

JavaScript Access to DICOM Network and Objects in Web Browser

Ivan Drnasin, Mislav Grgić & Goran Gogić

Journal of Digital Imaging

The Journal of the Society for Computer Applications in Radiology

ISSN 0897-1889

J Digit Imaging

DOI 10.1007/s10278-017-9956-7

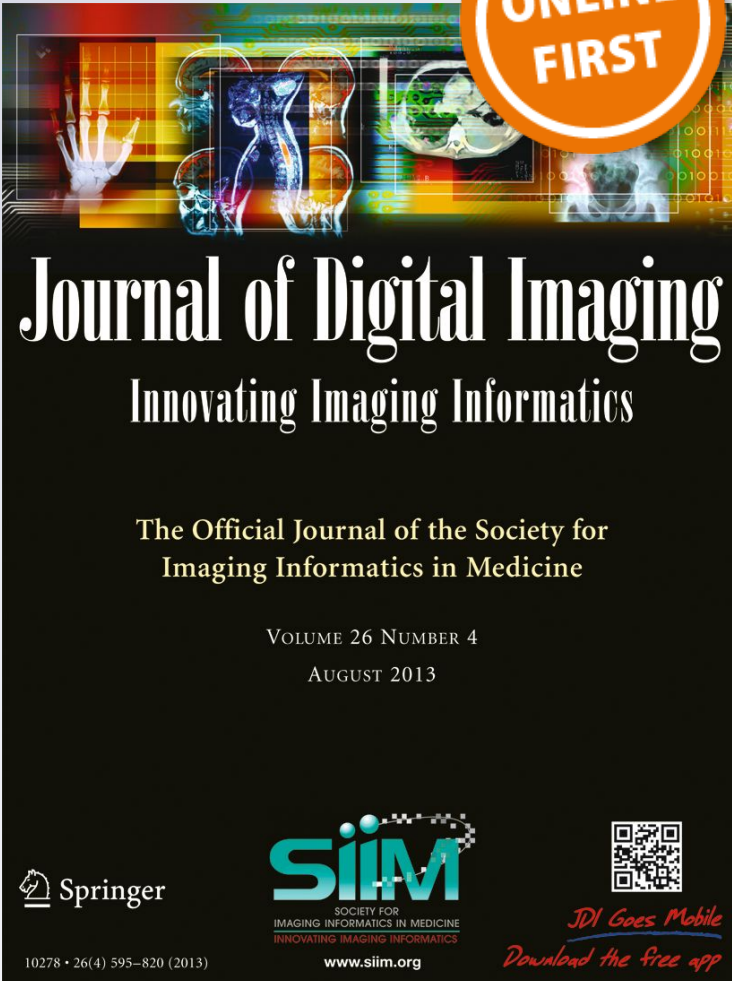
ONLINE FIRST


Journal of Digital Imaging


Innovating Imaging Informatics

The Official Journal of the Society for
Imaging Informatics in Medicine

VOLUME 26 NUMBER 4
AUGUST 2013

 Springer

 **SIIM**
SOCIETY FOR
IMAGING INFORMATICS IN MEDICINE
INNOVATING IMAGING INFORMATICS
www.siiim.org



JDI Goes Mobile
Download the free app

10278 • 26(4) 595–820 (2013)

Your article is protected by copyright and all rights are held exclusively by Society for Imaging Informatics in Medicine. This e-offprint is for personal use only and shall not be self-archived in electronic repositories. If you wish to self-archive your article, please use the accepted manuscript version for posting on your own website. You may further deposit the accepted manuscript version in any repository, provided it is only made publicly available 12 months after official publication or later and provided acknowledgement is given to the original source of publication and a link is inserted to the published article on Springer's website. The link must be accompanied by the following text: "The final publication is available at link.springer.com".

JavaScript Access to DICOM Network and Objects in Web Browser

Ivan Drnasin¹ · Mislav Grgić² · Goran Gogić¹

© Society for Imaging Informatics in Medicine 2017

Abstract Digital imaging and communications in medicine (DICOM) 3.0 standard provides the baseline for the picture archiving and communication systems (PACS). The development of Internet and various communication media initiated demand for non-DICOM access to PACS systems. Ever-increasing utilization of the web browsers, laptops and handheld devices, as opposed to desktop applications and static organizational computers, lead to development of different web technologies. The DICOM standard officials accepted those subsequently as tools of alternative access. This paper provides an overview of the current state of development of the web access technology to the DICOM repositories. It presents a different approach of using HTML5 features of the web browsers through the JavaScript language and the WebSocket protocol by enabling real-time communication with DICOM repositories. JavaScript DICOM network library, DICOM to WebSocket proxy and a proof-of-concept web application that qualifies as a DICOM 3.0 device were developed.

Keywords DICOM · WebSocket · HTTP · HTML5 · JavaScript · PACS · Teleradiology · Internet

Introduction

Three and half billion people use Internet [1]. Internet services include e-mail, news services, file transfer, and world wide web—an information space where documents and other web resources are identified by Uniform Resource Locators (URLs), accessed via HyperText Transfer Protocol (HTTP) protocol, presented as web pages rendered and displayed by web browser. Web pages are designed in HyperText Markup Language (HTML), cascading style sheets (CSS), and JavaScript language. Nowadays, web pages are mostly accessed by mobile web browser, found in smartphones and tablets instead of desktop web browsers. Web browsers are found on all operating systems and on all devices, both handheld and desktop. While web is mostly accessed by web browsers, web services allow access to web via HTTP protocol to any HTTP client outside of the web browser and web pages.

Web Applications

Web applications (web apps) are task-oriented web pages. They are not meant for sole presentation of web resources but aimed to accomplish some task and to provide similar user experience as in the native applications. The main differences are that the program code for web application resides on a web server, can be managed remotely for all the clients, and is executed by the web browser or web server rather than the operating system itself.

✉ Ivan Drnasin
ivan.drnasin@infomedica.hr

Mislav Grgić
mislav.grgic@fer.hr

Goran Gogić
goran.gogic@infomedica.hr

¹ Infomedica, Research and Development, Split, Croatia

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

Web apps can be both thin and thick clients. The task can be executed on a web server and the results (can be) sent back to a web browser, or the web browser can load data from the web server or from user input, and execute JavaScript code on the data. The rationale is that always increasing CPU and memory power is usually unused on personal computers by web browsers. Until recently, those resources could be only used by third-party plug-ins such as Adobe Flash, Java, or Silverlight. The reason for that lies in the fact that web browsers are unable to access OS interfaces and are lacking in performance, due to slow JavaScript code execution.

HTML5 and JavaScript

The HTML5 standard is a successor of HTML4 and XHTML1 standard. Published in October 2014, HTML5 has addressed the weaknesses of previous standards, improved multimedia and graphical support, added various new JavaScript application programming interfaces (APIs), and allowed access to peripheral devices. HTML5 standard is designed with low-power footprint and is meant for cross-platform application development. Such improvements include raw binary data access via TypedArray data types, full duplex communication via WebSocket protocol, rendering pixels in Canvas element, access to GPU via WebGL API, multithreading via WebWorkers API, and offline data storage via IndexedDB API and FileSystem API.

The JavaScript language, as an important part of the new HTML5 standard, was greatly improved and published in December 2009 in its fifth edition. Parallel with that, web browser vendors also improved the execution speed of JavaScript codes and began a race to reach the native speed of C/C++ languages. Such improvements in HTML standard and JavaScript execution have boosted single-page web app development and enabled cross-platform desktop and mobile application development with web languages (HTML, CSS, and JavaScript). Such projects like Apache Cordova, React Native, NativeScript, and Electron framework are very popular among web developers. Server side programming in JavaScript has also become popular during the last few years. According to some relevant surveys, JavaScript is the most commonly used programming language on earth [2].

HTTP Protocol

HTTP is a unidirectional protocol, built on top of a TCP/IP protocol. Request is initiated by the client, while the server processes and returns a response. HTTP allows the request message to go from the client to the server and then the server sends a response message to the client. Usually, a new TCP connection is initiated for an HTTP request and terminated after the response is received. A new TCP connection needs to be established for another HTTP request/response.

Web Standards for Medical Imaging

Challenge to access medical imaging repositories (PACS, Modality, Workstation) via ubiquitous web browsers and HTTP protocol was even more appealing with the expansion of web browsers to mobile and tablet devices. Image access from the palm of the radiologists' hands is even attractive today as it was before. Simplicity of the web browser, usually preinstalled on a device, together with central application management and deployment, is beneficial both for vendors and users. Articles from the early nineties dealt with this topic already; however, the web access to medical images increased lately, even in closed hospital network systems.

DICOM networks, which are the backbone of medical imaging repositories, are built on top of a TCP/IP protocol where DICOM compliant devices exchange DICOM messages and objects. To communicate with DICOM devices, DICOM standard uses its own network language, which is contained from DICOM Message Service Elements (DIMSEs). More DIMSEs create one DICOM network service (Table 1). The main goal of DICOM communication is data exchange in strictly defined format and order [3]. DICOM standard data exchange is service based. DICOM device requesting service from another DICOM device is labeled as service user class (SCU), while the service provider class (SCP) label is used for service provider. All DICOM messages and objects are binary data chunks that travel through both networks.

DICOM standard officials were aware of the global demand for web access to DICOM repositories, leading to the development of Web Access to DICOM Objects (WADO) standard and its ratification in 2004. WADO allows access to DICOM objects on web-enabled DICOM server through the Hypertext Transfer Protocol/Secure (HTTP/S) protocol using DICOM unique identifiers (UIDs) (Fig. 1). Data may be retrieved either in a presentation-ready format (JPEG or GIF) or in a native DICOM format. WADO does not support web searching of DICOM repositories [4, 5].

The Medical Imaging Network Transport (MINT) group was formed in 2010 to improve transfer speed of DICOM studies, eliminate need for DICOM routing, and address some

Table 1 DIMSE network services

| Service | Description |
|---------|--|
| C-ECHO | Connection check |
| C-STORE | DICOM objects store |
| C-FIND | Query o DICOM objects |
| C-GET | Transfer of DICOM objects, where receiver initiates connection |
| C-MOVE | Transfer of DICOM objects, where receiver does not need to initiate connection |

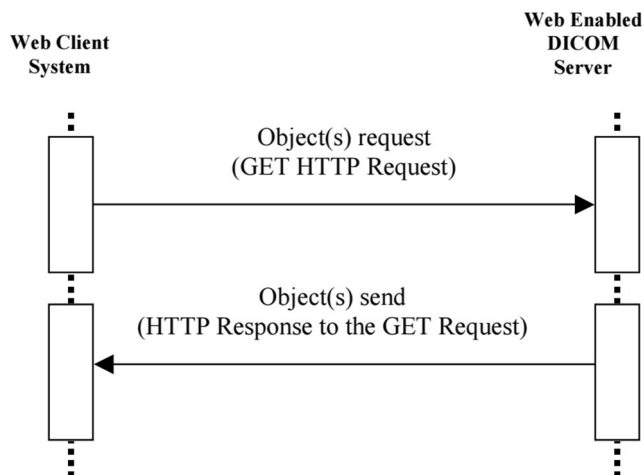


Fig. 1 WADO request

WADO weaknesses [6]. The MINT proposal was rejected by the DICOM committee [7].

WADO by means of Web Services (WADO-WS) supplement was added to the DICOM standard in 2011 [8]. WADO-WS defines Web Services for providing DICOM images and other persistent objects to an Electronic Medical Record/Electronic Health Record (EMR/EHR) system. The supplement deals only with retrieval, corresponding to the evolution of the existing WADO to Web Services. Query and notification mechanisms are not defined within this supplement. Both native DICOM and rendered images can be retrieved as well as total or partial metadata of the object without the image pixels.

WADO by means of RESTful Services (WADO-RS) is another supplement to the DICOM standard, added in 2013 [5]. It defines representational state transfer (REST) services for providing DICOM images and other persistent objects to an electronic medical record/electronic health record (EMR/EHR) system. This supplement deals with retrieval, corresponding to the evolution of the existing WADO to RESTful Services. Native DICOM can be retrieved, as well as it can separate bulk data, pixel data, or metadata of the object.

Query based on ID for DICOM Objects by RESTful Services (QIDO-RS) defines search for DICOM studies, series and instances, by specified search parameters. It is also a part of DICOM standard since 2013 [5].

Demand for both retrieving as well as pushing of new DICOM objects to DICOM repositories was stated as important on DICOM working groups. Store Over the Web RESTful Service (STOW-RS) enables storing of specific instances to the DICOM server and was standardized in 2013 [5].

UPS-RS Worklist Service defines a RESTful interface to the UPS SOP classes, which allows to query, retrieve, and update work items. It also describes how to open event channel through WebSocket connection for receiving event report messages.

DICOM RS Capabilities Service defines discovering of the supported services of a particular DICOMweb endpoint. Make the HTTP OPTIONS query against an endpoint, and a WADL response will be returned explaining the various options and what it supports [9].

All these web services are known as DICOMweb, the web standard for medical imaging [9].

DICOMweb Benefits

Web services have been already proved in real-world applications. DICOMweb inherited all benefits and limitations from web services, REST architecture, and HTTP protocol. They have numerous benefits, such as basic and digest authentication, authorization header, custom headers, different status codes, reduced number of DICOM negotiations over large DICOM dataset (they potentially pose a problem for many existing implementations [10]), ability to consume DICOMweb APIs in non-browser clients, easy integration via standard HTTP verbs (GET, PUT, POST, DELETE), simple API creation, caching control, TLS encryption, compression, huge developer ecosystem, and shorter learning curve than DICOM DIMSE services.

DICOM Network Benefits

DICOM network protocol, on the other side, is traditionally used in numerous institutions, radiology modalities, and non-radiology devices such as ECG and endoscopy. It also acts as interface to different workstations: advanced image processing or CAD systems. DICOM protocol also addresses encryption, authentication, and digital signatures [11]. A crucial feature is bidirectional data movement across DICOM network via DIMSE C-STORE, C-MOVE, and C-GET operation which is not only used in local networks but also in regional and national networks.

DICOMweb has several constraints compared to “classical” DICOM. It is not possible to implement DICOM SCPs, for example, C-STORE service. DICOM data cannot be pushed from PACS to WADO clients—send new images to client, such as emergency angiography data, or postprocessing. C-MOVE is not possible via current DICOMweb solutions. It is not possible to perform C-ECHO. Direct access to DICOM modalities such as CT, MR, or XA is not possible.

Those constraints limit the full potential of DICOM services and can limit radiology workflow in web browsers. Also, upgrading the current DICOM system to DICOMweb can take some time, especially for modalities, which can have up to 20 years’ lifecycle.

In this paper, we propose an alternative approach to access the DICOM network directly via web browsers. Such system can also be an upgrade to the existing DICOMweb services. Together with the benefits of using DICOM web services, we

propose adding another layer of DICOM communication in web browsers which will address benefits of classical DICOM network. Such system will act as a DICOM 3.0 compliant device inside web browsers, which will be able to send and retrieve data from any DICOM device, including modalities. It will be able to provide both SCP and SCU for the DIMSE network services in web browsers. Therefore, we propose the following:

1. Implementation of DICOM 3.0 standard in JavaScript language
2. Communication with DICOM network via WebSocket protocol
3. Implementation of DICOM SCP and SCU in web browsers
4. Full compatibility with existing DICOM infrastructure, without need for upgrade

Materials and Methods

JavaScript DICOM Network Library

The DICOM network library is a backbone of any DICOM communication system and is responsible for handling complexities of DIMSE services. Such library should work well with binary data and TCP protocol. To develop such library for web browsers, one must use JavaScript language, the only programming language available in web browsers.

The mDCM C# DICOM library [12] was used as a role model for class structures and program flow. C#-based DICOM library is chosen since syntax is much closer to JavaScript than C/C++. The library is designed to support all DICOM PDUs (Table 2). PDU is a core element of the DICOM network library—data packets exchanged at lower level between peers. PDU contains control information and user data. PDUs are constructed by mandatory fixed fields followed by optional variable fields that contain one or more items and/or subitems [13]. PDUs can be read from incoming connection or created for sending back to the connection. *jDataView* library [14] was used to simplify creation of some PDUs (Fig. 2). For P-Data-TF PDU, *DataView* API was used.

Table 2 List of supported DICOM PDUs

| PDU |
|--------------------|
| A-Associate-RQ PDU |
| A-Associate-AC PDU |
| A-Associate-RJ PDU |
| P-Data-TF PDU |
| A-Release-RQ PDU |
| A-Release-RP PDU |
| A-Abort PDU |

```
var reject = jDataView.createBuffer(0x03, 0x00,
0x00, 0x00, 0x00, 0x04, 0x00, 0x01, 0x03, 0x01);
```

Fig. 2 Simple A-Associate-RJ PDU created with *jDataView*

The *DataView* API provides low-level interface for reading and writing multiple number types in an *ArrayBuffer* irrespective of the platform's endianness [15].

Typed array views are in the native byte-order of user platform, while *DataView* byte-order can be controlled. By default, it is big-endian, but can be set to little-endian, which we used for this project. DICOM reserves little-endian as its default byte-ordering type. It means that all DICOM applications, regardless of architecture, must understand and process little-endian byte-order [3].

PDUs were received from WebSocket connection (described in the next section) and were processed in DICOM network base object (Fig. 3).

P-Data-TF differs from other PDU types; it is the only PDU responsible for transmitting the actual data. P-Data-TF sends DICOM objects, cut into chunks known as “protocol data value” (PDV) [3]. The code detects PDVs in P-DATA-TF PDU and processes chunk by chunk—appending chunks in temporary byte buffer. The system detects the received PDVs: are they DICOM command or DICOM file? Upon receiving the last PDV fragment, the system finishes appending the buffer and saves the DICOM file or executes the DICOM command. If needed, DICOM response is sent back through WebSocket connection (Fig. 4).

DICOM to WebSocket Proxy

Since JavaScript in web browsers does not support access to TCP/IP protocol or DICOM protocol, WebSocket protocol was used to communicate with the DICOM network, via DICOM to WebSocket traversal bridge (proxy). DICOM messages and objects were transported from the DICOM network to the WebSocket network, and vice versa.

WebSocket is a web technology providing full-duplex bi-directional communication channels over a single TCP connection, meaning that web servers can push data to web clients in real time, thus making possible implementation of DICOM SCP services. The WebSocket protocol was standardized by the Internet Engineering Task Force (IETF) as request for comments (RFCs) 6455 in 2011 [16], and the WebSocket API is being standardized by the W3C [17]. WebSocket differs from TCP in that it enables a stream of messages instead of a stream of bytes, real-time push messages, smaller message header than HTTP, and generally reduced latency over

```
wsclient.onmessage = function(evt){
//process PDU
dcm.ProcessNextPDU(evt, data);
};
```

Fig. 3 PDU received from WebSocket connection

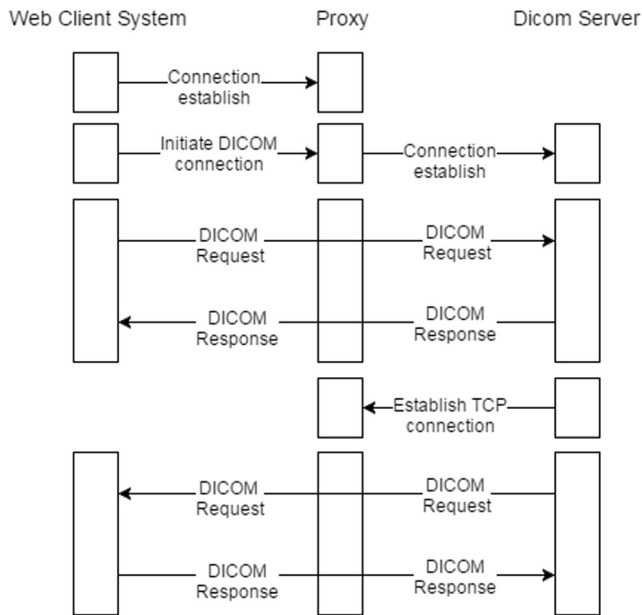


Fig. 4 WebSocket requests

HTTP. That said, WebSocket is a protocol built for high throughput of messages from both ends.

Proxy was implemented in JavaScript language, by using the Node.js platform [18] (Fig. 5). Node.js is a platform built on Google Chrome’s JavaScript runtime for easy building of fast, scalable network applications on all the operating systems [19]. Node.js uses an event-driven, non-blocking input/output (I/O) model that makes it lightweight and efficient, suitable for data-intensive real-time applications that run across distributed devices. Node.js WebSocket library ws [20] was used for WebSocket communication. Node.js net module, which provides asynchronous network wrapper for creating both TCP servers and clients, was used.

The proxy server listens on two ports for incoming DICOM or WebSocket connections. In the case of DICOM connection, binary DICOM messages are transferred from the TCP port to the WebSocket connection (Fig. 6). PDUs are received as chunks, not as a single message. They need to be concatenated. By using PDU length information, the proxy knows what PDU size to expect from incoming TCP connection. When all chunks are received, PDU is sent to the web client over WebSocket connection.

Proxy and JavaScript DICOM network libraries are available to download as open-source project [21, 22]. The tools

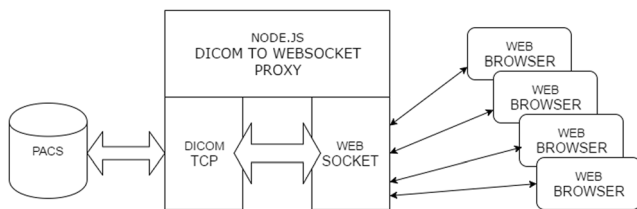


Fig. 5 Node.js proxy

```

// Load the net module to create a tcp server.
var net = require('net');
// Create tcp server
var tcpserver = net.createServer({ allowHalfOpen: false },
function (listener){

    listener.addListener("data", function (data)
    {
        //Process and send PDU to WebSocket connection
        ProcessPDU(data);
    });
});

// Start tcp server bound to port 104 on localhost
tcpserver.listen(104, "0.0.0.0");
    
```

Fig. 6 TCP server receives DICOM PDU chunks

that were used are Microsoft Visual Studio 2012, Sublime Text 2 and CodeAnywhere code editor. WebSocket connection in POC system was not secured via WebSocket over SSL/TLS (WSS).

Testing Methods

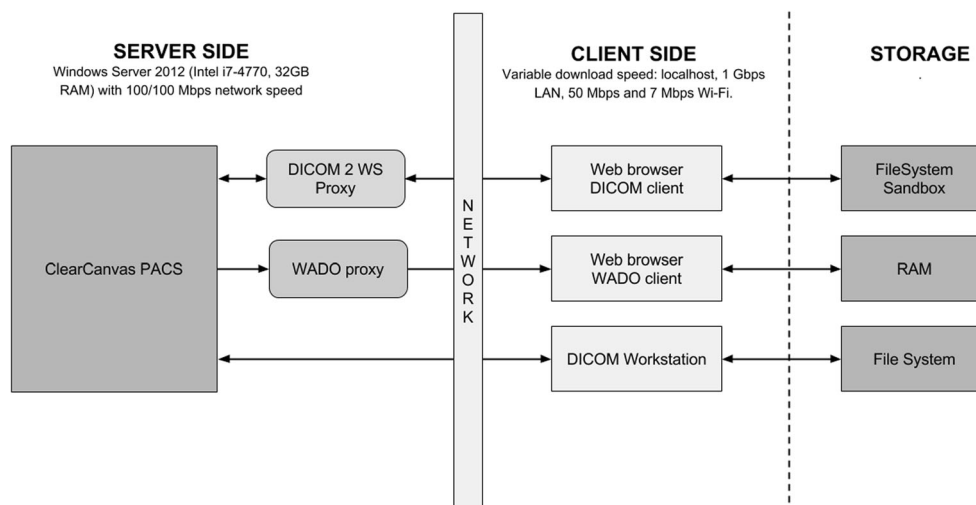
The system was tested against different DICOM devices: ClearCanvas Server and Workstation [23], Osirix Viewer [24], E-Film [25], Fellow Oak DICOM toolkit [26], Dcm4Chee [27], DCMTK toolkit [28], and Dicom Objects toolkit [29].

Tests should answer the question, “are JavaScript library and WebSocket protocol able to deal with DICOM network and DICOM traffic particularities: large DICOM files and multiple smaller DICOM files, e.g. CR and CT images?”

Before answering this question, it is necessary to say that the system is built as proof of concept, meaning that optimizations have not taken place, security protocols have not been implemented, concurrency problems have not been addressed, and special JavaScript features were not used to speed up execution time.

Download time from PACS was compared against the WADO client and the DICOM client (Fig. 7). PACS was installed together with DICOM to the WebSocket proxy and the WADO server on Windows Server 2012 (Intel i7-4770, 32 GB RAM) with 100/100 Mbps of Internet bandwidth. Clients (Windows 7, Intel Xeon E7525, 8 GB RAM) were tested with different download speeds: localhost, 1 Gbps Intranet, 50-Mbps cable Internet and 7-Mbps Wi-Fi Internet. ClearCanvas Image Server 2.0 was used as a PACS, while ClearCanvas Workstation 2.0 was used as DICOM client. We developed a simple WADO server with fo-dicom library [25]. Images were pre-cached on the WADO server by sending them from ClearCanvas PACS. Every WADO request would read a DICOM file from the file system and stream it to the WADO client. The WADO client was a simple JavaScript application using XHR requests to fetch images. The client could be tuned to execute one to four simultaneous XHR requests. Test DICOM cases were sent from PACS to clients. Seconds were used as measure parameter.

Fig. 7 Testing configuration for DICOM file transfer



Results

The JavaScript DICOM network library was used to develop proof-of-concept (POC) application [21]. The POC system should act as a DICOM device that works in any web browser supporting WebSockets (Fig. 8). POC should be able to initiate and receive DICOM connections from and to any DICOM 3.0 compatible device by using the JavaScript DICOM network library and WebSocket connection to proxy server. Several DICOM classes were developed for demonstration: C-ECHO SCU/SCP, C-STORE SCP, C-FIND SCU, and C-MOVE SCU (Table 3). The system accepts all storage SOP classes and all transfer syntaxes, except big-endian. Decoding of compressed transfer syntaxes can be supported via third-party JavaScript libraries [30].

DICOM to WebSocket proxy can listen on TCP port for DICOM connections and translating them to WebSocket connection—server mode. It is also capable of connecting to the TCP port of DICOM devices to send DICOM messages from WebSocket connection—

client mode. This device is lightweight, fast, and simple although not optimized for production use (security, concurrency, stability). It is fully compatible with the existing DICOM systems without the need of upgrading them. Therefore, it can be used with legacy PACS systems and modern vendor neutral archive (VNA) systems.

C-FIND SCU: Query Patient, Study, and Series

The POC system is capable of issuing C-FIND query on DICOM devices that support C-FIND SCP. Patient and series query level is not implemented in the POC system but is straightforward. Query parameters are currently not supported, so the system will return all studies from the DICOM device, if possible. C-FIND was tested against ClearCanvas PACS and Workstation, Osirix workstation, Medical Connection public DICOM server (Fig. 9).

Fig. 8 Proof-of-concept system screenshot

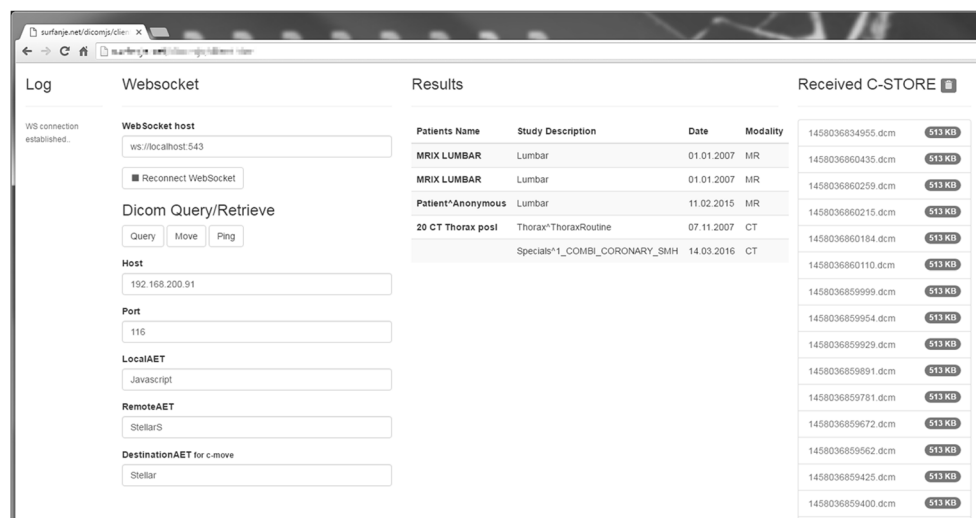


Table 3 Main features of JavaScript DICOM library and proxy

List of features

- Fully DICOM-compliant JavaScript DICOM library
- DICOM protocol to WebSocket protocol function
- DICOM “ping” SCU and SCP service
- Near “native” C-STORE SCP performance
- C-MOVE SCU—Moving of studies between DICOM applications
- C-FIND SCU—Search for patients, studies, series and instances
- Support for all web storage options

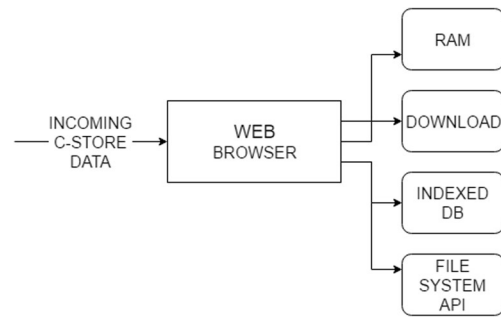


Fig. 10 Options to store DICOM data

C-STORE SCP: Receive “Pushed” DICOM Data

The POC system can receive DICOM “push” messages: C-STORE. POC is flexible in storing binary DICOM data when C-STORE response is received (Fig. 10). Data can be downloaded to the computer’s hard disk, kept temporarily in the computer random access memory (RAM) or stored in a web browser offline storage: Indexed Database (IndexedDB) or FileSystem API. IndexedDB is supported by all major web browsers [31] and supports asynchronously storing and retrieving blobs binary large objects of data in key-value fashion. Key-value database for DICOM use have already been investigated with the CouchDB database [32].

For this research, the POC system uses Google Chrome FileSystem API, supported by 44.8% of web browsers [33]. Although not standardized by W3C, the FileSystem API provides a simple way to create, read, navigate, manipulate, and write blobs of data to a sandboxed section of the user’s local file system asynchronously [34]. Therefore, incoming DICOM files from C-STORE connections were written asynchronously to sandboxed file system, by using Web Workers.

C-MOVE SCU: the system was used to C-MOVE images from ClearCanvas PACS to other DICOM workstations—Osirix and ClearCanvas Workstation.

C-ECHO SCU/SCP: the system was used to test DICOM “Ping” with various DICOM devices successfully.

Fig. 9 C-Find results from Medical Connections public DICOM server

| Log | Websocket | Results |
|-----------------------------|------------------------------------|--|
| WS connection established.. | WebSocket host: ws://localhost:88 | KOWALIK Merge Study 02.04.2012 CR |
| WS connection is closed.. | Reconnect WebSocket | KOWALIK 14.10.2014 MR |
| WS connection established.. | Dicom Query/Retrieve | KOWALIK 14.10.2014 MR |
| WS connection is closed.. | Host: 213.165.94.158 | KOWALIK 18.06.2015 CR |
| | Port: 104 | KOWALIK 24.06.2015 CR |
| | LocalAET: Javascript | KOWALIK 02.07.2015 CR |
| | RemoteAET: StellarS | KOWALIK US First trimester 23.10.2015 US |
| | DestinationAET for c-move: Stellar | KOWALIK US App 29.10.2015 US |
| | | KOWALIK 04.02.2016 US |
| | | ? IMAGEnet 19.08.2013 XC |
| | | 27.05.2010 RF |
| | | 00002 Patient Resting ECG 17.10.2013 ECG |
| | | 00002 Patient Resting ECG 17.10.2013 ECG |
| | | 00002 Patient Resting ECG 17.10.2013 ECG |
| | | 00002 Patient Resting ECG 17.10.2013 ECG |

Table 4 C-Storing 271 CT images 512×512 , uncompressed (514 KB per image), in seconds

| Client download speed | Client | | |
|-----------------------|------------|----------------|-------|
| | DICOM | DICOM JS | WADO |
| Localhost | 12.9 | 12.4 | 6.2 |
| 1 Gbps | 22.3 | 10.5 | 7.1 |
| 50 Mbps | 86.8 | 43.9 | 38.7 |
| 7 Mbps | – | 172.9 | 165.6 |
| Storage | Filesystem | FileSystem API | RAM |

DICOM File Transfer

Although we tested the system with many different DICOM files, we recorded only few image transfer sessions, to prove that the system is feasible in various network settings.

CT study (Table 4), with 271 uncompressed images (512×512 pixels, 514 KB size per image) and DX study (Table 5) with 6 uncompressed images (2787×2472 pixels, ~14 MB per image) was used for testing.

High load test was also performed (Table 6) by sending 2391 uncompressed CT images (512×512 pixels, 514 KB size per image) to clients in LAN network only. For that test, the WADO client was using four simultaneous XHR requests to download four images per request, which improved results and showed benefits of HTTP protocol.

Discussion

The system qualifies as a DICOM 3.0 capable device in web browsers by successfully performing given tasks. Development focus was on C-STORE SCP and C-FIND SCU method: performance and compatibility with different DICOM implementations and devices. System has answered several important topics from the beginning: implementation of DICOM DIMSE network services in JavaScript language is

Table 5 C-Storing 6 DX images 2787×2472 , uncompressed (14 MB per image), in seconds

| Client download speed | Client | | |
|-----------------------|------------|----------------|-------|
| | DICOM | DICOM JS | WADO |
| Localhost | 3.0 | 9.1 | 6.9 |
| 1 Gbps | 10.5 | 18.5 | 5.5 |
| 50 Mbps | 23.4 | 15.5 | 16.2 |
| 7 Mbps | – | 78.1 | 101.1 |
| Storage | Filesystem | FileSystem API | RAM |

Table 6 C-Storing 2391 CT images 512×512 , uncompressed (514 KB per image), in seconds

| Client download speed | Client | | |
|-----------------------|------------|----------------|------|
| | DICOM | DICOM JS | WADO |
| LAN | 231 | 210 | 110 |
| Storage | Filesystem | FileSystem API | RAM |

possible and communication with the DICOM network via the WebSocket protocol is feasible. The POC system is compatible with various open-source DICOM implementations and some popular proprietary implementations.

Performance

We are aware that the C-STORE performance test is somewhat limited. We tried to give an overview of the system performance in various network environments, trying to avoid and overcome “WebSocket vs REST” debate. System performance tests are conducted to prove that the system can deal with DICOM traffic via WebSocket connection. The results prove that such implementation is feasible.

Tests also show positives of WADO-based systems for large studies of transfer, although results are much helped by pre-caching, multiple XHR requests and storing in RAM only.

Besides, the system’s C-STORE SCP performance in limited testing environment, in terms of speed, is comparable to native DICOM protocol. Although we recorded the test only against ClearCanvas DICOM implementation, which is based on customized mDcm library, the system worked in the same manner with other DICOM implementations.

Securing WebSockets

WebSocket protocol, just like HTTP, is not safe by default. It needs effort to keep WebSocket applications secure. WebSocket traffic should be encrypted by using WSS. It protects against man-in-the-middle attacks. WebSockets reuse the same authentication information that is found in the HTTP request when the WebSocket connection was made [35]. The system should validate DICOM data, e.g., only allow DICOM and authentication data to pass through. Origin header should be used as an advisory mechanism; it helps differentiate WebSocket requests from different locations and hosts, but developers should not rely on it as a source of authentication [36]. DOS attack should also be handled by DICOM to WebSocket proxy. The WebSocket protocol does not handle authorization and/or authentication. Application-level protocols should handle that separately in case sensitive data is being transferred. Tunneling DICOM connections

through WebSocket proxy can put DICOM network on risk as it would enable access to services in the case of a Cross Site Scripting attack; therefore, data coming through a WebSocket connection should be always validated [36].

Proxy Considerations

Node.js proxy is a feasible technology for proxy system, because it is proved in real-world applications and used by many world class systems, like PayPal, Uber, Netflix, and others. Node.js developers also enjoy a huge module ecosystem, known as npm. Benefit of Node.js JavaScript environment is that it can reuse the same JavaScript DICOM network library for server-based operations.

Several problems were encountered during the development phase: DICOM TCP packet detection in Node.js and SCU to SCP switching methods with Node.js proxy. Sometimes, larger PDU sizes can reset TCP connection to DICOM clients. When sending smaller chunks of data, like compressed CT images, problems do not occur. During the initial testing phase, with older Node.js versions (0.12) and older WS library version (0.6.5), problems did not occur. Problems will further be investigated and resolved. For the production system, security upgrade is needed, which implements all WebSocket security features. Concurrency should also be addressed.

Conclusions

It is technologically possible to develop DICOM device exclusively with JavaScript language. The only real problem was the TCP network access and that was overcome with WebSocket protocol. The WebSocket protocol enables development of DICOM real-time applications. This paper shows that access to DICOM devices with JavaScript language from a web browser is fast, reliable, and backward DICOM compatible. To our knowledge, a new type of DICOM web application is presented.

DICOM to WebSocket proxy proves to be a simple DICOM upgrade option that enables all DICOM systems to communicate via the WebSocket protocol with JavaScript DICOM network library.

The JavaScript DICOM network library can be further optimized in terms of speed, compatibility, embeddability, simplicity, and better object-oriented library core. Unit test should be added. By using new JavaScript ES6 features or specific JavaScript subsets like asm.js [37], processing speed comparable to native DICOM C++/C# applications could be achieved. We will also investigate options to compile C++ and C# DICOM libraries to JavaScript via specific tools like Emscripten [38] and JSIL [39].

For further research, the system will be optimized as mentioned. C-FIND SCU will be upgraded with query parameters and all search levels. C-STORE SCP storage will be upgraded to IndexedDB API, which means that the system will index DICOM data. Therefore, C-STORE SCU and C-FIND SCP function could be implemented to send data back to the DICOM applications. We will also consider Web real time communication (RTC) API to enable web-based peer to peer DICOM content sharing.

In terms of connectivity, content sharing, device access, software accessibility, software updates, and user reach, the JavaScript DICOM network library with DICOM TCP to WebSocket proxy presents an optimal web access upgrade to the existing DICOM and DICOMweb applications.

References

1. Internet live stats, www.internetlivestats.com/internet-users/#sources—Last accessed Jan 10, 2017
2. Stack Overflow Developers Survey 2016, <http://stackoverflow.com/research/developer-survey-2016#developer-profile-experience> – Last accessed Jan 10, 2017
3. Pianykh, OS, DICOM practical introduction and survival guide, Springer, 2008
4. DICOM Part 18, Supplement 85, Web Access to DICOM Persistent Objects (WADO), ftp://medical.nema.org/medical/dicom/2011/11_18pu.pdf - Last accessed Jan. 6, 2015
5. Lipton P, Nagy P, Sevinc G: Leveraging internet technologies with DICOM WADO. *J Digit Imaging* 25:646–652, 2012
6. Medical Imaging Network Transport, <https://code.google.com/p/medical-imaging-network-transport/>—Last accessed Dec. 11, 2014
7. Clunie D: Framing big study problem, <http://dclunie.blogspot.com/2011/06/framing-big-study-problem.html>—Last accessed Jan. 15, 2015
8. DICOM PS3.18 2015a—Web Services, <http://dicom.nema.org/medical/dicom/current/output/pdf/part18.pdf> - Last accessed Feb 22, 2015
9. DICOMweb, <https://dicomweb.hcintegrations.ca/services/>—Last accessed Jan 10, 2017
10. Clunie D: How many (medical image exchange) standards can dance on the head of a pin?, <http://dclunie.blogspot.hr/2016/03/how-many-medical-image-exchange.html>—Last accessed Jan 10, 2017
11. DICOM PS3.15 2013—Security and system management profiles, <http://dicom.nema.org/dicom/2013/output/chtml/part15/PS3.15.html>, Last accessed Jan 10, 2017
12. Dillion C: mDCM DICOM library, <https://github.com/fo-dicom/mdcm>—Last accessed Jan 10, 2017
13. DICOM upper layer protocol for TCP/IP data units structure, http://dicom.nema.org/dicom/2013/output/chtml/part08/sect_9.3.html—Last accessed Jan 10, 2017
14. jDataView, <https://github.com/jDataView/jDataView>—Last accessed Jan 10, 2017
15. DataView, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/DataView—Last accessed Jan 10, 2017

16. The WebSocket protocol specification, Internet Engineering Task Force <http://tools.ietf.org/html/rfc6455>—Last accessed Jan. 15, 2015
17. The Websoket API, World Wide Web Consortium <http://www.w3.org/TR/2011/WD-websockets-20110929/>—Last accessed Jan. 15, 2015
18. Nodejs, <https://nodejs.org/>—Last accessed Apr. 12, 2015
19. Chaniotis IK, Kyriakou KID, Tselikas ND: is Node.js a viable option for building modern web applications? A performance evaluation study. *Computing* 1–22, 2014
20. ws: a node.js websocket library, <https://github.com/websockets/ws>—Last accessed Feb 12, 2015
21. JavascriptAccessToDicomObjects library, https://bitbucket.org/willy_skipper/javacriptaccesstodicomobjects—Last accessed March 18, 2016
22. Dicom to WebSocket library, https://bitbucket.org/willy_skipper/dicom2websocket—Last accessed March 18, 2016
23. Clear Canvas, <https://github.com/ClearCanvas/ClearCanvas>—Last accessed Feb 12, 2015
24. Osirix Viewer, <http://www.osirix-viewer.com/>—Last accessed March 18, 2016
25. E-Film, <https://estore.merge.com/na/index.aspx>—Last accessed March 18, 2016
26. Fellow Oak DICOM for .NET, <https://github.com/fo-dicom/fo-dicom>—Last accessed March 18, 2016
27. Dcm3che, <http://www.dcm4che.org/>—Last accessed March 18, 2016
28. DCMTK toolkit, <http://dicom.offis.de/dcmthk.php.en>—Last accessed March 18, 2016
29. DicomObjects DICOM toolkit, <https://www.medicalconnections.co.uk/DicomObjects>—Last accessed March 18, 2016
30. Daikon, <https://github.com/rii-mango/Daikon> - Last accessed Jan 10, 2017
31. IndexedDB API usage, <http://caniuse.com/#feat=indexeddb> - Last accessed March 18, 2016
32. 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
33. Filesystem & FileWriter API usage, <http://caniuse.com/#search=filesystem>—Last accessed Feb 12, 2015
34. Google Chrome Storage API, https://developer.chrome.com/apps/app_storage—Last accessed Feb 12, 2015
35. WebSocket authentication, <http://docs.spring.io/spring-security/site/docs/current/reference/html/websocket.html#websocket-authentication>—Last accessed Jan 10, 2017
36. WebSocket security, <https://devcenter.heroku.com/articles/websocket-security>—Last accessed Jan 10, 2017
37. Asm.js, low-level subset of JavaScript, <http://asmjs.org/spec/latest/>—Last accessed Feb 12, 2015
38. Emscripten, <https://github.com/kripken/emscripten>—Last accessed Jan 10, 2017
39. JSIL, <http://jsil.org>—Last accessed Jan 10, 2017