Hybrid Approach to WebRTC Videoconferencing on Mobile Devices

Bakary Diallo [(k)], Abdelaziz Ouamri, Mokhtar Keche

University of Sciences and Technology of Oran - Mohamed BOUDIAF (USTO - MB), Oran, Algeria
{bakary.diallo,ouamri.abdelaziz, mokhtar.keche}@univ-usto.dz

Abstract. This paper provides an in-depth comparative study and an interoperability study between a WebRTC browser-based P2P videoconferencing solution and a hybrid mobile app based one, built with the React Native framework. The comparison is in terms of CPU load, RAM occupancy and network data usage. To carry out our experiments, we designed and implemented a WebRTC P2P videoconferencing prototype, including a signaling server and two separate client applications based on the same algorithm written in JavaScript. The first application is a WebRTC web client (compatible desktop and mobile) and the second is a WebRTC React Native hybrid mobile application. According to the results obtained after several video calls performed over WLAN and LTE networks, our WebRTC hybrid app consume less CPU (~-10%) compared to the web browser-based one. The two types of applications show comparable RAM occupancies. In short, our results showed that implementing WebRTC real-time video streaming in a hybrid mobile app can be a better alternative in the WebRTC videoconferencing on mobile devices, while most of the scientific researches carried out around WebRTC still focus on the web browser.

Keywords: WebRTC · Videoconferencing · Hybrid App · Web App · CPU consumption · RAM occupancy · Bandwidth occupancy · Quality of Experience (QoE).

1 Introduction

Nowadays, especially since the Covid-19 pandemic, the growing demand for real-time multimedia data transmission applications such as video conferencing, distance education, IP television, video on demand, medical tele-operation, and remote video surveillance are increasingly challenging, on the one hand, the application developers in terms of technologies (API, framework and platform) and topologies, and on the other hand, the device manufacturers in terms of physical performances (CPU, RAM, and power) and mobile operators in terms of bandwidth and coverage. Moreover, most of the solutions available in the literature are paid, proprietary or require some external plugins. WebRTC is a free and open-source framework that does not require any plugin and integrates several powerful tools for encoding, decoding, and securing audio and video streams [1]. The technologies behind WebRTC are implemented as an open web standard and available as regular JavaScript APIs in all major web browsers. For native and hybrid clients, like Android and iOS applications, a library is available that provides the same functionality.

WebRTC is intended to work in web browsers, native and hybrid clients on different kind of devices (desktop, mobile and IoT devices). To the best of our knowledge, there is no research that rigorously studied the usage of WebRTC outside the web browsers. In this paper, we present an in-depth comparative study and an interoperability study between a WebRTC browser-based videoconferencing solution and a hybrid mobile app based one. The comparison is in terms of CPU load, RAM and network occupancies on mobile devices, and on different types of networks (WLAN, and LTE). To carry out our experiments, we designed and implemented a WebRTC videoconferencing prototype containing a signaling server and two separate client applications based on the same algorithm written in JavaScript: The first one is a responsive web application, compatible with a mobile and a desktop. The second client is a WebRTC hybrid mobile application, developed with the React Native framework [2].

To push further our study, the "video calling functionality" of two mobile multimedia applications "Facebook Messenger" and "Facebook WhatsApp Messenger" were included in our experiments to find out whether a simple WebRTC React Native hybrid application can be compared, on some technical performances aspects (CUP load, RAM occupancy and network data usage), to these high-end applications supported by the dynamic working groups of Facebook.

The rest of this paper is organized as follows. Section 2 presents the related works. Section 3 describes our proposed prototype. Experimental evaluations and results obtained are presented in Section 4. Section 5 provides a discussion of the obtained results. Finally, a conclusion is drawn and future works are suggested in Section 6.

2 Related Works

Since its introduction by Google in 2011, WebRTC attracted the curiosity of several developers and researchers around the world. Several studies and researches were carried out on the subject.

The authors of [3] describe some measurements collected from a WebRTC implementation operated from real mobile nodes within the pan European MONROE platform. They experimentally investigated and compared the performance of WebRTC for static and mobile cellular users of several networks across Europe. They observed that mobility is still an important challenge for WebRTC, since mobile broadband operators do not yet cope with full quality coverage for users on the move. Their studies were limited to Goggle Chrome web browser and the factors related to networks (video frame rate, video bit rate, packet delay, and jitter delay).

Asif et al. [4] compared WebRTC video conferencing functionality against a Skype-based solution to determine whether an integrated approach could provide an experience as good as or better than the off the shelf solution on certain aspects. They achieved this by implementing WebRTC into an existing groupware web application, PowerMeeting, and compared this with PowerMeeting's existing Skype-based solution. They found that whilst users felt that WebRTC was capable of delivering a solution that could be used without any major issues, the quality and reliability of the Skype solution provided a more stable experience for groupware activities overall. Since the implementation of their solution, there was many progresses in WebRTC technology. Their solution was based on the SimpleWebRTC library, which was developed before the WebRTC first stable release [5]. A WebRTC hybrid application could be a better solution.

The authors in [6] presented a test bed whose aim is to enable some experiments about real-time multimedia sessions by using the WebRTC paradigm over a LTE-capable infrastructure with the assistance of the NS-3 simulator. They analyzed some typical performance figures of multimedia calls such as throughput, jitter and packet loss trough different mobile network scenarios simulated using the NS-3 simulator. This study is also based on the web browser.

Kundan et al. [7] presented seven different cross platform apps built using Chrome App (for desktop) and Apache Cordova (for mobile) frameworks and tools. These apps use WebRTC for real-time audio and video streaming. They described some challenges and techniques (like media capture and playback, network connectivity, interoperability, and multi-way call) related to audio, video, network, power conservation, and security in such applications. The authors of this paper did not present any comparative study between web-based WebRTC solutions and hybrid-based ones in terms of technical performances related to hardware, such as CPU load and RAM occupancy, which can drastically impact the quality experienced by a user in a videoconferencing solution.

In this paper, we used the React Native hybrid framework and established detailed comparative and interoperability studies between a WebRTC browser-based videoconferencing solution and a WebRTC hybrid mobile app based solution, in terms of CPU load, RAM and network occupancies.

Authors of [8] created and implemented a WebRTC videoconferencing solution that can offer bi-directional communication over different networks, such as wired and Wi-Fi of LAN and WAN networks. A deep evaluation of the physical implementation was done regarding CPU performance, bandwidth consumption and QoE. They concluded that the bandwidth consumption of audio communication in WebRTC exhibited (53 - 54 Kbit/s) bandwidth rate over LAN and (48 - 50 Kbit/s) over WAN, while the CPU exhibited a range of 13% to 17% as an average needed rate, according to their testing environment. Their studies were only based on the Google Chrome web browser and they did not address the case of mobile devices.

While implementing a P2P videoconferencing system based on WebRTC, the authors of [9] tracked the CPU and memory states according to the number of users in the conference. Although brief and limited to web browsers, this study is a good introduction to technical performances (CPU time, and RAM occupancy) evaluation of a P2P videoconferencing solution based on WebRTC.

2 Our Proposed Solution

A WebRTC solution comprises two parts: a client part and a server part. The client part can be a web or mobile or desktop application, coded in JavaScript / HTML / CSS or other native languages (Java, C ++, Oject-C). The HTML / CSS part is used for HMI (Human-Machine Interface) and the JavaScript part for communications with the servers (signaling, STUN and TURN) and the exploitation of WebRTC functions (connection establishment, data exchange, etc.).

Our WebRTC videoconferencing prototype includes a signaling server and two separate client applications.

2.1 Signaling Server

WebRTC repose on three main APIs:

- MediaStream, which represents a media stream composed of audio/video tracks obtained via the mediaDevices.getUserMedia() method allowing access to user's multimedia resources (camera, and microphone), or from other multimedia data files (.mp3, .mp4) already available on the user's device.
- RTCPeerConnection, which makes it possible to establish a P2P connection between two users via the ICE protocol to exchange audio /video streams.
- RCTDataChannel, which represent an arbitrary data channel between two end users.

Since WebRTC standard does not specify a signaling protocol between the clients and the signaling server, each developer has the possibility to implement its own signaling mechanism. In this work, we designed and implemented a signaling server based on the Node.js framework [10], the JavaScript Session Establishment Protocol (JSEP) [11] and the WebSocket API [12, 13]. Our signaling protocol includes five control messages: “connexion”, “offer”, “answer”, “candidate”, and “fin”.

Fig. 1 illustrates a scenario in which client 1 sends a message “offer” to client 2, via the signaling server, to initiate a call. Client 2 receives this message and sends a message of type “answer” to client 1, to answer its call. Then the two clients exchange “candidate” messages via the signaling server until a P2P connection (RTCPeerConnection) is fully established between them (for further information on WebRTC signaling, see [1] and [11]).

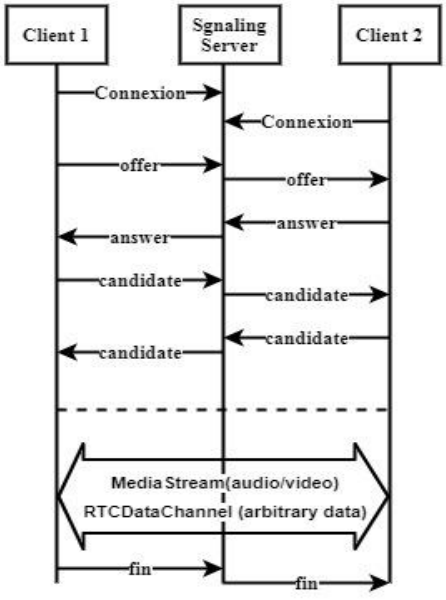

Fig. 1. Our proposed signaling protocol

2.2 Client Applications

We implemented two WebRTC client applications based on the same main algorithm written in JavaScript. The first one, is a responsive web application (for desktop and mobile), and the second one, is a hybrid mobile application developed with the React Native framework.

Web Client. WebRTC is embedded in web browsers as a JavaScript API. Nowadays, it is available in most modern web browsers (Google Chrome, Mozilla Firefox, Safari and Opera). HTML5 and CSS3 languages were used to implement the graphical user interface, where local and remote videos were embedded in HTML5 <video> tags. The "Responsive design" functionality of CSS3 was used to adapt the GUI to all screen sizes (desktop and mobile). JavaScript was used to provide communication with the signaling server via the WebSocket API available in the web browser, and for real- time acquisition and transmission of audio and video streams, via the WebRTC API. Some screenshots of this application are shown in Fig.2.

Hybrid Mobile Client. The structure of a React Native app is quite different from a website. It combines the best of native development with React (a JavaScript library for building user interfaces) [14]. The graphical interface is not programmed in simple HTML5 but in JSX [15], which allows to manipulate the user interface in the JavaScript code. React Native possesses its own tags (graphical elements, also called components) such as <RTCView> to display a real-time video stream. The React Native code is not interpreted in a web browser, it is rendered as platform native code. The graphical elements are converted to their native equivalents, and the back-end can be controlled with both JavaScript and Java (for Android) or JavaScript and Ojective-C (for iOS). WebRTC APIs are not available by default in React Native framework, React Native possess a specific Node.js module for WebRTC (react-native-webrtc) [16]. JavaScript functions required some modifications but their basic algorithms remained reusable. Some screenshots of this application are also shown in Fig. 2.

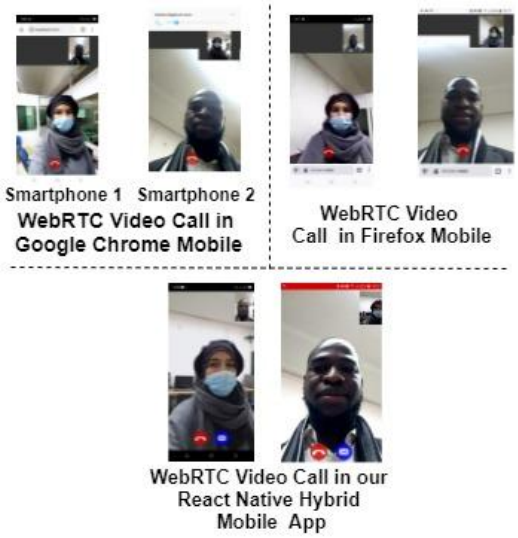

Fig. 2. Our WebRTC web and hybrid clients

3 Evaluation and Results

3.1 Experimental Environments

We carried out experiments according to the network topologies illustrated in Fig. 3. The first one is a WLAN (Wireless Local Area Network) containing:

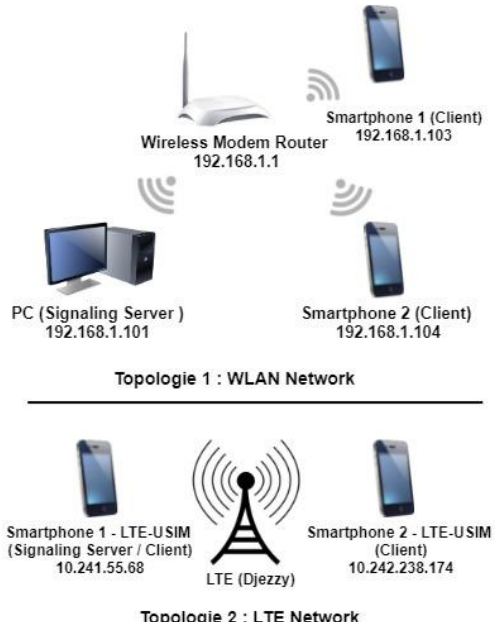

Fig. 3. Our Test Bed

- A wireless modem-router TD-W8901N TP-LINK 150Mbps.
- A TCHNO CF7 smartphone (Smartphone 1), 1.5 GHz ARMv7 Quad Core CPU, 3 GB of RAM, with an operating system Android 8.1.0 (Oreo).
- An LG-M700A Smartphone (Smartphone 2), Octa Core 1.09 GHz Qualcomm Technologies, Inc MSM8940, 3GB of RAM and Android 8.1.0 (Oreo).
- A desktop computer (PC) Microtower HP 600B, Intel Pentium G645T CPU 2.50 GHz, 4 GB of RAM with a Windows 7 Professional 64-bit operating system.

The second network in Fig. 3, represents the topology that was used on the LTE network. It consists of the two smartphones (Smartphone 1 and Smartphone 2) equipped with a LTE USIM card from the mobile telephony operator Djezzy in Algeria.

Evaluated Parameters. The 4 main parameters evaluated in our studies are: the CPU consumption, RAM occupancy, Number of bytes sent per second, and Number of bytes received per second. These parameters can significantly affect the user perceived quality of experience in a video streaming application.

To accomplish our studies, we performed 15 video calls over the WLAN and the LTE networks, each video call took 5 minutes. We performed 61 measurements, at a rate of one measurement every 5 seconds, of CPU consumption, RAM occupancy, number of bytes sent per second, and number of bytes received per second by using the Android mobile application “Simple System Monitor”.

The purpose of calls performed over the WLAN network was to accomplish a comparative study, in terms of CPU consumption, RAM and bandwidth occupancies, and an interoperability study, between the WebRTC web client and the WebRTC hybrid-based one.

The purpose of video calls performed over the LTE network was to evaluate our WebRTC hybrid mobile client over the LTE and to establish a comparative study between this one and the “video calling functionality” of two popular and major multimedia mobile applications “Facebook WhatsApp Messenger” and “Facebook Messenger”, in terms of CPU consumption, RAM and bandwidth occupancies.

The tests were conducted with a video definition of 320x240 pixels and a frame rate of 25 fps, in each client application.

3.2 Evaluation over WLAN

Comparative Study. Over WLAN, we performed 6 video calls between the two smartphones, which we divided into 2 scenarios:

- Scenario 1: In which we performed, between the two smartphones, a video call in the Google Chrome Mobile browser, a video call in Mozilla Firefox Mobile browser and a video call in our WebRTC hybrid mobile app.
- Scenario 2: We made the same calls as in Scenario 1 by changing the sense of calls (the calling smartphone becomes the callee one).

The average values over the 61 measurements performed from the CPU consumption, the RAM occupancy, the number of bytes sent per second, and the number of bytes received per second are displayed in Table 1 for Smartphone 1, and Table 2 for Smartphone 2, for both Scenario 1, and Scenario 2.

The numerical average values displayed in Table 1 show that the mobile web browsers Google Chrome and Mozilla Firefox consume more CPU (~100%) compared to our React Native hybrid mobile application (~85%) on the Smartphone 1, and that the difference is negligible in RAM occupancy (~55%). They also show that the Google Chrome Mobile browser consumes much more data (~65 KB/s) compared to Mozilla Firefox Mobile (~17 KB/s) and our hybrid application (~15 KB/s).

The results obtained on the Smartphone 2 (Table 2) confirm those of Smartphone 1. From this table, we can observe that the Google Chrome and Mozilla Firefox mobile browsers display an average CPU load of (~45%), while our hybrid app displays an average CPU load of (~36%). The difference in RAM occupancy is still negligible (~58%). We can observe again that the Google Chrome Mobile browser exceed the two other applications in terms of binaries data usage (~50 KB/s vs. ~15KB/s) during a video call. Besides this, we can observe that the two scenarios of tests give approximately the same average values on both smartphones.

Table 1. Average values of CPU consumption, RAM occupancy, number of bytes sent/s and number bytes received/s on Smartphone 1 (in Chrome Mobile, Firefox Mobile, and React Native app) - Scenario 1 and Scenario 2.

Measurement	Before calls	Application					
		Chrome Mobile		Firefox Mobile		React Native hybrid app	
		Sce[1]. 1	Sce. 2	Sce. 1	Sce. 2	Sce. 1	Sce. 2
CPU Consumption (%)	50,21	100,00	100,00	100,00	100,00	85,29	84,17
RAM Occupancy (%)	43,49	55,59	53,24	55,52	53,75	56,23	57,35
Bytes Sent/s (KB/s)	1,28	54,14	60,84	11,00	23,16	17,38	14,75
Bytes Received/s (KB/s)	0,07	67,92	81,97	13,17	23,76	16,55	11,58

Table 2. Average values of CPU consumption, RAM occupancy, number of bytes sent/s and number of bytes received/s on Smartphone 2 (in Chrome mobile, Firefox mobile, and React Native app) - Scenario 1 and Scenario 2.

Measurement	Before calls	Application					
		Chrome Mobile		Firefox Mobile		React Native hybrid app	
		Sce. 1	Sce. 2	Sce. 1	Sce. 2	Sce. 1	Sce. 2
CPU Consumption (%)	36,43	43,24	48,31	46,48	41,74	38,35	33,82
RAM Occupancy (%)	51,03	63,13	59,72	58,66	58,38	59,94	58,40
Bytes Sent/s (KB/s)	1,12	54,98	38,89	16,45	19,82	16,51	13,59
Bytes Received/s (KB/s)	0,10	48,48	59,94	4,19	6,74	13,57	13,82

Interoperability Study. To check the interoperability between our WebRTC brower-based solution and the hybrid-based one, we performed 3 video calls between the two smartphones over the WLAN, which we considered as Scenario 3 (on the WLAN). In this scenario, we performed a call between Google Chrome Mobile browser (Smartphone 1) and Mozilla Firefox Mobile (Smartphone 2), a call between Google Chrome Mobile (Smartphone 1) and our hybrid mobile application (Smartphone 2), and finally, a video call between Mozilla Firefox Mobile (Smartphone 1) and our WebRTC hybrid mobile application (Smartphone 2).

All mixed video calls were performed successfully, (see Table 3), where we can observe that the web browsers display average values that are similar to those obtained in the precedent study (Table 1 and Table 2). The same observation holds for the hybrid app also. On the basis of this study, we can affirm that the latest release of WebRTC (WebRTC 1.0) ensures the interoperability between the mobile web browsers Google Chrome and Mozilla Firefox on one hand, and on the other hand, between these browsers and a WebRTC hybrid mobile application developed with the React Native framework, on Android operating system.

Table 3. Average values of CPU consumption, RAM occupation, bytes sent/s and bytes received/s on Smartphone 1 and Smartphone 2 (in Chrome Mobile, Firefox Mobile, and React Native app) - Scenario 3.

Measurement	Call Scheme					
	Chrome to Hybrid		Firefox to Hybrid		Chrome to Firefox	
	Chrome (Sma[2]. 1)	Hybrid (Sma. 2)	Firefox (Sma. 1)	Hybrid (Sma. 2)	Chrome (Sma. 1)	Firefox (Sma. 2)
CPU Consumption (%)	99,51	34,19	100,00	34,11	100,00	40,17
RAM Occupancy (%)	58,66	59,02	56,75	59,09	55,86	59,97
Bytes Sent/s (KB/s)	4,70	14,11	2,32	1,75	6,42	11,35
Bytes Received/s (KB/s)	20,58	2,69	2,49	2,35	13,41	5,02

[1] Sce. means Scenario in the two Table 1 and Table 2.

[2] Sma. means Smartphone in Table 3

3.3 Evaluation over LTE

For a further evaluation of our approach (implementing WebRTC videoconferencing in a mobile hybrid application), we included in our studies the "video calling functionality" of two out of range multimedia applications "Facebook WhatsApp Messenger" and "Facebook Messenger". To figure out how our approach can be situated among the current state of art of mobile videoconferencing solutions, according to certain technical aspects, such as CPU consumption, RAM occupancy, and mobile data usage, which are some major factors that impact the quality experienced by a user.

LTE (Long Term Evolution) also known as 4G, represents a major evolution in the mobile networks field. It is the extension of GSM (2G) and UMTS/HSPA (3G/3G+). LTE has the advantage to be an all-IP network, any device connected to a LTE network is automatically assigned an IP address, internal to this network [17]. This IP address can then be used to reach the device across the operator's mobile network. This is the functionality that we used in this study to make video calls over the LTE network, with our hybrid mobile application.

Over the Algerian Djezzy LTE mobile network, we performed 6 video calls which we divided into 2 scenarios:

- Scenario 1: We performed, between the two smartphones, a video call in "Facebook WhatsApp Messenger", a video call in "Facebook Messenger" and a video call in our WebRTC hybrid mobile app.
- Scenario 2: We repeated the same calls as in Scenario 1 by switching the sense of calls (the calling smartphone becomes the callee one).

As on the WLAN, we performed 61 measurements, at a rate of one measurement every 5 seconds, of CPU consumption, RAM occupancy, number of bytes sent per second, and number of bytes received per second for each call, on each smartphone. The average values over the 61 measurements performed from the CPU usage, RAM occupancy, number of bytes sent per second, and number of bytes received per second are displayed for the two scenarios in Table 4 for Smartphone 1, and Table 5 for Smartphone 2.

According to these values, our WebRTC hybrid mobile application shows a little better CPU performance (~67% on Smartphone 1 and ~33.34% on Smartphone 2), compared to the two other applications (~73% on Smartphone 1 and ~33.40% on Smartphone 2). Our hybrid app shows comparable RAM occupancy with WhatsApp Messenger on both smartphones (~53% on Smartphone 1 and ~57% on Smartphone 2), while Facebook Messenger displayed a little higher RAM occupancy (~57% on smartphone 1 and ~61% on Smartphone 2). In terms of data usage, the WhatsApp Messenger is much better optimized than our hybrid app and Facebook Messenger.

These results show that the WebRTC, running in a hybrid mobile app built with React Native framework, can be a better and up to date alternative to real-time video streaming on the mobile devices.

Table 4. Average values of CPU consumption, RAM occupation, number of bytes sent/s and bytes number of received/s on Smartphone 1 (by Facebook Whatsapp Messenger, Facebook Messenger and our React native Hybrid app) over LTE - Scenario 1 and Scenario 2.

Measurement	Application					
	WhatsApp		Messenger		React Native hybrid app	
	Sce. 1	Sce. 2	Sce. 1	Sce. 2	Sce. 1	Sce.2
CPU Consumption (%)	77,11	89,92	73,43	73,31	65,92	68,24
RAM Occupancy (%)	54,06	54,53	57,86	53,79	52,07	53,43
Bytes Sent/s (KB/s)	0,08	0,15	17,33	9,76	20,82	19,23
Bytes Received/s (KB/s)	0,04	0,55	5,96	7,56	11,63	10,27

Table 5. Average values of CPU consumption, RAM occupation, number of bytes sent/s, and number of bytes received/s on Smartphone 2 (by Facebook Whatsapp Messenger, Facebook Messenger and the React Native hybrid app) on LTE - Scenario 1 and Scenario 2.

Measurement	Application					
	WhatsApp		Messenger		React Native hybrid app	
	Sce. 1	Sce. 2	Sce. 1	Sce. 2	Sce. 1	Sce. 2
CPU Consumption (%)	33,41	33,40	33,42	33,36	33,34	33,34
RAM Occupancy (%)	57,15	57,24	61,09	62,63	58,67	57,23
Bytes Sent/s (KB/s)	3,77	1,01	49,95	61,18	64,84	40,71
Bytes Received/s (KB/s)	0,64	0,02	51,55	53,81	86,86	8,13

4 Discussion

In this paper, a WebRTC browser-based videoconferencing solution is compared to a hybrid-based one, in terms of CPU consumption, RAM occupancy and network data usage on mobile devices. A check of the interoperability between these solutions is also performed.

The results obtained after several video calls performed over WLAN and LTE networks showed that our WebRTC React Native hybrid app consume less CPU (around 10%), compared to web browser-based one on mobile devices, while the difference in RAM occupancy is insignificant. It was also found that our hybrid mobile app and Mozilla Firefox Mobile consume less data than does Google Chrome Mobile.

According to our experiments, a WebRTC hybrid app is more customizable, smoother and more efficient compared to a web browser-based one. For the end-user, there is no difference between a hybrid app and a native one. However, a WebRTC browser-based solution is simpler and faster to develop, distribute and to update compared to a hybrid-based one. The WebRTC is not directly available in the React Native framework, it needs a free and open-source third-party module “react-native-webrtc” that requires a lot of configuration to be supported in React Native.

Experiments carried out over the LTE network allowed us to evaluate our React Native hybrid app over LTE, and compare it against the “video calling functionality” of two popular multimedia applications “Facebook WhatsApp Messenger” and “Facebook Messenger”, in terms of CPU consumption, RAM occupancy, number of bytes sent per second and number of bytes received per second during a video call.

We found out that our WebRTC hybrid app shows results that are comparable to those of these two major applications, in terms of CPU load and RAM occupancy. This demonstrates the efficiency of the WebRTC technology and the prodigality of our approach to implement the WebRTC video streaming in a hybrid mobile app.

In addition, if we make a comparison between the results obtained over Wi-Fi and those obtained over LTE, we can observe that the later ones are slightly better than the former ones, in terms of CPU load and RAM occupancy, as shown in Tables 6 and 7.

Table 6. Comparison between WLAN and LTE, in terms of CPU consumption, RAM occupancy, number of bytes sent/s and number of bytes received/s, by our react native hybrid app on Smartphone 1.

Measurement	Network	
	WLAN	LTE
CPU Consumption (%)	85	68
RAM occupancy (%)	55	53
Bytes Sent/s (KB/s)	14	20
Bytes Received/s (KB/s)	13	11

Table 7. Comparison between WLAN and LTE, in terms of CPU consumption, RAM occupation, number of bytes sent/s and number bytes received/s, by our react native hybrid app on Smartphone 2.

Measurement	Network	
	WLAN	LTE
CPU Consumption (%)	34	33
RAM occupancy (%)	59	58
Bytes Sent/s (KB/s)	13	40
Bytes Received/s (KB/s)	13	81

Finally, our interoperability study revealed that the first stable release of WebRTC, WebRTC 1.0 : Real-Time Communication between Browsers, supports the interoperability between a WebRTC hybrid app, built with React Native framework, and a browser-based one, which gives the possibility for a developer to provide web and hybrid versions of its application, according to necessities.

5 Conclusion and Future Works

In this paper, we demonstrated the feasibility of the implementation of WebRTC real-time videoconferencing in a React Native hybrid mobile application, and the interoperability between it and a browser-based one.

We designed and implemented a WebRTC videoconferencing solution containing a signaling server and two separate client applications based on the same algorithm and written in JavaScript: a WebRTC browser-based client, and a WebRTC hybrid mobile app built with React Native framework.

After several video calls performed over WLAN and LTE networks, we found out that our WebRTC React Native hybrid app consume less CPU (~-10%) compared to web browser-based one, while showing comparable RAM occupancy. We also found out that the interoperability is ensured between the two types of application.

Our prototype can find many applications in:

- E-Learning.
- E-commerce (video customer service).
- Healthcare (doctor-patient communication system).

In future works, we plan to introduce the iOS operating system, and other hybrid mobile technologies, like Google Flutter framework, which is becoming an undisputable competitor to React Native framework. We also plan to include some hybrid desktop technologies like Electron, to make a comparison on the desktop also.

References

1. Real-time communication for the web, https://webrtc.org, last accessed 2021/06/25.
2. React Native, https://reactnative.dev, last accessed 2021/06/25.
3. Moulay, M., Mancuso, V.: Experimental performance evaluation of WebRTC video services over mobile networks. In.: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 541-546. Honolulu (2018). doi: 10.1109/INFCOMW.2018.8407020.
4. Hussain, A., Wang, W., Xu, D.-L.: Comparing WebRTC video conferencing with Skype in synchronous groupware applications. In: 2017 IEEE 21st International Conference on Computer Supported Cooperative Work in Design (CSCWD), pp. 60-65. Wellington (2017). doi:10.1109/cscwd.2017.8066671.
5. WebRTC 1.0: Real-Time Communication between Browsers, https://www.w3.org/TR/webrtc, last accessed 2021/06/25.
6. Carullo, G., Tambasco, M., Mauro, M. D., Longo, M.: A performance evaluation of WebRTC over LTE. In: 2016 12th Annual Conference on Wireless On-demand Network Systems and Services (WONS), pp. 1-6. Cortina d'Ampezzo (2016).
7. Singh, k., Buford, J.: Developing WebRTC-based team apps with a cross-platform mobile framework. In: 2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC), pp. 236-242. Las Vegas, NV (2016). doi:10.1109/ccnc.2016.7444762.
8. Edan, N. M., Al-Sherbaz, A., Turner, S.: Design and evaluation of browser-to-browser video conferencing in WebRTC. In: 2017 Global Information Infrastructure and Networking Symposium (GIIS), pp. 75-78. St. Pierre (2017). doi: 10.1109/GIIS.2017.8169813.
9. Apu, K. I. Z., Mahmud, N., Hasan, F., Sagar, S. H, P2P video conferencing system based on WebRTC. In: 2017 International Conference on Electrical, Computer and Communication Engineering (ECCE), pp. 557-561, Cox's Bazar (2017), doi: 10.1109/ECACE.2017.7912968.
10. Node.js, https://nodejs.org/en, last accessed 2021/06/25.
11. JavaScript Session Establishment Protocol, https://tools.ietf.org/html/draft-ietf-rtcweb-jsep-26, last accessed 2021/06/25.
12. The WebSocket API (WebSockets), https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API, last accessed 2021/06/25.
13. The WebSocket Protocol, https://tools.ietf.org/html/rfc6455, last accessed 2021/06/25.
14. React, https://reactjs.org, last accessed 2021/06/25.
15. Introducing JSX, https://reactjs.org/docs/introducing-jsx.html, last accessed 2021/06/25.
16. React Native WebRTC, https://github.com/react-native-webrtc, last accessed 2021/06/25.
17. Yannick, B., Éric, H., François-Xavier, W.: LTE et les 4G. Eyrolles, Paris (2012).