Performance Evaluation of Distributed Data Delivery on Mobile Devices Using WebRTC

Arto Heikkinen
Center for Internet Excellence
University of Oulu
Oulu, Finland
e-mail: firstname.lastname@cie.fi

Timo Koskela
Center for Internet Excellence
University of Oulu
Oulu, Finland
e-mail: firstname.lastname@cie.fi

Mika Ylianttila
Center for Internet Excellence
University of Oulu
Oulu, Finland
e-mail: firstname.lastname@cie.fi

Abstract—**Direct peer-to-peer connectivity between web browsers is becoming reality with the emerging and constantly developing WebRTC technology stack. This opens possibilities for new kind of plugin-free web applications, such as browser-to-browser file transfers and multi-party conferencing. In this paper, the performance of WebRTC on mobile devices is evaluated with different mobile device, wireless network connectivity and web browser configurations. The evaluation was conducted with a WebRTC test environment that was implemented based on PeerJS JavaScript library and PeerServer signaling server. The measurements include session establishment delay and overhead, session maintenance overhead, resource consumption of multiple simultaneous file transfers and efficiency of different file transfer approaches. Based on the results, the delay for establishing a WebRTC connection may in the worst cases exceed even 10 seconds making it a serious bottleneck. However, from the standpoints of memory consumption and CPU load, high-end mobile devices are very capable of running multiple simultaneous WebRTC connections for data transfers. The results of this paper provide new insight to researchers, application and browser developers and WebRTC standardization bodies.**

Keywords—Web Browser, Peer-to-Peer, P2P, Delay.

I. Introduction

The origins of peer-to-peer (P2P) networking can be traced back to the end of 1960s, when the development of the first P2P system called ARPANET was initiated. However, the concept of P2P networking became widely known only at the turn of the millennium when P2P systems such as Napster, Gnutella, JXTA and KaZaa were introduced. Today, P2P systems are defined as ”distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources” [1]. Typically, P2P systems are highly scalable and robust due to their capability for self-organization and due to the decentralization of all or the majority of communication. From the service provider standpoint, deployment of P2P systems also decreases expenses, because the communication load on the centralized servers can be partially shifted to the clients (i.e. peers) residing at the edges of the network. However, the decentralization of communication also introduces privacy and security issues, and may compromise the overall system performance on account of heterogeneous capabilities of participating peers [2].

Currently, a substantial effort is taken to bring P2P networking also into the Web technology stack. Web Real-Time Communications (WebRTC) is an Application Programming Interface (API) definition that encompasses multiple standards, protocols and JavaScript APIs to enable direct browser-to-browser communications [3]. WebRTC provides support for streaming audio and video as well as exchanging arbitrary application data. The advantage of using WebRTC is that the implemented P2P applications are inherently cross-platform supported and no plug-in installations are required as with earlier approaches [4][5]. Although WebRTC has already been implemented in most widely used web browsers, the standardization work in the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C) is still on-going.

For mobile users, WebRTC enables versatile application scenarios ranging from P2P file transfers and multi-party conferencing to augmented reality [5][6]. Use of mobile devices, however, also introduces performance challenges due their limited hardware capabilities, wireless bandwidth and battery life [2]. Furthermore, mobile devices are typically behind Network Address Translations (NATs) and firewalls, which may delay or entirely prevent the establishment of WebRTC connections. For traversing NATs, WebRTC uses Interactive Connectivity Establishment (ICE) technique that relies on Session Traversal Utilities for NAT (STUN) and Traversal Using Relay NAT (TURN) [3]. In order to gain an understanding of WebRTC performance on mobile devices and to identify potential performance bottlenecks, in-depth performance measurements are required [7][8][9].

In this paper, performance of WebRTC on mobile devices is evaluated in terms of session establishment delay and overhead, session maintenance overhead, resource consumption of multiple simultaneous file transfers and efficiency of different file transfer approaches. The evaluation was conducted using the implemented WebRTC test environment with different mobile device, wireless network connectivity and web browser combinations. The results of this paper provide new knowledge for researchers, application and web browser developers as well as WebRTC standardization bodies.

The paper is organized as follows: Section II presents the related work on the use of WebRTC with mobile devices. Section III describes the experimental setup including the implemented test environment and the used methodology. Section IV presents the results of the experiments, and finally, Section V concludes the paper.

II. Related Work

Although the standardization work of WebRTC is still on-going, this emerging technology has already received a lot of attention in the research community. The recent work introduces some preliminary results on the performance of WebRTC (e.g. [5][10]), yet only a few studies use mobile devices as the experiment platform (e.g. [7]).

In [7], use of WebSockets and WebRTC was examined from the standpoint of mobile device power consumption. This work investigated means how the power saving and quality of service (QoS) features of cellular networks could be taken advantage of in WebRTC communications. Based on a power consumption analysis with 4G/LTE, recommendations for future standardization were given. These included, for instance, recommendations that the W3C Battery API should provide accuracy queries on the battery status and that an indication mechanism should be implemented advising if the cellular QoS is applicable to the current WebRTC communication.

In [8], the performance of WebRTC's adaptive video streaming capabilities were evaluated in a mobile setting, where measurements were collected while moving at walking speeds through an 802.16e WiMAX network with a laptop PC. As a result, it is stated that WebRTC did not achieve acceptable service quality under challenging mobile network conditions, especially in setting with high packet loss. However, this work focuses only on video streaming performance and does not take into account other factors such as connection establishment or data transmission. Also, genuinely mobile devices, such as smartphones or tablets, were not used in the experiments.

In [9], the applicability of HTML5 and WebRTC standards for P2P video streaming was investigated. The work describes a BitTorrent-like P2P video streaming solution which was used to identify the performance bottlenecks in WebRTC-based P2P video streaming implementations. As a result, it was stated that hashing used in the video metadata creation is generally very burdensome to execute in web browsers. However, video decoding itself can be quite easily performed even with mobile devices. The authors concluded that in theory it should be feasible to implement a WebRTC-based P2P video streaming service for mobile devices with carefully chosen optimizations.

In [5], the performance of full-mesh and star network topologies for WebRTC-based communication was evaluated when transferring multimedia content. This work focused particularly on the performance of the congestion control mechanism used with WebRTC. Based on the experimental results, the congestion control mechanism in WebRTC works well when the latencies are below 200ms. It was also discovered that there is an under-utilization issue with the full-mesh network topology resulting from two (i.e. sender and receiver) independent congestion controllers working on the same media stream. The work did not consider mobile aspects, but emphasized that evaluation with mobile devices is required.

In [10], the feasibility of live video streaming protocols using WebRTC was investigated. The preliminary experimentation data shows that it is possible to implement a pull-based P2P streaming protocol with WebRTC, at least, for small-scale P2P networks. The work concludes that building large-scale P2P networks still remains a challenge due to the current limitations of WebRTC implementations. This work did not consider mobile aspects.

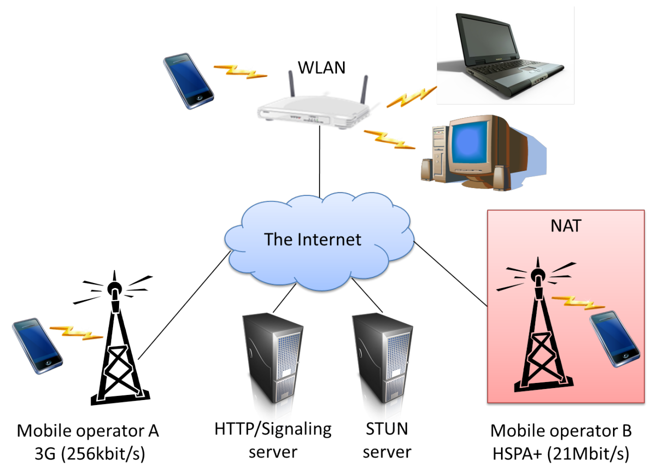


Fig. 1. WebRTC test environment.

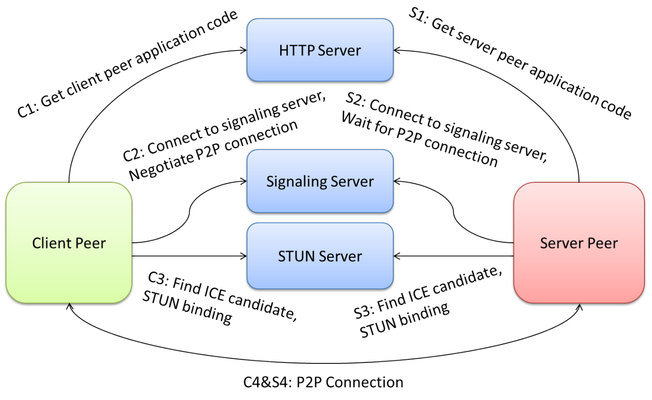
III. Experimental Setup

A. *WebRTC Test Environment*

For conducting the experiments, a real-world test environment[1] was implemented as illustrated in Figure 1. An open source JavaScript library called PeerJS 0.3.13 was used for implementing the WebRTC prototype applications running on a web browser. PeerServer 0.2.8 was used as a signaling server and Apache HTTP server 2.2.22 as a HTTP server. Both HTTP and signaling servers were run on the same server machine located in Germany. A public STUN server provided by Google was used. For providing wireless network connectivity, a WLAN (802.11n) hosted by the University of Oulu, Finland and two Finnish mobile operator networks were used. Mobile operator A provided slow speed 3G connectivity (256kbit/s) with a public IP. Mobile operator B provided broadband HSPA+ connectivity (21Mbits/s) with a private IP behind NAT. In addition, a live network emulation tool called dummynet [11] was used for limiting the available bandwidth in the WLAN.

The experiments were conducted with four different devices: two smartphones, a laptop and a server-grade PC. The first smartphone was Samsung Galaxy S3 4G with quad-core 1.4 GHz ARM Cortex-A9 CPU running Android 4.3. The second smartphone was Sony Xperia Z3 with quad-core 2.5 GHz Qualcomm Snapdragon 801 CPU running Android 4.4. The laptop was HP EliteBook 2760p with Intel i5 CPU running Win7. The server PC had Intel i7 CPU and was running Ubuntu Linux 12.04. It was also running dummynet for limiting the available network bandwidth. The used web browsers were Chrome 39.0, Chromium 41.0 (Linux) and Firefox 33.1. The mobile devices were stationary during the experiments. Only Android-based smartphones were used, as Android is currently the only mobile platform where mainstream web browsers support WebRTC.

[1]https://github.com/Chiru/webrtc-testbench.



HTTP Server
Signaling Server
STUN Server
Client Peer
Server Peer
C1: Get client peer application code
S1: Get server peer application code
C2: Connect to signaling server, Negotiate P2P connection
S2: Connect to signaling server, Wait for P2P connection
C3: Find ICE candidate, STUN binding
S3: Find ICE candidate, STUN binding
C4&S4: P2P Connection

Fig. 2. WebRTC connection establishment procedure.

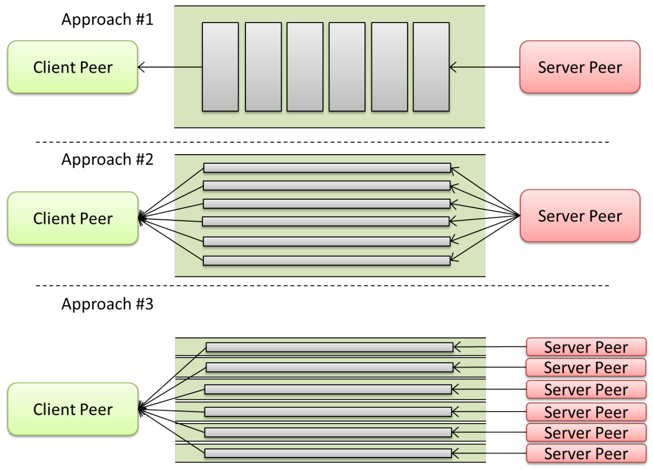


Fig. 3. Different file transfer approaches.

B. Methodology

In order to evaluate the performance of WebRTC on mobile devices, the following four research questions were examined:

1) What is the cost for establishing a WebRTC connection between web browsers?
2) What is the overhead for maintaining an open, but idle WebRTC connection between web browsers?
3) What is the level of resource utilization caused by multiple simultaneous data transfers in a mobile device using WebRTC?
4) How efficiently WebRTC can handle multiple simultaneous data transfers?

For examining the research questions, we conducted several experiments using our WebRTC test environment. In each experiment, both peers implemented the connection establishment procedure as described in Figure 2. First, the application code was downloaded from the HTTP server; second, the signaling server was connected in order to negotiate the WebRTC connection; third, ICE candidates were discovered for NAT traversal and STUN binding was conducted; fourth, the WebRTC connection between the web browsers was established. In each test, we performed several repeated measurements in order to obtain reliable results. The repetitions were conducted in a rapid fashion, which prevented the wireless radios from entering idle or sleep state.

A WebRTC connection consists of two main elements: RTCPeerConnection and RTCDataChannel, which are both exposed as separate APIs to a JavaScript program. RTCPeerConnection automatically handles the ICE process, STUN keepalive messaging and media stream management and provides functionality for setting up the P2P connection. RTCDataChannel enables sending arbitrary application data between the peers. In all experiments, in addition to a RTCPeerConnection, we also establish a RTCDataChannel between the peers as our focus is on transmitting arbitrary data. TURN-based connectivity, where all the communication goes through a relaying server, was not included in the experiments, as this paper focuses on P2P communication.

For the first research question, both session establishment delay and overhead were measured using varying web browser combinations. For measuring the session establishment delay, the time required for both establishing a connection with the signaling server and the other peer were measured. The measurements were conducted during office hours (08:00-16:00) with varying wireless network combinations. In this case, mobile network A (slow, public IP) was used with Samsung Galaxy S3 4G and mobile network B (fast, private IP) was used with Sony Xperia Z3. WLAN was used with both smartphones and the laptop. Each experiment was repeated 100 times using a loop in a JavaScript code. For measuring the session establishment overhead, the amount of transmitted data between the laptop (client peer) and Samsung Galaxy S3 4G (server peer) was measured only in the WLAN as the network traffic patterns of WebRTC were identical in the mobile network. However, some amount of retransmissions can occur in a mobile network, depending on the network conditions. This would increase the overhead for session establishment compared to our results where no retransmissions occurred. The network traffic was captured using a network protocol analyzer called WireShark running on the laptop. The P2P network traffic on the smartphone can be measured at the laptop end as both peers see the same P2P messaging. Capturing the network traffic was conducted three times for each web browser combination as the results varied only marginally between consecutive experiments.

For the second research question, session maintenance overhead including both the memory consumption and the data transmission rate of WebRTC keepalive messages was measured with varying web browser combinations. The measurements were conducted only in the WLAN using Samsung Galaxy S3 4G (client peer) and the laptop (server peer), as the data rate of WebRTC keepalive messaging between the peers was identical in the mobile network. Multiple server peers were running in the laptop and a single client peer in the smartphone opened simultaneous connections to them. The number of idle WebRTC connections with an open RTCDataChannel was started from 1 and increased up to the maximum number of supported connections (Firefox: 50 and Chrome: 256). Memory consumption measurements were conducted using the smartphone and the internal monitoring tools provided by both web browsers (about:memory). With Chrome, the memory consumption of the WebRTC application tab, and with Firefox,

TABLE I. Signaling server connection establishment time using the smartphones.

Mobile Network A (256 kbits/s)	Mobile Network B (21 Mbits/s)	WLAN (802.11n)
200-1359 ms	273-396 ms	148-310 ms

the memory consumption of the main process were manually recorded after each increment of idle WebRTC connections. Data transmission rate of WebRTC keepalive messages was measured on the laptop using WireShark for a period of one minute in each experiment.

For the third research question, the CPU load caused by multiple simultaneous WebRTC file transfers was measured. A single client peer was running on the Samsung smartphone, which opened multiple simultaneous WebRTC connections to multiple server peers running on the server PC. Both Firefox and Chrome were tested on the client side, while the server peers were running on Firefox. Both of the devices were connected to the WLAN. In this experiment, three different network speeds were emulated using dummynet on the server PC: 5 Mbits/s, 10 Mbits/s and 15 Mbits/s. The number of simultaneous WebRTC connections was varied from 1 to 50. In each connection, a 1 MB binary file was transmitted from the server peer to the client peer via RTCDataChannel. The data transmission for each connection was running in an endless loop, where the client peer constantly requested a new file as the previous file transfer was completed. The CPU utilization of the smartphone during the data transmission was measured for approximately one minute period for each experiment. For measuring the CPU utilization, a tool called ”Usemon” was utilized, capable of instrumenting the CPU load of an Android device and plotting a graph of the CPU load history. Based on this graph, the average CPU utilization during a single test run was manually inspected.

For the fourth research question, the total transfer time for 1 MB of binary data, split into 100 kB chunks, was measured using different data transfer approaches illustrated in Figure 3. A single 100 kB binary file was transmitted ten times in each experiment in order to transfer a total of 1 MB of binary data. Each experiment was repeated 30 times using a loop in a JavaScript code. In the first data transfer approach, a client peer was connected to a server peer. The 100 kB file was transmitted ten times in a row via a single RTCDataChannel, one file at a time. The second approach utilized also only a single WebRTC connection between the client peer and a server peer. However, multiple RTCDataChannels were opened and a 100 kB file was transmitted in each simultaneously. In the third approach, the client peer initiated several simultaneous WebRTC connections to different server peers. In each WebRTC connection, a single RTCDataChannel was utilized for transmitting a single 100 kB file concurrently. The Samsung smartphone was running a single client peer, while the server PC was running multiple server peers in parallel. Both of the devices were connected to the WLAN. The same network configurations were emulated using dummynet as for the third research question. Both Firefox and Chrome were tested at the client end. At the server end, in addition to Firefox also Chromium web browser was utilized in the experiment. Chrome is not available on Linux, so Chromium 41.0 was used instead. It is the open source equivalent of Chrome available on Linux.

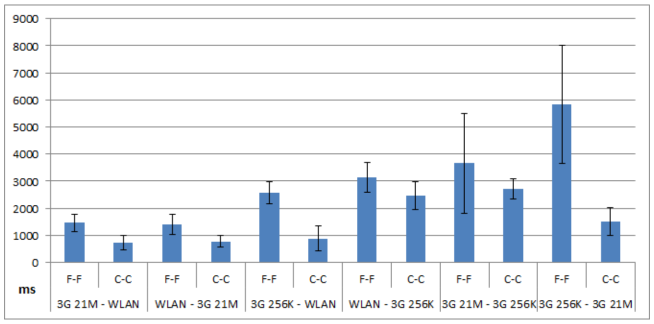


Fig. 4. Peer connection establishment time with mobile networks and the WLAN using homogeneous browser pairs.

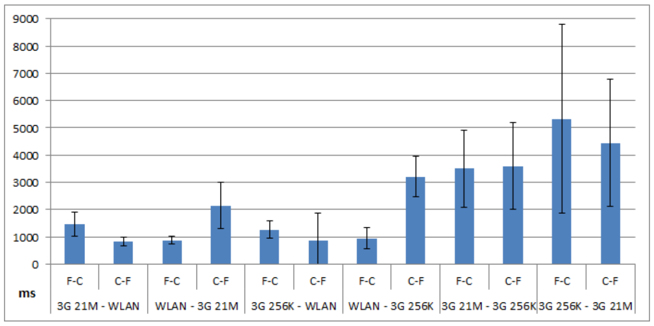


Fig. 5. Peer connection establishment time with mobile networks and the WLAN using heterogeneous browser pairs.

IV. Results

A. Session Establishment Delay

Table I shows the variation of the average delay for establishing a connection between a mobile peer and the signaling server using different wireless connectivities over all the experiments. It can be seen in Table I that especially with mobile networks, the variation in the delay can be huge. This was somewhat expected as the signaling server was intentionally located in another country. However, in majority of the experiments, the signaling server connection establishment time was less than 350ms, which can be further shortened by positioning the signaling server in close proximity.

In Figures 4, 5 and 6, the average delay and the standard deviation for establishing a connection between the peers is illustrated, where F stands for Firefox and C for Chrome. The web browser that is mentioned first acts as the initiator (i.e. client) of the peer connection establishment. In overall, it appears that using Chrome both as the client and the server is always the fastest and using only Firefox is always the slowest. This presumably results from the fact that Firefox does not yet implement the Trickle ICE extension to the ICE protocol that allows incremental ICE candidate gathering and connectivity checks between the peers. In contrast to Trickle ICE, the original ICE first waits the ICE gathering process to finish before proceeding with the WebRTC connection establishment.

It can be seen in Figures 4 and 5 that when using only Chrome the average delay for establishing a connection between two mobile peers located in the same country is less than 3000ms, which can still be adequate for many application

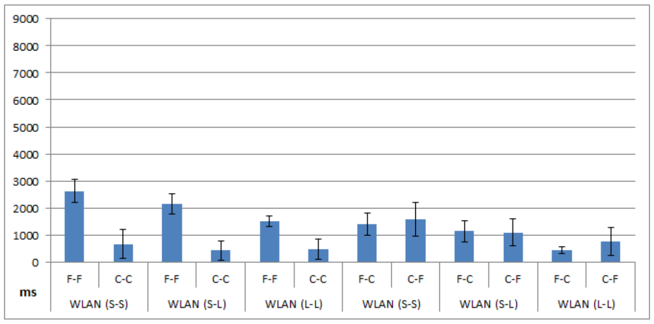


Fig. 6. Peer connection establishment time with the WLAN.

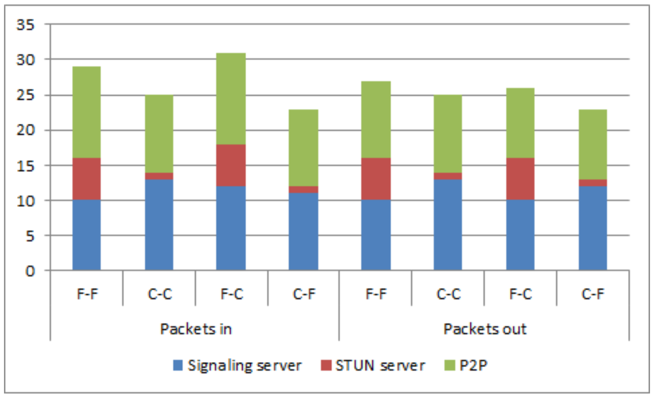


Fig. 7. The average number of packets transmitted in WebRTC connection establishment.

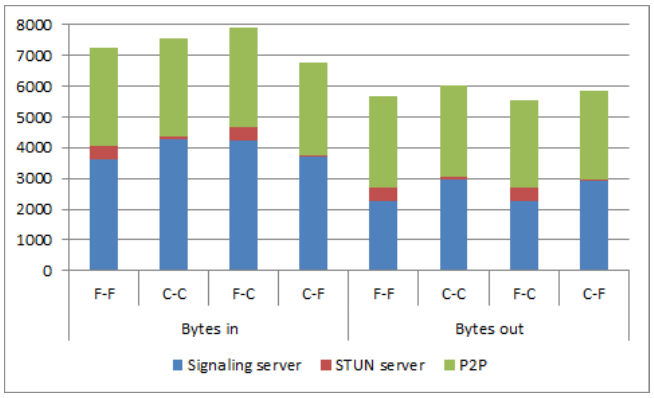


Fig. 8. The average number of bytes transmitted in WebRTC connection establishment.

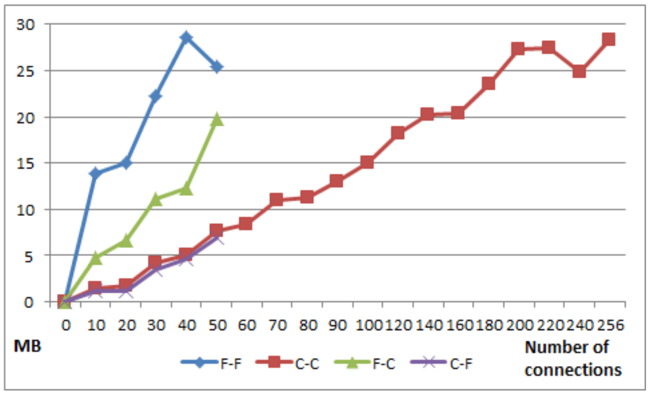


Fig. 9. Memory consumption of multiple idle WebRTC connections (Chrome supports 256, Firefox 50 connections).

scenarios. However, with Firefox (or Firefox and Chrome combinations), the average delay and especially the variation in the delay grow significantly. When examining Firefox and Chrome combinations in Figure 5, it can be seen that Firefox performs substantially better (either as a client or a server) when used with the laptop connected to the WLAN than with a smartphone connected to a mobile network. However, when the laptop and a smartphone are both connected to the WLAN, there is no significant difference in the performance as shown in Figure 6. The most interesting finding in this experiment is that the session establishment time using Firefox is significantly longer compared to Chrome especially when Firefox is used in slow wireless networks.

In Figure 6, S stands for smartphone and L for the laptop. As there were no differences in the connection establishment delays between the two smartphones, Samsung Galaxy S3 4G was chosen for running the client peer. In the WLAN, establishing a WebRTC connection between the peers was expectedly very fast with all web browser combinations, yet Firefox was still clearly the slowest.

B. Session Establishment Overhead

Figures 7 and 8 clearly illustrate that the overhead for establishing a WebRTC connection is rather marginal in terms of required data transmissions. It can be seen in Figure 8 that less than 6KB is sent and 8KB is received when successfully establishing a WebRTC connection. The only notable difference between Firefox and Chrome can be seen in the messaging frequency with the STUN server as shown in Figure 7. When Firefox is used as a client, a substantially larger number of packets is exchanged with the STUN server. The difference presumably results from the different implementations of the ICE protocol as explained in Section IV.A. However, in overall, this more frequent messaging with the STUN server has only negligible effect on the overall amount of bytes transmitted as illustrated in Figure 8.

C. Session Maintenance Overhead

The memory consumption of idle WebRTC connections is presented in Figure 9. The reason for the non-linearity of the curves lies in the garbage collection procedure that was initiated a few times during the experiments. Based on Figure 9, it appears that Firefox consumes more memory for maintaining idle WebRTC connections. However, the overall memory consumption of both web browsers is quite modest. Possible memory leakage of long maintained connections should be investigated further.

The rough estimates regarding the memory consumed by each idle WebRTC connection are presented in Table II. The estimates were calculated as shown in (1). It should be noted that Firefox runs all tabs in a single process, whereas Chrome creates a separate process for each tab.

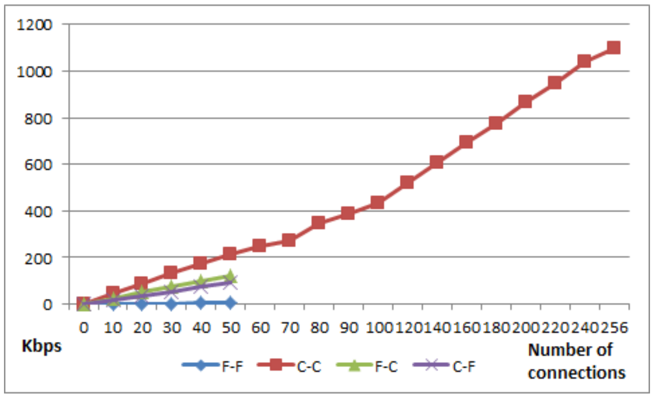


Fig. 10. Data transmission rate for WebRTC connection keepalive messaging between multiple peers (Chrome supports 256, Firefox 50 connections).

TABLE II. AVERAGE MEMORY CONSUMPTION OF A SINGLE IDLE WEBRTC CONNECTION.

F-F	C-C	F-C	C-F
509 KB	111 KB	397 KB	138 KB

$$m_1 = \frac{m_{max} - m_0}{d_{max}} \tag{1}$$

d_{max} = the maximum supported number of WebRTC connections.
m_0 = the memory consumption without any WebRTC connections;
m_1 = the average memory consumption of an idle WebRTC connection;
m_{max} = the memory consumption of the maximum supported number of idle WebRTC connections;

In Figure 10, data transmission rates for idle WebRTC connection keepalive messaging occurring directly between the peers are shown. Chrome (approx. 4 Kbps) uses substantially more bandwidth for keepalive messaging per each idle WebRTC connection than Firefox (approx. 0.1 Kbps). Based on the captured network trace, Chrome sends keepalive messages approximately every 500 ms, whereas Firefox only every 30 s. On one hand, a very short keepalive interval unnecessarily consumes the limited wireless bandwidth and the battery life especially when using mobile networks [12], but on the other hand, a very long keepalive interval may result in long delays in case of network failures.

D. Resource Consumption of Multiple Simultaneous File Transfers

In Figure 11, the CPU load is presented with a varying number of WebRTC connections and wireless network speeds. The general trend in Figure 11 is that the CPU load grows very slowly with the increasing number of WebRTC connections. This is due to the fact that already a single RTCDataChannel occupies the available wireless bandwitdh, and thus, increasing the number of WebRTC connections does not significantly influence the actual amount of data transmitted. Therefore, the increase in the CPU load results primarily from the increased maintenance burden of WebRTC connections.

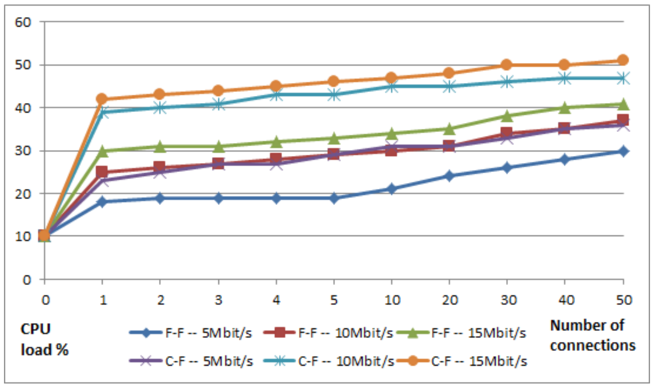


Fig. 11. The CPU load percentage when transmitting data using multiple RTCDataChannels.

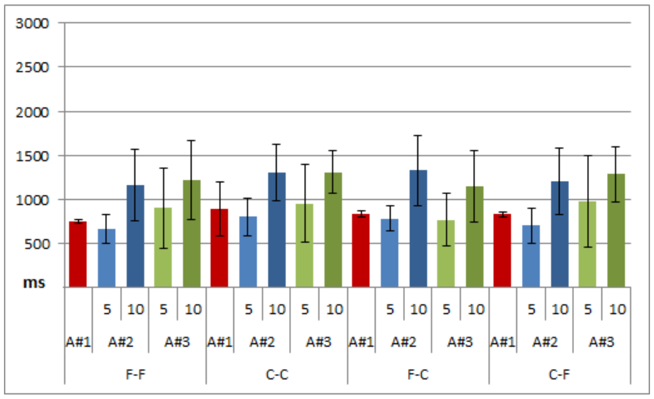


Fig. 12. Transmission times of 1 MB of data with different data transfer approaches (bandwidth: 15 Mbits/s).

It is also evident in Figure 11 that Chrome utilizes the CPU resources more heavily than Firefox. Based on the network trace captured with Wireshark, it was discovered that Chrome sends keepalive messages approximately every 500 ms even during an active file transfer between the peers. This explains at least a part of Chrome's higher CPU utilization.

E. Effectiveness of Different File Transfer Approaches

In Figure 12, the transmission times of 1 MB of data with different data transfer approaches are presented. Although the test was conducted using several different network speeds, only the results for 15 Mbit/s are presented as the different network configurations showed similar behavior. The figure shows average transfer times and the standard deviations calculated based on 30 measurements. Five different measurements were performed for each web browser combination. In the first one (red bar), approach 1 was utilized where the 100 KB data chunks were transmitted one after the other. The second and third measurement utilized approach 2 with five (light blue bar) and ten (dark blue bar) parallel RTCDataChannels. The fourth and fifth measurement utilized approach 3 with five (light green bar) and ten (dark green bar) parallel WebRTC connections.

The results indicate that with approaches 2 and 3 using five parallel data transfers instead of ten provides better performance. This is probably caused by the fact that with more concurrent data transmissions the probability of a single transmission running into congestion problems increases. If congestion occurs, the total transmission time increases. This theory is also supported by the high standard deviations in the measurements with ten parallel data transfers.

In general, approach 3 seems to be the slowest in nearly every measurement. However, approach 3 will logically provide good performance if server peers have lower network bandwidth than the client peer. Approach 2 seems to be the fastest when using five parallel data transfers. The exact circumstances where approach 2 is faster than approach 1 should be studied further, as it seems that approach 2 could boost data transfers between two peers. Similar kind of findings were also discovered in [13] about the use of parallel TCP connections. Firefox seemed to perform slightly better, possibly due to the frequent keepalive messaging sent by Chrome.

V. Conclusions

In this paper, the performance of WebRTC on mobile devices was evaluated with different mobile device, wireless network connectivity and web browser configurations. Using the implemented test environment, we investigated the performance of WebRTC in terms of session establishment delay and overhead, session maintenance overhead, resource consumption of multiple simultaneous file transfers and efficiency of different file transfer approaches.

Based on the evaluation results, it appears that establishing a WebRTC connection between web browsers (i.e. peers) may take a considerable amount of time in certain conditions. When using only Chrome, the average WebRTC connection establishment delay stayed reasonable (below 3 seconds), but with Firefox the delay even exceeded 10 seconds, which is obviously too long time for an end-user to wait. Furthermore, the delay was typically longer when both peers were connected to a mobile network. Poor performance of Firefox is at least partially explained by the fact that Firefox does not seem to implement the Trickle ICE extension to the ICE protocol. Based on our findings, the Trickle ICE extension considerably quickens the WebRTC connection establishment process, and in the future, it should also be implemented in Firefox.

From the standpoint of WebRTC connection maintenance, both Chrome and Firefox behaved quite modestly regarding memory consumption. However, Chrome kept sending a considerable amount of keepalive messages (approx. 4 Kbps per WebRTC connection), which did not cease even in the middle of file transmissions. The keepalive messaging could be piggybagged into the payload data transmissions in order to optimize application performance. From the application developer's standpoint, it would be beneficial to implement the keepalive interval as an adjustable parameter. In addition, measured or estimated peer-wise network attributes, such as latency and available download and upload bandwidth, might be useful in many applications. These features could be valuable additions to the WebRTC API.

Our findings on the CPU utilization indicate that the today's high-end smartphones have enough capacity to effortless run multiple simultaneous file transfers using WebRTC. With the WLAN limited to 15Mbits/s and 50 WebRTC connections transmitting data, the maximum CPU utilization of a smartphone with Firefox was only 40%. With Chrome, the maximum CPU utilization was slightly higher (approx. 50%), but this presumably resulted from the excess amount of unnecessary keepalive messages.

In our future work, we plan to extend our study to cover the WebRTC performance on mobile devices also from the standpoints of multimedia streaming and energy-efficiency. We will also consider real application scenarios and LTE networks.

VI. Acknowledgments

This work is supported by the European Celtic-Plus project CONVINcE and was partially funded by Finland, France, Romania, Sweden and Turkey. This work is also supported by the CADIST3D project funded by the Academy of Finland.

References

[1] S. Androutsellis-Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM Comput. Surv.*, vol. 36, no. 4, pp. 335–371, Dec. 2004. [Online]. Available: http://doi.acm.org/10.1145/1041680.1041681

[2] T. Koskela, O. Kassinen, E. Harjula, and M. Ylianttila, "P2p group management systems: A conceptual analysis," *ACM Comput. Surv.*, vol. 45, no. 2, pp. 20:1–20:25, Mar. 2013. [Online]. Available: http://doi.acm.org/10.1145/2431211.2431219

[3] C. Jennings, T. Hardie, and M. Westerlund, "Real-time communications for the web." *IEEE Communications Magazine*, vol. 51, no. 4, pp. 20–26, 2013. [Online]. Available: http://dblp.uni-trier.de/db/journals/cm/cm51.html#JenningsHW13

[4] T. Koskela, O. Kassinen, E. Harjula, J. Pellikka, and M. Ylianttila, "Case study on a community-centric mobile service environment," *International Journal of Web Applications*, vol. 2, no. 3, pp. 187–205, 2010.

[5] V. Singh, A. Lozano, and J. Ott, "Performance analysis of receive-side real-time congestion control for webrtc," in *2013 20th International Packet Video Workshop (PV)*, Dec 2013, pp. 1–8.

[6] S. Nickels, H. Sminia, S. Mueller, B. Kools, A. Dehof, H. Lenhof, and A. Hildebrandt, "Proteinscanar - an augmented reality web application for high school education in biomolecular life sciences," in *Information Visualisation (IV), 2012 16th International Conference on*, July 2012, pp. 578–583.

[7] G. Mandyam and N. Ehsan, "Html5 connectivity methods and mobile power consumption," Qualcomm, Tech. Rep., 2012.

[8] F. Fund, C. Wang, Y. Liu, T. Korakis, M. Zink, and S. Panwar, "Performance of dash and webrtc video services for mobile users," in *Packet Video Workshop (PV), 2013 20th International*, Dec 2013, pp. 1–8.

[9] J. Nurminen, A. Meyn, E. Jalonen, Y. Raivio, and R. Marrero, "P2p media streaming with html5 and webrtc," April 2013.

[10] F. Rhinow, P. Veloso, C. Puyelo, S. Barrett, and E. Nuallain, "P2p live video streaming in webrtc," in *2014 World Congress on Computer Applications and Information Systems*, Jan 2014, pp. 1–5.

[11] M. Carbone and L. Rizzo, "Dummynet revisited," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 2, pp. 12–20, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1764873.1764876

[12] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: A measurement study and implications for network applications," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '09. New York, NY, USA: ACM, 2009, pp. 280–293. [Online]. Available: http://doi.acm.org/10.1145/1644893.1644927

[13] T. J. Hacker, B. D. Athey, and B. Noble, "The end-to-end performance effects of parallel tcp sockets on a lossy wide-area network," in *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, ser. IPDPS '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 314–. [Online]. Available: http://dl.acm.org/citation.cfm?id=645610.661894