FH JOANNEUM (University of Applied Sciences)

Usage possibilities of WebRTC in a cross-platform developed mobile app

Bachelor Thesis

submitted in conformity with the requirements
for the degree of
Bachelor of Science in Engineering (BSc)

Bachelor's degree program **Internettechnik**
FH JOANNEUM (University of Applied Sciences), Kapfenberg

supervisor: Dipl. Ing. Johannes Feiner

submitted by: Michael Stifter
personal identifier: 1310418054

06 / 2016

Obligatory signed declaration:

I hereby declare that the present Bachelor's thesis was composed by myself and that the work contained herein is my own. I also confirm that I have only used the specified resources. All formulations and concepts taken verbatim or in substance from printed or unprinted material or from the Internet have been cited according to the rules of good scientific practice and indicated by footnotes or other exact references to the original source.

The present thesis has not been submitted to another university for the award of an academic degree in this form. This thesis has been submitted in printed and electronic form. I hereby confirm that the content of the digital version is the same as in the printed version.

I understand that the provision of incorrect information may have legal consequences.

Michael Stifter Graz, 13.06.2016

Table of contents

Abstract

Web Real-Time Communication (WebRTC) enables developers to create peer-to-peer real-time communication applications with audio and video streams using web technologies such as Hypertext Markup Language version 5 (HTML5), JavaScript and Cascading Style Sheets (CSS). However, because not all web browsers and mobile platforms currently support this technology, developers are faced with a problem when they want to ensure that their applications are functioning on a majority of mobile devices. This thesis proposes cross-platform developed apps as a solution for reliably using WebRTC inside mobile applications without the necessity to invest in development efforts for multiple native platforms. The first part of the thesis discusses WebRTC and its functionality in detail along with the advantages and limitations of using it on mobile devices. In the second part, cross-platform mobile development is presented as a possible solution for ensuring that WebRTC can be used reliably in mobile applications and various framework options for simplifying the development process are compared. In a third step, a reference app was developed using two promising options, Crosswalk and OpenWebRTC. The reference apps of both frameworks were evaluated using a set of criteria defined during the research process. The evaluation showed that both frameworks are suitable for developing cross-platform apps using WebRTC, however, Crosswalk proved to be the more attractive option because it offers better access to underlying device-specific features and simpler build and deployment mechanisms. The conclusion section offers several possibilities to extend the underlying work.

Kurzfassung

Web Real-Time Communication (WebRTC) ermöglicht es Softwareentwicklern, Echtzeitkommunikationsanwendungen zu erstellen, bei denen Audio- und Videokanäle direkt zwischen den Benutzern übertragen werden, ohne dabei von einem Webserver weitergeleitet zu werden. Diese Anwendungen können mittels Webtechnologien wie zum Beispiel Hypertext Markup Language Version 5 (HTML5), JavaScript und Cascading Style Sheets (CSS) realisiert werden. Zurzeit unterstützen allerdings nicht alle Webbrowser und mobile Plattformen diese Technologie, was Entwickler vor ein Problem stellt wenn sie sicherstellen wollen, dass die Anwendungen auf einer Vielzahl von mobilen Geräten funktionieren. Diese Arbeit schlägt Cross-Platform Apps als eine Lösung vor, um WebRTC verlässlich in mobilen Apps verwenden zu können, ohne dieselbe Anwendung für mehrere mobile Plattformen neu entwickeln und betreuen zu müssen. Da sich die Technologie hinter WebRTC derzeit noch in Entwicklung befindet und sich Teilkomponenten in unregelmäßigen Abständen ändern können, liegt ein wesentliches Augenmerk dieses Ansatzes darauf, dass der Web View einer solchen Anwendung auf dem aktuellen Entwicklungsstand von WebRTC ist. Der erste Teil dieser Bachelorarbeit erläutert WebRTC und die dahinterstehende Technologie mitsamt seinen Vor- und Nachteilen näher. Der zweite Teil stellt Cross-Platform Entwicklung als eine mögliche Lösung vor, um WebRTC verlässlich in mobilen Anwendungen verwenden zu können und vergleicht mehrere Optionen für Frameworks, die den Entwicklungsaufwand vereinfachen können. Aufgrund der vorgenommenen Recherche wurde eine Referenzanwendung mit zwei vielversprechenden Frameworks, Crosswalk und OpenWebRTC, entwickelt. Diese beiden Apps wurden anschließend aufgrund einer Zusammenstellung von Kriterien beurteilt, die im Zuge der Recherche erstellt wurden. Die Beurteilung ergab dass beide Optionen die Entwicklung von Cross-Platform Apps mit WebRTC brauchbar sind, Crosswalk zeigte sich jedoch als die ansprechendere Wahl. Der Grund dafür liegt am besseren und einfacheren Zugriff auf gerätespezifische Funktionen und die mitgelieferten Build- und Deployment-Werkzeuge. Abschließend werden noch einige Möglichkeiten vorgestellt, um den derzeitigen Entwicklungsstand der Arbeit zu erweitern.

1 Introduction

Over the last years, Web Real-Time Communication (WebRTC) has seen a significant rise in popularity especially in browser-based web applications, and is expected to continuously grow over the following years[1]. It offers numerous advantages for communication technology because it enables real-time communication in web browsers without any additional plugins or software. This was not possible before the introduction of WebRTC (cf. Stifter 2016, p. 23). Apart from that, it uses peer-to-peer connections, where the data is transferred directly without a third-party server involved, which results in decreased network latency. However, its biggest disadvantage to date is the fact that not all web browsers support WebRTC.

This poses a problem for developers who want to use WebRTC in web applications today, because it is impractical to require users to switch to a web browser that fully supports WebRTC. On mobile devices, however, the situation is different. Besides web apps, there is also the possibility to use native apps. Native apps have become massively popular over the last years and are able to deliver substantial advantages when it comes to user experience (cf. Cailean 2013). This stems from the fact that it is possible to integrate and access numerous components of the user's device, such as the list of contacts, the calendar and various other sensors into an application with ease. While it is possible to develop a native app that uses WebRTC, it also increases the development effort considerably, since it is necessary to implement the same functionality on multiple platforms, such as Android, iOS and Windows Phone.

One solution to this problem could be the use of a suitable cross-platform development framework that facilitates the use of WebRTC. With a cross-platform developed mobile app it is possible to develop the application only once, instead of multiple times for each platform it should run on. However, since WebRTC is a technology that can be considered relatively new and is still under development, it is not guaranteed that cross-platform development frameworks fully support the latest version of WebRTC.

This thesis takes a deeper look into popular cross-platform mobile development frameworks and examines them on their ability to support current versions of

[1] http://webrtcstats.com/are-we-at-the-tipping-point-of-webrtc-adoption/ [4 June 2016]

WebRTC. To analyse this evaluation, a set of criteria is defined in order to identify suitable framework options for developing cross-platform apps that use WebRTC. During this process, two suitable framework options are presented and a reference app is developed using both frameworks.

The thesis is structured as follows: The first part discusses the history and functionality of WebRTC, together with its benefits and shortcomings. The second part mentions various ways of implementing a mobile app and highlights the advantages and disadvantages of each method in detail. In a third step, the possibilities of using WebRTC on mobile devices are addressed and two reference apps are developed using possible options for WebRTC cross-platform development frameworks. Following that, the essential insights of the development process with regard to the implementation of the reference apps are pointed out. Chapter 5 discusses the evaluation process and its results. The final section concludes the thesis by summarizing the essential findings and suggesting possibilities to expand the underlying work.

2 WebRTC

“Web Real-Time Communication (WebRTC) is a new standard that lets browsers communicate in real time using a peer-to-peer architecture” (Loreto & Romano 2014, p. vii). This technology development is particularly promising since it enables real-time telecommunication applications within web browsers, without the need of third-party extensions or plugins. Furthermore, WebRTC is open source software, which means that the entire source code is publicly available. This is beneficial for developers because they can get a full understanding of the inner functionality and, additionally, the code can be extended and improved by an online community.

The following chapter provides an overview of the history and functionality of WebRTC, along with its benefits and limitations. The end of chapter looks at the possibilities of using WebRTC outside of web browsers.

2.1 History

WebRTC started as a project conducted by Google and was presented to the public for the first time in May 2011 (cf. Alvestrand 2011). Later that year, the company decided to publish the entire source code under a permissive Berkeley Software Distribution (BSD) license, enabling the internet community to contribute ideas to the project. At the same time, in November 2011, the first rudimentary version of WebRTC was added to the Google Chrome browser. At the beginning of 2013 the technology passed an important milestone, as Mozilla published its implementation of WebRTC in their Firefox browser and it was possible to start peer-to-peer connections from Chrome to Firefox and vice versa. In that same year, both companies also released the first mobile versions of their browsers supporting WebRTC (cf. WebRTC Tutorial 2014).

2.2 Architecture

WebRTC is built upon a rather complex architecture, which offers an Application Programming Interface (API) with a simple set of functions for interaction with applications. Figure 1 illustrates an outline of the whole architecture. The Web Application API acts as the top layer, which is written in JavaScript and can be accessed by a standard web page in a browser. This is the only layer that developers have to work with, while all other architecture layers fulfil their tasks independently upon requests on this top layer. The Web Application API interacts with the internal WebRTC API, which is written in C++. The internal API is

responsible for the handling of PeerConnections and their session management (cf. Stifter 2016, p. 22).

The concern of the following layer is the management of media-related components. This includes the codecs of audio and video engines, echo cancellation, image enhancement and, most importantly, the correct synchronization of media tracks in order to provide a valuable user experience. The layer below handles the capturing of audio and video streams, and is therefore directly communicating with the lowest level of the architecture, which is the physical device hardware, e.g. cameras and microphones that are built-into or attached to the device (cf. Grigorik 2013, p. 311).

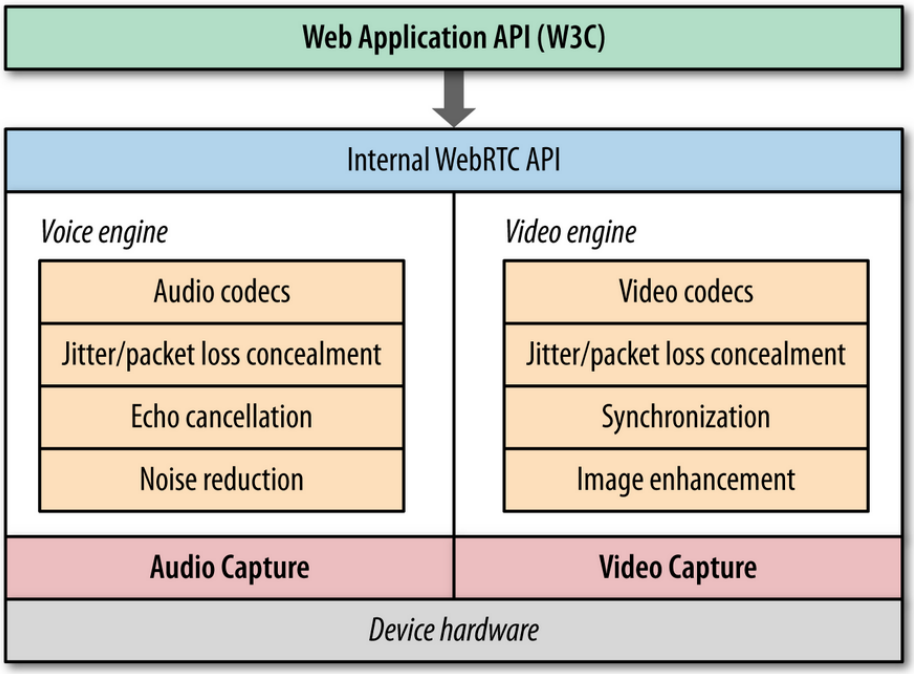


Figure 1: WebRTC architecture (Grigorik 2013, p. 311)

2.3 Components

WebRTC is based upon three different components, namely MediaStream, PeerConnection and DataChannel. While the first two are mandatory, DataChannel is an additional optional component. The components are described in more detail in the following subsections.

2.3.1 MediaStream

The MediaStream interface is responsible for everything related to audio and video streaming. It can hold any number of MediaStreamTracks. In a traditional video conferencing scenario, this would be one video track and two audio tracks, a left and

a right channel (see Figure 2). However, developers have the option to add or remove tracks (cf. Loreto & Romano 2014, p. 12).

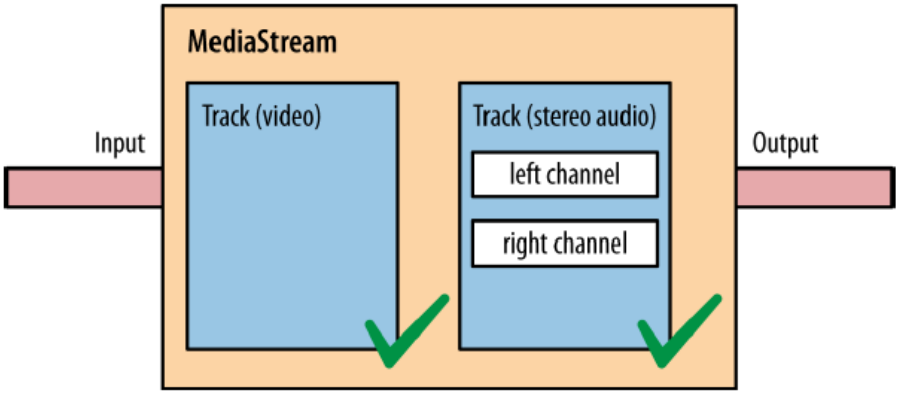


Figure 2: A WebRTC MediaStream object that contains one video and two audio tracks (Loreto & Romano 2014, p. 13)

To create a MediaStream object, developers can use navigator.getUserMedia() API, which obtains access to media equipment attached to the user's device. This typically includes cameras and microphones. Listing 1 describes a simple example on how to request access to an audio and video stream.

When this request is issued in line 21, the web browser informs the user about the request from the web page. The user can then grant access to the requested media sources or deny it. In other words, it is necessary to explicitly get the user's approval for using the device's media components. Furthermore, browsers display a red recording icon on top of the web page's tab to indicate that it has currently access media resources. This adds an additional security layer on behalf of the user, since it is not possible to use the media devices without knowledge and consent from the user.

If the request was approved, the browser tries to access the requested resources and subsequently calls the success callback function described in line 8. In case the user denied the request or if an error occurred during the initialization stage, the error callback function in line 16 will be called.

```
01.  // The constraints object determines the type of stream that is requested
02.  var constraints = { audio: true, video: true };
03.
04.  // DOM reference to the video element on the HTML page
05.  var video = document.getElementById("video");
06.
07.  // Success callback in case user grants access to the request
08.  var successCallback = function(stream) {
09.      // attach stream to video element
10.      video.srcObject = stream;
11.
12.      video.play();
13.  };
14.
15.  // Error callback in case user denies media access or an error occurrs during
     initialization
16.  var errorCallback = function(error) {
17.      console.error("navigator.getUserMedia() failed:", error);
18.  };
19.
20.  // Request media access
21.  navigator.getUserMedia(constraints, successCallback, errorCallback);
```

Listing 1: Simple example for requesting access to camera and microphone of user device

2.3.2 PeerConnection

A PeerConnection object in WebRTC “represents an association with a remote peer, which is usually another instance of the same JavaScript application running at the remote end” (Loreto & Romano 2014, p. 7). In other words, it holds the actual WebRTC peer-to-peer connection between two users. It is responsible for managing every aspect of the connection, from initialization to teardown. The developer is only required to implement the initial start-up and the termination of a connection, the management part in between is automatically handled by the WebRTC API (cf. Leaver, Iwase & Katsura 2015).

The initialization of a PeerConnection is accomplished over a signalling channel, which is usually JavaScript code inside the web page. The data transfer at this stage is handled by the web server. The whole signalling process is mentioned in more detail in Chapter 2.6. When the PeerConnection between two users has been successfully established, it is possible for both parties to exchange MediaStream objects. This could mean, for instance, that they can now see and talk to each other in a video chat directly from browser to browser (cf. Loreto & Romano 2014, p. 7f).

2.3.3 DataChannel

The DataChannel API is the only optional component and is therefore not required to be implemented by the developer. Its purpose is to provide an additional communications layer, in which developers can send arbitrary data between the two users. One PeerConnection object can hold any number of DataChannels. The API functions of DataChannels were modelled after the ones from WebSockets and resemble them closely (cf. Leaver, Iwase & Katsura 2015).

The main configuration options for a DataChannel is its priority inside the PeerConnection and if the messages should be delivered in reliable or unreliable mode. In reliable mode, messages sent over the DataChannel are guaranteed to be delivered in the order they were sent, adding some administration overhead to the data transfer, which might result in slower transmission times. In unreliable mode, on the other hand, this overhead is not included, thus resulting in faster transmission without guaranteeing successful delivery (cf. Ristic 2014).

2.4 Protocols

WebRTC uses several protocols that are essential to deliver its real-time communications functionality. As can be seen in Figure 3, WebRTC uses User Datagram Protocol (UDP) at the transport layer, since it offers low latency and has little protocol overhead. It does, however, not guarantee the order of the packets or that a packet has been delivered at all. In an audio and video streaming environment, this is a compromise that application designers are willing to accept, since the human brain is able to fill small gaps easily, while it is highly sensitive to transmission delays (cf. Grigorik 2013, p. 315f).

With UDP alone, however, it is not possible to establish and maintain peer-to-peer connections. WebRTC needs Interactive Connectivity Establishment (ICE), Session Traversal Utilities for Network Address Translation (STUN) and Traversal Using Relays around Network Address Translation (TURN) as mechanisms to determine public Internet Protocol (IP) addresses and traverse Network Address Translation (NAT) layers and firewalls. On top of that layer, all data transferred between two peers is secured with Datagram Transport Layer Security (DTLS). After this stage, the Stream Control Transport Protocol (SCTP) and the Secure Real-Time Transport Protocol (SRTP) handle higher-level networking tasks like multiplexing of streams and provide congestion and flow control (cf. Leaver, Iwase & Katsura 2015).

Together, all layers described in this section, provide the functionality of the PeerConnection and DataChannel API.

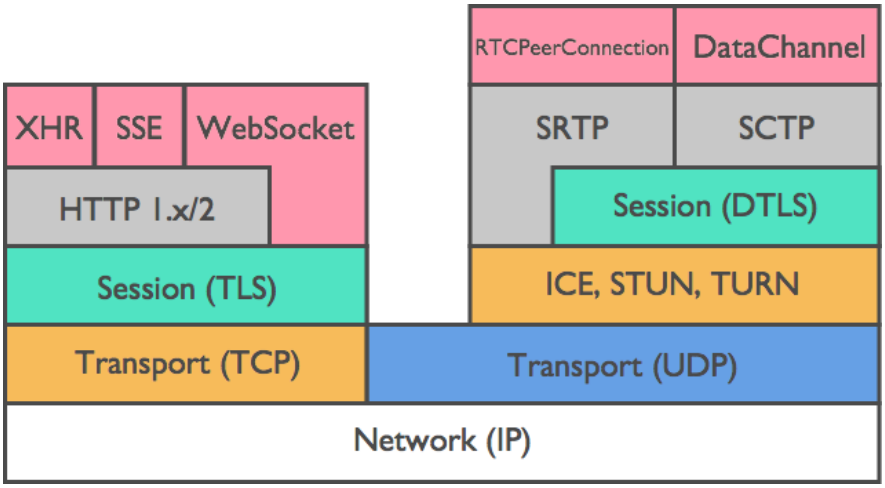


Figure 3: WebRTC protocol stack (Leaver, Iwase & Katsura 2015)

2.5 Functionality

The majority of web applications is based upon the client-server principle, which means that the client (i.e. the web browser) requests a web page from the server, who in turn fulfils the request by delivering the Hypertext Markup Language (HTML) source code of the page. In WebRTC, this model is extended by introducing a peer-to-peer communication layer (cf. Loreto & Romano 2014, p. 2). As depicted in Figure 4, both peers Alice and Bob request a web page from a server, which also acts as the signalling server. The signalling server is responsible for various tasks, such as determining the best possible option for a direct network path between the peers and finding suitable audio and video stream encodings and resolutions. After this initial signalling stage, Alice and Bob now share a peer-to-peer connection, where all media data is exchanged directly between them, without the server being involved. The media data consist of audio and video streams and, optionally, arbitrary data transferred over a DataChannel.

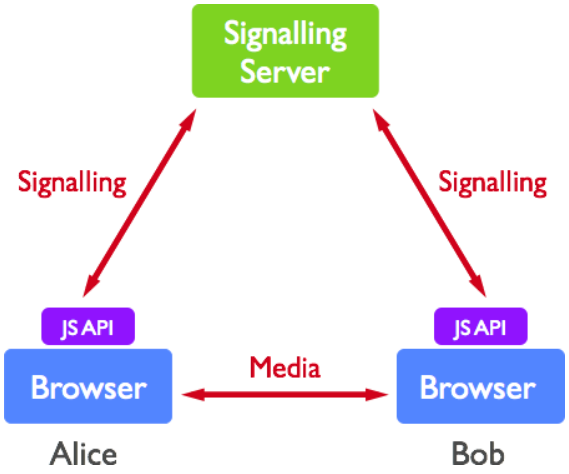


Figure 4: WebRTC call topology (Leaver, Iwase & Katsura 2015)

2.6 Signalling

The signalling stage is responsible for setting up peer-to-peer connections between users. Figure 5 describes an example of the signalling process. Sticking to the call setup from above, Alice is trying to call Bob. To start a peer-to-peer connection, Alice's web application first creates a new PeerConnection object and, upon success, adds her own media stream tracks to it. This could either be audio or video tracks, or both. Afterwards, a signalling offer message is sent to the remote peer, in this case Bob. This offer message includes meta data about Alice's media streams, such as codecs and media types, information about the network that Alice is part of as well as key data used to create secure connections (cf. Loreto & Romano 2014, p. 9).

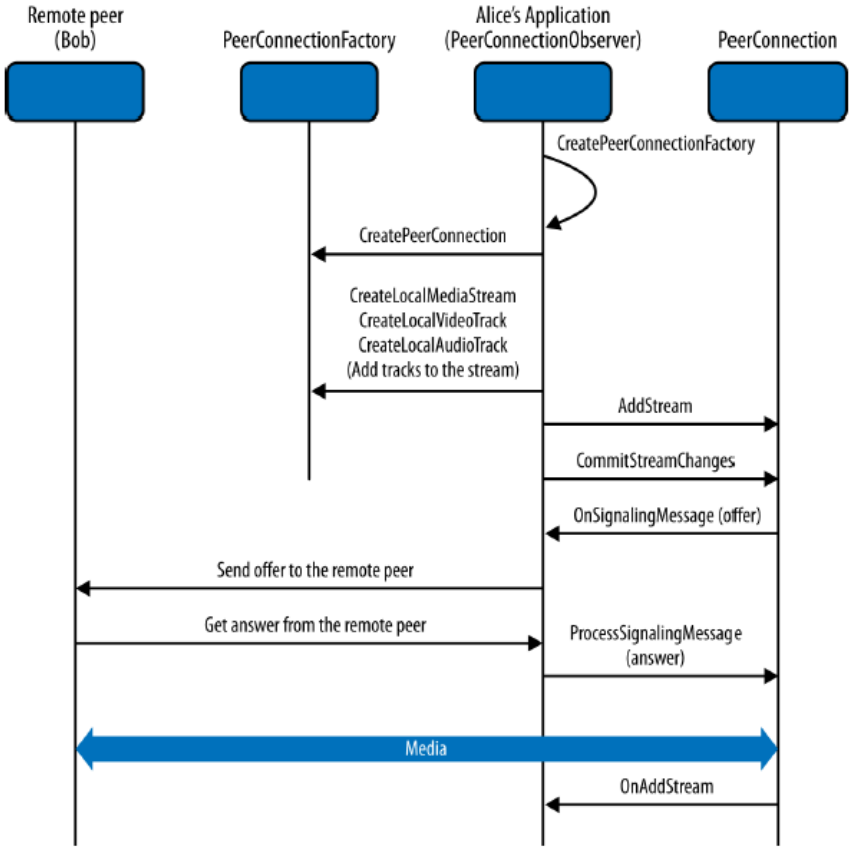


Figure 5: Signalling process to start a PeerConnection with another user (Loreto & Romano 2014, p. 10)

Bob's application receives this offer and starts a similar process, where the same information about Bob's endpoint is added to the PeerConnection object, which is then sent back to Alice as a signalling answer message. This process can be repeated several times, until both peers have found a suitable network path for their peer-to-peer connection. To determine this path, WebRTC uses ICE, which is responsible for locating the public IP address of both peers. Since this might be problematic if one or both users are part of a network that uses NAT, this is accomplished by using STUN and TURN servers (cf. Loreto & Romano 2014 p. 8, p. 37).

The technical implementation of the signalling process can be achieved by a variety of options. One popular approach is to use WebSockets. This offers the advantage that the client and the signalling server share a persistent full-duplex connection, on which both parties can send data at any time. Furthermore, it might be beneficial since the whole signalling process is of an asynchronous nature and requires the sending and receiving of multiple session description offers and answers. A different

approach is to use XMLHttpRequest (XHR), which is as viable as the WebSockets approach from a technical point of view. For a developer, however, the use of XHR requires a more complex application architecture since it is built upon stateless, unidirectional Hypertext Transfer Protocol (HTTP) requests (cf. Khan 2015).

2.7 Advantages

One significant advantage of WebRTC is the fact that it is platform independent (cf. Grégoire 2015). With classic desktop applications, developers have to ensure that they are functioning across a number of operating systems, such as Microsoft Windows, Linux or Apple's Mac OS. This is not necessary with web applications because they usually run in web browsers. The fact that the main execution environment of WebRTC is a web browser yields the additional advantage that there are several device types that have browsers installed, which further broadens the number of platforms where web applications can be used on. This includes desktop computers, laptops, smartphones and lately also wearable devices like smart glasses and smart watches. It has to be noted, however, that not all web browsers offer the same range of features.

Another large benefit of WebRTC is that it can be used free of charge. This is especially interesting for companies who need to use real-time communication as part of their daily business, for instance for exchanging information with branch offices abroad. Several proprietary business solutions, such as Skype Business[2], bill the use of their service per customer and per month. WebRTC could offer businesses an option to easily develop such a real-time communication application themselves and save monthly license charges for a proprietary product.

Additionally, WebRTC offers the advantage that all components of the communication process are securely encrypted. This is of special interest to a large number of users, who are concerned about data privacy on the internet. WebRTC uses DTLS to encrypt the data transfers between two users in a PeerConnection. On top of that, the SRTP is used together with the SCTP to handle the real-time communication functionality, such as reliable delivery, flow control and multiplexing of media streams (Leaver, Iwase & Katsura 2015).

To further improve the security of the whole WebRTC environment, Google Chrome in December 2015 removed the possibility to obtain access to a device's camera

[2] https://www.skype.com/de/business/

and microphone via the MediaStream API if the web page was not loaded using Hypertext Transfer Protocol Secure (HTTPS) (cf. What's next for WebRTC 2015). Consequently, developers who want to use this feature are encouraged to run their applications in a more secure environment. If a web page does not use HTTPS, user inputs, such as data submitted in a form, are transferred to the web server in plain text. With applications like Wireshark[3], it is possible for anyone in the same network to read the submitted data. When using HTTPS, on the other hand, the entire data transfer is encrypted with Transport Layer Security (TLS).

The design of the WebRTC architecture relies on a peer-to-peer model. In reality, this means that once a PeerConnection between two users has been established, there are no third party servers involved in the data transfer. This offers two advantages at the same time. First, it reduces network latency because the peers are connected directly, and second, it increases security by removing one component in the transfer that could be a potential target for attacks (cf. Azevedo, Lopes Pereira & Chainho 2015).

2.8 Limitations

A significant limitation of WebRTC is its browser compatibility. As depicted in Figure 6, the web browsers that fully support WebRTC are Google Chrome, Mozilla Firefox, both together with its mobile counterparts, and Opera.

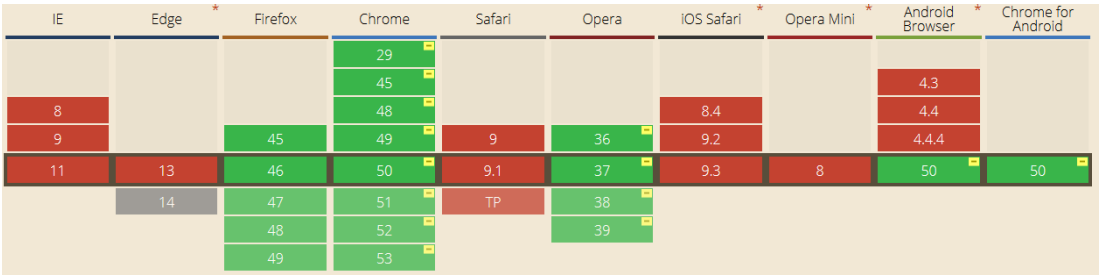


Figure 6: Overview of browsers that have a working implementation of WebRTC PeerConnections[4]

[3] https://www.wireshark.org/
[4] http://caniuse.com/#search=webrtc [27 May 2016]

As can be seen in Figure 7, these three browsers represent approximately 61% of the total web browser market share in Austria in 2015[5]. By comparison, this figure has risen by three percentage points from 58% in 2014 (cf. Stifter 2016, p. 26).

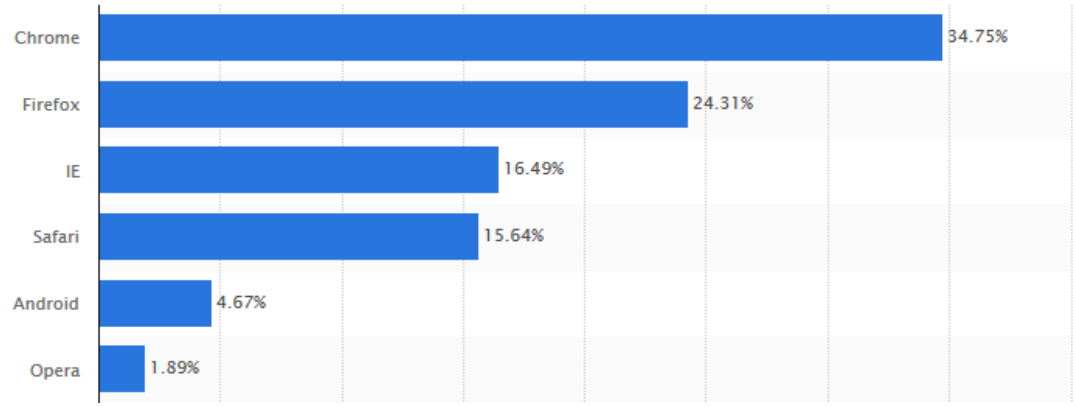


Figure 7: Web browser market share in Austria in 2015[5]

As for the other browsers, Microsoft's Internet Explorer has been discontinued and cannot be installed on the current operating system Windows 10. As a result, the market share of Internet Explorer is expected to decline in the near future. Its successor, Microsoft Edge, has started to implement Object Real-Time Communication (ORTC), which is compatible to WebRTC in its current state with some additional workarounds. In September 2015, the first features of ORTC were integrated into Edge[6]. Apple, on the other hand, has not yet revealed any plans on integrating WebRTC into their Safari browser.

One reason for the incomplete browser compatibility is that WebRTC is still under development. It has a working draft API definition[7], which is maintained by the World Wide Web Consortium (W3C) and is updated on a non-regular basis. The fact that the definition is not yet finished could possibly discourage developers, as some API functions and methods might be changed in the future. Additionally, some web browsers still use vendor prefixes for certain methods, which makes development difficult. For instance, the method for obtaining camera access is called navigator.getUserMedia() in the W3C specification, however, if developers want to

[5] http://www.statista.com/statistics/421152/wbe-browser-market-share-in-austria/ [27 May, 2016]
[6] https://blogs.windows.com/msedgedev/2015/09/18/ortc-api-is-now-available-in-microsoft-edge/ [27 May 2016]
[7] https://www.w3.org/TR/webrtc/

ensure that it works across all browsers, they have to use the following lines of code in their application.

```
01.    navigator.getUserMedia = navigator.getUserMedia ||
02.                             navigator.webkitGetUserMedia ||
03.                             navigator.mozGetUserMedia;
```

Listing 2: Necessary variable assignment to deal with vendor prefixes in web browsers

There are a number of other methods that suffer from the same issue, especially when it comes to WebRTC-specific functions, for instance RTCPeerConnection or RTCSessionDescription, which both must be assigned in the same way as described in Listing 2 for navigator.getUserMedia.

2.9 Usage possibilities on mobile devices

The main goal of WebRTC is to bring real-time communications to the web browser (cf. Loreto & Romano 2014, p. 1). This raises the question if there are other ways to use WebRTC on mobile devices.

2.9.1 Web applications

One option that requires less effort is a browser-based web application that uses Responsive Web Design. Responsive Web Design aims at the “creation of web sites that take into account different types of devices, usually ranging from mobile phones to desktop, and optimise viewing experience for the device at hand” (Voutilainen, Salonen & Mikkonen 2015). It makes use of a special Cascading Style Sheets (CSS) version 3 feature, media queries. Media queries enable developers to define CSS rules based on custom criteria which have to apply in order to be used by the web browser. These criteria include the screen size, the orientation of a device or the pixel density of the screen (cf. Johansen, Pagani Britto & Cusin 2013).

Broadly speaking, Responsive Web Design can be achieved by two different approaches: First, it can be implemented from the ground up using media queries. This allows the developer a maximum of flexibility, since it is possible to define the query breakpoints and the responsive behaviour of components as detailed as desired. Second, it is possible to use a responsive web design framework. The advantage of such a framework is that it can be used without additional effort for setup. On the other hand, it can be laborious compared to the first approach to add customised behaviour.

A popular example of such a framework is Bootstrap[8]. Bootstrap's main focus lies on a "mobile-first" approach, and has a considerable amount of functionality built in. It makes use of a grid system, which by default contains twelve columns per row (cf. Voutilainen, Salonen & Mikkonen 2015). The behaviour of the columns can be adapted according to the size of the display the web page is viewed on. Bootstrap classifies device types into four categories, namely extra small devices, small devices, medium devices and large devices.

2.9.2 Native app

WebRTC offers the possibility to be used in native mobile apps. As described in Chapter 2.2, the internal WebRTC API is written in C++, and offers an API for web applications. For native apps, however, there are no native WebRTC APIs available yet. Google offers a WebRTC library for both the Android and iOS operating system. In order to use this library, the source code first must be compiled on the development system. The official WebRTC page offers step-by-step guidelines[9] for the compilation process. It must be noted that for compiling the native Android code, it is necessary to use a machine which is running the Linux operating system. For the iOS code, a machine with Mac OS X is required, respectively.

2.9.3 Native app with web views

Another option would be to use a standard native app that contains only one web view, which loads and displays a web application like a browser does. As a result, it is possible to provide users with a native app, without having to implement the application logic for the appropriate operating system.

Android features an implementation of WebRTC since version 4.4 of its operating system. However, as Hart (2015) points out, it is based on Chromium 36 and does not guarantee an update to the latest WebRTC version. On iOS, on the other hand, there is no possibility to use WebRTC inside a UIWebView in a native app. The reason for this is that the Safari browser, as an underlying foundation, does not feature any implementation of WebRTC at all, as mentioned in Chapter 2.8.

2.9.4 Cross-platform developed mobile app

When it comes to using WebRTC inside a cross-platform developed mobile app, developers currently have two options.

[8] http://getbootstrap.com/
[9] https://webrtc.org/native-code/

Firstly, there is Crosswalk[10], which is an open source web application runtime. The Crosswalk Project is backed by technology company Intel and is built upon Apache Cordova. The substantial advantage of using Crosswalk is the fact it always uses the latest version of the Google Chrome browser for its web view. This is a considerable benefit when developing a WebRTC application, since Google Chrome is generally the first web browser to implement new WebRTC features (cf. Hart 2015). Due to the fact that Crosswalk uses Apache Cordova internally, applications for it are written using HTML5, JavaScript and CSS. Thus, apps created with Crosswalk are hybrid apps.

The second option is OpenWebRTC, which is a project by Swedish company Ericsson Research and describes itself as a "mobile-first WebRTC client framework for building native apps" [11]. It is built on top of the GStreamer multimedia framework. Like Crosswalk, the source code is open source and publicly available on the internet. OpenWebRTC cannot be considered a traditional cross-platform mobile development framework. It rather offers natively compiled libraries that provide the WebRTC functionality with API methods similar to the ones used in web applications in the browser. Using OpenWebRTC, developers can decide to use it either in a native or in a hybrid app. When used in native apps, it offers the advantage of a compiled library, without the need to build the Google WebRTC library. In hybrid apps, the OpenWebRTC helper library provides custom web views for Android and iOS, which are derived from their parent classes, WebView in Android and WKWebView in iOS. Consequently, it is possible to use WebRTC inside a web view on an iOS device.

2.10 Potential applications

Due to its technical design, WebRTC excels when it comes to providing real-time peer-to-peer communication. In the near future, this will most likely open up a number of new potential use cases, especially for web applications. Three possible use cases are briefly outlined in the following subsections.

2.10.1 Real-time communication

An evident use case for WebRTC is the field of real-time communication. Before the introduction of WebRTC, developing a real-time communication application meant that programmers had to obtain vast knowledge about audio and video codecs,

[10] https://crosswalk-project.org/
[11] http://www.openwebrtc.org/

communication protocols, data transfer and encryption. WebRTC simplifies this process significantly by providing a plain JavaScript API (cf. Leaver, Iwase & Katsura 2015). As a result, web developers can easily build such an application simply by calling the right API methods at the right time, without needing special knowledge about telecommunications technology.

Furthermore, due to the fact that WebRTC runs natively within web browsers, it is not necessary for the users to install any kind of software or plugin to use it. This could be an additional encouragement for people to use it, as there is no entry barrier in the form of downloading and installing software from the internet. This opens up new opportunities for e-commerce businesses to communicate with their customers personally and face-to-face, directly on their web page. For instance, these communication opportunities could include customer support or personal consulting.

In general, the field of real-time communication ranges from plain text messaging, telephone-like audio connections to video streaming. In its simplest form, there are only two users involved, but especially in a business environment, it is likely that there are a larger number of people involved in a conference call (cf. Leaver, Iwase & Katsura 2015).

2.10.2 Peer-to-peer file sharing

An interesting application use case for WebRTC is peer-to-peer file sharing. Nurminen et al. (2013) examined the feasibility of such an application. The idea behind it was that popular video-on-demand platforms, for example You Tube, need to invest heavily in content delivery networks (CDN) in order to provide their videos promptly to a constantly increasing number of users. They designed a system where all users are part of a peer-to-peer network. Theoretically, it is only necessary for one user of the network to download a video from the server. If another user requests the same file, the web application asks if an active peer already has downloaded the file. If that is the case, it gets the file directly from this user, saving a considerable amount of bandwidth for the video platform and its content delivery network.

Nurminen et al. (2013) concluded that it was not possible due to the fact that web browsers at the time did not support the DataChannel API. Nowadays, however, a large number of browsers already support this API, including Mozilla Firefox, Google

Chrome and Opera, making it a feasible option for platforms dealing with enormous bandwidth traffic.

2.10.3 Integrating real-time sensor data

Azevedo, Lopes Pereira & Chainho (2015) propose a solution for a standardised API for the integration of real-time sensor data into web applications. They mention an example from the field of medicine, where a patient is able to remotely send data from a medical device, for instance a blood pressure sensor, to a doctor. In order for this concept to work, it is necessary to implement a middleware that handles the communication between the sensors and the web browser. This could be realised with a web browser extension (cf. Azevedo, Lopes Pereira & Chainho 2015).

Another field that could benefit greatly from such a solution is manufacturing, namely large factories. Especially since the rise of the “Industry 4.0” era, there have been several attempts to create “smart factories”, which offer increased flexibility in the production process. In such environments, malfunctions can be discovered significantly faster. They could even be detected before they happened, when the devices are able to issue warnings if measurement readings are not within pre-defined thresholds (cf. Dhungana et al. 2015).

2.11 Reference use case: Remote support for domestic use

One tangible use case, which belongs to the real-time communications category mentioned in Chapter 2.10.1, is remote support for domestic use. This term stands for the assistance of less-skilled persons from better-skilled ones, by using a real-time communications application, during the execution of domestic chores. While the conventional use case for such remote support applications concentrates on industrial factories (cf. Stifter 2016, p. 8), it can be similarly used in a domestic context. This includes, for instance, the repair of bicycles, the assembly of furniture, failure or damage analysis of technical devices and components and similar tasks.

A typical situation could look like this: Someone does not know how to replace a flat tyre of their bicycle and uses a remote support app on their smartphone to call a friend or family member and show them the problem through the shared video stream of their smartphone’s camera. This other person can then guide them verbally through the repair process and, ideally, assist them by highlighting important tools or tasks to perform by drawing helpful indicators on top of the

camera feed. As a result, the person could be able to repair the flat tyre themselves, as opposed to having to take the bicycle to a repair shop.

2.12 Conclusion

It has been established that the WebRTC technology can be used in a telecommunications environment and because it is independent of specific operating systems there is a considerable benefit. In addition, the entire data transfer is securely encrypted, which is particularly compelling to companies concerned about the privacy of their data.

However, the limitations mentioned in this chapter still remain of serious nature, especially in a consumer environment. It is hardly feasible to coerce an end customer to switch to a certain web browser in order to use a web application. The same applies to a business context. While native mobile apps can provide a reliable solution for this problem, they also come with a significantly intensified effort, both in the development and maintenance stage.

On the whole, one potential compromise that can be considered both economic and user-friendly is the use of cross-platform apps. While they suffer from certain detriments with regard to user experience, the development effort is minimised and it is guaranteed that users will be able to use the app, on condition that they have a smartphone or tablet that is not older than five years.

This chapter discussed the architecture and technology behind WebRTC and compared its strengths and weaknesses. In this context, potential use cases were discussed and possibilities to use them in applications on mobile devices displayed. Based on this research it can be concluded that cross-platform developed apps might offer a practicable approach to achieve widespread dissemination on mobile devices by using a single codebase. The following chapter discusses cross-platform mobile development in detail along with possible options for cross-platform development frameworks.

3 Cross-platform mobile development

Cross-platform mobile development aims at providing developers with opportunities to write code for mobile applications once and run it on all platforms. In theory, this method results in significantly reduced development resources compared to native app development. However, it also includes some possible drawbacks, such as diminished user experience (cf. Angulo & Ferre 2014).

This chapter provides an overview of cross-platform mobile development. It discusses the advantages and limitations of this technology along with its different approaches and presents several possible framework options to simplify the mobile development process.

3.1 Background

Beginning with the introduction of Apple's iPhone back in 2007, mobile applications have become massively popular. Back then, it was self-evident to implement an app natively because there were no other feasible options. However, the following years have seen a substantially increasing number of popular mobile platforms, such as Android, Windows Phone and the aforementioned iOS. Since all these platforms use different programming languages, there was no possibility to reuse the programming code written for one platform. As a result, it had to be rewritten in the exact same way for all platforms that should be supported. Furthermore, making changes to an app again meant going through all platforms and implementing the changes separately for each platform, which results in increased development effort and expenses (cf. Xanthopoulos & Xinogalos 2013).

This problem has led to a rethink in application development. A common approach nowadays is to use an API for retrieving and manipulating the underlying data. This offers the benefit that there is only one standardised way of interaction that various application types, such as a web application, a desktop application and mobile apps can use (cf. Azevedo, Lopes Pereira & Chainho 2015). However, each application type still needs to develop the business logic individually.

Another solution to this problem is cross-platform mobile development. It enables developers to write code for an app only once and subsequently use it from that code base in applications for all desired platforms. In most cases, the code is written using web technologies, such as HTML5, JavaScript and CSS. Incidentally, as Charland & LeRoux (2011) point out, it was the original plan for iPhone apps to be

written using these tools. In the end, however, Apple decided that third-party apps for their operating system have to be written natively in Objective-C, which was followed by Swift in 2014.

A common approach used in this context is Model-Driven Development (MDD). In MDD, the business logic is separated from the technical implementation and a model is used to describe the problem domain (cf. Charkaoui, Adraoui & Benlahmar 2014). To achieve that, the functionality of the application and how it behaves under different circumstance is described using a modelling language (cf. Xanthopoulos & Xinogalos 2013). A variation of this approach is used for generating cross-platform mobile apps, which is discussed in Chapter 3.5.4 in more detail.

3.2 User experience

One area that was severly influenced by the spread of smartphones over the last years is user experience, a term that refers to a person's overall experience when using a software application or system (cf. Mullins 2015). Charland & LeRoux (2011) define two primary categories for user experience: The context and the implementation of the application or system. Both categories are described in more detail in the following section.

3.2.1 Context

The context of the application consists of components which need to be understood by the user, such as platform-specific user interface conventions. These conventions can differ across mobile platforms. Charland & LeRoux (2011) describe a common functionality that is implemented differently on Android and iOS: the ability to go back to a previous view inside an app. On Apple devices for instance, there are virtual buttons that provide this functionality, typically in the top left corner of the screen. The majority of Android devices, on the other hand, are equipped with a physical back button in the lower right area of the device. In cross-platform apps, developers need to consider these conventions, as users of a mobile platform might expect an app to behave in a way they are used to from native apps (cf. Charland & LeRoux 2011). If it is crucial that the user interface components align with the respective platform's conventions, developers can use interpreted or generated cross-platform apps, which are addressed in more detail in Chapters 3.5.3 and 3.5.4.

3.2.2 Implementation

The implementation of the application involves all elements “that can be controlled in an application” (Charland & LeRoux 2011). This part is solely the responsibility of the developer and includes the handling of input errors, the graphical layout of an application and additional helpful features that improve the app’s workflow. With smartphones it is possible, for instance, that an application behaves differently based on the location it is used in. An app that offers information about restaurants could use the device’s GPS position to automatically show high-rated restaurants nearby. Another simple example for a valuable user experience is a chat app that lets users select contacts from the smartphone’s address book to start chats, without having to manually add them to the app. These examples enhance the user’s experience while interacting with the app by fulfilling basic tasks automatically.

In some scenarios, traditional web apps struggle to provide a similar user experience in comparison to native apps: For instance, of the two examples described previously it is only possible to determine the user’s GPS position in the browser, after getting explicit consent from the user for this action. There is, however, no possibility to access the contact list of the device inside a traditional web app, even if it is running on a smartphone. The same is true for accessing the phone’s built-in calendar app or issuing push notifications about events of interest to the user, like a new chat message.

Cross-platform developed apps offer developers the possibility to introduce a similar level of user experience to web apps. In most cases, they can access native device features through a special API of the cross-platform development framework. These features typically include the previously mentioned contact list, the calendar and geolocation as well as usage of native dialog elements, the device’s battery status, vibration control, network information and the motion of the device. During app execution, the framework then calls the respective native methods depending on which operating system the app is run on.

3.3 Advantages

A significant advantage of cross-platform mobile development is reduced development time and costs. This is due to its approach of “write once, run anywhere” (cf. Xanthopoulos & Xinogalos 2013). Angulo & Ferre (2014) identify

several further implications of this circumstance, for instance the reusability of developer skills and one common shared codebase to work on. With cross-platform mobile development, it is also considerably simpler to synchronise app releases. Furthermore, the advantage of diminished development effort affects all stages in the product life-cycle, for all subsequent app updates or maintenance works.

Additionally, cross-platform mobile development provides developers with a simple possibility to achieve a uniform app “look and feel” easily across all supported operating systems. Although users generally “seek apps that resemble native apps” (Heitkötter, Hanschke & Majchrzak 2012) in terms of user interfaces, it can be an advantage to have an app that looks and behaves the same way on all platforms, especially if the users are already thoroughly familiar with the functionality, for instance from an existing web app. Facebook, for example, deliberately decided to ignore certain platform-specific interaction conventions when they developed the mobile apps for their web application in order to provide users with interaction paradigms they were familiar with (cf. Angulo & Ferre 2014).

Another benefit of cross-platform mobile development stems from the fact that there is less platform-specific special knowledge necessary when developing an app with HTML5 technologies (cf. Xanthopoulos & Xinogalos 2013). In general, HTML5 can be considered as easier to learn than native app development, since in most cases it is not necessary to study specific platform development practices, which for instance include the build process for compiling the app, proper memory management, user interface conventions and a multitude of devices that might behave differently under certain scenarios, especially in the Android environment.

As Charkaoui, Adraoui & Benlahmar (2014) further point out, cross-platform developed apps can be downloaded from its respective platform’s app market place, such as the Android Play Store or the iOS AppStore. This is a considerable advantage since it provides a single point of access for the users of the platform where they can obtain the applications. Cailean (2013) labels this characteristic as the “discoverability” of an app. In fact, traditional web apps that run solely in a browser do not share this trait.

3.4 Limitations

Cross-platform developed apps commonly do not exhibit the same performance measures as native apps do. One reason for this is that native apps are usually

compiled, which in most cases results in faster execution times because the programming code is translated into machine code before the execution of the program. Cross-platform developed apps, on the other hand, mostly use interpreted languages such as JavaScript, which execute its code instructions step-by-step without compiling them first (cf. Charland & LeRoux 2011). This circumstance particularly affects hybrid apps, since their user interfaces are not utilising the optimised native components (cf. Xanthopoulos & Xinogalos 2013). This does not apply to other types of cross-platform mobile developed apps, such as generated apps and, to some extent, interpreted apps.

Apart from that, cross-platform developed apps are not capable of directly accessing device-specific features, such as the contact list, file system and various sensors. The communication to this underlying hardware is achieved by using an API layer (cf. Singh & Buford 2016). For developers, this process can be inconvenient and might not provide results with the same accuracy compared to native app development. Because of the additional API layer between the application and the device hardware, the execution of the app's business logic might be further slowed down.

The fact that there is no standardised interface for the API that bridges the communication between the native operating system and the cross-platform developed app has additionally led to the situation that each framework has implemented their own version of such an API (cf. Charland & LeRoux 2011). This can be inconvenient for developers because although there are few differences in terms of the functionality, the programming syntax can vary widely. As a result, switching from one cross-platform development framework to another can be an arduous task.

Another detail to consider when developing cross-platform mobile apps is that it might not be enough to simply wrap an existing web page inside the web view of a native app to gain approval of Apple's reviewing team to include the app in the App Store. According to Xanthopoulos & Xinogalos (2013), it is possible that such an app will be rejected by Apple's reviewing team. In its official review guidelines, Apple clearly states that apps will be rejected if they are "simply web sites bundles as apps"[12]. Consequently, it is necessary that an app provides some sort of additional

[12] https://developer.apple.com/app-store/review/guidelines/ [1 June 2016]

value in comparison to a simple web page. This could be achieved, for instance, by enhancing the app's functionality by integrating the device's contact list or calendar to provide supplementary functions and enhance the user experience.

3.5 Approaches

Xanthopoulos & Xinogalos (2013) define four different categories for cross-platform developed apps: Web, hybrid, interpreted and generated apps. All four approaches are thoroughly discussed in this section.

3.5.1 Web apps

Web apps are applications that run within a web browser. Typically, they use HTML5 and JavaScript. The advantage of web apps is that nowadays, almost any smart mobile device has a web browser installed, thus providing a broad range of dissemination. One disadvantage of web apps is the limited access to the device's sensors, file system and features like contact list and calendar. Native apps, on the other hand, can exploit the device's full potential when it comes to these features.

Unlike native apps, web apps do not need to be physically installed on the device and, furthermore, also do not have to be upgraded when a new version is available. At the same time, this becomes a disadvantage when users are not connected to the internet. In this case, the web app is not accessible to the user (cf. Xanthopoulos & Xinogalos 2013). There are modern HTML5 technologies like the Application Cache (AppCache) to eradicate this problem. AppCache allows developers to store programming logic and data on the user's device. However, this technology requires substantial additional programming effort (cf. Xinogalos, Psannis & Sifaleras 2012).

3.5.2 Hybrid apps

Hybrid apps are a combination of native apps and web apps. They are "primarily built using HTML5 and JavaScript, and a detailed knowledge of the target platform is not required" (Xanthopoulos & Xinogalos 2013). The essential difference to web apps is that they run within a native app container. The code is still executed by a web browser, but can be bundled together with the application, thus removing the necessity of an active internet connection to download the programming logic. With hybrid apps, it is also possible to access the device's special features through APIs provided by the cross-platform development framework (cf. Xanthopoulos & Xinogalos 2013).

3.5.3 Interpreted apps

Interpreted apps use pre-defined commands to build the user interface with native components when the app is started. This means that on the Android platform users will interact with typical Android-styled buttons, while on iOS users will interact with iOS-styled buttons, without any additional effort of the developer. Despite this advantage in user experience, the developer is completely dependent on the used framework. This could especially pose a problem when a new version of an operating system is released, because it is not clear if the app will automatically have access to new features or if all previously used components will look and behave the same way (cf. Xanthopoulos & Xinogalos 2013).

3.5.4 Generated apps

This type of cross-platform developed apps use the code to generate native apps from it. The main difference to interpreted apps is the fact that they are generated before the app is started. They benefit from a high overall performance due to the use of compiled native code. One downside of generated apps is the increased build time that has to be carried out each time a change is made to the app (cf. Xanthopoulos & Xinogalos 2013).

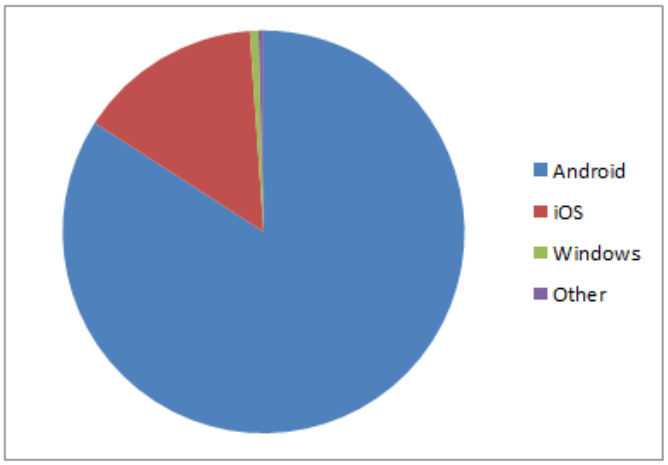
3.6 Criteria for choosing a framework

In order to determine a suitable cross-platform development framework, it is necessary to define a set of criteria that a framework needs to fulfil to be considered a useful option for developing a mobile app that uses WebRTC.

3.6.1 Support for mobile platforms

A significant factor for selecting a cross-platform mobile development framework is the support of mobile platforms (cf. Angulo & Ferre 2014). Ideally, such a framework should support a variety of platforms, and most importantly include the ones that offer the highest dissemination. According to recent statistics (cf. Statista 2016), the three most popular mobile platforms, namely Android, iOS and Windows Phone, represent 99.6% of the global smartphone operating system market share (see Figure 8). Consequently, developers can assume that their application reaches a vast majority of available smartphones if a framework supports these three platforms.

Android	84.1%
iOS	14.8%
Windows	0.7%
Other	0.4%



Source:
http://de.statista.com/statistik/daten/studie/73662/umfrage/marktanteil-der-smartphone-betriebssysteme-nach-quartalen/ [4 June 2016]

Figure 8: Global market share of smartphone operating systems in the first quarter of 2016

3.6.2 Access to device-specific features

Angulo & Ferre (2014) argue that a fundamental feature of a cross-platform mobile development framework is its ability to offer access to underlying device-specific features, such as the current Global Positioning System (GPS) position, the address book, the calendar and various other sensors like the accelerometer and the device orientation. These features are commonly provided by an API. It is essential that a framework offers a similar level of functionality compared to native app development, although the program execution is likely to be slower in comparison since the access to the features happens over the API of the framework.

3.6.3 Legal background

Another important criterion for choosing a cross-platform mobile development framework is the legal background. This includes the question whether the framework is distributed as free software or if there are usage fees, which can be further categorised into one-time and recurring payments. Furthermore, a crucial issue is the license under which the framework is published. This determines the fact whether a developer is allowed to create commercial software products with the framework without additional fees, which could be essential especially for companies that want to sell software products created with the framework (cf. Heitkötter, Hanschke & Majchrzak 2012).

3.6.4 WebRTC capability

Because this thesis aims at developing a WebRTC-capable cross-platform developed app, an essential criterion in this case is that a framework supports WebRTC. Ideally, it should possible to use the same JavaScript programming logic for creating peer connections as in browser-based web applications, without having to add method prefixes for a plugin. This would minimise the development effort for creating a cross-platform developed app when a fully functional web application already exists.

3.6.5 Side issues

Angulo & Ferre (2014) further define some less important side issues, for instance if the framework in question can be integrated with a widespread Integrated Development Environment (IDE) or even includes an own solution for such an environment. Furthermore, the programming language that the framework uses for developing cross-platform apps could be of interest for companies, which might favor a framework that uses a well-known programming language in order for new developers to be become easily acquainted with the project. Lastly, it could be beneficial if the framework offers an integrated solution for automatically building and distributing apps to further minimise deployment times (cf. Heitkötter, Hanschke & Majchrzak 2012).

3.7 Cross-platform development frameworks

Over the last years, a multitude of cross-platform development frameworks has emerged. This section gives a brief overview of some of the most popular frameworks and their particular characteristics.

3.7.1 Apache Cordova (PhoneGap)

One of the most popular cross-platform development frameworks is Apache Cordova[13]. Apps built with this framework belong to the category of hybrid apps. Developers can write applications with HTML5, JavaScript and CSS files, which are then be executed inside a native application in a web view. Due to the fact that these tools are also used to develop web applications, this framework offers a relatively comfortable entry point into cross-platform mobile development. Access to underlying features such as sensors and file system is provided via an API, the

[13] https://cordova.apache.org/

Cordova Plugins. Apache Cordova provides support for numerous platforms, such as Android, iOS, Windows Phone, Blackberry and Ubuntu.

Apache Cordova is open-source, although its owner, the software company Adobe, also released a similar version of it called PhoneGap[14]. PhoneGap is built on the same core application as Apache Cordova, but is part of a product package that also offers various additional tools, for instance a desktop application, a build tool and a variety of third-party libraries and plugins.

3.7.2 Xamarin

Xamarin[15] is another popular framework that builds generated native apps. Instead of web technologies it uses the programming languages C# or Ruby. Xamarin apps use native user interface components, thus providing app users with well-known interaction tools. It supports the most popular operating systems, namely Android, iOS and Windows Phone and also offers a native API to access device sensors. Xamarin also offers additional services such as an automated build tool.

3.7.3 Appcelerator Titanium

Appcelerator Titanium[16] is an example for a framework that creates interpreted apps, which means that apps created with this framework will use native user interface components. However, it also features some aspects from hybrid apps by providing developers with the possibility to write reusable modules in JavaScript.

Titanium is one product of the Appcelerator platform, together with tools like *Arrow*, which is a framework for easily building APIs or *Push*, which is a pre-built service for push notifications that can be integrated into apps. Furthermore, Appcelerator provides a multitude of analytics tools. It has to be noted that while Titanium itself is open-source and free-to-use, all other previously described tools from Appcelerator are only available in paid plans.

3.7.4 Ionic

Ionic[17] is a relatively new cross-platform mobile development framework that relies heavily modern web technologies like AngularJS[18], Sass[19] and virtual Document

[14] http://phonegap.com/
[15] https://www.xamarin.com/
[16] http://www.appcelerator.com/titanium/titanium-sdk/
[17] http://ionicframework.com/
[18] https://angularjs.org/
[19] http://sass-lang.com/

Object Model (DOM) rendering for data-intensive apps with rapidly changing user interfaces. It is built upon Apache Cordova. Ionic is an open-source project and its entire source code can be found on Github, where users are also able to report bugs or suggest improvements to the code.

3.7.5 Sencha Touch

Sencha Touch[20] focuses primarily on creating user interfaces. It has special features to simulate user interface components from Android and iOS within a web application. It does not, however, provide tools to build native apps. In order to use the code in a native app, it is either necessary to create a blank native app containing a web view for each platform the app should run on or use another framework like the previously mentioned Apache Cordova to fulfil the task.

3.7.6 Other frameworks

There are also a number of smaller, lesser known cross-platform mobile development frameworks. These include jQuery Mobile[21], Mobile Angular UI[22], Kendo UI[23] or the lightweight app.js[24]

In conclusion, this chapter highlighted important aspects of cross-platform mobile development. User experience was identified as a crucial factor in mobile application development and some examples were provided on how it can be applied to cross-platform developed apps to provide convenient additional features in comparison to traditional web apps. Afterwards, the advantages and limitations of cross-platform mobile development were discussed along with framework options and the approaches they use and a set of criteria which they need to fulfil in order to enable programmers to develop a WebRTC-capable cross-platform app. The following chapter describes the process of integrating WebRTC into a cross-platform developed app.

[20] https://www.sencha.com/products/touch/#overview
[21] https://jquerymobile.com/
[22] http://mobileangularui.com/
[23] http://www.telerik.com/kendo-ui
[24] http://code.kik.com/app/2/index.html

4 App development

Considering the findings of the underlying research, the following step is the actual creation of cross-platform developed apps that feature WebRTC functionality. For this purpose, the two promising options mentioned in Chapter 2.9.4, Crosswalk and OpenWebRTC were used to develop a prototypical app, which are oriented towards the reference use case of domestic remote support outlined in Chapter 2.11. Afterwards, these two apps were evaluated in order to determine their viability to be used in WebRTC-capable applications. The results of this evaluation are presented in Chapter 5. Additionally, the same application logic was integrated into a native Android app containing a web view, in order to enable the possibility to compare the results to those of the two cross-platform apps.

4.1 Introduction

A WebRTC application commonly consists of two parts: First, a management server that is responsible users to discover each other and assist in the creation of peer connections between them. Second, a user interface that displays local and remote video streams and enables users to interact with each other, for instance start a call or send text messages. This user interface can either be a web application running in a browser, or a mobile app, which is in this case a cross-platform developed app.

4.2 Previous work

Stifter (2016) implemented a browser-based real-time remote support application that uses WebRTC for peer-to-peer communication. This application consists of a management server written in Node.js and a web application using HTML5 web pages for the user interface. A significant advantage of this technical setup is the fact that there is only one programming language, JavaScript, involved, which simplifies development and facilitates reuse of code components between frontend and backend (cf. Stifter 2016, p. 35).

With this application, users can start peer-to-peer audio and video conferences with each other. The video feed can be paused if one user wants to take a closer look at the video image, which might be difficult with a moving video feed. Additionally, they are able to exchange plain text messages, which can be useful in case one person is located in a loud environment where it is difficult to understand the other person. A special feature of the remote support application is that users are able to assist each other by drawing coloured indicators on top of the video feed. This is especially

helpful when one person has more experience with the matter at hand than the other person.

Since the aim of this thesis is to evaluate possibilities of using WebRTC in cross-platform mobile apps, the previously described existing management server from Stifter will be used for the signalling part. The user interface will be realised with cross-platform apps. In the end, it should be possible to start video calls from the existing web application to the cross-platform apps and vice versa. Furthermore, the same should be possible from app to app.

4.3 Necessary adjustments

In order to successfully communicate with the management server, it is first necessary to open a secure WebSocket connection. This is achieved by calling the URL of the server with the “wss://” protocol prefix instead of the “https://” used when requesting a web page. When the WebSocket connection has been successfully established, the management server responds with a welcome message that contains the new client’s unique ID. This ID will further be used to start peer connections between users.

Each time a new user connects to the management server, a message is broadcast to all connected clients in order to inform them about the available peers. This process is repeated when a user disconnects from the server, for instance by closing the application. The application is then responsible for providing a user interface that enables users to start calls with connected peers. When one user decides to call another, this is achieved by sending a call message with containing the ID of the collocutor to the management server. The server then relays this message to the peer with the matching ID (cf. Stifter 2016, p. 36). After that, the two peers start the signalling process – described in Chapter 2.6 – by exchanging their respective session description offers and answers, which results in the creation of a shared peer connection. Additionally, a data channel is opened to enable the two users to send text messages to each other. This data channel is also used for exchanging the drawing path information for the overlay indicator feature.

Due to the fact that the web application from Stifter (2016) was written in JavaScript, large portions of the existing code could be reused inside the cross-platform app with only small adaptations necessary to facilitate a working signalling communication between management server and app. As a result, the main task of

the app development was to ensure a functioning integration of the cross-platform development framework and the plugins that enable the WebRTC functionality.

4.4 Additional features to enhance user experience

As mentioned in Chapter 3.2, a substantial advantage of mobile apps is the circumstance that they can deliver significantly enhanced user experience compared to web apps. Furthermore, it might not be enough to simply wrap an existing web page inside a web view in order to be granted access to the iOS App Store, as Chapter 3.4 pointed out. Therefore, it was decided to implement two additional features in order to create an appealing user experience for the cross-platform developed apps.

4.4.1 Contact list

First, the Apache Cordova contact plugin (“cordova-plugin-contacts”) was used to enhance the information displayed about available users. The feature retrieves all entries from the device’s contact list and tries to match the first and last names with the usernames provided by the management server. If a match is found, additional information about this contact is displayed in the available user section. This includes this user’s phone numbers and address. Consequently, it is possible that users of the cross-platform app start a phone call directly from the app, for instance when the internet connection has been lost and no WebRTC video streams are possible. Furthermore, app users could start navigation to the address of the other user if there are issues to be discussed in person. The technical implementation of the contact retrieval process is outlined in Listing 3.

It is important to note that this feature does not pose a threat with regard to data privacy, since only information can be displayed that has been entered manually into the contact list by the user. It is not possible to access or retrieve information from the other person’s device.

Considering the reference use case mentioned in Chapter 2.11, this feature enables users to directly call a friend or family member if they need help while working on a domestic chore, such as putting together pieces of furniture or repairing a bicycle or car. The possibility to call these acquaintances without the necessity of them having the app opened could further enhance the usefulness of this feature. Consequently, some form of push message service needs to be added to the app’s functionality in

order to provide the opportunity to deliver messages to users regardless of their presence inside the application.

```
01.   // define contact find options
02.   var options = new ContactFindOptions();
03.   options.filter = collocutor.username;
04.   options.multiple = true;
05.   options.desiredFields = [navigator.contacts.fieldType.id,
      navigator.contacts.fieldType.displayName, navigator.contacts.fieldType.name,
      navigator.contacts.fieldType.phoneNumbers];
06.   options.hasPhoneNumber = true;
07.
08.   var fields = [navigator.contacts.fieldType.displayName,
      navigator.contacts.fieldType.name];
09.
10.   // start finding contacts process
11.   navigator.contacts.find(fields, function(contacts) {
12.           // success callback
13.
14.           // process contacts array and display the desired information
15.           // e.g. full display name, phone numbers to enable
16.           // starting a phone call directly from the app or the address
17.           // of the contact to start a navigation
18.           addContactInformationToUserList(parseContacts(contacts));
19.       }, function(error) {
20.           // error callback
21.           console.error("Error finding contacts", error);
22.   }, options);
```

Listing 3: Retrieve information for another user from the contact list of the device

4.4.2 Battery status

Another useful feature in the context of video streaming are notifications about the device's battery status. In the implementation of the apps, users are notified if they intend to start a video stream when the battery status drops beneath a certain threshold, which usually varies across different device types, but typically ranges within fifteen to twenty percent of the total battery charge. As a result, users are informed about the low battery status beforehand, and are less likely to use the app until the battery is completely drained.

For instance, information about the device's battery status can be easily obtained using Apache Cordova's plugin "cordova-plugin-battery-status". It adds three events

to the app's *window* object. First, *batterystatus*, which returns an object containing the remaining percentage of the battery's charge available as well as information whether the device is plugged in or not. This event is called every time the percentage value changes. Second, *batterylow*, which is called when the battery charge drops beneath a device-specific threshold value. Third, *batterycritical*, which works in the same way as *batterylow*, but means that the device should be charged immediately.

Listing 4 provides a simple example of how a user can be informed about the device's low battery status using a built-in notification inside the app. Additionally, it would be possible to further enhance the programming logic using this information, for instance by automatically requesting a reduced video resolution and frame rate in order to prolong the battery's performance when dropping below an adjustable threshold value.

```
01.    var onBatteryStatusLow = function(status) {
02.        var n = new Notification({
03.            title: "Battery low",
04.            content: "Battery status at " + status.level + "%",
05.            type: "warning"
06.        });
07.
08.        // issue notification to screen
09.        n.notify();
10.    };
11.
12.    // listen for battery low events
13.    window.addEventListener("batterylow", onBatteryStatusLow, false);
```

Listing 4: Example of informing the user about a low battery status inside the application

4.5 Implementation of cross-platform mobile apps

This section briefly describes the implementation process of the two WebRTC-capable cross-platform frameworks, Crosswalk and OpenWebRTC. Their technical backgrounds and histories were mentioned in more detail in Chapter 2.9.4.

4.5.1 Crosswalk

In order to use the existing web application programming logic in a cross-platform developed app with Crosswalk, the first step is to create a blank Apache Cordova

project and add the Crosswalk plugin, “cordova-plugin-crosswalk-webview”. This ensures that the foundational browser engine of the app’s web view will be the latest version of the Google Chrome browser. Furthermore, this provides developers with additional methods to test their applications, since they can be certain that the web view environment functions in the same way as Google Chrome. It also enables them to test their applications with well-known browser testing tools such as Selenium[25].

The inclusion of the Crosswalk Cordova plugin enables the use of the entire WebRTC components and functionality like it is possible in Google Chrome. For the implementation of the reference use case in Crosswalk, there were no extensions to the existing programming code of the web application necessary, except for the adding the Cordova initialisation wrapper file, which contains a collection of events that are executed during different stages of the app lifecycle. For instance, such events include the online or offline state of the app, which enables developers to react to changing network conditions and various common events such as “load” or “deviceReady”.

4.5.2 OpenWebRTC

Unlike Crosswalk, OpenWebRTC is not a complete cross-platform development framework, but rather a *client* framework. In other words, it is always necessary to create a native app for each mobile platform that should be supported in order to use OpenWebRTC. These apps need to contain the OpenWebRTC library source files and at least a web view to wrap the web application code. It is also possible to create full native apps with OpenWebRTC. The WebRTC functionality is provided by native libraries that have to be integrated with the native apps. OpenWebRTC offers libraries for Android and iOS, but not Windows Phone.

The OpenWebRTC iOS helper library yields a simple opportunity to use WebRTC inside a hybrid iOS app. As mentioned in Chapter 2.9.3, it is not possible to achieve that with plain iOS tools. OpenWebRTC enables this functionality by providing a custom “OpenWebRTCWebView” class that is extended from iOS’s built-in web view class, “WKWebView”. Developers can decide between packaging the complete application code inside the web view, or using it only for the video display and WebRTC connection handling. In the latter case, it is possible to use native user

[25] http://www.seleniumhq.org/

interface components for controlling the application. Consequently, it is necessary to connect user input, such as clicks on buttons, to the respective JavaScript function calls. While this increases the development efforts, it could also be an opportunity to leverage the operating system's native capabilities and offer native user experience to the users.

The same functionality can also be achieved on the Android platform, although technically it is not necessary to use OpenWebRTC to obtain access to the WebRTC technology for devices using Android version 4.4 or above. However, in order to ensure the WebRTC functionality inside the app, it is advisable to use an appropriate framework to gain predictability of the app's way of functioning (cf. Hart 2015). For the implementation of the reference use case, it was sufficient to add the web view provided by OpenWebRTC to achieve the desired real-time communication functionality.

4.6 Implementation of web app in web view

In order to create a frame of reference for the cross-platform apps, one native Android app was created that contains only a single web view. The entire web application source code was packaged into this web view. It has to be noted that this procedure was only possible for Android, since there is currently no option to use WebRTC inside a plain iOS web view.

The setup of the native Android app with one web view could be realised within thirty minutes. Because the web application code requests access to the camera using navigator.getUserMedia(), it was necessary to use Android's WebChromeClient class for the initialization of the web view. As shown in Listing 5 in line 15, it is necessary to override the WebChromeClient's onPermissionRequest method. In line 20, a check is performed if the requesting resource equals the local HTML source file. Consequently, no other web page is allowed media access on the device. In contrast to the procedure inside a real web browser, where the user is required to manually grant access to the request, in this case the developer has to specify in advance which resources will be allowed to use the camera and microphone, without knowledge of the user of the device.

```
01.  // index file of web app
02.  private static String LOCAL_FILE = "file:///android_asset/webrtc.html";
03.
04.  private WebView webView;
05.
06.  @Override
07.  protected void onCreate(Bundle savedInstanceState) {
08.      super.onCreate(savedInstanceState);
09.      setContentView(R.layout.activity_main);
10.
11.      webView = (WebView) findViewById(R.id.webView);
12.      webView.setWebChromeClient(new WebChromeClient() {
13.          // Handle the permission request inside the web application
14.          @Override
15.          public void onPermissionRequest(final PermissionRequest request) {
16.              MainActivity.this.runOnUiThread(new Runnable() {
17.                  @Override
18.                  public void run() {
19.                      // The request is only granted for our local file
20.                      if(request.getOrigin().toString().equals(LOCAL_FILE)) {
21.                          request.grant(request.getResources());
22.                      }
23.                      else {
24.                          request.deny();
25.                      }
26.                  }
27.              });
28.          }
29.      });
30.
31.      // load the web application into the web view
32.      webView.loadUrl(LOCAL_FILE);
33.  }
```

Listing 5: Handling of camera and microphone access request with navigator.getUserMedia() inside an Android web view

4.7 Insights

A valuable insight during the cross-platform development process was that unlike desktop computers and laptops, smartphones are commonly equipped with multiple media devices, such as cameras and microphones. Thus, it is helpful to provide users with the possibility to select the desired media input. Especially for video conferencing, users might want to switch the camera mode between front-facing for

face-to-face communication and back-facing for sharing the camera feed with another person.

Listing 6 describes a simple example of how an application obtains a list of all attached media devices. In order to enable the user to select a specific camera, the content of the *videoDevices* array could be displayed inside a select box. Upon selection of an option, the device ID from line 10 could be passed inside the constraints object of the navigator.getUserMedia() call, resulting in the selection of the desired camera. Listing 7 below in line 8 provides insight on how this device ID can be used to request a specific camera.

```
01.  // arrays for storing the media devices
02.  var audioDevices = [];
03.  var videoDevices = [];
04.
05.  navigator.mediaDevices.enumerateDevices()
06.      .then(function(devices) {
07.          // iterate over all devices
08.          devices.forEach(function(device) {
09.              // internal ID of the device - this can be used to request a
     specific device inside the navigator.getUserMedia() constraints
10.              var id = device.deviceId;
11.
12.              // label of the device (e.g. "HP HD Webcam") - this can be used to
     display to the user
13.              var label = device.label;
14.
15.              switch (device.kind) {
16.                  case "audioinput":
17.                      audioDevices.push(device);
18.                      break;
19.
20.                  case "videoinput":
21.                      videoDevices.push(device);
22.                      break;
23.
24.                  default:
25.                      // unknown device kind
26.              }
27.          });
28.      );
```

Listing 6: Determine all available media devices connected to the physical device

Furthermore, developers can provide additional user experience in their apps by considering changing network conditions and react to them in a sensible way. During the WebRTC connection setup, for instance, it is possible to request lower video quality and framerates when a device is located inside a network with low bandwidth. Additionally, the app could automatically try to re-establish a PeerConnection if it was interrupted due to network loss (cf. Singh & Buford 2016).

An example of how lower video quality can be requested using navigator.getUserMedia() is outlined in Listing 7. In line 7, an additional optional parameter array is passed to the video access request, containing special attributes such as a specific camera and a video resolution of 640 x 480 pixels at most, thus limiting the data transfer for the video stream.

```
01.  // device ID from the navigator.mediaDevices.enumerateDevices() call
02.  var deviceId = "...";
03.
04.  var constraints = {
05.      audio: true,
06.      video: {
07.          optional: [
08.              { sourceId: sourceId },
09.              { width: { max: 640 } },
10.              { height: { max: 480 } }
11.          ]
12.      }
13.  };
14.
15.  // request access to the media devices (success and error callbacks omitted for
     brevity)
16.  navigator.getUserMedia(constraints, success, error);
```

Listing 7: Example of requesting media devices with specific attributes

This chapter summarised the development of the two cross-platform apps using a framework along with one implementation of a hybrid Android app without the help of a framework and described the insights and findings of this process. The following chapter evaluates these three apps considering a list of criteria of noteworthy characteristics.

5 Evaluation

Upon completion of the app development phase, the three created apps were evaluated in due consideration of a set of criteria related to the overall app development process. This set of criteria was constructed under consideration of the overall criteria for cross-platform development (see Chapter 3.6) along with the findings of the research on usage possibilities of WebRTC on mobile devices (see Chapter 2.9).

It has to be noted that the used evaluation criteria are not applicable to cross-platform development in general, but should rather provide guidelines in context of using it in combination with WebRTC. In the end, developers need to choose the parameters for their use case with regard to their respective requirements.

5.1 Method

There were a total of three different methods used during the evaluation process: First, the CPU and memory consumption of the apps were measured. Second, the apps were assessed in accordance with the constructed assumptions. In a third step, a select group of test users was asked to rate the created apps with regard to their level of usefulness, i.e. if they could imagine using the app in their daily lives in a simple user test. In addition, they were asked to rate the overall “look and feel” of the apps in comparison to other familiar mobile apps that they use regularly.

5.2 Setup

This subsection describes the setup process for the three different methods used to evaluate the created apps.

5.2.1 Technical

The following hardware was used for the technical setup of the evaluation process: One Samsung Galaxy J5 smartphone running the Android 5.1.1 operating system, for measuring the CPU and memory consumption of the app. For the user tests, a Samsung Nexus 10 tablet running Android 5.1 was used in addition to the previously mentioned Samsung Galaxy J5 in order to test the real-time communication capability of the apps.

5.2.2 Constructed assumptions

The assumptions for the evaluation process were constructed considering the findings of the research process and, in particular, the reference use case described in Chapter 2.11.

- The app should be functioning on more than 90% today's smartphones
- The development effort should be minimised, i.e. it should not be necessary for the same app functionality to be developed multiple times for multiple platforms. If a framework is used, the setup process should not exceed more than one hour.
- Created apps should be distributable at will for the developer and not be subject to legal restrictions.
- The app should support the latest version of WebRTC.
- The created apps should have a similar Central Processing Unit (CPU) and Random Access Memory (RAM) consumption as a comparable native app, i.e. not exceed the metrics of the native app by more than twenty percent.
- There should be a possibility to utilise device-specific features such as GPS sensors, the file system and built-in applications like the contact list or the calendar.
- Potential users of the app should be able to navigate the apps without additional help and should perceive the created apps as helpful.
- It is advantageous if a framework provides additional tools to simplify the overall development process. This includes automated deployment and testing tools or supplementary apps that let developers try the current version of an app on a real device and update automatically when new changes are made.

5.2.3 User tests

In total, a group of eight users was asked to use the created apps in the context of a domestic remote support scenario as outlined in Chapter 2.11. Afterwards, they were asked to rate their perception of the app's usefulness in such a scenario on a binary scale ("I would not use this app in this case" versus "I would use this app in this case"). In addition, they were asked to rate app's overall impression and resemblance in comparison to native apps they use regularly on similar scale ("Usage of this app does not provide any of the experience familiar from native apps" versus "Usage of this application looks and feels like using a native app").

5.3 Results

Table 1 illustrates an overview of the entire evaluation results. For the technical part of the process, the apps created with the Crosswalk and OpenWebRTC frameworks exhibited a higher CPU and RAM consumption than the native counterpart with a web view, although the increase stayed within twenty percent of the native app's reference value.

The second category includes the constructed assumptions. In the first item of this category, the device dissemination, both Crosswalk and OpenWebRTC reached approximately 97 percent of today's smartphones. This metric includes all devices which run the Android, iOS or Windows Phone operating system as mentioned in Chapter 3.6.1, considering devices that are not capable of using the WebRTC technology due an outdated operating version, namely Android devices that use a version lower than 4.0[26]. Using the groundwork of Stifter (2016), the development efforts for each app implementation stayed within one hour. The setup process for Crosswalk needed around twenty minutes less than the one for OpenWebRTC, which be a result of Crosswalk's comprehensive online documentation, since it is a plugin of the popular Apache Cordova framework. This circumstance is also advantageous on the issue of accessing underlying device-specific features. Apache Cordova provides developers with a well-documented API for interacting with these features. With OpenWebRTC, on the other hand, developers need to implement this functionality inside the native wrapper application in Java code. The same applies when it comes to automated build and deployment tools, where Apache Cordova offers various instruments, while OpenWebRTC cannot provide such tools.

The third category, user tests, summarises the experiences of the eight test users of the cross-platform developed apps. Seven out eight found that they would use the Crosswalk app in the underlying reference use case. When using the OpenWebRTC app, six out of eight assessed the app as useful in this scenario. However, when asked if the experience while using the app was similar to native apps that the users were familiar with, only three out of eight concluded that the Crosswalk app matched this description and only two out of eight users for the OpenWebRTC implementation. This could be caused by the fact that except for the additional features mentioned in Chapter 4.4, no further measures were implemented to enhance the user experience of the apps.

[26] https://developer.android.com/about/dashboards/index.html [13 June 2016]

Framework	Crosswalk	OpenWebRTC	Android WebView
App type	hybrid app	hybrid app	hybrid app
CATEGORY			(for technical reference)
TECHNICAL			
CPU consumption	9%	8%	7%
RAM consumption	57,3 MB	54,8 MB	48,2 MB
CONSTRUCTED ASSUMPTIONS			
Device dissemination	ca. 97%	ca. 97%	ca. 82%
Created apps are freely distributable	yes	yes	yes
Development effort for implementation (incl. framework setup time)	40 minutes	60 minutes	25 minutes
Supports latest version of WebRTC	yes	only on software update	only on software update
Access to device-specific features (contact list, calendar, sensors, …)	yes	circuitous	circuitous
Includes automatic build and deployment tools	yes	no	
USER TESTS			
Test users would use the app in domestic remote support scenario	87,5%	75,0%	
Test users perceive the app usage experience as similar to a native app	37,5%	25,0%	

Table 1: Overview of evaluation results

5.4 Result matrix

In addition to the overall results overview, Table 2 summarises the results in order to provide a simple, appealing overview of the evaluation with regard to the assumptions constructed in Chapter 5.2.2.

Framework	Crosswalk	OpenWebRTC
Constructed assumptions		
The app should be functioning on more than 90% of today's smartphones	✓	✓
The development effort should be minimised e.g. not exceed more than one hour	✓	✓
Created apps should be distributable at will for the developer and not be subject to legal restrictions	✓	✓
The app should support the latest version of WebRTC	✓	~
The created apps should not exceed the CPU and RAM consumption of a similar native app by more than 20%	✓	✓
The app should be able to utilise device-specific features such as contact list, calendar, sensors…	✓	~
Test users would use the app in domestic remote support scenario	✓	✓
Test users perceive the app usage experience as similar to a native app	~	~
Additional tools to simplify overall development process, e.g. automatic build or deployment tools	✓	✘

Table 2: Evaluation results summarised in result matrix

In conclusion, the evaluation process showed a preference towards the Crosswalk framework. While it is evident that OpenWebRTC is also a suitable framework for using WebRTC in cross-platform developed mobile applications, the user tests and memory consumption values along with the answers to the constructed assumptions all resulted in a tendency towards Crosswalk.

The following chapter observes the current the state and possible future of WebRTC along with opportunities to extend the apps created in the development stage.

6 Outlook

In May 2016, the WebRTC technology turned five years old. Although there have been hardly any new technical additions to WebRTC lately, it still is a long way before it can be considered finished. The majority of the processes at the moment focus on raising the number of WebRTC-capable web browsers. Similarly, the underlying work on the prototype application is far from being finalised. There are numerous possibilities for further extending the current project. Some forecasts on technical extensions are mentioned in this chapter along with possible enhancements for the prototype.

6.1 The future of WebRTC

WebRTC evangelist Tsahi Levent-Levi assumes that Apple will introduce the first implementation of WebRTC into their Safari browser within the next year, around the end of 2016 (cf. Levent-Levi 2016). This is based on the fact that the development team of WebKit, which is Safari's rendering engine, have started adding WebRTC functionality. He further believes that there will be no advance notice from Apple concerning this matter, but rather an official announcement once the functionality is fully implemented into Safari. The same holds true for Microsoft, which have announced at the BUILD conference in April 2016 that in addition to their implementation of ORTC, which was mentioned in Chapter 2.8, they are also working on adding WebRTC as it is currently specified (cf. Levent-Levi 2016).

6.2 User management and authentication

The backend server that handles the WebRTC connection setup and distribution of information about available peers is currently not authenticating user requests. This does not mean that communication with the server is insecure, all requests to and from the server use HTTPS and are therefore encrypted with TLS. However, there is currently no user management system implemented, which would allow users to log into the application with conventional username and password combinations. For now, anyone who knows the Uniform Resource Locator (URL) of the application is able to use it.

To eradicate this problem, it is either possible to design and implement an authentication solution from the ground up or use an existing application like for

instance Passport[27], which is an authentication middleware for Node. With Passport, it is possible to use local username and password authentication as well as authenticating users with an authorization protocol like OAuth[28].

6.3 Multi-user sessions

At this time the application allows for any number of parallel peer-to-peer sessions, meaning that one session cannot contain more than two users. While this entails a number of advantages previously discussed in this thesis, it might sometimes be necessary to invite more than two users to a session. Especially in remote support environments it might be beneficial to get the opinion of another expert to solve certain problems.

Due to its peer-to-peer design, WebRTC only supports two users in one session. If three or more users want to participate in a session, one solution would be to use a Multipoint Control Unit (MCU) as a central communication point which handles the routing of audio and video streams between all participating parties. There are publicly available open-source solutions like Erizo[29] or Janus[30] which could perform this task without requiring considerable development efforts.

[27] http://passportjs.org/
[28] http://oauth.net/
[29] https://github.com/ging/licode/tree/master/erizo
[30] https://janus.conf.meetecho.com/

7 Conclusion

The advantages of WebRTC are undoubtedly compelling: it can be used directly in web browsers without any licensing fees, 100% of the communication path is securely encrypted and there is less network latency due to the peer-to-peer setup. Additionally, the risk of unwanted data fraud is significantly reduced since there are no third-party servers involved in the data transfer.

However, the circumstance of WebRTC still being in development is a severe limitation for developers. The same is true for the fact that currently only 61% of web browsers in Austria support WebRTC. An effective remedy for this problem would be the implementation of WebRTC in the browsers of the Microsoft and Apple, which would raise the support level to over 90%. While there are first signs that this could happen in the near future as mentioned in Chapter 6.1, there are no predictions on when WebRTC's internal implementation will be finished.

If developers want to utilise the advantages of WebRTC in mobile applications, a practical approach at the moment is to use a cross-platform app. As a result, a significantly larger device dissemination can be reached. The Crosswalk project offers a simple method for ensuring that the browser rendering engine inside a web view is compatible to the latest release of Google's Chrome browser. This circumstance removes a comprehensive level of concern for developers since it increases predictability with regard to the behaviour of the web view and simplifies the testing process. In addition, because it is a plugin of Apache Cordova, it comes with a comprehensive set of tools, which can simplify the entire development process substantially. This includes a large collection of other plugins, simple access to native device features and automated build and deployment tools.

While this technology offers a simple way of reducing development efforts and costs in comparison to native apps, special attention must be put towards the topic of user experience. Each mobile platform employs their own interaction conventions, which their users are familiar with. As a result, users have certain expectations with regard to the behaviour of apps, for instance how the navigation between different views inside the app works. These issues have to be addressed by the developer in order to prevent users to switch to a native app that offers similar functionality.

List of tables

List of figures

List of listings

List of abbreviations

AJAX	Asynchronous JavaScript And XML
API	Application Programming Interface
AppCache	Application Cache
BSD	Berkeley Software Distribution
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DOM	Document Object Model
DTLS	Datagram Transport Layer Security
GPS	Global Positioning System
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protcol
HTTPS	Hypertext Transfer Protocol Secure
ICE	Interactive Connectivity Establishment
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
IP	Internet Protocol
MCU	Multipoint Control Unit
MDD	Model-Driven Development
NAT	Network Address Translation
ORTC	Object Real-Time Communication
RAM	Random Access Memory
RPC	Remote Procedure Calls
SRTP	Secure Real-time Transport Protocol
STUN	Session Traversal Utilities for Network Address Translation
TLS	Transport Layer Security
TURN	Traversal Using Relays around Network Address Translation

URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WebRTC	Web Real-Time Communication
WLAN	Wireless Local Area Network
XML	Extensible Markup Language

Bibliography

Alvestrand H. 2011, *Google release of WebRTC source code*. Availble from: http://lists.w3.org/Archives/Public/public-webrtc/2011May/0022.html. [10 June 2016]

Angulo E & Ferre X 2014. 'A Case Study on Cross-Platform Development Frameworks for Mobile Applications and UX'. In *Proceedings of the XV International Conference on Human Computer Interaction* (Interacción '14). ACM, New York, NY, USA, , Article 27 , 8 pages. doi: 10.1145/2662253.2662280

Azevedo J, Lopes Pereira R & Chainho P 2015, 'An API proposal for integrating Sensor Data into Web Apps and WebRTC'. In *Proceedings of the 1st Workshop on All-Web Real-Time Systems* (AWeS '15). ACM, New York, NY, USA, Article 8. ISBN: 978-1-4503-3477-8. doi: 10.1145/2749215.2749221

Cailean I 2013, *Why native apps remain unrivalled by web apps in user experience and discoverability*. Available from: http://www.trademob.com/why-native-apps-remain-unrivalled-by-web-apps-in-user-experience-and-discoverability/ [6 June 2016]

Charkaoui S, Adraoui Z & Benlahmar EH 2014. 'Cross-platform mobile development approaches'. In *2014 Third IEEE International Colloquium in Information Science and Technology (CIST)*, Tetouan, 2014, pp. 188-191. doi: 10.1109/CIST.2014.7016616

Charland A & LeRoux B 2011. 'Mobile Application Development: Web vs. Native'. *Queue* 9, 4, Pages 20 (April 2011), 9 pages. doi: 10.1145/1966989.1968203

Dhungana D, Falkner A, Haselböck A & Schreiner H 2015. 'Smart factory product lines: a configuration perspective on smart production ecosystems'. In *Proceedings of the 19th International Conference on Software Product Line* (SPLC '15). ACM, New York, NY, USA, 201-210. doi: 10.1145/2791060.2791066

Grégoire, JC 2015, 'On Embedded Real Time Media Communications'. In *Proceedings of the 1st Workshop on All-Web Real-Time Systems* (AWeS '15). ACM, New York, NY, USA, Article 7 , 4 pages. ISBN: 978-1-4503-3477-8. doi: 10.1145/2749215.2749224

Grigorik I, 2013, *High Performance Browser Networking*, 1st edn., O'Reilly Media, Sebastopol. ISBN: 978-1-449-34476-4.

Hart C, 2015, *Developing mobile WebRTC hybrid applications*. Available from: https://webrtchacks.com/webrtc-hybrid-applications/. [29 May 2016]

Heitkötter H, Hanschke S & Majchrzak TA 2012, 'Comparing Cross-Platform Development Approaches for Mobile Applications'. Available from: https://www.wi1.uni-muenster.de/pi/veroeff/heitkoetter/Comparing-Cross-Platform-Development-Approaches-for-Mobile-Applications.pdf [6 June 2016]

Johansen RD, Pagani Britto TC, Cusin CA 2013, 'CSS Browser Selector Plus: A JavaScript Library to Support Cross-browser Responsive Design'. In *Proceedings of the 22nd International Conference on World Wide Web* (WWW '13 Companion). International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 27-30. ISBN: 978-1-4503-2038-2.

Khan M, 2015. *WebRTC Signalling Concepts*. Available from: https://www.webrtc-experiment.com/docs/WebRTC-Signalling-Concepts.html. [28 May 2016]

Leaver E, Iwase Y, Katsura K 2015, *A Study of WebRTC Security*. Available from: http://webrtc-security.github.io/. [6 June 2016]

Levent-Levi T, 2014. *What's Behind Ericsson's OpenWebRTC Project?* Available from: https://bloggeek.me/ericssons-openwebrtc-project/. [29 May 2016]

Levent-Levi T, 2016. *Microsoft, Apple and WebRTC in 2016.* Available from: https://bloggeek.me/microsoft-apple-webrtc/ [4 June 2016]

Loreto, S & Romano SP, 2014, *Real-Time Communication with WebRTC*, 1st edn., O'Reilly Media, Sebastopol. ISBN: 978-1-449-37187-6.

Mullins C 2015, 'Responsive, mobile app, mobile first: untangling the UX design web in practical experience'. In *Proceedings of the 33rd Annual International Conference on the Design of Communication* (SIGDOC '15). ACM, New York, NY, USA, , Article 22 , 6 pages. doi: 10.1145/2775441.2775478

Nurminen JK, Meyn AJR, Jalonen E, Raivio Y & Garcia Marrero R 2013, 'P2P media streaming with HTML5 and WebRTC'. In *Computer Communications Workshops*

(INFOCOM WKSHPS), 2013 IEEE Conference on, Turin, 2013, pp. 63-64. doi: 10.1109/INFCOMW.2013.6970739

Ristic D, 2014. *WebRTC data channels*. Available from: http://www.html5rocks.com/en/tutorials/webrtc/datachannels/. [28 May 2016]

Singh K & Buford J 2016, 'Developing WebRTC-based Team Apps with a Cross-Platform Mobile Framework'. *2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, Las Vegas, NV, 2016, pp. 236-242. doi: 10.1109/CCNC.2016.7444762

Statista 2016, 'Marktanteile der führenden Betriebssysteme am Absatz von Smartphones weltweit vom 1. Quartal 2009 bis zum 1. Quartal 2016'. Available from: http://de.statista.com/statistik/daten/studie/73662/umfrage/marktanteil-der-smartphone-betriebssysteme-nach-quartalen/ [4 June 2016]

Stifter M, 2016. *WebRTC – Development of a browser based real-time peer-to-peer remote support application*. Bachelor thesis, FH JOANNEUM Kapfenberg.

Voutilainen JP, Salonen J & Mikkonen T 2015. 'On the Design of a Responsive User Interface for a Multi-device Web Service'. In *Mobile Software Engineering and Systems (MOBILESoft), 2015 2nd ACM International Conference on*, Florence, 2015, pp. 60-63. doi: 10.1109/MobileSoft.2015.16

WebRTC Tutorial, 2014 (video file). Available from: https://www.youtube.com/watch?v=5ci91dfKCyc. [27 May 2016]

What's next for WebRTC, 2015 (video file). Available from: https://www.youtube.com/watch?v=HCE3S1E5UwY. [6 June 2016]

Xanthopoulos S & Xinogalos S 2013. 'A comparative analysis of cross-platform development approaches for mobile applications'. In *Proceedings of the 6th Balkan Conference in Informatics* (BCI '13). ACM, New York, NY, USA, 213-220. doi: 10.1145/2490257.2490292

Xinogalos S, Psannis KE & Sifaleras A 2012. 'Recent advances delivered by HTML 5 in mobile cloud computing applications: a survey'. In *Proceedings of the Fifth Balkan Conference in Informatics* (BCI '12). ACM, New York, NY, USA, 199-204. doi: 10.1145/2371316.2371355