Journal of Digital Imaging (2020) 33:697–707
https://doi.org/10.1007/s10278-019-00305-0

Exploring WebRTC Potential for DICOM File Sharing

Ivan Drnasin [1,3] · Mislav Grgic [2,3] · Gordan Gledec [3]

Published online: 12 December 2019
© Society for Imaging Informatics in Medicine 2019

Abstract
This paper explores the potential use of WebRTC set of protocols for DICOM file exchange. We have developed a simple proof-of-concept peer-to-peer DICOM file-sharing web application based on a set of WebRTC protocols. Application performance is compared with contemporary DICOM applications and transfer protocols which showed that WebRTC has its place in the DICOM file-sharing domain.

Keywords DICOM · DIMSE · DICOMWeb · WebRTC · HTTP

Introduction

Digital Imaging and Communications in Medicine (DICOM) is the international standard used to standardize transfer, storage, retrieval, printing, processing, and display of digital radiology and non-radiology images and objects.

DICOM transfer is built on top of the Transmission Control Protocol/Internet Protocol (TCP/IP) suite and uses two distinct protocols to exchange messages in precise format and order [1]: DICOM Message Service Elements (DIMSE) and DICOMWeb protocol. DIMSE is working directly on the top of the TCP/IP, while DICOMWeb is built on top of the Hypertext Transfer Protocol (HTTP). The DICOM committee first implemented Web Access to DICOM Objects (WADO) standard and then DICOMWeb – more refined Representational State Transfer (RESTful)-based WADO not limited only to file retrieval services but also for storage, Modality Worklist (MWL), and DICOM service discovery. DICOMWeb is a very young standard and still not widely supported, especially in imaging devices.

DICOM storage defines file structure as a header (metadata) and data (sequence of bytes: pixels, video, or audio sequence). One DICOM imaging procedure can vary from 1 file to 50,000 files and can take from 10 KB to 30 GB of storage. DICOM data can be compressed to save storage space and to speed up the transfer.

A very important concept in the DICOM world is sharing/transferring DICOM datasets among the radiologists, specialists, researchers, and patients outside of the imaging department for referral purposes, testing, education, scientific cooperation, research, or teleradiology [2].

DICOM network protocols are widely used for primary diagnosis and review, inside integrated imaging department and other departments which also support DICOM protocols and less frequently used in working from home situation, sharing, teleradiology, testing, education, research, etc. [2].

This gap is closed by VPNs, file-transferring protocol (FTP), email (DICOM standardized), CDs/DVDs (standardized by IHE Portable Data for Imaging), and nowadays via web/cloud technologies including new DICOMweb protocols. Some factors have influenced the popularity of DICOM file sharing via web/cloud services:

- CDs/DVDs are clumsy, insecure, easily damaged, stolen, lost, and can have compatibility issues.
- Email services often have attachment size limits (~20 MB) and are inherently not meant for file sharing, especially big data sets.
- DICOM protocols, FTPs, VPNs, and web services need network infrastructures such as public IP addresses, servers,

✉ Ivan Drnasin
ivan.drnasin@infomedica.hr

Mislav Grgic
mislav.grgic@fer.hr

Gordan Gledec
gordan.gledec@fer.hr

[1] Infomedica, Research and Development, Sime Ljubica 55, 21000 Split, Croatia

[2] Department of Wireless Communications, Faculty of EE and Comp, University of Zagreb, Zagreb, Croatia

[3] Department of Applied Computing, Faculty of EE and Comp, University of Zagreb, Zagreb, Croatia

special software, special configuration, and additional effort to manage and secure them [3].

• The Internet was originally designed as a direct communication system [4], but the depletion of IPv4 addresses and NAT (Network Address Translation) devices has lingered direct Internet connections needed for running personal FTP, VPN, web, or DIMSE services.

• Rise of cloud computing and file-sharing web services, such as Dropbox, Google Drive, Microsoft OneDrive, or specialized cloud medical imaging platforms, has made file sharing via web simple and efficient: user uploads files via a web browser to the cloud and sends link to the recipient.

In the recent decade, some authors, encouraged by the success of P2P file-sharing networks, such as BitTorrent, Skype, Gnutella, Freenet and due to limitations of accessing DICOM nodes via Internet infrastructure, have proposed improving medical data exchange via P2P methodologies, mostly based on research experiments [5]. The main drive for introducing P2P in the DICOM domain was the "discovery of huge volumes of clinical information" for discovery and research [5–8]. But due to the nature of DICOM repositories, security, policies, and legal problems related to the P2P domain, P2P techniques have not found solid ground in the medical imaging domain.

Improving data transfer time is not a new topic in the medical imaging domain. Bandwidth limitations and image size provoked researchers to investigate compression methods, multi-frame grouping, parallelism, bandwidth optimization, and other techniques. In order to address such situations, researchers are exploring new methods. Le Maitre et al. concluded when comparing DIMSE protocol versus DICOMWeb protocol "revealed that the STOW protocol is faster than C-STORE, especially when transferring a large number of DICOM objects" [9]. Rascovsky et al. assume that "document-based databases may be better suited to store current medical imaging data volumes" [10]. Langer et al. believe "to achieve performance that will scale as exam sizes grow, it would seem that fundamentally different protocols and infrastructure may need to be employed as health care providers attempt to do more and faster at a distance" [12]. Maani et al. show that "the combination of compression methods and parallelism also improves transmission time over both LAN and WAN" [13].

Meanwhile, while cloud computing has become a popular way of sharing (DICOM) files via omnipresent web browsers, Google has introduced a Web Real-Time Communication (WebRTC) set of protocols (Table 1). WebRTC is a collection of communication protocols and application programming interfaces (APIs) that standardize and enable real-time peer-to-peer (direct) communication between web browsers [14].

WebRTC uses User Datagram Protocol (UDP) as a transport layer [15] which does not guarantee delivery or security but uses other protocols on top of the UDP for such tasks. In order to encrypt and maintain data integrity, WebRTC uses the Datagram Transport Layer Security (DTLS) protocol. DTLS is the UDP equivalent of TLS (Transport Layer Security) protocol, which is popular and widely used for securing Internet transactions.

For video and audio exchange, RTC protocol and its secure version Secure RTP (SRTP) are used. To guarantee data delivery, WebRTC uses Data Channel API for reliable (TCP like) and simple (WebSocket like) data transfer, such as files or messages. Data channels use Stream Control Transmission Protocol (SCTP), which provides reliable (configurable), stable, message-oriented communication even over unreliable networks such as UDP. SCTP messages are carried by the DTLS protocol [15].

WebRTC needs four types of server-side functionality to make P2P connection work:

1. For hosting WebRTC applications, one can use ordinary web hosting.
2. For user discovery and communication between peers, the signaling server is used.
3. For NAT traversal, WebRTC uses STUN (Session Traversal Utilities for NAT) server.
4. When NAT traversal fails, WebRTC uses TURN (Traversal Using Relays around NAT) server, for relaying data as a last resort.

In order to establish a real-time P2P connection between web browsers (or any other clients on the Internet), WebRTC needs to cope with NAT and firewall devices. IPv4 address depletion has lingered direct P2P connections among the Internet consumers; therefore direct connections are almost impossible. Various techniques have been used to enable P2P VOIP (Voice Over IP) telephony, P2P multiplayer games, P2P computing, and generally P2P services through NAT. The most important task is to traverse NAT devices.

NAT is a mechanism which rewrites IP address in IP packet with another IP address. Main motive for NAT was IPv4 address depletion: 2^{32} (4,294,967,296) possible addresses. NAT was established as a temporary solution until the new IPv6 protocol comes, but it stayed even after introducing the IPv6 protocol [16]. Actually, NAT has slowed down IPv6 introduction [16]. IPv6 should make NAT needless because it can assign a unique IPv6 address to any Internet device. But NAT has become popular in the meantime. Experts assume that NAT will live together with IPV6 for many years [16]. There are two main reasons for that:

1. Reduces the need for global Internet addresses (e.g., entire college or hospital may be located behind one IPv4 address).
2. Security reasons: NAT filters and analyzes incoming internet traffic. For example, bans connections that are not

Table 1 WebRTC features

WebRTC features	
Feature	**Explanation**
Real-time direct data exchange	Enables direct browser to browse data exchange (audio, video, and arbitrary data) in maximum quality in certain environments via UDP-based protocols for data exchange and NAT traversal
Security and integrity	All transferred data (audio, video, and arbitrary data) must be encrypted while in transit via DTLS protocol, an equivalent of popular TLS protocol in TCP/IP network
Simple programming interface	WebRTC hides programming complexity via JavaScript API. Three main APIs are: MediaStream: acquisition of audio and video streams RTCPeerConnection: communication of audio and video data RTCDataChannel: communication of arbitrary application data
Relaying data when direct P2P connection fails	WebRTC uses TURN servers for relaying data when a direct P2P connection fails. TURN serves do not store data, just relaying. They cost money
Multi-platform	Available on all popular desktop and mobile web browsers and desktop operating systems

initiated internally by an organization. It provides a single point for incoming network requests from the Internet (outside) and gives control for all organization computers.

NAT stores data about private IP address inside the LAN network and connects them with public IP addresses assigned to the router device. NAT stores tables/maps which contain IP addresses and ports (IP-port tuple). With NAT, every computer can connect to the Internet with a joint IP address, but can't act as a service provider (e.g., web server) because it is not reachable by its joint public IP address.

In order to cope with NAT traversal, WebRTC uses ICE framework and UDP hole punching technique [15, 17]. When NAT maps IP-port tuple, then the application is allowed to exchange data through NAT in both directions, during some time. Such mappings are called pinholes. Pinholes are set and removed dynamically. The technique which enables direct communication between programs behind different NAT networks is called hole punching [17]. To pin a hole, the client contacts the known public server, and that network request sets mapping in local NAT device. When another client contacts the same public server, the server then has connections to both clients and knows their public IP-port tuples. Then public server exchanges IP-port tuples with both clients Fig. 1. After that, direct communication can start Fig. 2. For that, WebRTC uses signaling and STUN server. In case STUN fails, WebRTC uses already mentioned TURN servers for relaying. There is an estimate that 92% of the connections can take place directly (STUN) while 8% of the connections require a relaying (TURN) [15].

W3C consortium and IETF have standardized [18] WebRTC for P2P data exchange in the desktop and mobile web browsers. WebRTC has already used widely for video/audio communication such as WhatsApp [19], Zoom [20] content sharing, online gaming, distributed video delivery content delivery networks (CDN) [21], torrent networks [22], Internet of Things, etc.

Materials and Methods

With computational power and network connections improving in the consumer domain, we wanted to prototype WebRTC-based DICOM file-sharing service that provides real-time P2P DICOM image exchange, which guarantees security and integrity with integrated DICOM Viewer module. Also, we wanted to explore and compare WebRTC protocols with contemporary DICOM protocols for DICOM file sharing: DIMSE and DICOMWeb.

To test our idea, a simple proof-of-concept (POC) WebRTC web application was developed for DICOM file sharing and visualization between two users. The application was developed in JavaScript/HTML/CSS.

Surge.sh [23] is used for hosting the POC application. The publicly available SignalMaster [24] server is used to coordinate communication and control messages. SignalMaster utilizes the SocketIO library [25] for signaling over WebSocket. SimpleWebRTC.js [26] was used to access the WebRTC API which simplifies the use of the DataChannel API to exchange DICOM files between web peers. Prior to transferring, DICOM files are grouped and compressed by the *JSZip* [27] library in order to shorten data transfer time. For processing DICOM files, the Cornerstone dicomParser.js [28] library was used. Libopenjpeg.js [29] and libCharLS.js [30] libraries are used for decompressing DICOM pixels. Cornerstone Viewer [30] was used to display images, with basic functions such as zoom, pan, windowing, and stacking. User interface was built with the Bootstrap toolkit [31]. Figure 3 presents all modules and system architecture.

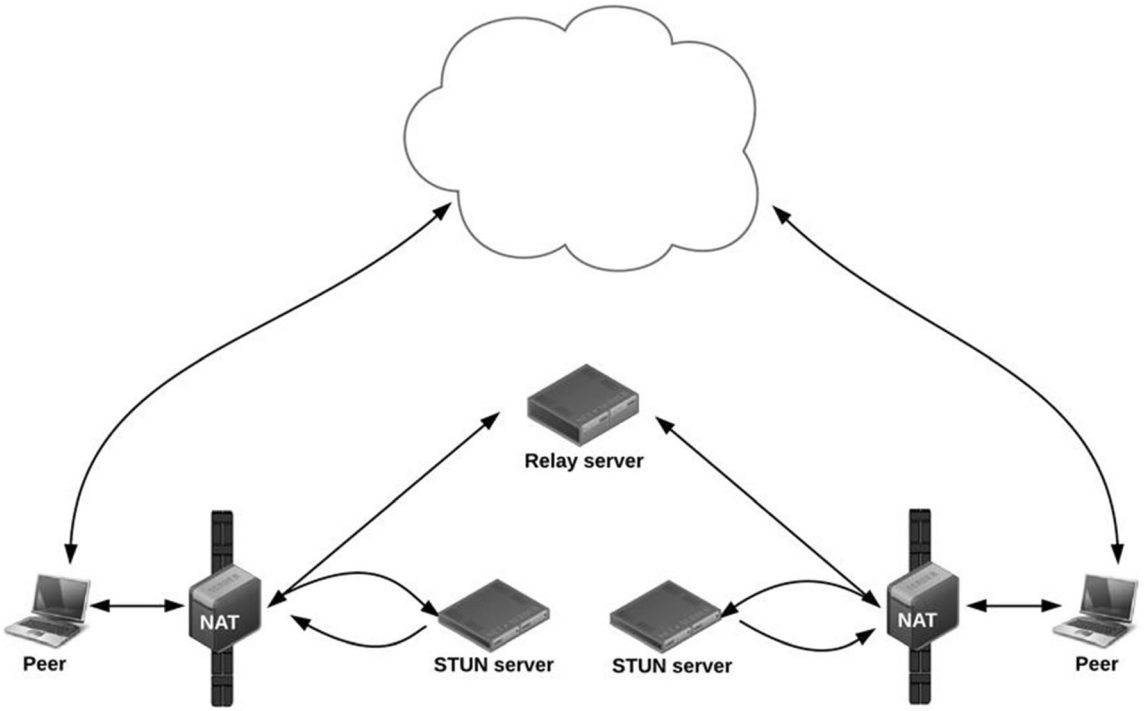

Fig. 1 WebRTC connection initiation [17]

We have tested performance on application and protocol level. Application data transfer time was compared to ClearCanvas Workstation/Server [32] (DIMSE protocol) and DICOMCloud [33] (DicomWeb STOW-RS protocol). Four DICOM studies (Table 2) were used for testing the performance of both systems. Five specific programs were prepared for protocol-level testing (Fig. 4):

1. JavaScript DataChannel program using simple-peer library [34] for testing WebRTC transfer: we have written

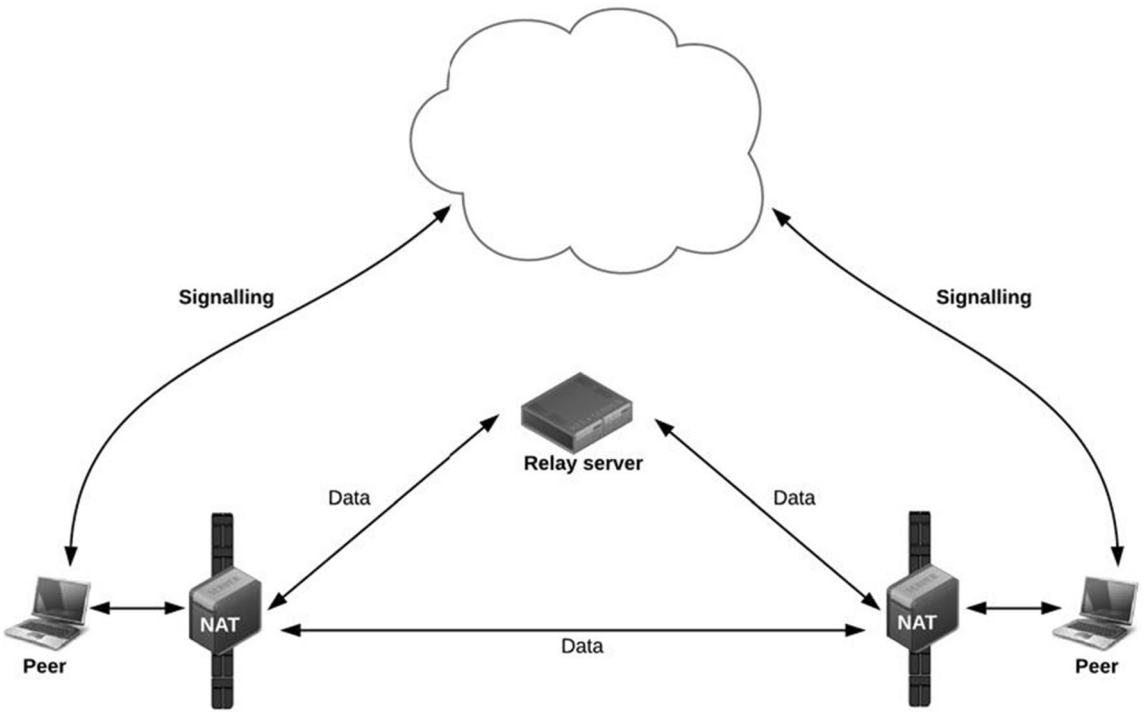

Fig. 2 WebRTC data pathways [17]

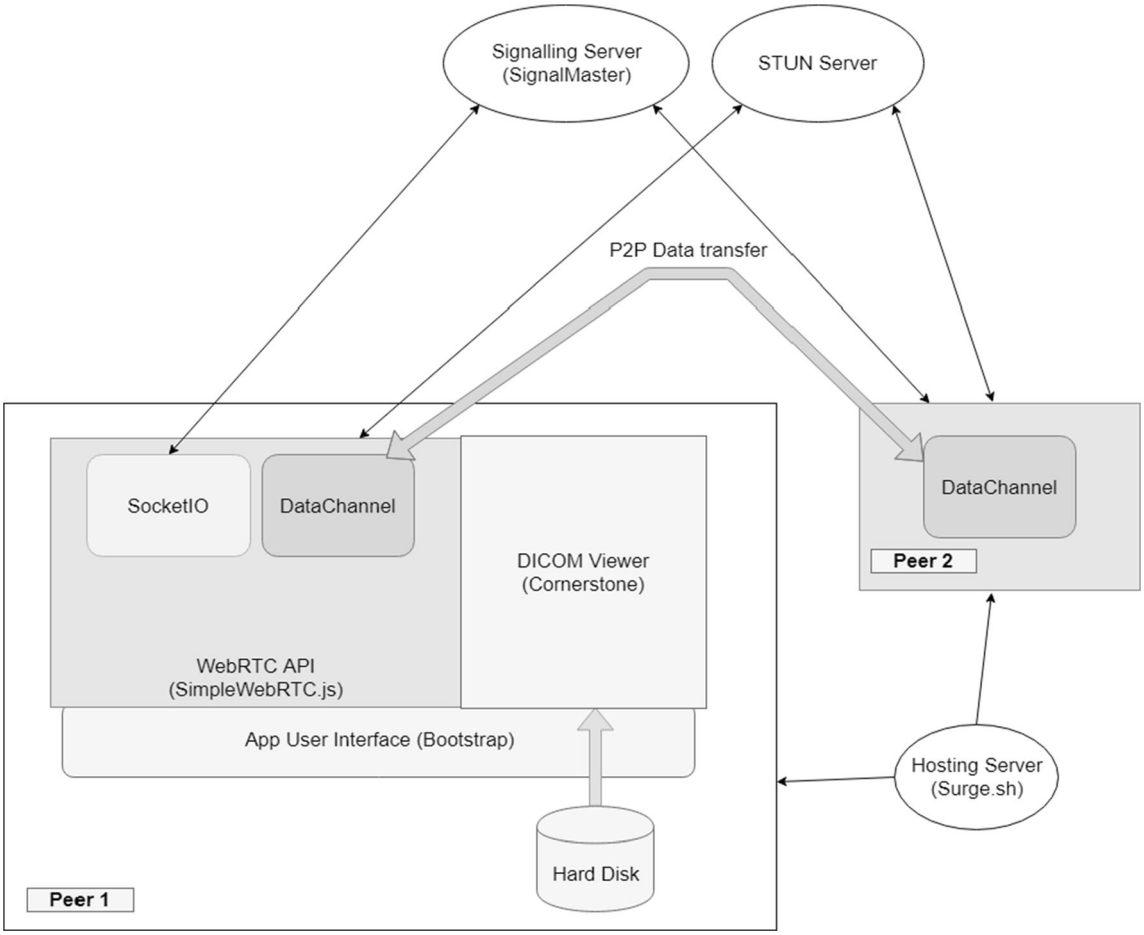

Fig. 3 The system architecture of DICOM file-sharing POC application

a simple JavaScript program [35] that transfers N files from peer A to peer B, without storing images.

2. Modified DICOMWeb JavaScript client [36] for uploading DICOM files to the STOW-RS web server. Modifications include synchronous operations and no limitations for file uploading.
3. Modified DICOMWeb STOW-RS web server: we have removed image processing disk storing operations and left only reading the image from memory to temporary memory buffer.
4. C# DIMSE SCU based on Fo-Dicom library [37] for sending DICOM files to STORE SCP server, single-threaded, synchronous application.
5. DIMSE Store SCP server based on Fo-Dicom library [38]: no image reading, no storing to disk, just receiving images from network.

Image transfer was measured in 3 different network environments: LAN 1 Gbps bandwidth; Internet 50 Mbit bandwidth, and LAN 10 Mbit bandwidth simulated via NetLimiter software [39].

Files were transferred using 3 different protocols (Fig. 4) from source computer (Windows 10, i7 8 GB RAM, 50 Mbit upload bandwidth) to local Windows Server 2012 machine in LAN (i5, 8 GB RAM, 1 Gbps bandwidth) and to remote

Table 2 DICOM datasets used for testing POC application and protocol performance

ID	Dataset	Compression	Size of 1 file	Files number	Total size	Comment
A	US	No	127 MB	1	127 MB	Single large file
B	MR	No	516 KB	232	65 MB	Small MR case
C	MG	No	54 MB	4	217 MB	A small number of large files
D	PET	Yes, Lossy	~ 68 KB	4333	298 MB	A large number of small files

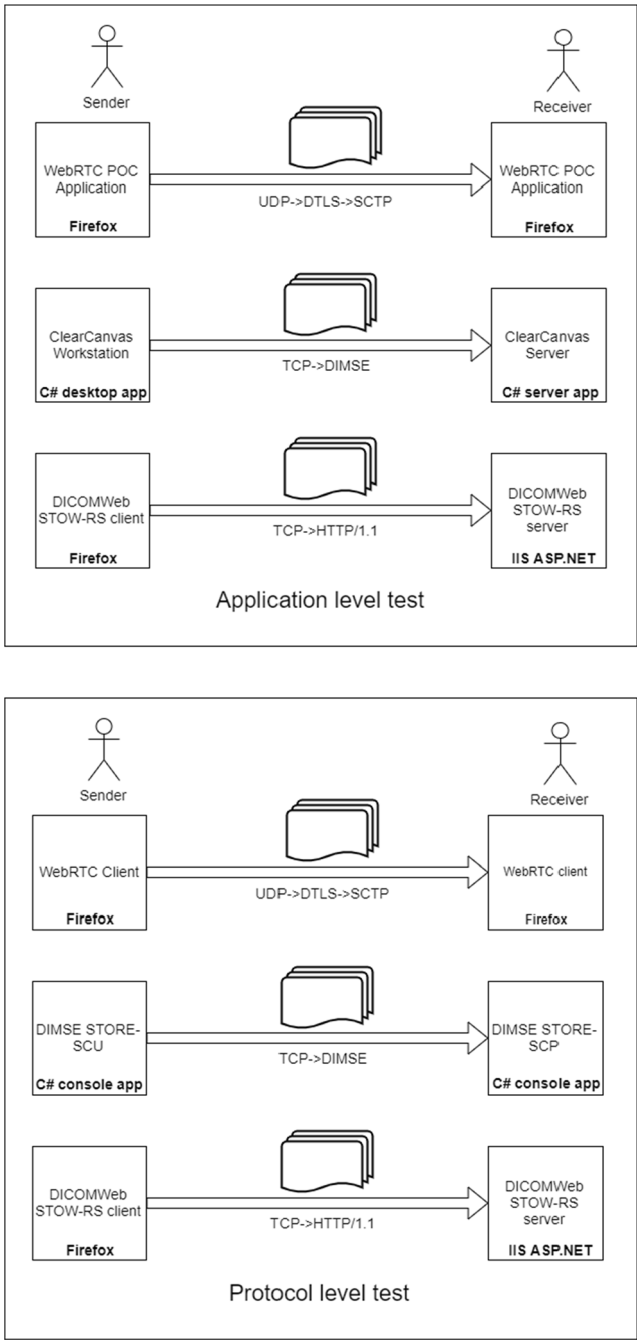

Fig. 4 Sending and receiving programs for application-level test (POC, ClearCanvas, DICOMCloud) and protocol-level testing programs (WebRTC client, DIMSE STORE SCU/SCP, DICOMWeb client/server)

Windows Server 2012 (Amazon Frankfurt t2.medium EC2 instance, 503/749 Mbps bandwidth).

Mozilla Firefox was used (version 69.0.1) for testing web clients (WebRTC and DicomWeb) since it allows usage of larger WebRTC packets size – 65 KB [40]. Measurements were performed 3 times; average time and standard deviation were calculated. Files were sent one after another, synchronously, no multithreading. Only transfer time was measured, excluding image loading and preparation. A programmable timer was used, started on sending first, and finished on receiving the last file; all measurements are in milliseconds. IIS was a tuned server to allow maximum worker processes with "unlimited" memory. All computers had Windows firewall enabled and were behind a corporate network (NAT, router, firewall). For establishing P2P connections, a publicly available Google STUN server was used. ClearCanvas Workstation uses multiple threads (6) for DICOM connections. DICOMWeb STOW-RS client [41] was modified to use 4 simultaneous AJAX connections for uploading files to the STOW-RS server [41].

Results

Developed application includes features: reading DICOM files from hard disk; automatically organizing DICOM files in tree hierarchy; displaying DICOM images with basic set of tools; creating room for connecting with peers via URL; sending study or series to another party; visualizing the receiving of DICOM files; optionally compressing (depends on transfer syntax) and grouping DICOM files before sending; and downloading received DICOM files from sender. The running application can be temporarily accessed at Surge.sh hosting [42]. Test applications are available on Git servers [35–38].

The simple workflow of the application includes the following steps (Fig. 5):

1. Once the app is loaded in a web browser, communication room or channel needs to be defined.
2. Room URL will be used for other parties to access the same communication channel.
   a. URL can be shared with other parties (email, instant messaging, SMS, etc.).
3. Open DICOM images from hard disk, USB drive, or CD/DVD drive.
4. Choose what to share.
5. Wait for the receiver to connect.
6. Send study or series as a grouped ZIP archive (compressed optionally).

The application interface is shown in Fig. 6. The application automatically detects DICOM images and sorts images and series in the DICOM tree. The application does not communicate with DICOM nodes via DIMSE protocol. It is a simple DICOM file-sharing application between 2 users, meant for patients, doctors, or scientists.

The application performance test is presented in Fig. 7.

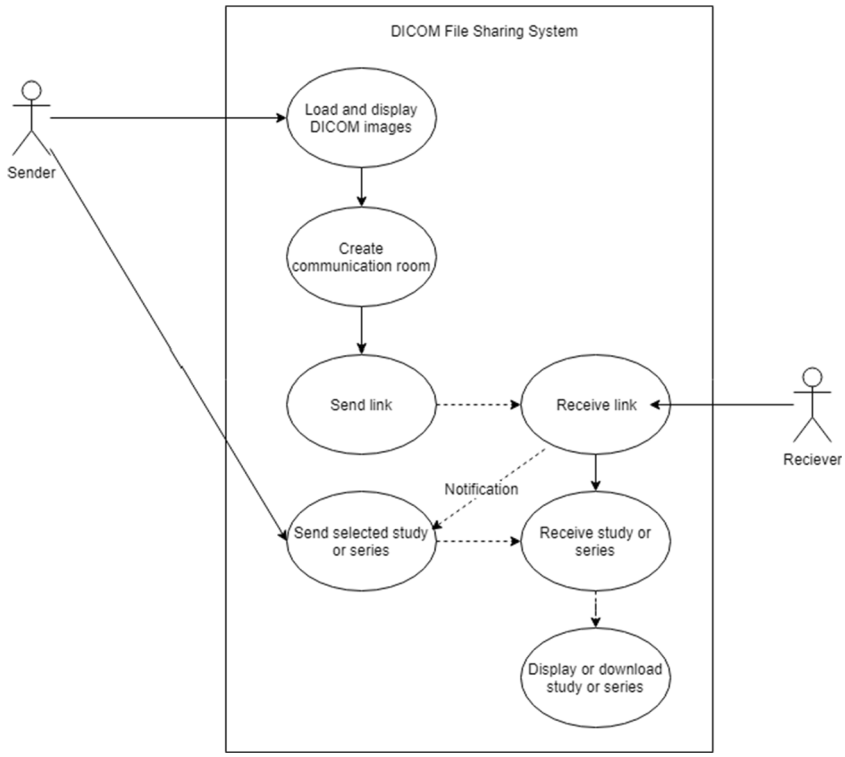

Fig. 5 Use case diagram of DICOM file-sharing POC application

Protocol level applications were also developed and used for measurements, presented in Figs. 8, 9, 10.

For developed POC application and test programs, there are some caveats:

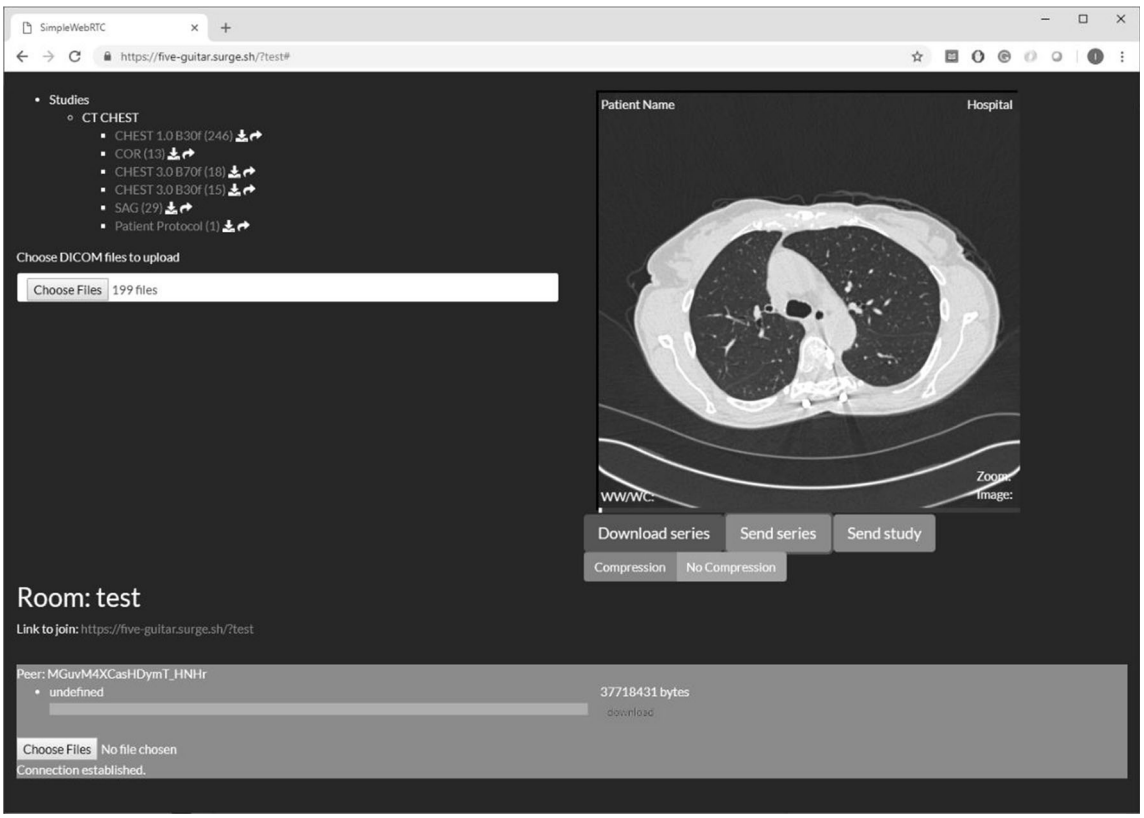

Fig. 6 POC application screenshot

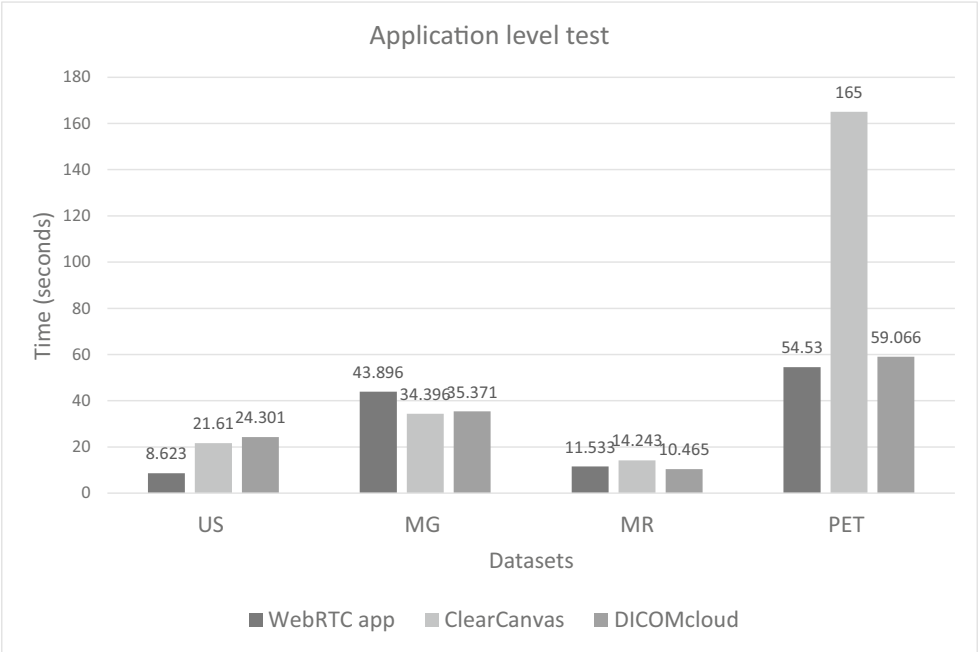

Fig. 7 Application-level test results for 50 Mbps bandwidth on the Internet

• WebRTC encrypts data via DTLS protocol, which means more CPU cycles and additional handshakes for transmission, but it does not slow transmission noticeably. [43].

• WebRTC test protocol program does not signal when the complete file is received; it is simple as possible. It means that protocol could be a little slower, but according to our experience, no significant difference would occur.

• We have used HTTP/1.1 protocol for DICOMWeb STOW-RS, whereas HTTP/2 could bring more performance.

• We have not encrypted communication with TLS protocol in DIMSE and STOW-RS, TLS adds more delay but does not affect transmission noticeably [43].

• We have not used the TURN server; therefore P2P connections behind strict NAT devices will not work (never less, production-grade TURN servers are available on the market).

Discussion

WebRTC is not a new technology anymore, being widely in both desktop and handheld devices, offering different services such as audio/video P2P communication, data transfer offloading, P2P streaming, web torrents, file sharing, etc. From a technical point of view, such concepts are responsible

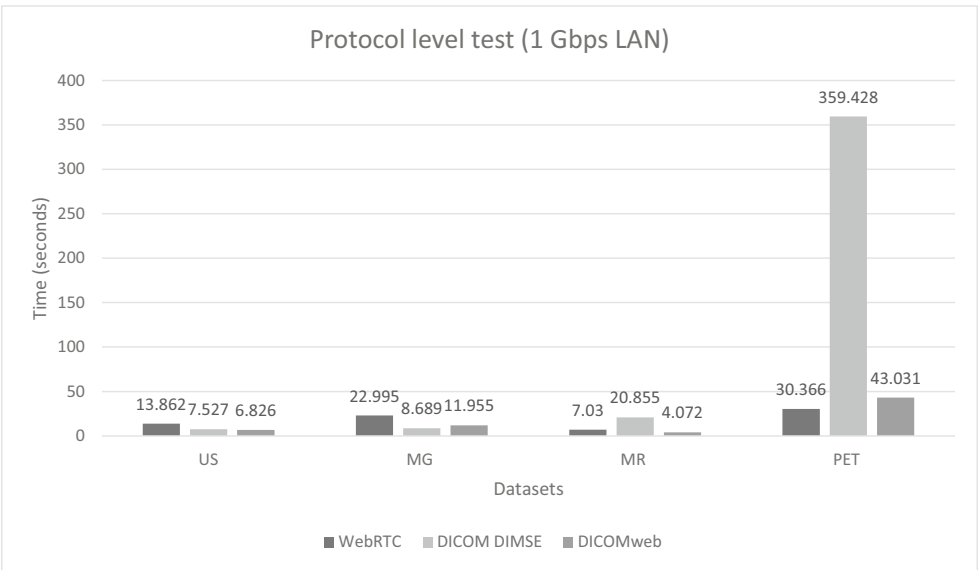

Fig. 8 Protocol level results for 1 Gbps bandwidth in LAN

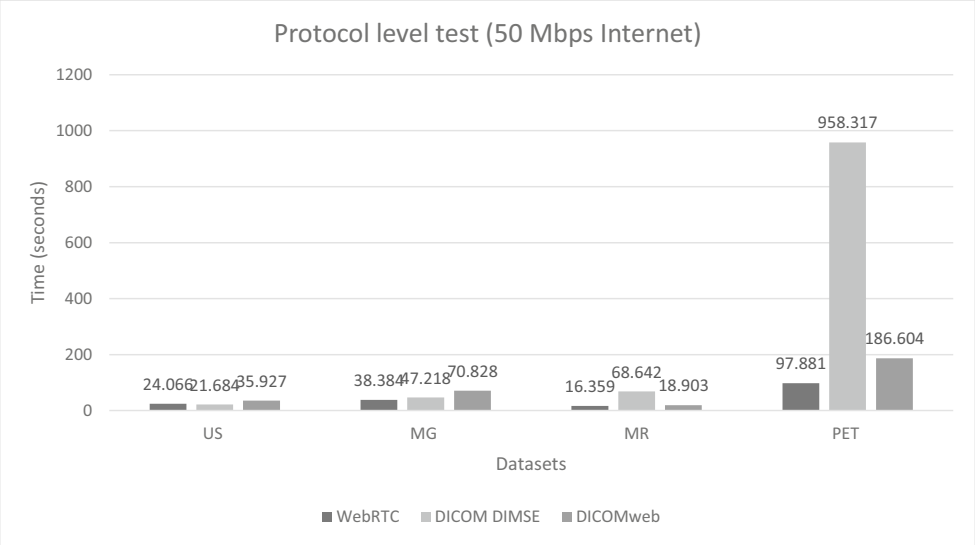

Fig. 9 Protocol level results for 50 Mbps bandwidth on the Internet

for massive data transfer among Internet users [11]. WebRTC has brought different technical tools to achieve such an amount of direct communication on the Internet.

We have conducted measures and compared the WebRTC data transfer protocol with 2 standard DICOM communication protocols.

First, we have compared the POC application with ClearCanvas and DICOMCloud applications. There is no "winner." POC application is fastest in transferring uncompressed files due to the automatic compression. ClearCanvas uses multiple threads for transactions and therefore achieves best results with smaller datasets, while DIMSE limitation is visible, even with multithreading, when working with a large number of files. DIMSE loses some time in negotiating presentation contexts, transfer syntax, and other parameters before actual file transmissions [1] and is not able to send a group of files without multiple negotiations, therefore slowing transfer rates even more. DICOMCloud, on the other hand, is the slowest when working with a single file while highly effective when multiple smaller files are involved. It is important to say that POC application is not able to use multiple threads: everything operates in a single thread, whereas DIMSE and STOW-RS use multiple connections to the servers.

Protocol level test programs were single-threaded and weaknesses of DIMSE are more visible. DIMSE wins for larger files in smaller numbers but only in high-speed environments. On slower networks, WebRTC and STOW-RS win over DIMSE. For a large number of files, DIMSE shows all protocol problems. WebRTC is almost identical to STOW-RS.

When comparing protocol level results with Le Maitre et al. [9], DIMSE was also the slowest when the number of

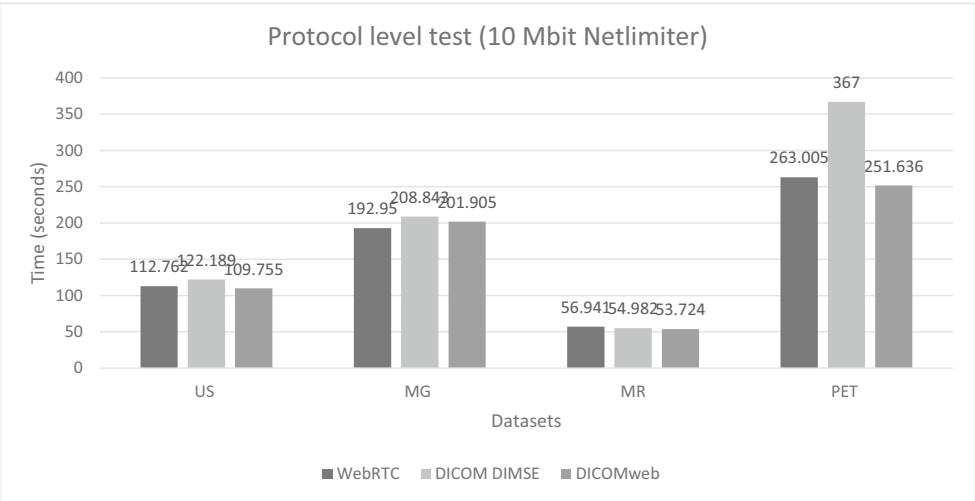

Fig. 10 Protocol level results for 10 Mbps bandwidth in LAN (simulated via NetLimiter)

files is increasing. For such cases as MG and CR, DIMSE was the best option, while STOW-RS showed slower transmission rates when sending a smaller number of large files (30–50 MB).

There are currently limitations in POC application: (1) only 2 peers are supported, so it is meaningful to share when both peers are online at the same time. Advanced libraries like WebTorrent support multiple peers communicating at the same time. (2) If any of peers turns off the device or disconnects, the transfer is interrupted; we have not implemented automatic reconnecting and detection.

For any WebRTC application, the biggest problem is certainly NAT restriction which can block direct connections. Therefore, TURN servers must relay all traffic, and relaying is not free (e.g., production-grade global TURN relaying for 150 GB monthly bandwidth cost $83 per month [44]).

WebRTC technology is now standardized by W3C and IETF; transmission encryption and integrity are guaranteed by DTLS, used, and proved in many different applications. It is made for connecting people. It brings the potential for connecting health institutions too, especially for broadcasting data between the group of people. It could be used for emergency cases since it needs less infrastructure for enabling direct network connections. One can think of a future where every DICOM device can send datasets to anywhere in a secure manner without complex server/network infrastructure. WebRTC is constantly improving. One of the initiatives is to include QUIC (Quick UDP Internet Connection) protocol in WebRTC specification [45].

Conclusion

In this article, we have presented a simple proof-of-concept peer-to-peer DICOM file-sharing application for both doctors and patients. Such a concept is not meant to replace any existing standardized way of DICOM file exchange to share but rather to offer a simplified and safe way of exchanging DICOM files for doctors and patients, by utilizing mobile/desktop web browsers and Internet connections. POC application automatically groups and compresses DICOM files and therefore shortens the time to exchange files. Additional improvement comes from the absence of the middle web server/cloud system between peers. It means that data exchange is direct and there is no need for additional time to download files from the server. Benefits are especially visible when working with uncompressed DICOM datasets and with a large number of DICOM files.

WebRTC data transfer protocol was compared to DICOM standard transmission protocols. We can conclude that P2P protocol based on WebRTC in some cases improves transfer time, by reducing overall time to transfer the DICOM dataset. Since WebRTC protocol is flexible and configurable, therefore one can customize protocol to suit better for the purpose.

We believe that WebRTC could find its place in personal DICOM file sharing and maybe could help in reducing overall turnaround time in emergency cases, such as stroke cases where “time is a brain” or on-call duty. WebRTC is a proven and mature technology that might find its place in the medical imaging and communication domain.

References

1. Pianyhk OS: DICOM practical introduction, and survival guide. Berlin: Springer, 2008
2. DICOM Standards, Supplement 54: DICOM MIME Type: http://dicom.nema.org/DICOM/supps/sup54_pc.pdf
3. Kammerer FJ, Hammon M, Schlechtweg PM, Uder M, Schwab SA: A web-based cross-platform application for teleconsultation in radiology. Journal of Telemedicine and Telecare 21(6):355–363, 2015
4. Oram A: Peer-to-peer: "harnessing the benefits of a disruptive technologies". California: O'Reilly Media, Inc, 2010
5. Costa C, Ferreira C, Bastião L et al.: Dicoogle. J Digit Imaging 24: 848, 2011
6. Blanquer I, Hernadez V, Mas F: "A peer-to-peer environment to share medical images and diagnoses providing context-based searching". In Proceedings of the 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing. Washington: IEEE Computer Society, 2005
7. Maglogiannis I, Constantinos D, Kazatzopoulos L: Enabling collaborative medical diagnosis over the internet via peer-to-peer distribution of electronic health records. J Med Syst 30(2):107–116, 2006
8. I. Maglogiannis, C. Andrikos, G. Rassias, P. Tsanakas. "A DICOM based collaborative platform for real-time medical teleconsultation on medical images". In: Vlamos P. (eds) GeNeDis 2016. Advances in experimental medicine and biology, Vol. 989, Springer, Cham.
9. Le Maitre A, Fernando J, Morvan Y, Mevel G, Cordonnier E: Comparative performance investigation of DICOM C-STORE and DICOM HTTP-based requests, 2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 2014, pp. 1350–1353
10. Rascovsky SJ, Delgado JA, Sanz A, Calvo VD, Castrillom G: Informatics in radiology: use of CouchDB for document-based storage of DICOM objects. Radiographics 32(3):913–927, 2012
11. 10 Massive Applications Using WebRTC, available at: https://bloggeek.me/massive-applications-using-webrtc/ (last accessed: 2019–10–17)
12. Langer SG, French T, Segovis C: C. TCP/IP Optimization over Wide Area Networks: implications for teleradiology. Journal of Digital Imaging 24:314–321, 2011
13. Maani R, Camorlinga S, Arnason N: A parallel method to improve medical image transmission. Journal of Digital Imaging 25:101–109, 2012
14. WebRTC, available at: webrtc.org (last accessed: 2018-09-09)
15. Grigorik I., High Performance Browser Networking
16. Stevens W: TCP/IP illustrated volume 1, 2nd edition, 2014
17. State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs)
18. W3C Editor's Draft, WebRTC 1.0: Real-time Communication Between Browsers, **available at** https://w3c.github.io/webrtc-pc/

19. What's up with WhatsApp and WebRTC?, available at: https://webrtchacks.com/whats-up-with-whatsapp-and-webrtc/ (last accessed: 2019-07-08)
20. Hancke P., How Zoom's web client avoids using WebRTC (DataChannel Update), **available at** https://webrtchacks.com/zoom-avoids-using-webrtc/
21. Peer5 The World Cup and P2P Streaming, available at: https://docsend.com/view/ern48k8 (last accessed: 2018-09-09)
22. WebTorrent, available at: https://webtorrent.io/faq (last accessed: 2018-09-09)
23. Surge.sh static hosting, available at: surge.sh (last accessed: 2018-09-09)
24. SignalMaster signaling server, available at: http://github.com/andyet/signalmaster (last accessed: 2018-09-09)
25. Socket.IO library, available at: socket.io (last accessed: 2018-09-09)
26. Simple WebRTC library, available at: http://github.com/andyet/SimpleWebRTC (last accessed: 2018-09-09)
27. Javascript ZIP library, **available at:** http://github.com/Stuk/jszip (last accessed: 2018-09-09)
28. Cornerstone DICOM parser, available at: http://github.com/cornerstonejs/dicomParser (last accessed: 2018-09-09)
29. Cornerstone Open JPEG library, **available at:** http://github.com/cornerstonejs/openjpeg/ (last accessed: 2018-09-09)
30. Cornerstone project, available at: http://github.com/cornerstonejs/ (last accessed: 2018-09-09)
31. Bootstrap, available at: http://github.com/twbs/bootstrap (last accessed: 2018-09-09)
32. Clear Canvas, https://github.com/ClearCanvas/ClearCanvas (last accessed: 2018-07-08)
33. DicomCloud, https://github.com/DICOMcloud/DICOMcloud (last accessed 2019-07-08)
34. Simple-peer library, available at https://github.com/feross/simple-peer (last accessed: 2019-10-17)
35. Simple-peer DataChannel protocol test, available at: https://bitbucket.org/snippets/willy_skipper/KroBa7 (last accessed: 2019-10-17)
36. DICOMWeb client protocol test, available at: https://bitbucket.org/snippets/willy_skipper/znMoxn last accessed: 2019-10-17)
37. C# DIMSE SCU protocol test, available at: https://bitbucket.org/snippets/willy_skipper/qnBox7 (last accessed: 2019-10-17)
38. DIMSE Store SCP server based on Fo-Dicom library, **available at:** https://bitbucket.org/snippets/willy_skipper/7nBoxk (last accessed: 2019-10-17)
39. Netlimiter, available at: https://www.netlimiter.com/ (last accessed: 2019-10-08)
40. Large Data Channel Messages, available at: https://blog.mozilla.org/webrtc/large-data-channel-messages/ (last accessed: 2019-10-08)
41. DICOMWeb JS client, **available at:** https://github.com/DICOMcloud/DICOMweb-js (last accessed: 2019-10-08)
42. Dicom.live - Dicom file-sharing project, available at: http://dicom.live (last accessed: 2019-07-08)
43. Is TLS fast yet?, **available at:** https://istlsfastyet.com/ (last accessed: 2019-10-08)
44. Xirsys, available at: https://xirsys.com/pricing/ (last accessed: 2019-10-17)
45. QUIC API for Peer-to-peer Connections, available at https://w3c.github.io/webrtc-quic/ (last accessed: 2019-15-10)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.