Inamdar, Kibriya; Kottayi Pilapprathodi, Oormila Ramanandan; John, Jopaul; u. a.

A Web Architecture for E-Health Applications Supporting the Efficient Multipath Transport of Medical Images

Date of secondary publication: 27.04.2026

Accepted Manuscript (Postprint), Conferenceobject

Persistent identifier: urn:nbn:de:bvb:473-irb-114833x

Primary publication

Inamdar, K.; Kottayi Pilapprathodi, O. R.; John, J.; u. a. (2022): A Web Architecture for E-Health Applications Supporting the Efficient Multipath Transport of Medical Images, in: F. Phillipson, G. Eichler, C. Erfurth, u. a. (Ed.), Innovations for Community Services : 22nd International Conference, I4CS 2022, Delft, The Netherlands, June 13–15, 2022, Proceedings, Cham: Springer International Publishing, pp. 136–152, doi: 10.1007/978-3-031-06668-9_11.

A Web Architecture for E-Health Applications Supporting the Efficient Multipath Transport of Medical Images

Kibriya Inamdar, Oormila Ramanandan Kottayi Pilapprathodi, Jopaul John, Markus Wolff, Marcel Großmann, and Udo R. Krieger(✉)

Fakultät WIAI, Otto-Friedrich-Universität,
An der Weberei 5, 96047 Bamberg, Germany
udo.krieger@ieee.org

Abstract. New advanced e-health applications are required to support the effective processing of diagnostic and therapeutic healthcare protocols in modern societies. Looking at the work flow handled by the consulted medical staff in the first level of a medical treatment chain such as family doctors, an effective treatment usually requires the processing of pre-recorded medical images of a patient during the first diagnostic phase. We consider the development of a Web server architecture that offers the transport of medical images by an Android application and illustrate its design by a realized PACS prototype. The effective data transport of medical images is realized by a multipath-QUIC protocol which is integrated into a DICOM proxy server. Its further development can integrate other fog computing systems which support additional interconnected e-health applications employed by a consortium of users in a medical treatment process.

Keywords: e-Health · DICOM · Android applications · Multipath-QUIC

1 Introduction

Nowadays, new advanced e-health applications are required to support the effective processing of diagnostic and therapeutic healthcare protocols in a modern society as the pandemic has recently shown. Looking at the work flow handled by the medical staff in the first level of a medical treatment chain such as family doctors, an effective treatment usually requires the processing of pre-recorded CT and/or MRT images of a sick person during the first diagnostic phase. These applications provide an important domain within the rapidly evolving Internet-of-things and the related cloud and fog computing concepts, cf. [1,14].

Considering a single patient, this set of digital medical images such as X-rays, CTs, MRTs, PETs and others is generated during a first diagnostic phase within a radiological analysis department by sophisticated imaging devices [24]. The latter produce these images in a standardized format based on the Digital

Imaging and Communications in Medicine (DICOM) standard [21]. This NEMA standard describes numerous details about the structure, storage, and transport of such medical images and their corresponding data files. The number of these medical images is increasing rapidly, promoting the need to upgrade both storage and retrieval systems [4,16,25]. Normally, a recorded set of DICOM compatible images contains additionally a lot of medical metadata and information on the collection of these images, i.e., a patient's image set is normally quite large and commonly its size is more than 100 MB as it may contain more than 100 images.

The DICOM format [21] ensures that medical images comply with high quality standards which allow a precise diagnosis and interpretation. Using teleradiology, in principle these images can be sent by different methods from one location to another one. As these images are large in size and number regarding a single patient, it is difficult to transmit such a large amount of medical data along existing communication networks with relatively low bandwidth, in particular along the last link, if it is a wireless one. Considering the data transfer between medical units at different sites in a standard treatment process, for this reason the images are often transferred by means of storage devices such as DVDs. Even in today's modern Internet environment incorporating wired and wireless network infrastructures this handling is still a day-by-day standard.

Experiments of loading such a DICOM image set via a cellular network into a Web application that is using a browser with Android OS [12] show that the actual loading time is rather high. The resulting latency to depict such a DICOM image by a Web browser rendering the transferred medical data generates an intolerable delay in the view of medical professionals. The latter personnel normally needs these images instantly when they are examining and giving advice to a treated patient. Hence, the efficient transfer of requested DICOM compatible images from a remote storage server to a potentially mobile Web client at the distant treatment site constitutes a serious technical problem as the medical staff can only provide a very limited time slice for each patient. Hence, the current way of transferring DICOM compatible images by client-server oriented, Web based e-health applications using a standard HTTP protocol stack is impractical and requires effective improvements of the related protocols.

In our study we have developed a Web based image retrieval architecture for a requested medical image set. It uses an Android based client to retrieve and render a remotely requested series of DICOM compatible images. Our architecture incorporates a standard Orthanc server [16] storing these medical images. Considering the image transport along the Internet, we apply the new multipath-QUIC transport protocol [5]. The latter is enhancing the well established Quick UDP Internet Connection (QUIC) protocol [15]. It has been designed to support transport and session services for HTTP similar to TCP with TLS on top of UDP. Multipath-QUIC [5] has the ability to exploit different paths which exist between a sender and a receiver and provides bandwidth aggregation and a seamless network failover. We use an existing implementation of multipath-QUIC by the programming language Go [11]. The client side of the e-health application has been developed using Android [12]. The latter is a popular open

source software system based on Linux and primarily designed for touch screen mobile devices such as smartphones and tablets. The DICOM server side of our Web application has been developed using Python [22]. Using Go, we have built and integrated the required multipath-QUIC client and multipath-QUIC server with the developed client-server based Picture Archiving and Communication System (PACS) architecture to transfer the DICOM compatible images.

The paper is organized as follows. In Sect. 2 we present the foundations of digital imaging and communication techniques for e-health applications based on the current medical standard DICOM. In Sect. 3 we describe an enhanced PACS architecture for the medical image transfers using a proxy server that is interconnected with a DICOM cloud infrastructure. In Sect. 4 we discuss our versatile Web-server architecture for the efficient multipath transport of medical images to an e-health app. Finally, some conclusions are presented.

2 Foundations of Digital Imaging and Communication Techniques for e-Health Applications

Nowadays, digital images are generated in medical applications during a diagnostic investigation by sophisticated technical devices. The latter imaging systems generate the required medical images of high resolution such as X-rays, CTs, MRTs, and PETs, in a standardized image format, cf. [24]. It is normally based on NEMA's Digital Imaging and Communications in Medicine (DICOM) standard [21]. This NEMA standard which partly overlaps with other areas of medical informatics has been developed with an emphasis on diagnostic medical imaging as practiced in radiology, cardiology, pathology, dentistry, and ophthalmology. It is used in image based therapies such as interventional radiology, radiotherapy, and surgery, cf. [16]. It primarily facilitates the interoperability of Picture Archiving and Communication Systems (PACS) claiming a DICOM conformance in a multi-vendor environment. However, DICOM by itself does not guarantee this interoperability in a straightforward manner in all use cases [21]. However, the DICOM technology and its related tools are also applicable to a much wider range of image and non-image related information exchange processes needed in today's clinical infrastructures, veterinary, and medical research.

2.1 The Scope of the Medical Imaging Standard DICOM

Regarding the handling of medical images, DICOM [21] is currently the major global standard. It covers the processing, communication, and management of medical images and their related metadata. In particular, DICOM facilitates the interoperability of medical image processing equipment by specifying

- a set of processing, viewing, and networking protocols followed by those image generating devices claiming a DICOM conformance,
- the syntax and semantics of commands and associated information that can be exchanged by these DICOM protocols,

- a set of media storage services obeyed by those DICOM compatible systems, as well as a file format and a medical directory structure to facilitate the access to these images and related meta information stored at DICOM servers,
- and additional information which must be supplied with an implementation if DICOM conformance is claimed.

However, the DICOM standard does not specify

- the implementation details of any standard feature in a DICOM compatible system,
- the overall set of features and functions expected from a Picture Archiving and Communication System, which is implemented by an integration of a group of DICOM compatible devices,
- a testing and validation procedure to assess an implementation's conformance to the DICOM standard.

These shortcomings are handled by additional software systems and processes which have been developed by equipment providers and software developers in the last decades. In our current research perspective we shall focus on a lightweight, open source software development regarding this PACS/DICOM environment, cf. [16,17].

2.2 Objectives and Use Cases of a Medical Image Transfer

Medical professionals normally want to request and display the recorded medical images of a patient during a consultation phase in a timely manner without any significant delays or waiting times regarding the image rendering by an app. In this way, they are able to give their patients a fast, accurate diagnosis. Normal use cases include scenarios where a medical professional wants

- to retrieve relevant DICOM compatible images via standard end devices, e.g. a laptop, an Android tablet or even Android phone,
- to be able to search for DICOM images using a unique image identifier of a patient, the *PatientID*, on a server storing these images,
- to zoom into a specific DICOM image such that one can see specific details or get a general overview on the image,
- to use a medical Web application supported either by a wired in-house communication network or WiFi network, or by both types.

The DICOM protocol inherently targets primarily the intranet of a single hospital, not the Internet or the cloud. Thus, its protocols may be blocked by an outbound firewall. Furthermore, although the DICOM data exchange protocol supports TLS encryption, this feature is rarely enabled. Depending on the type of an e-health application, one can, of course, leverage the HTTP protocol in this context. Such Web-based protocols are easier to work with, can be transparently encrypted by HTTPS, and are compatible with a multiple-hospital scenario.

Regarding the processed DICOM objects of a medical study stored at a DICOM server like Orthanc [16,17], the diagram depicted in Fig. 1 illustrates the hierarchy of a patient's data set. Each study includes a set of a series of medical images. Each series is in turn a set of image instances, where the latter is a synonym for a single DICOM compatible file. The DICOM standard arranges the associated identifiers *StudyInstanceUID*, *SeriesInstanceUID* and *SOPInstanceUID* to be globally unique. Thus, it is mandatory for two different image generating devices not to generate the same identifiers, even if they are manufactured by different vendors. An Orthanc server exploits this rule to derive its own unique identifiers.

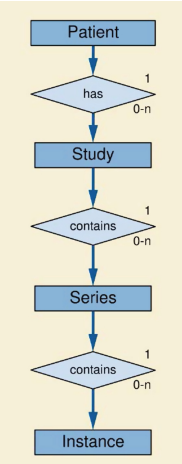

Fig. 1. Data hierarchy of a DICOM object (cf. [21])

However, even if the *PatientID* must be a unique index within a given hospital, it is not guaranteed to be globally unique. This means that different patients' image recordings at different hospitals might share the same *PatientID*.

A recorded collection of DICOM compatible images contains a lot of medical information about a specific patient's physical state and additional metadata. Such a patient's image set is normally large and commonly sized more than 100 MB as it may contain more than 100 images. Currently, loading such an image set along a communication network between a PACS server storing the required data and a client retrieving the latter reveals that the actual downloading period may be quite high. Medical professionals need these images instantly when they are examining a patient and giving advice. Thus, an experienced high latency may constitute a severe QoE problem as there is only very limited time for each patient during a medical consultation. Therefore, it can be concluded that the current way of transferring DICOM images by means of a standard client-server architecture employing a HTTP-TCP protocol stack is not a very practical approach and may not be effective by itself.

3 An Enhanced PACS Architecture for the Transfer of Medical Images

A classical Picture Archiving and Communication System (PACS) for medical images and their related e-health applications normally incorporates a client-server model. The PACS services include an easy access to a patient's reports and images, enhanced analysis viewing, and a chronological data management, among other functions. Such a PACS environment should be both user friendly and developer friendly. Examples of a PACS server include DCM4CHE and Orthanc, among others, cf. [16,17]. Examples of PACS clients are provided by 3D Slicer, Horos, DWV, and others, cf. [9].

In our approach we first follow the distributed image processing concept of Jodogne [16,17] and enhance a standard PACS by a proxy server and a client-centric improvement of its medical imaging application to manage the required image transfer to a mobile device more efficiently, see Fig. 2.

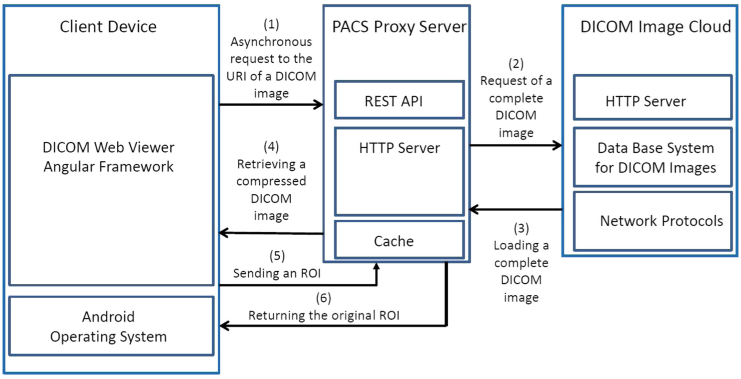

Fig. 2. An enhanced PACS architecture for the medical image transfer

The mobile client with such a new designed Web application embedded into Android can request the image data from the used Orthanc proxy server (1) using its REST API as interface. We can store these medical images that are retrieved from an attached cloud server or an interconnected DICOM image server infrastructure in the cloud (2, 3) by means of the DICOM network protocols in the cache of this proxy server. The latter can also change the format, split images into smaller blocks, the tiles, and compress those parts, cf. [16,18,21]. Then the proxy can handle related retrievals from its cache and reply to compressed image requests of a client (4) which are invoked by such an initial lightweight request (1) for an object of interest (ROI). On a user's ROI enquiry to show the full sized DICOM format of an image block, the proxy can also deliver this larger original object (6) from the cache (see Figs. 4, 5).

Similar to Jodogne's approach [16], our distributed DICOM system offers in this way some specific PACS proxy services regarding stored DICOM objects, and arranges the image access of the mobile client on top of Android to the specified DICOM proxy server as well as the related data transport more effectively than a classical PACS model.

Considering Web viewers and visualization tools for DICOM images, the DICOM Web Viewer (DWV) [9] is a widely adopted standard open source PACS client that can be used as Web based DICOM image viewer. Its advantage is that it accepts files both in a DICOM file format *.dcm* and a *.jpeg* format. Moreover, it has some advanced functions, e.g., paning, that allows the user to zoom into the images, draw, which is used to plot a line from one point to another one in a DICOM image (-these lines can be labelled and classified using titles

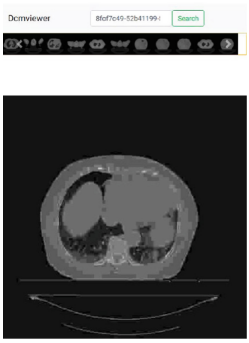

Fig. 3. Compressed version of the image in the Android app

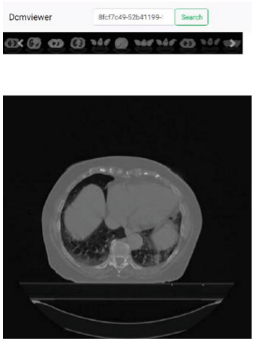

Fig. 4. Original version of the image in the Android app

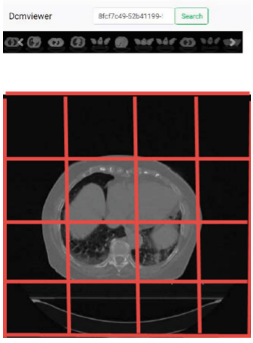

Fig. 5. Conceptional view on tiles splitting of an image in the Android app

and can be used for distance measurements in an image-), and window/level-windowing, also known as grey-level mapping, contrast stretching, histogram modification or contrast enhancement. It determines a process in which a CT image greyscale component of a DICOM image is manipulated via incorporated CT numbers. Performing this manipulation will change the appearance of the picture to highlight particular structures [24]. The window level is usually measured in Hounsfield units [20].

The widely adopted DWV-Angular [10] is a Web package based DICOM viewer that inherits these functionalities. As it is written in Angular CLI [2], it can be adopted by many popular operating systems for smart devices like Android [12] and iOS due to the availability of advanced Web browsers on top of these operating systems. We have used this Web programming and application framework for the Android app to realize a first prototype of the mobile client for a DICOM image retrieval in the described enhanced PACS model shown in Fig. 2.

Regarding this enhanced PACS architecture, the client uses this Android app as Web API to access the proxy server and employs unique predefined URIs as basic reference to the images and their tiles (see Figs. 3, 4 and 5). In Fig. 3 we show a compressed version of such a DICOM image. In our approach the images are displayed as 4×4 up to 7×7 grid in a tile structure using the Grid Layout Manager and Recycler View in Android. Whenever a particular region of an image area is selected in the compressed picture, the corresponding original image part is automatically loaded. Figure 4 shows the original version of the fully loaded DICOM image arising from that one in Fig. 3.

The image data are specified in terms of sub-blocks of a DICOM instance, called tiles (see Fig. 5), that are encoded as separate subframes. The latter are split up at the proxy server and transferred as compressed tiles to the client

after they were requested by the Web application encoded by Angular. It is embedded within the Android application to support the use cases of mobile users. There the tiles are represented as a matrix of HTML canvases. Each canvas gets a binding to a click event in our JavaScript code. Such a click, when fired, triggers a related JavaScript method that identifies the particular canvas and loads the corresponding original tile from the proxy server. Thus, tiles without useful information don't require to be loaded what can reduce the bandwidth demand. In this manner, we could save between 25% up to 39% of the transport bandwidth for the considered 4×4 and 7×7 grid structured images in some studied cases.

However, in this first client-centric PACS realization the data are sent along a TCP transport layer which can potentially cause long transfer times and high end-to-end latency. Considering the actors like doctors, these long waiting times provide a bad user experience and cause a loss of productivity. As doctors are likely to have an adequate number of patients during their daily medical round, it is essential that they can request these DICOM images associated with a patient at hand in a fast way. It may not always be possible to achieve this goal so easily using the sketched transport architecture of the PACS proxy.

4 A Versatile Web-Server Architecture for the Efficient Multipath Transport of Medical Images

Considering the DICOM proxy architecture for new e-health applications running on top of a client-server micro-service model, it is our key objective to present a server-centric extension of the sketched PACS model and its proof-of-concept realization in this section. This improvement is achieved by integrating a more efficient transport protocol stack. Therefore, the new design can support the use of mobile Web applications of medical professionals in local treatment units more efficiently. As before, the latter personnel is assumed to require the retrieval and transport of medical images stored in an associated accessible DICOM server in a cloud as quickly as possible.

The associated Web server system comprises local clients running Android apps that get access to a DICOM proxy server. The latter issues retrieval requests to a PACS image server like Orthanc storing medical data in conformance with the DICOM standard, see Fig. 6. The task of this extended DICOM proxy server, now implemented by Python [22], is to handle the DICOM requests from the client side and to process and save the requested DICOM files retrieved from a remote DICOM server or DICOM cloud. Then the latter image data are transferred as retrieval response back to the clients.

4.1 The Transport of Medical Data Along a Multipath-QUIC Route

Normally, a client-server architecture for Web applications applies a standard protocol stack employing a HTTP/1.1 or HTTP/2.0 application protocol on top of TCP as transport protocol. Recently, the transition to HTTP/3.0 has been

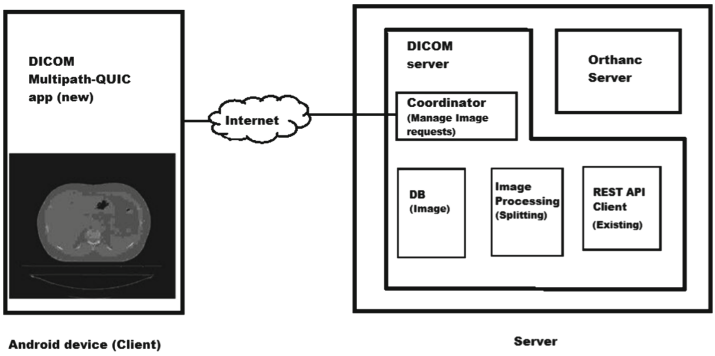

Fig. 6. Client-server architecture of the DICOM system enhanced by a multipath transport protocol

prepared which offers the use of QUIC as enhanced version of a UDP transport protocol with additional flow and congestion control functionalities [11,15], see Fig. 7.

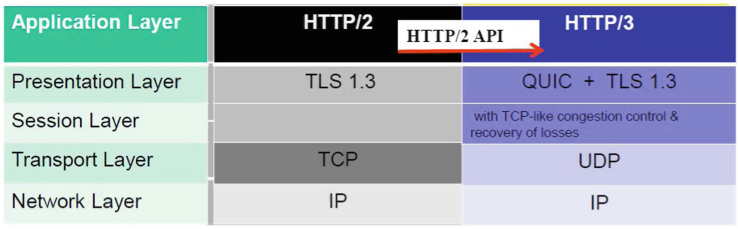

Fig. 7. HTTP/2.0 and HTTP/3.0 architectures of a communication system with TCP or QUIC based data transport

However, it is to be expected that either communication stack cannot provide the transport performance that is required for an efficient, mobile e-health application with retrieved medical image data. Therefore, one may consider existing multipath variants of the TCP or QUIC protocols at the transport layer [5]. The objective of our advanced server-centric PACS architecture is to mitigate these transport problems by improving the data transfer of DICOM images over a communication network. As QUIC is derived from the lightweight UDP protocol and more efficient than TCP, we have determined to integrate the multipath-QUIC (MPQUIC) protocol for the data transport of medical images in e-health applications. The related transport protocol stack is depicted in Fig. 8.

Multipath-QUIC [5] is basically a natural extension of the QUIC protocol at the transport and session layers. QUIC [15] achieves already performances improvements compared to TCP. Additional improvements are gained by the concept to multiplex several QUIC/UDP connections introducing an adequate

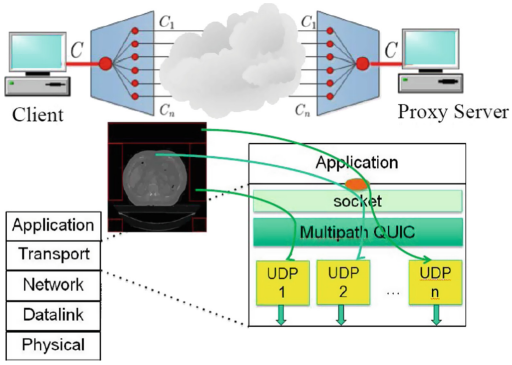

Fig. 8. Multipath-QUIC transport protocol stack (see also [5])

session functionality, cf. Fig. 8. By multipath-QUIC the medical data can now be transferred simultaneously along $n > 1$ established paths with their capacities $C_1, \cdots, C_n$. This scheme has several advantages including the aggregation of the bandwidth $C = \sum_{i=1}^{n} C_i$, failure resiliency and an easier failover in the case when one part of the underlying network fails. Given all these benefits, it is also quite likely that it can effectively support a DICOM image transfer scenario both reducing latency and improving the throughput.

Therefore, it is the objective of this second part of our proof-of-concept PACS study to produce a working prototype of a DICOM proxy server architecture with a multipath-QUIC component regarding the data transfer of sliced DICOM compatible images. Regarding the overall client-server architecture of the realized Web application, we have developed a new transport service using the programming language Python. It consists of DICOM related functionalities, like the communication with the Orthanc background server that stores the retrieved medical images, cf. [16]. Furthermore, the functionality to split up DICOM images into smaller tiles as well as certain caching functions have been provided as before, see Sect. 3. These basic functions are required for an effective transport of the large medical data sets along a pre-established multipath-QUIC route between the Python-server as DICOM proxy endpoint and an Android client retrieving these DICOM files, see Fig. 6.

Compared to the client-centric PACS model shown in Fig. 2, this new server-centric prototype consists of an extended native Android e-health application, which implements a multipath-QUIC client. It requests the data of a DICOM image from the corresponding multipath-QUIC endpoint in the proxy server. The latter is based on a server application written in Python which contains the sketched image processing functionality, like compressing the images and splitting them up into tiles, etc. It also interfaces with the associated Orthanc server which stores and manages the DICOM images in the background. An overview of this new client-server architecture with its multipath-QUIC stack is depicted in Fig. 9.

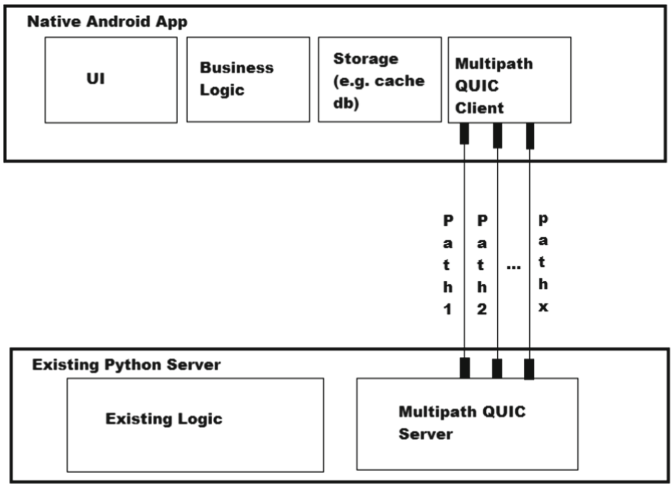

Fig. 9. Multipath-QUIC architecture of the communication subsystem

Regarding the new transport architecture, multipath-QUIC is not implemented from scratch, but an already existing implementation [5,11] in the programming language Go has been used. It can be accessed by Python ctypes [23]. First, a library from the multipath-QUIC Go code has been created in C-shared mode. When the code is executed for the first time after the start of the Python server application initiated by the client's request, then a listener of the multipath-QUIC server is spinned up, see Fig. 9. Thereafter and whenever an additional request arrives, a method to transfer the data to the multipath-QUIC client is executed.

In order to verify whether it is feasible to use this favoured approach identified during our analysis phase, i.e., basing the implementation of the data transport onto the existing multipath-QUIC Go code as proof-of-concept, the sketched approach has been assimilated with the previous enhanced PACS model of Fig. 2. To integrate the DICOM server application and the multipath-QUIC functionality at the transport layer, a coordinator component has been realized. It provides the interface between the existing functionality of the application and transport layers and processes the retrieval requests in such a way that a retrieved DICOM image can be transferred along an established multipath-QUIC route. Furthermore, the coordinator can check whether the requested image tiles of a requested DICOM image are already in the server's local transport buffer. In this case, the multipath-QUIC sender can use these elements. The associated Orthanc storage server is only contacted, if the required images are not locally available. This policy can save unnecessary round trips and, thus, further improve the transport performance of requested medical data sets. This implementation verifies the possibility to actually access the sketched network protocol stack by the Go code, which is then called via the Android application and the Python server

application. The prototype proves the feasibility of our advanced transport concept. Hence, the selected PACS/DICOM design approach can be used with the multipath-QUIC Go-implementation in the sketched e-health scenarios.

Moreover, we have figured out that the multipath-QUIC protocol can provide large benefits. As it is assumed that the Orthanc server and the DICOM proxy server are either located on the same machine or being connected by a high-speed network, it is sufficient to limit the usage of multipath-QUIC to the corresponding connection between the Android e-health application and the Python-proxy server in our new design.

4.2 Functionality of the DICOM Proxy Server

The major task of the DICOM proxy server in the PACS architecture of Fig. 6 is to handle the retrieval requests invoked at the client side, to process and to store the corresponding DICOM objects that are actually retrieved from the DICOM server realized by Orthanc and, finally, to respond to the client's request by transferring the appropriate DICOM objects.

If a DICOM object is not available in the cache of the proxy server, it is retrieved at the associated Orthanc server side or a cluster of DICOM servers. After retrieving the DICOM compliant object in a byte format from the Orthanc server, it is the proxy server's main job to convert splitted DICOM images into a *.jpeg* format and to compress them afterwards (see also [18]). Then the proxy server can handle the consecutive requests from the client side regarding the original DICOM images or their optional resizing.

The image retrieval process works as follows. Whenever the proxy server receives a client's request with a specified patient identifier *PatientID*, the server checks if the *PatientID* key is present in its local database. If it is present there, the associated data can be sent back to the client. However, if the*PatientID* key is not found in the cache, the proxy server has to load the requested object from the associated Orthanc server. The latter server then provides the required DICOM file set.

Considering the implementation of the described system, the main functionalities at the proxy server side comprise the following functions:

1. to split the images into tiles with a specified $n * n$-format, $n \in \{4, \ldots, 7\}$,
2. to save the processed image and its compressed tiles into a related database as local cache site, and
3. to prepare the transfer of the corresponding objects to the requesting client.

Regarding the implementation, the general-purpose programming language Python [22] has been used since it can be easily customized to the used Web frameworks and apps to build the designed Web server system with the proper tools and libraries such as Django [6] or Flask [27]. In particular, the Django REST-framework [7] which combines Django with a REST architecture [8] has been used. This approach can easily help to separate the client and server sides and, in turn, supports the portability and scalability of the designed architecture.

After the proxy server has downloaded the required DICOM files from the Orthanc server, it converts these DICOM files into corresponding files with a *.jpeg*-format to prepare their further transport. Regarding our implementation these image files in *.jpeg*-format are compressed with the help of the Python package Pillow (PIL) [3]. As PIL does a lossy *.jpeg*-compression, the image size is drastically reduced. For instance, a DICOM file with a size of 80 MB is reduced to 7 MB during such a *.jpeg*-conversion. These compressed image blocks are then ready for a fast client access and can be transferred to the client side by the applied multipath-QUIC transport protocol stack of the Web server architecture.

When the client selects any of these received images for viewing, the *.jpeg*-compliant image is loaded into the user's app. This transmission approach saves a large amount of bandwidth and reduces the latency time of a response to the medical object request. For this purpose the image is divided into multiple subparts, i.e., the tiles, on which the compression has been performed. When the client send the first request, the lowest resolution files are returned. Once the user clicks on any of the thumbnail images the better version of the still compressed *.jpeg*-tiles are sent back. Now if the user wants to see the original image of any part, he/she can simply click on the particular part and the corresponding original tile is finally transferred back to the client.

4.3 Client Realization by a Multipath Compatible Android App

Regarding the developed client-server architecture for the DICOM image retrieval and transfer in Fig. 6, the new client side has been developed using an Android system [12] and the programming language Go [19]. This selection is reasonable for the described e-health scenarios since Android is a versatile open source, Linux based software system designed for mobile touch screen devices such as smartphones and tablets. Regarding the development of an Android app, one can implement it by the Angular [2] framework and further embedded the Angular application as WebView into the Android system as shown in Sect. 3. However, this approach cannot be used in combination with multipath-QUIC, as Angular [2] applications are a subset of JavaScript applications which, in general, cannot access the networking stack of a browser directly. But such an access scheme would be necessary to use the developed multipath-QUIC API.

Therefore, it has been necessary to implement a completely new native Android application compared to the previous one of Sect. 3. To support multipath-QUIC, two possible approaches have been identified: either to use the existing multipath-QUIC implementation [5,11] in the language Go, which might be the simplest solution, or to use the Cronet library applied by Chromium on Android [13]. In the first case, one needs to check whether the Go code is able to access the used Android network stack. In conclusion, accessing the network layer functionality is no problem at all. However, as Cronet has only implemented QUIC (cf. [11,13]), but not multipath-QUIC, this second case has been disregarded in our implementation.

Regarding the development of the new client application, Android version 11 is used here. Go is a statically typed, compiled programming language that

provides a high performance in networking and concurrency scenarios [19]. Therefore, it has been employed in the multipath-QUIC implementation [5]. The front-end design of the new Android application is very similar to the previously developed Angular Web app in Sect. 3. The existing multipath-QUIC implementation [5] written in Go is also used for our network communication stack. Furthermore, the Go tool gobind [26] is applied to generate an Android archive file from the multipath-QUIC Go-client. This archive is then used in the Android application, to fetch the image data from the proxy server. All the client-server communication takes place by means of this Go code. The search box uses an input text string to load the medical images of a patient using the *PatientID* as key. This request is issued by the Go-client. Then this client establishes the multipath connection with the Go-endpoint as counterpart in the proxy server and requests data via the multipath-QUIC implementation. If the related *PatientID* key has been found, all the available compressed image data in *.jpeg* format that belong to the patient are returned to the client side. These image data are stored in the cache of the application and immediately displayed as thumbnails or references. Whenever the user clicks on a particular thumbnail, then corresponding image tiles are loaded by the Go code as before.

4.4 Effective Medical Object Retrieval and Communication by an e-Health Application

The message sequence diagram in Fig. 10 depicts the prototypical communication flow between the various components of the resulting enhanced client-server PACS architecture. An e-health application based on the Android app first requests the required medical data at the proxy server using a patient's identifier. This request is passed by the multipath-QUIC client which is invoked at the communication layer by the related DICOM image request. Using the Go implementation of multipath-QUIC, a corresponding connection with the

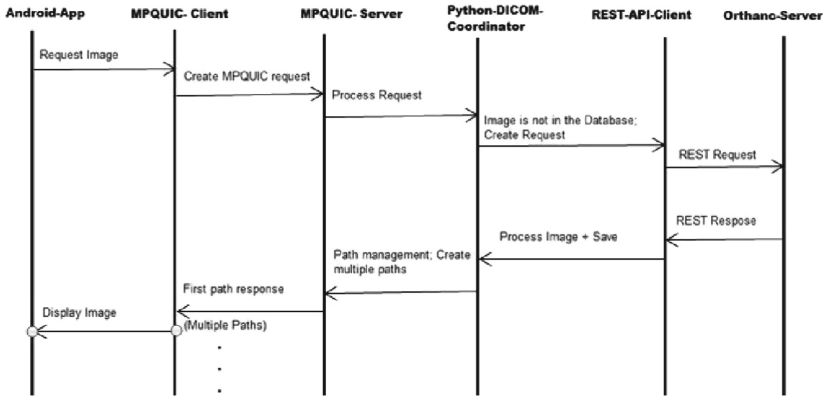

Fig. 10. Message flow of a proxy server retrieval regarding a DICOM image transfer

addressed multipath-QUIC server that is residing at the backend of the e-health application is then established. This medical data request is finally lifted to the application layer of the Python-proxy server. The latter validates the request and looks for the requested data in the associated medical database. If the data are not present in this cache of the proxy, they are fetched from the interconnected Orthanc server using a REST request. In this way, the required date become available at the proxy side locally for the invoked data transfer. Then this cached data are transferred to the client application by the multipath-QUIC server using multiple QUIC/UDP paths. Once the data are received at the multipath-QUIC client, they are rendered and displayed by the Android e-health application.

5 Conclusions

Nowadays, new advanced e-health applications are required to support the effective processing of the implemented diagnostic and therapeutic healthcare protocols of a modern society as the pandemic has shown, see also [14]. A classical Picture Archiving and Communication System (PACS) for medical images and their related e-health apps incorporates a client-server system based on the DICOM [21] standard and Android Web clients, cf. [4,16,17].

In our study we have shown how such a PACS architecture can be enhanced by a DICOM server that manages the medical image transfer and uses a set of reformatted, compressed tiles of an original image to speed up the retrieval and transport processes. Furthermore, we have designed a Web server application that offers the effective transport of these medical image data by means of a multipath-QUIC [5] protocol. It is integrated into the PACS architecture of a proxy server storing the retrieved DICOM images. We have implemented the designed architecture by a prototype with an integrated multipath-QUIC transport layer which has been realized by the programming language Go, see [11]. Our experiments have revealed the feasibility of the developed enhanced PACS architecture and its benefits. Further performance studies are needed to evaluate the full potential of the developed multipath-transport method for retrieved DICOM images.

The realized prototype still has some weaknesses such as a lack of encryption with regard to the data transport and the data-in-rest of a transferred DICOM image in the stack. These deficiencies might violate compliance regulations like HIPAA and GDPR for such sensitive medical data. Thus, a more advanced security and privacy policy must be integrated into the developed protocol stack and a strong verification of the resulting code is required after that as well.

Moreover, the applied QUIC Go project [11], which provides the basis of our applied multipath-QUIC [5] Go coding scheme, has recently added some technical support for HTTP/3.0. It should be checked whether these API enhancements can also be ported to a multipath-QUIC scheme to support the presented DICOM application scenarios. Furthermore, a study of the proposed multipath transport concept is currently realized in a real industrial context of medical image analysis. After its completion comprehensive measurement studies will be

executed to reveal its performance compared to the single-path approach realized by current PACS systems. All these issues shall become a subject of our future research.

References

1. Aazam, M., Huh, E.-N.: Fog computing and smart gateway based communication for cloud of things. In: 2014 International Conference on Future Internet of Things and Cloud (FiCloud), pp. 464–470, August 2014
2. Angular - The modern web developer's platform. https://angular.io/
3. Clark, A., et al.: Pillow. https://pillow.readthedocs.io/en/stable/
4. Cornerstone Project: Medical Imaging, Simplified. https://cornerstonejs.org/
5. De Coninck, Q., Bonaventure, O.: Multipath QUIC: design and evaluation. In: Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies, CoNEXT 2017, pp. 160–166 (2017)
6. Django Software Foundation: Django. https://www.djangoproject.com/
7. Encode OSS Ltd.: Django REST framework. https://www.django-rest-framework.org/
8. Fielding, R.T.: REST: architectural styles and the design of network-based software architectures. Doctoral dissertation, University of California, Irvine (2000)
9. GitHub Project: DWV - DICOM Web Viewer. https://github.com/ivmartel/dwv
10. GitHub Project: DWV-angular. https://github.com/ivmartel/dwv-angular
11. GitHub Project: A QUIC implementation in pure Go. https://github.com/qdeconinck/mp-quic
12. Google: Android. https://www.android.com/
13. Google: Documentation for app developers. Developer Guides, Cronet. https://developer.android.com/guide/topics/connectivity/cronet/reference/org/chromium/net/CronetEngine
14. Islam, S.M.R., et al.: The internet of things for health care: a comprehensive survey. IEEE Access **3**, 678–708 (2015)
15. Iyengar, J., Thomson, M.: QUIC: a UDP-based multiplexed and secure transport. Internet draft, draft-ietf-quic-transport-04, June 2017. https://datatracker.ietf.org/doc/rfc9000/
16. Jodogne, S.: The Orthanc ecosystem for medical imaging. J. Digit. Imaging **31**(3), 341–352 (2018). https://doi.org/10.1007/s10278-018-0082-y
17. Jodogne, S.: Orthanc - the free and open-source, lightweight DICOM server. Orthanc Labs, Visé, Belgium. https://www.orthanc-labs.com/, https://www.orthanc-server.com/
18. Kaur, H., Kaur, R., Kumar, N.: Lossless compression of DICOM images using genetic algorithm. In: 2015 1st International Conference on Next Generation Computing Technologies (NGCT), pp. 985–989, September 2015
19. Kincaid, J.: Google's go: a new programming language that's Python Meets C++. TechCrunch. Retrieved, vol. 29 (2010)
20. Mah, P., Reeves, T.E., McDavid, W.D.: Deriving hounsfield units using grey levels in cone beam computed tomography. Dentomaxillofacial Radiol. **39**(6), 323–335 (2010). pMID: 20729181
21. National Electrical Manufacturers Association, NEMA PS3/ISO 12052. Digital Imaging and Communications in Medicine (DICOM) Standard (2017). https://www.dicomstandard.org/

22. Python Software Foundation: Python. https://www.python.org/
23. Python Software Foundation: ctypes - A foreign function library for Python. https://docs.python.org/dev/library/ctypes.html
24. Seeram, E.: Computed Tomography - Physical Principles, Clinical Applications, and Quality Control. Elsevier, Amsterdam (2015)
25. Tahmoush, D., Samet, H.: A new database for medical images and information. In: Horii, S.C., Andriole, K.P. (eds.) Proceedings of SPIE 6516, Medical Imaging 2007: PACS and Imaging Informatics, 65160G, pp. 140–148. International Society for Optics and Photonics (2007)
26. The Go Project: Documentation - Binding Go. https://godoc.org/golang.org/x/mobile/cmd/gobind
27. The Pallets Projects: Flask. https://palletsprojects.com/p/flask/