). It consists of one key frame followed by several P-Frames and optionally a B-Frames. The lower its size, the lower the key frame interval meaning there are less I-Frames per time unit of video. More frequent key frames enable to reduce distortion and artefacts while streaming in a lossy environment. However, a low GOP size increases the media size since key frames are the least compressible frame type.

![Group Of Pictures (GOP)](image)

**Figure 7:** Common distribution of the different frame types in a video stream. B-Frames are the rarest type of frame due to their coding complexity. [Rom11]

2.4.1.2 H.264/AVC

H.264 is the successor of H.263 and MPEG-4 Part 9. Developed in 2003 by the ITU-T VCEG together with ISO/IEC Moving Picture Experts Group (MPEG) both committees joined their endeavour under one common title: Advanced
Video Coding (AVC). Most important goals of this standardisation effort have been enhanced compression performance as well as network related optimisations regarding conversational and streaming services [Wie+03].

H.264 is one of the most commonly used formats for recording, compression, and distribution of high definition video and was even adopted by the Blu-ray Disc format for that purpose. Video content providers like Vimeo and YouTube provide their source material in H.264 amongst other formats.

There are currently 21 profiles, in general terms subsets of capabilities in the specification of H.264 [ISO12], each of which oriented at different types of applications and devices.

For instance the Constrained Baseline Profile (CBP) compromises many of the processing intensive robustness features of H.264, followed by the Baseline Profile (BP) and the Main Profile (MP) in increasing order of reliability and complexity. Since no profile more complex than BP is supported natively by Android and iOS they are the most relevant H.264 profiles for cross-platform streaming and broadly used in mobile applications and video conferencing. Table 2 summarises the major differences among these three profiles.

Table 2: Comparison of H.264/AVC profiles and their respective support for a different optimisation technologies.

<table>
<thead>
<tr>
<th></th>
<th>CBP</th>
<th>BP</th>
<th>MP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android Support</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>I- &amp; P-Frames</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>B-Frames</td>
<td>–</td>
<td>–</td>
<td>✓</td>
</tr>
<tr>
<td>Interlacing</td>
<td>–</td>
<td>–</td>
<td>✓</td>
</tr>
<tr>
<td>Flexible Macroblock Ordering</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>Arbitrary Slice Ordering</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>Redundant Slices</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
</tbody>
</table>

One of the more modern coding features added to the H.264 standard is Scalable Video Coding (SVC). It enables the construction of standards compliant partial bit streams as required by scalable encoding described in section 2.3.3.2. Moreover Multiview Video Coding (MVC) offers another bit stream composition, allowing more than one viewpoint for a video scene or stereoscopic video.

2.4.2 Audio

Generally the same rules as for video coding itemised in section 2.4.1 apply to audio, only at a smaller scale due to lower data rates. In the following section some of the most influential audio codecs are introduced. Their respective file extensions are the common ones and not exclusive, because of the codec support by different container formats.

**MP3** (*mp3*) [ISO93; Bra99] was initially developed by the Fraunhofer ISS and later published as a standard under MPEG-1 Audio Layer III in 1993. Its widespread appliance in hard- and software still makes it the de facto standard in digital audio compression for the transfer and playback of music. MP3 employs an efficient lossy compression algorithm
based on perceptual coding, which reduces accuracy of audio in parts beyond the human auditory perception [JJS93]. Although a variety of more efficient coding algorithms emerged since 1993, MP3 is still the most important in audio with a major consolidation by tradition of use.

**Vorbis** (*.ogg) [XOF12] is an open and patent-free, direct competitor to MP3. It is meant as an alternative to replace MP3 due to common issues with patented and restricted formats. Similarly it offers a wide range of lossy coding algorithms and is capable of higher compression factors. Despite its technical superiority, Vorbis never caught up to the public profile of MP3 [Smi10].

**Opus** (*.opus) is an open, cross-platform lossy audio compression specified by the IETF [VVT12]. Because of its low algorithmic delay Opus distinguishes itself from other audio formats. Opus incorporates concepts of two proven codecs, SILK by Skype Technologies’ optimised for speech as well as xiph.org Foundation’s CELT [Val+10] low-latency coding principle. It supports a variable bit rate encoding from 6 Kbps to 510 Kbps, which can adjust seamlessly among the different rates. Irrespective of the focus on latency a study has shown its efficiency in terms of quality per bit rate compared to the other codecs introduced above [Che+11]. In spite of the various patents covering Opus it is free, as they are offered on royalty free terms.

**AAC** (*.aac) or Advanced Audio Coding is extensively supported by a multitude of hardware manufacturers due to a fidelity comparable to MP3 at faster on-the-fly encoding rates and its ongoing integration into the specifications of MPEG-2 [ISO06] and MPEG-4 [ISO09]. Advanced Audio Coding (AAC) employs a modular system that categorises different audio formats into object types. Up to a certain degree the hardware manufacturers or software engineers may cherry-pick components to create a solution which fits their requirements. While AAC is already flexible and efficient since its publication in April 1997, later extensions added further optimisations for low bandwidth environments in terms of elaborated object type combinations into profiles under the common name High-Efficiency AAC (HE-AAC).

### 2.4.3 Container

A container is a meta file format which wraps different kind of data elements into a single file. They are frequently used for cross-platform exchange of data and for multimedia applications. Generally the data wrapped within a container is called the payload.

Basically containers enable to aggregate a variety of file types into a unified format reliably defined by a specification. Through such a common standard an application is able to cherry pick according to its features and ignore remaining data. For instance a video stream with subtitles and additional meta information, usually tags in multimedia applications, may be wrapped into a container. Since only an identifier of the decoding algorithm required to decode the payload will be provided, a client able to identify and open a container might not be able to use the data. This is a common problem with exceptionally versatile container formats like Matroska.
Containers differ primarily in their respective size overhead and codec support, presenting various features and limitations. The most important multimedia containers are briefly introduced below:

**AVI (.avi)** is a widespread container with a long history since 1992. Initially it was not designed to support B-Frames for zero latency utilisation in Video For Windows (VFW). Difficulties arise from MPEG-4 Advanced Simple Profile (ASP) enabling a packed bit stream to interlace B-Frames with their respective future reference. Because this referenced frame must be decoded at first it will be transmitted before. Regardless, decoders of the VFW API are not intended to read future frames which is the reason for H.264 abandoning support for VFW and therefore AVI as a container in 2006, deeming it irrelevant for a remote browsing solution. Moreover there is no equivalent to packed bit stream in the MPEG-4 AVC standard.

**MP4 (.mp4)** is a popular container format defined in the MPEG-4 Part 14 standard. It supports almost any kind of media data. Typically an MP4 container contains video and audio streams encoded with H.264 and AAC, respectively.

**3GP (.3gp)** is widely utilised on 3G mobile phones. 3GP is defined as an extension of MPEG-4 Part 12.

**MPEG TRANSPORT STREAM (.ts)** is defined in the MPEG-2 Part 1 standard and mostly used in digital television broadcast systems.

**MATROSKA (.mkv)** is an extensible, open, royalty-free container format. Playback profiles or precisely defined subsets of it are still missing in the standard. Thus the IETF never proposed a common internet media type [FKH13] for Matroska. Streaming per RTP is discouraged because of redundancy as the container already provides the same timing and channel mechanisms7.

**WEBM (.webm)** is an open, royalty-free container format structurally based on the Matroska container. WebM has gained noteworthy popularity, since it has been adopted as one of the most suitable formats for web content due to its patent-free and open nature. Accordingly only open-source codecs are recommended in the standard, namely VP8 video and Vorbis audio8.

Table 3 provides a comparison between some of the still relevant container formats, regarding their respective support for a variety of audio and video codecs.

8 URL: http://www.webmproject.org/.
Table 3: Comparison of the audio and video format support of different media containers.

<table>
<thead>
<tr>
<th></th>
<th>*.mp4</th>
<th>*.3gp</th>
<th>.ts</th>
<th>.mkv</th>
<th>.webm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>MPEG-4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>H.264/AVC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>VP8</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>✓</td>
</tr>
<tr>
<td>Audio</td>
<td>✓</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>MP3</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>AAC &amp; HE-AAC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>Vorbis</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Opus</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

2.5 ANDROID OPERATING SYSTEM

Android is an open source OS primarily specialising in mobile devices. While under maintenance by Google in collaboration with members of the Open Handset Alliance (OHA), a consortium of hardware, software, and telecommunication companies, it profits from contributions by the open source community.

At its core Android uses a modified version of the Linux kernel and its applications are usually written in the Java programming language. Nevertheless, because the developers implemented a custom Virtual Machine (VM) known as Dalvik VM (DVM), optimised for mobile devices with more efficient use of battery power, they sacrificed the byte code compatibility to common *.jar archives.

Applications for the Android OS are released and distributed via the Google Play Store, an open marketplace with a negligible review process and low requirements compared to its competitors. Furthermore the publication of applications is generally unrestricted, allowing installations from any other source as an application package file (*.apk).

Usually each device comes with a combination of free and proprietary software and relies on the continued update support by its manufacturer. This concept introduced a high version fragmentation into the Android ecosystem. Figure 8 gives an overview of the devices with recent access to the Google Play Store over a seven day period and allows to draw conclusions about the global version distribution. Currently Jelly Bean (4.1.x – 4.3) has a share of roughly 60% of the entire Android market while everything older than Gingerbread (2.3.3) is on the brink of extinction, adding up to less than 2%.

2.5.1 Android Media Support

Due to growing demands towards a mobile communication and entertainment platform, Android supports several formats and codecs natively. There-

9 url: https://play.google.com/store.
Figure 8: Graph of the version fragmentation of Android OS at the end of May 2014 measured by access statistics to the Google Play Store over the course of seven days.

fore a device is capable of viewing videos while browsing or sharing media recorded by its integrated camera.

Table 4 summarises the media codec support by container format and version of the Android OS. Since mobile devices operate exclusively on the media receiving end of remote browsing, only decoding capabilities are examined.

<table>
<thead>
<tr>
<th>Video</th>
<th>Decode</th>
<th>Container</th>
<th>Android Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPEG-4</td>
<td>✓</td>
<td>*.3gp</td>
<td>1.0+</td>
</tr>
<tr>
<td>H.264/AVC (CBP &amp; BP)</td>
<td>✓</td>
<td>*.mp4, *.ts</td>
<td>1.0+</td>
</tr>
<tr>
<td>VP8</td>
<td>✓</td>
<td>*.webm</td>
<td>2.3.3+</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*.mkv</td>
<td>4.0+</td>
</tr>
<tr>
<td>Audio</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MP3</td>
<td>✓</td>
<td>*.mp3</td>
<td>1.0+</td>
</tr>
<tr>
<td>AAC &amp; HE-AAC</td>
<td>✓</td>
<td>*.3gp, *.mp4</td>
<td>1.0+</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*.ts</td>
<td>3.0+</td>
</tr>
<tr>
<td>Vorbis</td>
<td>✓</td>
<td>*.ogg</td>
<td>1.0+</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*.mkv</td>
<td>4.0+</td>
</tr>
<tr>
<td>Opus</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
Fundamentally remote browsing is a subset of screen sharing with one major constraint and additional auxiliary feature requirements for an improved user experience. Nothing else than an arbitrary, modern browser with support of recent standards has to be made available to another party. Moreover it is not inherently a necessity to share the browser with more than one peer. A multitude of solutions exist to mirror a browser or the content of a screen in general.

For every existing solution a trade-off between transfer efficiency and flexibility against implementation complexity is recognisable. While more abstract communication with high level commands is fast and efficient it presumes plenty of previous knowledge decreasing universality. Any advantage is lost once outside of the abstractions domain [EM12]. The following sections will assign each related approach to a general mirroring category.

3.1 ORIGINAL OBJECT TRANSFER

One of the earliest and most obvious solutions to sharing screen content is original object transfer, sending facsimiles for remote interpretation.

Apple’s AirPlay\(^1\) transfers concrete audio and video files for playback to an Apple TV while Microsoft’s Windows Media Player\(^2\) sends the raw media to a device with Remote Desktop Protocol (RDP) support. In a similar way the already end-of-life Access Grid Distributed PowerPoint, a tool for parallel slide presentation on multiple sites, spreads data in the proprietary format of PowerPoint [Hof01].

Since the original objects are usually purposefully compressed they achieve high network efficiency without any capture overhead. Nevertheless some of the processing cost is postponed to the remote site while decoding the original file and the tight integration between the peers proves to be a disadvantage. There is no guarantee that the remote system is able to render the redirected object precisely because of the environmental and platform dependencies, for example a missing font or support for an embedded format. Furthermore the complete transfer raises privacy concerns and issues with material covered by patents because it is often undesired to grant ownership with full access of a file or to infringe the copyright protection.

\(^1\) URL: http://www.apple.com/airplay/
3.2 DIRECT MIRRORING

Approaches in this category share the common attribute, that their data to share was either already rendered or otherwise interpreted locally. They share and simultaneously impose their intrinsic perception of the data onto a potential receiver.

3.2.1 Pixmap Transfer

Screen capturing and sharing as a sequence of pixel maps, referred to as pixmaps, is another variant of direct mirroring used by Virtual Network Computing (VNC) [Ric+98]. It is extended by different research endeavours to exclusively share the pixmaps of a specific application window, for example browsers [WL07; BS08].

VNC implements the Remote Framebuffer (RFB) protocol [RL11] to share screen content and user input. RFB is a client driven remote access protocol sending the captured pixmaps in an uncompressed or compressed format using either lossless zlib4 or the lossy JPEG [ISO4] on a per pixmap basis. Therefore it leverages the broad applicability of the framebuffer but requires the host machine to run a VNC server monitoring changes. Due to the request-response principle of the protocol clients adapt to network conditions at the expense of missing out on some updates and increased latency. Upon receipt of the answer to a prior request the client requests a new update from the server which responds within a specific time interval or omits the transaction. Albeit their high bandwidth cost and inherently slow screen updates [WL07], solutions based on pixmap transfer are flexible and portable because many operating systems provide native mechanisms to capture and display pixmaps. While RFB is efficient for coherent pixmaps where changes are small and tightly concentrated it lacks in the dispatch of dynamic content [EM12].

An interesting example for a VNC implementation is TightVNC5. It distinguishes itself from other projects by supporting the DFMirage6 video mirror driver saving CPU cycles for other applications by an optimised detection of screen updates and pixel grabbing algorithm. Since the introduction of Windows 8 the mirror driver model is deprecated and superseded by a more convenient Desktop Duplication API7.

Chandra et al. focus on the measurement of the update rates required by different applications and note demanding increases for interactive scenarios [CBR14]. They empirically show that a high capture rate is essential for content mirroring and implement a solution based on pixmap transfer. By capitalising on the prior screen in a cache and temporal redundancy they are able to achieve large gains in compression ratio.

3 URL: http://metavnc.sourceforge.net/.
4 URL: http://www.zlib.net/.
5 URL: http://www.tightvnc.com/.
3.2.2 Streaming

Establishing a continuous media stream by capturing and encoding at different logical layers is a diversified field of research and a subset of direct mirroring, well suited for dynamic content.

Tying to the hardware layer are appliances like NCast\textsuperscript{8} and WHDI\textsuperscript{9}, which if necessary digitise and encode the raw VGA or HDMI output directly from the pin and basically function as a wireless connection. Their implied hardware requirements are irregularly met by mobile devices. Therefore, amongst others, a crowd funded device called Airtame\textsuperscript{10} solves this problem by reversing responsibility. In form of a HDMI stick Airtame will be attached to the receiver side. The sender has to rely on a cross-platform application and Wi-Fi Direct\textsuperscript{11} enabled hardware.

![Figure 9: Figure (a) shows the original I-Frame extracted from Sintel. Because the isolation of the Y component (b) removes the chroma components the image turns greyscale. For a clearer perception of the remaining isolated components in (c) and (d) the luminance gamma was corrected by a factor of 0.5.](image)

All solutions based on this concept share raw throughput as their bottleneck. For instance, raw video material in a resolution of 1080p, 24 FPS with a colour depth of 8 bit and colour model of YUV 4:4:4 (compare figure 9) requires 1.24 Gbps, enough to saturate a gigabit ethernet cable. Therefore the bandwidth of a wireless network is essential, either acquired by sacrificing range by the means of higher transmission frequency as harnessed by WHDI and Airtame or by CPU intensive, lossy real-time compression as used in Chromecast\textsuperscript{12} or Miracast\textsuperscript{13}. Many systems assume the lack of a deployed network infrastructure. Thus they implement network independent solutions, proprietary point-to-point wireless (e.g. WHDI) or Wi-Fi Direct (e.g. Miracast) to operate autonomously, missing opportunities of interoperability and competing with existing home networks.

Chromecast requires the implementation of the Google Cast Software Development Kit (SDK) equally on sender- and receiver-side. Usually a sender features service discovery and a secure network messaging channel, whereas the receiver is either the Chromecast device itself, running a modified release of Google TV\textsuperscript{14} executing a browser environment, or custom implementations complemented by different mirroring techniques. He et al. assess various video quality metrics with respect to Chromecast. For the purpose of

\textsuperscript{8} URL: http://ncast.com/.
\textsuperscript{9} URL: http://www.whdi.org/.
\textsuperscript{10} URL: http://www.airtame.org/.
\textsuperscript{11} URL: http://www.wi-fi.org/discover-wi-fi/wi-fi-direct.
\textsuperscript{12} URL: http://www.google.com/chromecast/.
\textsuperscript{13} URL: http://www.wi-fi.org/discover-wi-fi/wi-fi-certified-miracast.
\textsuperscript{14} URL: http://blog.gtvhacker.com/2013/chromecast-exploiting-the-newest-device-by-google/.
matching and comparing source and destination frames they encode meta
data as a bar code into the video stream to draw conclusions on the receiver
side, irregardless the closed nature of this device [He+14].

Hence Miracast is the more open standard it benefits from more mainstream
adoption. After Intel amended Miracast to their Wireless Display (WiDi)\textsuperscript{15}
technology it found more and more appliance in consumer electronics. Every
computer with a Haswell CPU and Microsoft Windows 8.1 has Miracast built
natively, capturing at the firmware level with major improvements to the
encoding performance. Even on older systems support may be granted by
specialised software. Finally, Android provides native Miracast support since
version 4.2.0 of the OS.

For rapidly changing content RDP switches to RemoteFX Media Redirection
[RFX] transferring the screen in the widespread H.264 format and Google
Chrome Remote Desktop\textsuperscript{16} encodes it to VP8, similar to WebRTC. A recent
endeavour by Valve announced as Steam In-Home Streaming\textsuperscript{17} deals with
the low latency and input lag requirements of digital games by heavy hard-
ware acceleration, because real-time encoding is a major time constraint.
Its coding back ends, VADPAU on Unix and DXVA on Windows, rely on
Graphics Processing Units (GPUs) to relieve the CPU from coding work
and achieve timeliness due to their parallelised algorithms. Latest advances
in GPU coding acceleration by Intel Quick Sync Video\textsuperscript{18} in Haswell CPUs
could be utilised as well and may even improve performance in comparison
to GPU solutions [CA13]. Similar technologies exist for other CPU architec-
tures, such as AMD’s Video Codec Engine (VCE).

GRID is a cloud gaming architecture focusing on pure GPUs coding on both
ends proposed by NVIDIA. Benchmarks have shown that it is capable to
asynchronously transcode and package H.264 video frames into the desired
format within 30 ms on the server side, while the client may introduce an
additional delay of as low as 16 ms [DGY14].

3.3 INDIRECT MIRRORING

In addition to the direct transfer methods explained before there is the so
called indirect mirroring. It describes how content may be mirrored by in-
terception and network transmission of rendering commands, issued by an
application to the respective graphics API such as OpenGL or DirectX. Such
approaches leave room for the receiver to interpret content accordingly and
although the original object transfer in section 3.1 does basically the same,
indirect mirroring is generally more intricate.

Instances of this approach are WireGL [Hum+01] and its successor Chro-
mium [Hum+02]. Both pass OpenGL commands and their arguments to
remote renderers and are limited by network performance on confrontation
with either complex geometry or many recipients. Alternatively ClusterGL
makes use of UDP multicast to distribute an OpenGL command stream

\textsuperscript{15} \textbf{URL:} http://www.intel.com/content/www/us/en/architecture-and-
technology/intel-wireless-display.html.
\textsuperscript{16} \textbf{URL:} https://chrome.google.com/remotedesktop/.
\textsuperscript{17} \textbf{URL:} http://store.steampowered.com/streaming/.
\textsuperscript{18} \textbf{URL:} http://www.intel.com/content/www/us/en/architecture-and-
technology/quick-sync-video/quick-sync-video-general.html.
across a large display array and increases performance by additional frame differencing and compression optimisations [NHM11]. Estes and Mayer-Patel split the display adapter and device on the framebuffer level. Their approach exploits application semantics to achieve a higher bandwidth efficiency limiting the generality of this approach and increasing implementation cost [EM12].

Adjusted to a windowing system as the X Window System (X) [SG86] mirroring solutions have a better understanding of underlying graphic primitives and their layout resulting into low bandwidth requirements. Protocols such as X or RDP send high level commands in a compactly serialisable form, for example to open a window of specific size and position. Major downsides are implementation costs and the hard to achieve full feature support on client devices. In theory the basic concept is transferable to calls to a graphics engine like Skia as already in use by Chrome, Firefox and Android. Anyway the benefit is hardly estimable and the implementation cost will be extraordinarily high.

Another combination of techniques is called OpenGL Extension to the X Window System (GLX) forking [SME02] additionally encodes OpenGL calls onto the X protocol stream for indirect rendering to profit from bandwidth efficiency as well as hardware acceleration on the remote device. Nonetheless, inherent problems stay the same. Especially media file streaming is a scenario rarely touched by indirect rendering research because the possible savings are negligible.

Baratto et al. where able to conclude from their research about thin-client computing that the mapping of application level drawing commands to protocol primitives and its delivery mechanisms significantly improve overall performance [BKN05].

3.4 HYBRID MIRRoring

As a consequence of the diversity of approaches and their respective strengths, emerging hybrid solutions rise in importance. Simoens et al. propose a composite of the VNC-RFB protocol and H.264 to lower CPU and bandwidth requirements imposed on the system [Sim+08].

Implementations of remote desktop technology are usually difficult to categorise, provided that they are a collection of specifications and APIs, each tailored for a specific purpose. Although X was specifically designed for use over network connections it lacks the features and performance of NoMachine’s NX Technology19, due to the appliance of differential protocol compression and caching. Nevertheless, it executes a complete window system on the client and does not address the maintenance costs tied to that procedure. On the contrary Microsoft Remote Desktop runs the graphical user interface server-side. RFB assures that the results of screen updates are immediately transferred to the client, equal to how a local windowing system would render itself into the framebuffer.

19 URL: https://www.nomachine.com/.
3.5 REMOTE CONTROL

In its traditional sense the remote control is a device to operate another in a wireless manner over a short, line of sight distance. Technology has continuously evolved in terms of transmission and user experience, from originally infrared light up to alternative control schemes such as motion- and voice control. Translated to a digital application the remote control is usually not limited to line of sight or short distances but fulfils the same purpose.

Besides the obvious solution to the problem by sending serialised commands in packets via standardised (e.g. UDP) or proprietary (e.g. RDP) protocols another technique receives increasing attention. For instance the Roku Streaming Stick employs a Representational State Transfer (REST) API to enable external control via simple HTTP requests to the so called External Control Service\footnote{url: http://sdkdocs.roku.com/display/sdkdoc/ExternalControlGuide.}. It is discoverable via Simple Service Discovery Protocol (SSDP) and could be accessed by programs on virtually every platform. Valve’s SteamOS also simplifies remote control in such a way, although it was never announced or officially documented\footnote{url: https://github.com/SteamDatabase/RemoteControlDocs.}.

3.6 REMOTE EXECUTION

Another related field is the general research of remote execution for mobile applications. Different implementations for partitioning the application into logical components, the state migration problem and adaptation to network conditions exist.

Common approaches to application partitioning try to compensate for weak performance or the lack of resources on mobile devices, such as a smaller CPU or Random Access Memory (RAM). Kremer et al. [KHR00] propose static analysis to select program components for remote execution for energy saving purposes. Hydra [Wei+08] introduces support for specialised processors like Network Interface Cards (NICs) and GPUs and uses an Integer Linear Programming (ILP) solver to decide what to offload.

MAUI [Cue+10] retrofits some of the ideas to maximize the advantage of code offloading with regard to energy consumption. It evaluates the cost and benefit of each function call with a specialised solver on the remote server to evaluate the feasibility of remote execution and taking advantage of it.

3.7 SUMMARY

Table 5 scores important attributes of the various remote browsing categories. Although the values are subjective and plain estimations they help to give a general overview of strengths and weaknesses for the later decision making.
Table 5: Comparison of the various mirroring techniques based on estimations. The scoring system goes from 0 to 10, with 10 being the highest score.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Mirroring Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original</td>
</tr>
<tr>
<td>Quality</td>
<td>9</td>
</tr>
<tr>
<td>Implementation Cost</td>
<td>7</td>
</tr>
<tr>
<td>Timeliness</td>
<td>4</td>
</tr>
<tr>
<td>Resource Efficiency</td>
<td>7</td>
</tr>
<tr>
<td>Cross-Platform</td>
<td>2</td>
</tr>
<tr>
<td>Interoperability</td>
<td></td>
</tr>
</tbody>
</table>
This chapter is mainly a discussion of possible conceptual designs to a remote browsing solution conforming to the goals in section 1.2, for the purpose of arriving at the conclusion what systems should be incorporated into the prototypical implementation.

As a background for further considerations the analysis in section 4.1 enumerates general requirements. Section 4.2 determines the most adequate approach to meet the defined goals and exceptions. The most promising parts of the concept get their own section for in depth discussion.

### 4.1 REQUIREMENT ANALYSIS

To allow for an objective discussion about the design and implementation of a potential remote browsing solution the most significant requirements to achieve the goals defined in section 1.2 will be illustrated here. In favour of assessment and comparability the main aspects are discussed in order of relevance and enumerations are ranked by their importance for the prototypical implementation.

**Flexibility** is the first and foremost requirement of remote browsing. Ideally the implementation is capable of supplying an ubiquity of personal devices, such as smartphones, tablets and netbooks with a modern browser. This diversity in size, screen resolution, computing power, storage and even input methodology should be taken into account. Especially because of the limited user interfaces of mobile devices, due to their size and extensibility constraints, innovative input methodologies such as acceleration and touch are utilised and require a mapping to other input actions. Because of the lack of respective network jacks in mobile devices as well as human mobility in general, the connection has to be wireless. Lastly a remote browsing session should not end on the disconnection of a controller to profit from the continuation of processes initiated by it.

1. cross-platform
2. input methodology translation
3. wireless connection
4. runtime detachment

**Performance** is relevant in many parts of the implementation. Especially a high input latency can ruin the user experience as a whole. Unfortunately there is no stereotypical usage scenario of remote browsing. Today’s web applications may cover the entire range from serving static content up to dynamic real-time applications such as digital content creation or games, each resulting in a different nature and rate of screen updates. Compression could be considered as a method to
lower network congestion and increase the overall speed of the implementation. Furthermore performance and quality have to be balanced to allow for a resource efficient, sustainable solution, even on mobile devices. Dobrian et al. highlight the importance of the stream buffering to viewing ratio and its impact on user engagement [Dob+11]. Therefore performance will always be a priority at the expense of overall quality.

1. input lag
2. real-time
3. resource efficient

**Quality** is measured in correctness, how true to the original the content will be shared. Although dynamic, computer generated animations update at rates only limited by the hardware of the host device and would therefore require a higher frame rate, it is more relevant to increase quality perception than to create an identical reproduction.

1. quality perception
2. correctness

## 4.2 Design Concept

This section balances reasons and proposes a basic design concept of the implementation regarding fulfilment of the requirements in section 4.1.

To guarantee the independence of view and controller at runtime because of the detachment required for energy saving purposes, the components need to be decoupled. Either by a client-server or a P2P model, the latter of which has higher implementation complexity and is already applied in WebRTC. With centralisation of responsibility comes the imperative that the server-side has to support and run a browser adhering to modern web standards. As this requirement is easily met a client-server architecture is proposed for the prototypical implementation.

Apparently the many approaches described in chapter 3 have their respective advantages and disadvantages, mostly lacking in detail and owing to their generality are missing out on optimisation potential for remote browsing.

Firstly, the notion of original object transfer is the very basic concept of the internet and a browser itself. Thus its usual advantages are disputable in the context of remote browsing because it imposes nothing more than a purposeless indirection. Nevertheless, even older generations of native browsers deal well with static content. A highly efficient solution would fall back to a basic server side HTTP proxy if it can guarantee the faithful reproduction of content on the client.

Suchlike functionality requires a pre-processing step which analyses the requirements imposed by a web page by parsing its Document Object Model (DOM) tree. Meanwhile the interpreter could register HTML media elements and catch media events to enable direct media controls within the remote browser. An interpreter is transferable to other use cases and should be the essential part of every remote browsing solution.
The most substantial argument for solutions based on screen sharing is that they are either completely or to a large extent independent of the browser capabilities of the remote device. Two general approaches are distinguished between, namely pixmap transfer and video streaming.

Beyond that the convenient reuse of login credentials, auto completion data and bookmarks provided by the host device is advantageous. Even uniquely customised browser environments stay equally rich on the client side. Finally, if the screen sharing coding complexity does not exceed the web applications resource requirements, a client can benefit from the computational relocation to the host, similar to the concepts of remote execution. Therefore a screen sharing solution is proposed.

4.3 IMPLEMENTATION CONCEPT

This implementation concept describes and motivates important available technologies for the realisation of the design in section 4.2 after a general discussion of imposed challenges in section 4.3.1. Not least these considerations are the base of the future prototypical implementation.

4.3.1 Challenges

Forman and Zahorjan elaborate on the universal challenges and risks accompanying mobile computing [FZ94]. From security concerns over wireless networking up to battery life and data loss, remote browsing is influenced by most of them. Only few of the relevant considerations about the challenges of remote browsing are briefly introduced, to limit the extensive scope of this section.

MOBILE NETWORKING, albeit its advantages compared to a fixed Local Area Network (LAN) imposes additional challenges, from the physical up to the application layer. Wireless transmission is lossy caused by a variety of outside influences. Within a vacuum signal strength decreases proportional to the square of the distance, even more in reality. Increasing loss rates due to interference, higher delays and jitter are the consequence for wireless communicating applications. Although the physical layer is outside the scope of this thesis it has to be kept in mind.

INPUT LAG is an essential part of the prototypical application and therefore one of the goals is to keep it as low as possible. Input lag is defined as the delay between user input and visual feedback. Inhibiting factors range from processing time of input to network lag and throughout influence the user experience negatively. MacKenzie and Ware measured how the responsiveness of an application impacts user movement times and error rates. Up to an input lag of 75 ms the human error rate increases by ~36% while lag of 225 ms degrades it by ~214% [MW93]. Hence a delay of less than 100 ms is desirable.

LOW-LATENCY STREAMING serves the flexibility requirements of the implementation concept well but also introduces a variety of challenges.
If multiple clients simultaneously want to remote control the server, the network traffic is a linear function of the number of subscribed clients. Subsequently a home network might congest and a protocol capable of multicast as described in section 2.1.1.2 becomes mandatory. In addition to this use case such an unreliable protocol imposes less overhead at the expense of transmission errors. Lost, delayed or out of order packets have to be avoided either by a reliable network protocol or with different technologies in the application layer.

One of the latter is Automatic Repeat-Request (ARQ). It tackles the problem of packet loss by re-requesting the lost packets, allowing a successful recovery on timely delivery [Con+01]. Another approach uses frame interleaving to reduce the perceptual damage of a media stream. This technique splits frames into different packets to spread potential damage across time, leading to a less noticeable type of disruption [Gla+98].

Lastly the heterogeneity of clients demands for an adaptive streaming solution. For example, smaller devices might not be able to decode a high definition live stream and would waste a lot of bandwidth and computational resources that way. Therefore the server has to provide the device with an appropriate stream as already explained in section 2.3.3.

**Synchronisation** is a common problem in the distribution of live content. If a client’s clock is precisely synchronised with the server it is possible to reliably predict when new content is available, saving bandwidth due to fewer unsuccessful queries. For this purpose the client would periodically synchronise time with the server via Network Time Protocol (NTP) in an adaptive streaming solution [Mil91].

Another issue related to media streaming is asynchronous video or audio. For instance the incoming video frames do not have to be in order of appearance because of coding efficiency optimisations characterised in section 2.4.1.1. Thus the concept of time stamping each frame tries to remedy the problem. While the Presentation Time Stamp (PTS) represents the exact moment of rendering of the frame to an output device the Decoding Time Stamp (DTS) indicates processing order by the decoder. Therefore the decoding rate can dynamically adapt to the environment and the stream synchronises. It is not essential to include a DTS in all frames, since it can be interpolated by the decoder from a PTS. Generally the PTS and DTS only differ if the stream utilises B-Frames due to their coupling with both neighbouring frames (compare table 6).

Table 6: An example stream of frames (see section 2.4.1.1) and their respective timestamps.

<table>
<thead>
<tr>
<th>Content</th>
<th>PTS</th>
<th>DTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

**Client-Side Exclusive** challenges appear mainly due to the heterogeneity of client devices and their mobility. Hardware abstraction layers
such as SDL\(^1\) and SFML\(^2\) provide a cross-platform, unified window management as well as input and media device abstraction. Although they simplify the development process, this concept does not fit well into continuously evolving platforms like Android with specified user experience design rules which applications have to meet. Furthermore it is more efficient to use native capabilities for the given concept to decode and render a video stream.

**SERVER-SIDE EXCLUSIVE** obstacles emerge from to many components involved into the process of content preparation and delivery. As proposed in section 4.2 the server has to run a modern browser. While the use of an existing browser distribution is recommended it complicates the interfacing between the involved components of a native implementation. Anyhow, the alternative approaches as a browser plug-in or a shared library proxy introduce either a browser and SDK or even a binary compatibility dependency with unforeseeable implementation complexity. Moreover the media support requirements of a potential controller have to be precisely met to allow for media streaming.

### 4.3.2 Client & Server Architecture

Due to rapid development, web technologies converging on the browser and recent endeavours for standardisation there never have been as many opportunities to implement a web application.

JS engines are an integral part of every web browser, giving the developer access to networking capabilities, DOM or HTML\(^5\) video amongst others. Since the introduction of AJAX the rising importance of JS in web development caused an engine performance race between the established browsers for the competitive advantage.

Late 2012 Mozilla Research proposed the intermediate programming language asm.js that cuts down on JS’s dynamic systems to increase performance and is already adopted by the major browsers [HWZ13]. Due to the common practice of native browser optimisations this technology still has additional performance gain potential. Lately asm.js was benchmarked as only 1.5 times slower in Firefox than a native implementation\(^3\).

Transcompilers like Emscripten\(^4\) or Duetto\(^5\) compile C or C++ into asm.js code allowing the compilation of libraries or entire game engines into a JS file. They are as long cross-platform, as the target platform has a browser with native asm.js support. Once this requirement is met by the majority of mobile browsers, these frameworks represent an elegant solution completely within the problem domain of remote browsing.

Irrespective of WebRTCs (see section 2.2.2) rapid adoption on many platforms some time will pass until the majority mobile web browsers supports it natively. In combination with the local screen as a media source a peer could easily share its screen with others. However, the performance of this

---

1 URL: http://www.libsdl.org/.
2 URL: http://www.sfml-dev.org/.
4 URL: http://emscripten.org/.
5 URL: http://leaningtech.com/duetto/.
solution would have to be subject to further evaluation and comparison of its VP8 to H.264/AVC. Nevertheless, because WebRTC is still in an early phase of its life cycle and has yet to find wide adoption and compatibility to mobile devices it does not qualify as a solution yet.

Lastly native applications should be taken into account. What they lack in comfort due to implementation cost they compensate with performance and user experience, for example the native user interface in Android. Notwithstanding the complexity native solutions introduce into cross-platform communication they increase flexibility on both ends. Developers profit from the existing infrastructure such as development environments, build systems and libraries.

4.3.3 Communication

At its very base remote browsing necessitates the use of a protocol for wireless communication. Although the choice would be dependent on the back end of a chosen type of web communication some protocols are preferable. Generally the reuse of existing ones provides advantages due to the optimisations that have been made to efficiently support them.

UDP qualifies for real-time data transfer in mostly reliable network conditions, given in a local wireless network. Applied to multimedia the checksum algorithm encompasses excessive and for H.264/AVC even redundant data which leaves optimisation potential due to the intrinsic error correction of this codec. Thus UDP Lite is the favourable protocol for streaming with it.

Especially on the application layer an implementation could capitalise on the large investments already gone into existing adaptive streaming infrastructures, for instance built into the Android OS itemised in section 2.5.

While UDP would guarantee the lowest input lag possible, one of the first and foremost goals is the flexibility of the prototype with cross-platform adaptability. Because every smart mobile device OS comes with native implementations of HTTP the use of a REST API is preferable.

4.4 HTTP ADAPTIVE STREAMING

Whereas adaptive streaming (see section 2.3.3) represents an universal solution to the challenge of low-latency streaming to a multitude of different devices, the use of HTTP makes it cross-platform and flexible. Therefore it has to be thoroughly considered.

Even though TCP is generally inappropriate for the transport of media streams, it got a wider acceptance as the back end transport protocol of HTTP adaptive streaming due to the latter’s benefits. Generally web applications are converging on web browsers because of the flexibility of such implementations and slight performance penalties compared to more complex native versions. Furthermore HTTP is economic and easy to deploy since it employs common HTTP servers which also favours its use in a CDN [Zam09].
TCP is a reliable protocol that delivers most part of the global World Area Network (WAN) traffic and it is able to guarantee the stability of the network by means of an efficient congestion control algorithm [Jac88]. This circumvents the implementation requirement of additional error correction logic for UDP on the application layer. Moreover TCP has built-in NAT traversal functionality.

Nevertheless, it is significantly more computationally complex than the traditional streaming technologies described in section 2.3.1 and section 2.3.2 if served to a single client exclusively. It introduces additional storage and encoding costs, and the operationally complex challenge of maintaining quality globally. Furthermore the rate-adaptation logic of adaptive streaming competes with the TCP congestion control behind HTTP, increasing the complexity of the adaptation problem [ABD11].

Experiments with the adaptive streaming have certainly not met the real-time requirements of the proposed solution. They introduce delays of more than 2 s due to a large minimum segment size. Although each of the popular mobile OSs either supports HTTP streaming natively or can be extended by a cross-platform media framework, the native media support of Android lacks in the regard of HTTP adaptive streaming in combination with its native MediaPlayer (compare section 2.5.1).

4.5 SUMMARY

An implementation of remote browsing imposes a variety of challenges to the developer. Either way, while original object transfer is obviously not the correct solution for remote browsing, pixmap transfer utilises a low level kernel module and therefore requires an Android device with root privileges. Only the approaches based on complex implementations remain, either completely indirect or hybrid mirroring, hardly an alternative for a prototypical implementation as stated in table 5.
The involved entities in remote browsing, content provider, view and controller, where already mentioned in section 1.1. Because of the remote nature of this topic their data exchange has to be specified.

Generally communication flows over HTTP and WebSockets (see section 4.3.3), both using reliable TCP in their back end to avoid the problems and unwanted interactions related to parallel UDP traffic and TCP congestion control. Because such a concept cannot utilise the advantages of multicast in a multi-user environment, a split between the content provider and the view is not advisable, unless connected via HDMI or a similar cable for the network independent transmission of video signals to a display.

5.1 CLIENT

Fundamentally the client is the remote control for content provider as well as a receiver because it needs to stay independent of the location or line of sight to other views. The main challenge of its implementation is the media playback, the controls are a simple REST API and only shortly introduced for the server in section 5.2.3.

5.1.1 Media Playback

Tests have shown the inapplicability of the native Android MediaPlayer for real-time streaming. A live stream via RTP (see section 2.1.2.2), thus a low-latency application layer protocol with UDP back end, is played back with a delay of roundabout five seconds at 640 × 480 px and a bit rate of 800 Bps. Source of this delay is a constant within libstagefright.so, the hardware accelerated library for media playback at the core of the Android OS. Independent of the media the cache size defaults to an interval of 4–20 MB1, representing a too large buffer size generally unsuited for real-time playback of this type of content.

Therefore the live presentation of a media stream requires either a Native Development Kit (NDK) implementation of a custom media player or a web application able to receive and draw a video stream, for example into a HTML5 canvas element. Although FFmpeg is verifiably deployable to Android the implementation complexity of cross-platform NDK solutions is extraordinary due to different CPU architectures and their specifics. Therefore the implementation of the final prototype is a web application based on JS and the HTML5 canvas element. At the unfortunate expense of flexibility, a minor limitation in platform support, it is a versatile concept. Therefore the

1 URL: http://androidxref.com/4.2.2_r1/xref/frameworks/av/media/libstagefright/include/NuCachedSource2.h#75.
final implementation consists of a JS port of MPEG-1 and utilises WebSockets to receive a continuous and reliable data stream.

5.2 SERVER

The remote browsing server is responsible for media preparation and provision. Each topic will be examined in detail in the following sections.

5.2.1 Media Preparation

This section explains the general content preparation pipeline for streaming and the various opportunities by using FFmpeg. Latter offers a complete, cross-platform solution for various content preparation tasks and for output, listed in order of appearance in a common pipeline:

1. Capture (Input Devices)
2. Transcode
3. Segment & Combine
4. Index
5. Output (Output Devices)

While item 3 and item 4 are only relevant in an adaptive streaming pipeline that does not meet the real-time requirements (compare section 2.3.3 and section 4.1), the item 5 will be elaborated in section 5.2.2.

FFmpeg supports a variety of input devices enabling it to read input from local displays up to webcams or remote devices. It follows a modular approach based on pipes and streams to achieve this flexibility. For instance, Video4Linux2 or V4L2 is a programming interface for video capture [Ver14] and output devices as well as a driver framework embedded into the Linux kernel, well suited for recordings per webcam and similar devices. Although there is no technical relation, the name V4L is a derivative of VFW which is sometimes stylised as V4W. Nevertheless, remote browsing requires the fastest and most flexible input device to capture a window. Recent versions of FFmpeg include x11grab, a module dedicated to that purpose. Basically the whole set-up of the capture device is done in a single line of code:

```bash
#!/bin/bash
ffmpeg \ 
  -v verbose -threads 0 \ 
  -f x11grab -s 1440x900 -r 30 -i :0+0.0+0 \ 
  ...
```

The first line of parameters are general options to make FFmpeg more verbose and enable full use of available CPU cores. Secondly the input device is specified, from its source (x11grab) over structural information about the screen resolution, frame rate and the upper left corner of the frame. This basic set-up will be abbreviated in the following code listings.

2 URL: http://www.linuxtv.org/.
Afterwards the content pipeline has to do the transcoding step as elaborated in section 2.4 and section 2.3.3.1. It is possible to append the encoding options to the existing capturing configuration. Sometimes the capture will be fed directly into a specialised transcoding server on a self-regulating CDN items 2 to 4 because it is advisable to know the requirements of a target device beforehand. A simple video encoding step into a widely supported format looks like this:

```
#!/bin/bash
ffmpeg ${FFMPEG_OPT} ${FFMPEG_CAPTURE} \
  -c:v libx264 -preset ultrafast -tune zerolatency \ 
  -profile:v baseline -pix_fmt yuv420p -b:v 2M \ 
  ...
```

This example transcodes video input from `${FFMPEG_CAPTURE}` into the low-latency, mobile device and browser friendly H.264 codec. The YUV 4:2:0 pixel format is a compromise between correctness and visual perception due to chroma subsampling. In general the decoding of H.264 is computationally expensive and makes heavy use of threading in its native implementation. The web application equivalent would have to translate the H.264 decoding algorithm to the HTML5 Web Workers [Hic12b] concept in order to achieve similar performance and therefore sacrifice compatibility to a broad spectrum of older systems3. Because the prototypical implementation is not intended for use over WAN it is easier to decrease the compression complexity and use more of the available bandwidth by utilising another, less intricate codec: MPEG-1.

MPEG-1 [ISO93] has many limitations, for instance a maximum resolution of $768 \times 576$ px, but is a good fit for portable devices, despite notebooks or tablets with high resolution displays. If nothing else, MPEG is fairly light to decode, even for a non-native implementation like a JS web application.

5.2.2 Media Provision

Software that acts as a data selector, reads and joins multimedia streams from different types of input devices is called a multiplexer (muxer). They are generally used to increase the amount of data that can be sent over the network. In the case of this implementation MPEG-1 is the joint format of transmission, with each of its respective limitations. Fundamentally the media will be forwarded to stream-server.js, a JS based streaming server.

```
#!/bin/bash
ffmpeg ${FFMPEG_OPT} ${FFMPEG_CAPTURE} \ 
  -vf "scale=768:576" \ 
  -f mpeg1video http://localhost:8082/secret/768/576
```

This script introduces an additional, no explicitly streaming related concept to the pipeline, the `-vf "scale=768:576"` parameter. It describes a filter to scale input to the desired output resolution. Furthermore it pipes the MPEG-1 video stream through the local node.js server to a single client.

3 URL: http://caniuse.com/#search=worker.
5.2.3 Enabling Remote Control

Employing a REST API for simple and cross-platform remote control is a common practice as shown in section 3.5. HTTP POST requests are a highly optimised way communicating the intent of changing the state of the server, in this case specifically the input state. The node.js web application framework express.js provides a simple way to implement such a system and the HTTP server is already supplied by node.js as well. In principle each input argument by any kind of device can be explicitly URL encoded into an expression like `/input?touch.0.x=300.0&touch.0.y=42.0&touch.0.op=down`, notwithstanding multi-touch or sensory input.

On the Linux OS uinput is a kernel module that allows to influence the input subsystem from the user-side. Therefore it can be utilised to create and control an input device in `/dev/input`. Its result is a virtual interface, not tied to any kind of physical device and can be independently influenced by code, with the limitation that only keyboard and mouse events are supported and it introduces an OS dependency on the server. Nevertheless, it can be written as a node.js native module, easily expandable by a cross-platform and multi-device input injection library for future work.

5.3 SUMMARY

From the NDK build system of the Android OS over issues regarding NAT traversal while experimenting with protocols other than TCP as well as high delay due to the MediaPlayer cache problem (see section 2.5), the implementation appears to be difficult.
In this chapter the scenarios and metrics to evaluate the performance of the techniques presented in chapter 5 are proposed and applied. At first the experimental environment will be introduced in section 6.1 for ranking and comparability. Section 6.2 comments on the possible quality metrics to evaluate the success of meeting the criteria defined in section 4.1. Afterwards, section 6.3 elaborates on the respective course of action of each test scenario and the measured results.

6.1 EXPERIMENTAL ENVIRONMENT

Due to the complexity of cross-platform development and respective platform dependencies the test scenarios have to take place on limited resources. Figure 10 illustrates the construction of the test.

![Network diagram of the experimental environment with the communication paths between involved devices.](image)

Table 7 provides a reference of the hardware specifications of the devices deployed to benchmark the implementation from chapter 5. On the one hand server-side responsibilities are covered by a single notebook, usually a Home Theater Personal Computer (HTPC) in a home entertainment context, running Ubuntu Linux 13.10. This server incorporates everything from a basic HTTP server to the low-latency content preparation described in section 5.2. Additionally it executes a network emulation during the experiment to achieve a broader test coverage and find possible weaknesses. The controller or client on the other hand is a mid-range tablet originally released in April 2012, running version 4.4.2 of the Android OS.
Table 7: Overview of hardware specifications of the experimental devices.

<table>
<thead>
<tr>
<th></th>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>Samsung Galaxy Tab 2 7.0</td>
<td>Lenovo ThinkPad T400s</td>
</tr>
<tr>
<td><strong>Model</strong></td>
<td>GT-P3100</td>
<td>2815-W19</td>
</tr>
<tr>
<td><strong>CPU</strong></td>
<td>$2 \times 1.0$ GHz</td>
<td>$2 \times 2.53$ GHz</td>
</tr>
<tr>
<td><strong>GPU</strong></td>
<td>PowerVR SGX540</td>
<td>Intel X4500MHD</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>RAM: 1 GB</td>
<td>RAM: 4 GB</td>
</tr>
<tr>
<td></td>
<td>Storage: 16 GB Flash</td>
<td>Storage: 120 GB HDD</td>
</tr>
<tr>
<td><strong>Network</strong></td>
<td>WLAN: 802.11 b/g/n 3G: HSDPA</td>
<td>WLAN: 802.11 b/g/n 2G: GSM</td>
</tr>
<tr>
<td></td>
<td>900/2100</td>
<td>850/900/1800/1900</td>
</tr>
<tr>
<td><strong>Display</strong></td>
<td>$600 \times 1024$ px (75:128)</td>
<td>$1440 \times 900$ px (8:5)</td>
</tr>
<tr>
<td><strong>OS</strong></td>
<td>Android 4.2.2 (Jelly Bean)</td>
<td>Ubuntu 13.10 (Saucy)</td>
</tr>
</tbody>
</table>

6.2 QUALITY METRICS

Quality can be measured for video and audio respectively. It describes the reproduction accuracy of the remote browser’s content on the client device. Chikkerur et al. describe a variety of image quality metrics to parametrise differences between images [Chi+11]. Because the Peak Signal-to-Noise Ratio (PSNR) is an approximation to human perception of reconstruction quality this traditional, pixel based metric is sufficient as a quality indicator for the evaluation. A simple way to define the PSNR is with the help of the Mean Squared Error (MSE):

$$\text{MSE}(I_s, I_c) = \frac{1}{MN} \sum_{m=0}^{M} \sum_{n=0}^{N} (I_s(m, n) - I_c(m, n))^2.$$  \hspace{1cm} (6)

This MSE is the average squared distance between a reference image $I_s$ and a distorted image $I_c$. By adding up the squared differences of all the pixels and dividing by the total pixel count given by the resolution $M \times N$ px it is computed on a pixel-by-pixel basis. Thereafter the PSNR can be described with the following term:

$$\text{PSNR}(I_s, I_c) = 10 \cdot \log_{10} \left( \frac{\text{MAX}^2}{\text{MSE}(I_s, I_c)} \right) = 20 \cdot \log_{10}(\text{MAX}) - 10 \cdot \log_{10}(\text{MSE}(I_s, I_c)).$$  \hspace{1cm} (7)

In this equation the constant number MAX is the maximum value of a 8 b pixel, $2^8 - 1 = 255$, the common colour depth of mobile displays. The PSNR describes the ratio between the reference signal and the distortion signal in an image, given in decibels (dB). The higher the PSNR, the closer the distorted image is to the original. In general, a higher PSNR value should correlate to higher image quality. As long as the PSNR is used to evaluate the same codec its a valid metric [GH08].

Sometimes the throughput $\tau$, given in bits per second (bps), is used synonymously with the theoretical rate of data transfer also known as maximum bandwidth. Since it is often an unreliable measurement of perceived or practical performance, throughput will be used as a term of successful delivery
Figure 11: MPEG-1 analysed by a frame sample from Sintel, magnified to show its common compression artefacts similar to the JPEG image format.

of transfer units, where latter could either be a media frame or a segment in adaptive streaming:

\[ T(i) = \frac{s_i}{t_i} \]  

(8)

where \( s_i \) is the size of one transfer unit \( i \) in bits and \( t_i \) the time it took to download it in seconds.

6.3 Testing

The Assessment of the impact of operating system specific performance or capabilities was not part of the evaluation because the different system API dependencies would exceed a prototypical implementation, although they were considered and avoided where possible. Each decision for an OS was based on its respective popularity.

At first the remote activities to benchmark are described in section 6.3.1 and afterwards profiled regarding the proposed quality metrics in section 6.3.2.

6.3.1 Test Scenarios

**WEBSITE:** A comprehensive example of a media based website is the promotional web installation for Warner Bros. Gravity. It is a dynamic website using three.js, a JS 3D library, as a back end. Among a variety of animations, for instance menu scrolling, buttons or a contextual image slider it features background video and audio.

1 URL: http://gravitymovie.warnerbros.com/
2 URL: http://threejs.org/
(a) Website  (b) Game  (c) Video

Figure 12: Screenshots of the different test scenarios.

**Game:** For the sake of input testing a playing session of the game Sinous\(^3\) with high interactivity and precision requirements will be measured. Although its simple concept of dodging dots seems trivial it is a demanding test of input lag and every error will be immediately punished. Furthermore the sound effects are tied to a specific action and will highlight synchronisation problems. Sinous is entirely based on HTML5 and its canvas element.

**Movie:** Popular CDNs such as YouTube, Twitch or Netflix are generally comprehensively supported by the different streaming devices and provide their users with dedicated applications for mobile access to the service. Nevertheless, because other usage scenarios are not automated for consistent testing and highly dependent on user input a video viewing session of the movie Sintel on YouTube\(^4\) in 720p is evaluated as well. Sintel is a 3D animation by the Blender Foundation and because of its open license well suited as a test scenario.

6.3.2 **Benchmarks**

This section briefly applies the proposed quality metric to the measurable systems. While MPEG-1 reaches an acceptable PSNR value on static pages in figure 13(a) is resolution and encoding quality is too low to be able to read everything. Especially in the game scenario in figure 13(b) the codec cannot predict where points enter the playing field and the continuous scrolling leads to new random patterns every time, therefore the codec would not suited for gaming purposes. Interestingly the movie analysed in figure 13(c) produced by far the most frame updates as recognisable by the values at the x-axis. Its dynamically changing scenes leads to periodic variation which causes screen artefacts similar to JPEG and would not lead to a pleasurable viewing experience.

6.4 **Summary**

Due to a variety of difficulties with the implementation, the evaluation falls short. Nevertheless it is interesting to see that the only codec fast enough to encode 720p video material without significant losses in frame rate on the server notebook trades portability and coding complexity with so much quality reduction.

\(^3\) URL: http://www.sinousgame.com/.

\(^4\) URL: http://www.youtube.com/watch?v=eRsGyueVLvQ.
Figure 13: Experimental PSNR results of transcoding the original screen recording into a MPEG-1 video stream (20 s).
7 CONCLUSION

The following chapter draws conclusions from the insights of the bachelor thesis’ project. Section 7.1 comments on the results achieved in the previous evaluation chapter and the general thesis on hand. Section 7.2 highlights some of the possible improvements to this project which should be considered as future work.

7.1 DISCUSSION

The prototype is able to request a remote browser session from the server. Despite transmission and coding of the view imposes an increased energy consumption onto the device, some resource extensive web applications, algorithms and games can theoretically profit in performance and save resources on the mobile device. In the context of home entertainment it is suggested to use the device as a plain remote without the redundant viewing capabilities because a proper view might already be in sight.

Although a MPEG-1 stream enables extraordinary low delays, even if implemented in pure JS, its major downside is the quality loss. Though flexibility is defined as one of the most important goals for the remote prototype the quality trade-off is too high. The current implementation would require a more appropriate codec and the server more CPU power to be able to transcode the live feed in real-time with at least 30 Frames Per Second (FPS). Thus either a native implementation or a better transcompiled JS video codec is required.

7.2 FUTURE WORK

Based on the assumption that mobile devices are unable to keep up with the most recent web standard as a consequence to the variety of platforms, the solution for barrier-free browsing will always be remote. Nevertheless a tendency for further unification of web and multimedia becomes apparent with standardisation efforts such as WebRTC and DASH. Due to this development web applications will get more flexible and even share their screen natively without any requirement to dive into a native development kit. Further codecs should be evaluated for their applicability in web applications and different path should be taken to provide a comfortable, power efficient and high quality solution to remote browsing. Especially HTML5 Web Workers, web application based threading, are already supported on Android since version 4.4.
LIST OF ACRONYMS

AAC  Advanced Audio Coding 21–23, 25
AJAX  Asynchronous JavaScript and XML 12, 13, 39
API  Application Programming Interface 12–14, 22, 30–32, 40, 43, 46, 49
ARQ  Automatic Repeat-Request 38
ASP  Advanced Simple Profile 22
AVC  Advanced Video Coding 17, 20, 22, 23, 25, 39, 40
AVT  Audio/Video Transport 10
B-Frame  Bidirectional Frame 19, 20, 22, 38
BP  Baseline Profile 20, 25
CBP  Constrained Baseline Profile 20, 25
CDN  Content Delivery Network 2, 14, 16, 17, 40, 45, 50
CPU  Core Processing Unit 16, 28–32, 43, 44, 48, 53
DASH  Dynamic Adaptive Streaming over HTTP 18, 53
DCCP  Datagram Congestion Control Protocol 10
DOM  Document Object Model 36, 39
DSL  Digital Subscriber Line 8
DTS  Decoding Time Stamp 38
DVM  Dalvik VM 23
FPS  Frames Per Second 29, 53
GLX  OpenGL Extension to the X Window System 31
GOP  Group Of Pictures 19
GPU  Graphics Processing Unit 30, 32, 48
HDMI  High Definition Multimedia Interface 2, 29, 43
HE-AAC  High-Efficiency AAC 21, 23, 25
HTML  Hypertext Markup Language 12, 13, 36, 43, 45, 50, 53
HTPC  Home Theater Personal Computer 47
HTTP  Hypertext Transfer Protocol 9–12, 14, 15, 18, 32, 36, 40, 41, 43, 46, 47, Section: 2.1.2.1
ICE  Interactive Connectivity Establishment 14
IEC  International Electrotechnical Commission 20
IEEE  Institute of Electrical and Electronic Engineers 7
IETF  Internet Engineering Task Force 7, 9–11, 13, 21, 22
I-Frame  Intra Frame 19, 20, 29, 38
IIS  Internet Information Services 18
ILP  Integer Linear Programming 32
IP  Internet Protocol 7, 8, 10, Section: 2.1
IPTV  Internet Protocol Television 10
ISO  International Organization for Standardization 20
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISP</td>
<td>Internet Service Provider</td>
</tr>
<tr>
<td>ITU</td>
<td>International Telecommunication Union</td>
</tr>
<tr>
<td>ITU-T</td>
<td>ITU Telecommunication Standardization Sector</td>
</tr>
<tr>
<td>JS</td>
<td>JavaScript</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>MMUSIC</td>
<td>Multi-Party Multimedia Session Control</td>
</tr>
<tr>
<td>MP</td>
<td>Main Profile</td>
</tr>
<tr>
<td>MPEG</td>
<td>Moving Picture Experts Group</td>
</tr>
<tr>
<td>MSE</td>
<td>Mean Squared Error</td>
</tr>
<tr>
<td>MVC</td>
<td>Multiview Video Coding</td>
</tr>
<tr>
<td>NAT</td>
<td>Network Address Translator</td>
</tr>
<tr>
<td>NDK</td>
<td>Native Development Kit</td>
</tr>
<tr>
<td>NIC</td>
<td>Network Interface Card</td>
</tr>
<tr>
<td>NTP</td>
<td>Network Time Protocol</td>
</tr>
<tr>
<td>OHA</td>
<td>Open Handset Alliance</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer-To-Peer</td>
</tr>
<tr>
<td>PDA</td>
<td>Personal Digital Assistant</td>
</tr>
<tr>
<td>P-Frame</td>
<td>Predicted Frame</td>
</tr>
<tr>
<td>PPP</td>
<td>Point-to-Point Protocol</td>
</tr>
<tr>
<td>PSNR</td>
<td>Peak Signal-to-Noise Ratio</td>
</tr>
<tr>
<td>PTS</td>
<td>Presentation Time Stamp</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RDP</td>
<td>Remote Desktop Protocol</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>RFB</td>
<td>Remote Framebuffer</td>
</tr>
<tr>
<td>RTC</td>
<td>Real-Time Communication</td>
</tr>
<tr>
<td>RTCP</td>
<td>Real-Time Transport Control Protocol</td>
</tr>
<tr>
<td>RTP</td>
<td>Real-Time Transport Protocol</td>
</tr>
<tr>
<td>RTSP</td>
<td>Real-Time Streaming Protocol</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SDP</td>
<td>Session Description Protocol</td>
</tr>
<tr>
<td>SIP</td>
<td>Session Initiation Protocol</td>
</tr>
<tr>
<td>SSDP</td>
<td>Simple Service Discovery Protocol</td>
</tr>
<tr>
<td>SSE</td>
<td>Server Sent Events</td>
</tr>
<tr>
<td>SVC</td>
<td>Scalable Video Coding</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>UDP Lite</td>
<td>Lightweight User Datagram Protocol</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>VCE</td>
<td>Video Codec Engine 30</td>
</tr>
<tr>
<td>VCEG</td>
<td>Video Coding Experts Group 20</td>
</tr>
<tr>
<td>VFW</td>
<td>Video For Windows 22, 44</td>
</tr>
<tr>
<td>VGA</td>
<td>Video Graphics Array 29</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine 23</td>
</tr>
<tr>
<td>VNC</td>
<td>Virtual Network Computing 28, 31</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium 9, 13</td>
</tr>
<tr>
<td>WAN</td>
<td>World Area Network 40, 45</td>
</tr>
<tr>
<td>WebRTC</td>
<td>Web Real-Time Communication 13, 14, 30, 36, 39, 53, Section: 2.2.2</td>
</tr>
<tr>
<td>WiDi</td>
<td>Wireless Display 30</td>
</tr>
<tr>
<td>WLAN</td>
<td>Wireless Local Area Network 4, 48</td>
</tr>
<tr>
<td>X</td>
<td>X Window System 31</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language 12</td>
</tr>
</tbody>
</table>
BIBLIOGRAPHY


<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
</table>


I hereby declare that the bachelor thesis submitted was in all parts exclusively prepared on my own and that other resources, than those explicitly referred to, have not been utilised.

Magdeburg, Germany, the 19th June 2014

Tim Benedict Jagla