An AAL-Oriented Measurement-based Evaluation of Different HTTP-based Data Transport Protocols

Thomas Zinner*, Stefan Geissler*, Fabian Helmschrott* Susanna Spinsante‡, An Braeken§
*Institute of Computer Science, University of Würzburg, Am Hubland, 97074 Würzburg, Germany
Email: {zinner|stefan.geissler|fabian.helmschrott}@informatik.uni-wuerzburg.de
‡ Università Politecnica delle Marche, Ancona, Italy
Email: s.spinsante@univpm.it
§ Vrije Universiteit Brussl, INDI, Brussels, Belgium
Email: an.braeken@vub.ac.be

***Abstract*—A key requirement for Active and Assisted Living (AAL) environments is the exchange of data between different communication endpoints to support wide range of services and applications. Used communication protocols need to support the bidirectional flow of information and have to be optimized with regard to security or latency constraints. To address these issues, RESTful approaches have recently gained much attention from the community. In this context, different application layer transport protocols can be used to realize the required data exchange. Besides HTTP/1.1, developed and standardized in the 1990s, new protocols like HTTP/2 and the QUIC transfer protocol my be suitable candidates. The impact of the different protocols on the overall performance for web and AAL services is still an open research question. This paper narrows this gap by conducting a measurement-based comparison of the three described protocols with regard to their performance in terms of web page loading times for Google web services.**

***Keywords*—AAL;ELE;QUIC;HTTP;Measurements;RESTful;**

I. Introduction

Active and Assisted Living (AAL), as well as Enhanced Living Environments (ELE), deal with a huge variety of data types, among which, in a coarse classification, the following ones can be mentioned: behavioral (or habits-related) data [1], [2], physiological data [3], [4], environmental (or ambient-related) data [5], [6], and healthcare data [7], [8]. Data are both *produced* by sensors implemented in AAL and ELE systems, and *consumed* by the same systems, to support services and applications; it is possible to state that AAL and ELE systems act as data *prosumers*. As a consequence, data communication and exchange protocols play a very fundamental role in AAL and ELE platforms, and it is necessary to analyze the performance different protocols may provide, with respect to the requirements posed by the systems and services adopting them. In safety-related applications, such as fall detection systems [9] or systems monitoring anomalous escapes of subjects with dementia [10], an almost real-time delivery of the alarm notification to the final receivers (i.e. rescue operators, nurses, caregivers) is required to effectively support the monitoring service. On the other hand, applications focused on long-term monitoring, like behavioral analysis [11], require the collection of large amounts of data. To allow the reliable and efficient transmission of sensor data, modern application layer transfer protocols are required.

The data *variety* mentioned above, the *volume* of data that AAL or ELE platforms are able to collect from sensors, and the data transmission *velocity* requested by AAL and ELE systems, with some services expected to be delivered in real-time (like emergency notifications), make AAL and ELE domains requirements very similar to those associated to the Big Data and Internet of Things (IoT) paradigms. At the same time, however, AAL exhibits specific requirements and constraints that shall be carefully accounted for, when adopting an IoT-oriented design approach. Data handling and management in AAL require a review of some commonly adopted strategies in IoT, with respect to processing (that can be locally or remotely executed), delivery format (raw data can be transmitted, or aggregated information generated by their processing), and sharing options (privacy and security concerns related to the use of personal and health-related data). As an example, research [12] demonstrated that to increase the compliance with, and the success of a therapy, it is necessary to proactively send reminders and feedback to the patient, according to the so-called Closed Loop Principle. Communication protocols for AAL shall consequently support bidirectional flows of information. They shall be optimized to trade-off lifetime, security level, and pre-processing of the data from the sensors. In fact, latency possibly due to pre-processing and security operations applied to the raw data generated by clinical sensors may become unacceptable in applications where monitored physiological signals are needed [13].

Many AAL and ELE architectures and platforms proposed in the literature and prototyped in research projects rely on the HTTP-based Representational State Transfer (REST)-based APIs [14]–[16]. In fact, REST has gained much attention from the community as a lighter technology compared to SOAP-based web services. RESTful web services promote a resource-centric conceptualization, even if the dynamic discovery and eventing of RESTful services are yet considered a major hurdle to a full potential use of REST-based approaches.

In this context, different transport protocols can be used to realize the communication between AAL and ELE devices and their respective RESTful web service endpoints. First, there is the widely used HTTP/1.1 protocol that represents the current defacto standard. Developed and standardized in the 1990s, the original use case of delivering simple, text based websites has changed drastically. In order to overcome the drawbacks of HTTP/1.1 (e.g. head-of-line-blocking, single file requests), HTTP/2 has been released in 2015. Although the new version

improved the performance of the HTTP protocol in many ways, one of the main drawbacks still remains. Especially in the IoT context, the TCP handshake required to establish a HTTP connection reduces the performance drastically when it comes to small and frequent requests. This problem is finally tackled by the QUIC transfer protocol developed by Google [17]. In contrast to HTTP, QUIC uses UDP as its underlying transport protocol and realizes packet retransmission on application layer. This results in much faster connection setup times and thus improved performance.

This paper aims to compare the three described protocols, HTTP/1.1, HTTP/2 and QUIC with regard to their performance. Since QUIC is an experimental, non-standardized protocol, its most recent versions is not freely available. To be able to conduct comparable measurements of the latest protocol versions we evaluate the transmission performance of each of the protocols using Google web services. This allows a direct comparison of the protocol performance with respect to various metrics like connection setup time or page load time. Evaluation of these metrics allows to contextualize the performance of the protocols to AAL and ELEs. In fact, connection setup time is a critical parameter when dealing with the fast connection establishment needed to deliver alarms and emergency notifications in AAL. Similarly, page load time may be used as an indication of the responsiveness of HTTP-based user interfaces that are typically used in AAL platforms, to enable human-system interaction. Consequently, we can use the results to evaluate the potential of each of these protocols for different use cases and can thus assess whether QUIC or HTTP/2 might be suitable candidates in the AAL context.

The remainder of this paper is organized as follows: Section II presents the related work and background, and a performance comparison of HTTP-based Transport Protocols. The evaluation methodology adopted in this paper is discussed in Section III, and the related experimental results are presented in Section IV. Concluding remarks are provided in Section V.

II. Background and Related Work

A. HTTP-based Network Protocols

HTTP is an application level request/response protocol running on top of TCP [18]. The first formal standardization of HTTP was HTTP/1.1, which was released in January 1997 as RFC 2086 [19]. Common drawbacks of HTTP/1.1 are the sequential request/response exchange and the resulting head of line blocking on application layer, which negatively affect web page loading times. In mid-2009, Google announced their new experimental protocol called SPDY, which should solve some well-known HTTP/1.1 performance issues and therefore improve the web delivery latency. HTTP/2, which is based on SPDY and profits from the lessons learned during the design and operation of this protocol was standardized in May 2015 [20]. It overcomes the outlined head of line blocking on application level by dynamically multiplexing requests on one TCP connections. Besides the standardized protocols, Google designed an experimental protocol, QUIC based on UDP. It provides a couple of new features like faster connection establishment, connection migration, flexible congestion control and a support of cryptographic mechanisms by design.

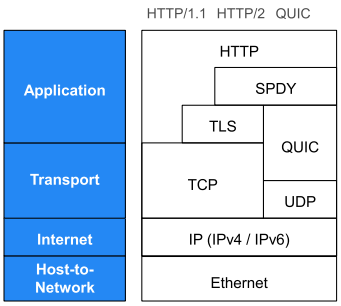

Fig. 1: Protocols Assigned to the Network Stack

In order to illustrate the fundamental difference between the protocols, Figure 1 shows their assignment to the network stack. As can be seen, HTTP/x are pure application layer protocols while QUIC is also partly located at transport layer. The main difference, is the particular underlying transport protocol.

Despite the obvious architectural differences, there are also different approaches in supporting a fast data transport, connection establishment, and encryption. HTTP/1.1 introduces keepalive connections, allowing to reuse already existing connections and hence to benefit from the properties of TCP. To further take advantage of the TCP connection, HTTP/2 introduced its binary framing layer which enables the possibility of multiplexing several streams over the same TCP tunnel. Even though QUIC is built on top of UDP, it uses stream multiplexing over a single connection just like HTTP/2.

HTTP/1.1 request pipelining was meant to allow multiple requests sent to the server simultaneously, however, it was never supported. Browser developers addressed this by opening up to six different TCP connections per origin. HTTP/2 got rid of the one request at a time constraint and introduced the concept of stream multiplexing over one connection. Hence, the browser can request multiple files simultaneously without the need of additional TCP connections. This multiplexing mechanism was also adopted by QUIC.

The main limiting factor concerning web latency is the Round Trip Time (RTT), thus it is desirable to reduce the required RTTs as much as possible. With the introduction of keepalive connections as default, HTTP/1.1 was able to get rid of additional RTTs caused by the constant re-establishment of TCP connections after file transfers. In order to further reduce the needed RTTs, web developers used tricks like resource inlining, image sprites or domain sharding as mentioned before. With the multiplexing, HTTP/2 did not only improve the keepalive concept, but also got rid of additional TCP connections, which also introduced additional RTTs. Beside the stream multiplexing, QUIC introduces two other mechanisms which help to reduce RTTs: its 0RTT connection establishment and its connection migration. With the 0RTT connection establishment QUIC provides the ability to open an encrypted connection in at most one RTT instead of 3 RTTs when using TCP with TLS. QUIC's connection migration helps to re-establish a connection to an already known server faster, so that the client can request new files immediately without having to wait for the client-server handshake.

This is quite common case in AAL and ELEs, where devices and sensors typically communicate with the same server (usually a single server is used for an AAL system), that can be local to the AAL platform or remotely connected.

B. Performance Comparison of HTTP-based Transport Protocols

In 2013, a first comparison between QUIC and HTTP/1.1 was performed [21]. The evaluations are based on a 10 MB file download for different packet loss, bandwidth, and RTT configurations. During the experiments, HTTP/1.1 performed better than QUIC. This is mainly due to missing optimizations in the early stage of the QUIC protocols at this time, and the missing TLS integration for the HTTP/1.1 experiments.

The author of [22] picked up on the results of [21] in 2014. He repeats the measurements in order to investigate the improvement of QUIC by comparing the results of the different versions. Additionally, the runs are also performed using HTTPS. In case of packet loss, the newer QUIC version clearly prevails against HTTP/1.1 and HTTPS regarding goodput. Looking at the increasing RTT measurements, QUIC is able to improve its performance on low latency links yielding in a result equal to HTTP. However, RTTs higher than 200 ms still have a huge negative influence on QUIC's goodput. Beside these comparisons, the author also evaluates the multiplexing mechanism of QUIC by requesting 10 files of 100 KB each over a single connection. In order to compare QUIC with HTTP/1.1 and HTTPS in this scenario, they are limited to a single connection. Experiments with the multiplexing mechanisms reveal that the goodput of QUIC continues to grow with available bandwidth, while the goodput of HTTP/1.1 and HTTPS is capped due to the fact that they cannot request more than one file at a time. When experiencing jitter on packet delays, QUIC is highly negatively affected by even small values like 0.5 ms standard deviation to the normal RTT.

In [23], the author compares the performance of QUIC, SPDY, and HTTP/1.1 using web page emulation. The evaluation of the mean page load time of all scenarios shows that SPDY performs almost always better than HTTP/1.1. On links with low bandwidths QUIC preveals over HTTP/1.1 due to its multiplexing and compression features. Altogether, it can be seen that QUIC improves as the RTT increases and for RTTs higher than 210 ms QUIC clearly outperforms HTTP/1.1. Nevertheless, HTTP/1.1 performs better than QUIC on low-RTT links with high bandwidth.

The performance of QUIC on transport and on application level is investigated by the authors of [24]. On transport level, QUIC's flow dynamics as well as the friendliness of QUIC's and TCP's congestion control algorithms are evaluated. In order to investigate the flow dynamics, the authors look at the impact of the link capacity, random losses, and enabled FEC on the channel utilization. Beside QUIC with and without FEC, the measurements are also performed using TCP. It turns out that the overhead introduced by FEC worsens the overall performance of QUIC. Further, for 1 % as well as for 2 % induced losses, the TCP goodput is significantly reduced, while the QUIC link utilization is not. A performance evaluation on application level is done by measuring the page load time of websites using QUIC, SPDY over TLS, and HTTPS. The measurements are performed for a small, a medium, and a large web page containing only images on a 3 Mbps and a 10 Mbps link, each experiencing 0 % and 2 % random losses while having a RTT of 50 ms. The authors observe that during the measurements, HTTPS opens six parallel TCP connections, SPDY opens one TCP connection multiplexing as many streams as needed to request all resources simultaneously, and QUIC opens one UDP connection multiplexing six streams. As metric, the percentage page load time improvement with respect to HTTPS is chosen. For the scenarios without losses, both SPDY and QUIC outperform HTTPS. Further, SPDY outperforms QUIC on the 10 Mbps link for large web pages. When inducing 2 % random losses, the use of SPDY always increases the page load time regardless of bandwidth and web page size. Additionally, QUIC outperforms SPDY on lossy links.

The authors of [25] examine the effects of HTTP/1.1, SPDY and QUIC on page load time. To this aim, they deploy four simple websites on Google Sites, a web hosting service by Google. The authors conclude that page load time is decreased significantly in more than 40 % of the scenarios when using QUIC. Especially on links with high RTT, QUIC seems to help the most, while it does not perform well when downloading a large amount of data. Nevertheless, HTTP/1.1 performs best for downloading large objects.

Altogether it can be said, that several aspects regarding the performance of QUIC have been evaluated including several metrics on transport and application level. This paper enhances the state of the art by investigating the network transport delay for real world deployments of these protocols using Google web services. Further, our evaluations include a close look on the impact of QUICS 0RTT connection establishment mechanism.

III. Evaluation Methodology

This section highlights the implemented tools, the utilized metrics and the measurement setup and the corresponding course of events.

A. Browser Plugin

In order to evaluate the performance of web browsing on application layer, we developed a plugin for the Google Chrome browser. Since the browser itself already provides extensive network logging and debugging capabilities, the plugin can simply hook into the logging process by listening to event calls provided by the browser engine. The functionality and components of the plugin are described in the following.

The plugin uses the network analysis tool provided by Google Chrome *DevTools*[1] in order to measure resource loading times during a website request. Furthermore, the DevTools allow the export of a JSON log containing all requests and corresponding responses. In addition, the *Navigation Timing API*[2] enables measurements of different latencies during the website loading process based on browser events. The final timestamp collected by the plugin is provided by the browser function `window.chrome.loadTimes()` which

[1] https://developers.google.com/web/tools/chrome-devtools/

[2] https://www.w3.org/TR/navigation-timing/

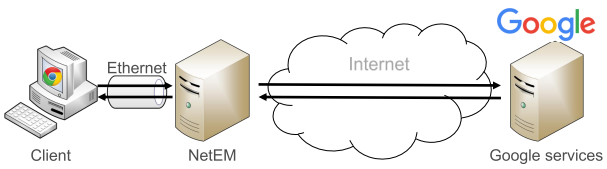

Fig. 2: Measurement Setup

Parameter	Used Values
Bandwidth (Mbps)	{2, 6, 16}
Additional RTT (ms)	{20, 100, 200}
Random Packet Loss (%)	{0, 1, 2}
0RTT CE of QUIC	{with, without}

TABLE I: Evaluation Parameters

represents the time at which the browser renders the first object. Finally, the plugin uses the *net-internals* tool built into Google Chrome in order to export the gathered data in JSON format for further processing.

B. Evaluation Metrics

A key metric regarding the performance of transfer protocols in the AAL context is the *Request Response Delay (RRP)*. It measures the time between a request is sent and the client receives the corresponding response. As the data volume that needs to be transferred in most AAL scenarios are small, the key factor influencing the RRP is the initial connection setup time.

By measuring productive Google web services we rely on the monitoring information provided by the browser plugin mentioned above. The corresponding metrics available for regular web site requests are i.e *Page Load Time (PLT)*, *Time to First Paint (TTFP)* or *DOM Content Loaded (DOM CL)*. In this paper, we focus on the evaluation of the PLT as it provides a solid indication for the delay between request and response during a web site request. The PLT is defined by the two browser events `navigationStart` and `loadEventEnd`, which represent the time at which the request is sent and the page is fully loaded and presented to the user respectively. The PLT, and especially the *perceived* PLT, has a relevant effect on users [26]. As web and app design optimization pushes the limits of fast and responsive service and content delivery, milliseconds can make a huge difference on the perceived responsiveness of a web or mobile application. In the AAL context, responsiveness is a crucial performance indicator for ensuring the usability of a web-based service. It is fundamental for the deployed application level transfer protocol to minimize transmission delays and thus increase the quality of service.

In addition to that, we calculate the *Speedup* in order to compare the performance of different protocols. The speedup thereby is defined as the ratio of the average page load times for different protocols.

C. Measurement Setup and Course of Events

The measurements performed in the context of this work have been done in a dedicated testbed comprised of two phyiscal machines that are directly connected to the internet. The measurement setup is depicted in Figure 2.

The middle machine, labeled *NetEM*, is running the Network Emulator Kernel module and shapes passing network traffic using the `tc` tool. This machine is responsible for introducing additional delay, packet loss and limit the available bandwidth according to the respective measurement scenario. The evaluated influence factors and the used values for each of the parameters are listed in Table I.

The *Client* machine on the left is running the Google Chrome browser with our plugin. It generates the website requests, monitors responses and extracts and stores relevant metric data.

The measurement workflow itself is comprised of two essential steps. First, the respective network parameters are configured using network emulation via `tc`. This enables the simulation of different network characteristics which influence the performance of the evaluated network protocols.

Second, the client machine requests a set of Google web services and measures the respective metric values mentioned in the previous section. The services used in this measurement step are the Google front page with averagely 212 KB, Google Drive with averagely 812 KB and Youtube with an average size of 1206 KB. Each of these services is queried 20 times for each parameter combination and each of the protocols HTTP/1.1, HTTP/2, QUIC and QUIC 0RTT CE. Important to note is that the measurements for each protocol are done in an alternating manner. After two consecutive website requests the used protocol is changed. This results in a spread of the measurement points in time. Thus, the impact of varying network and web service behavior, which cannot be controlled or quantified, is distributed allover the different protocols thus allowing a fair comparison.

This measurement workflow results in a dataset consisting of about 2000 measurements in total. The data is evaluated in detail in the following section.

IV. Evaluation of Page Load Times for Web Browsing

Firstly, we highlight the main effects of the investigated parameters and protocols on the page load times (PLT).

The influence two parameters may have due to an interaction is denoted as interaction effect. These effects provide a first indication for the actual measured influence of the parameters. The used results are the mean value over all measured repetitions for the particular parameter combination. Furthermore, the shown 90 % confidence intervals are calculated using the *t*-distribution.

The main effects plot for all investigated parameters and protocols can be seen in Figure 3. The packet loss scenarios have no significant impact on the page loading time and have thus not been included in this figure.

The first main effect to be evaluated is the RTT. An increase of this parameter results in an increase of the PLT. Even though the confidence intervals for 20 ms and 100 ms overlap, there is

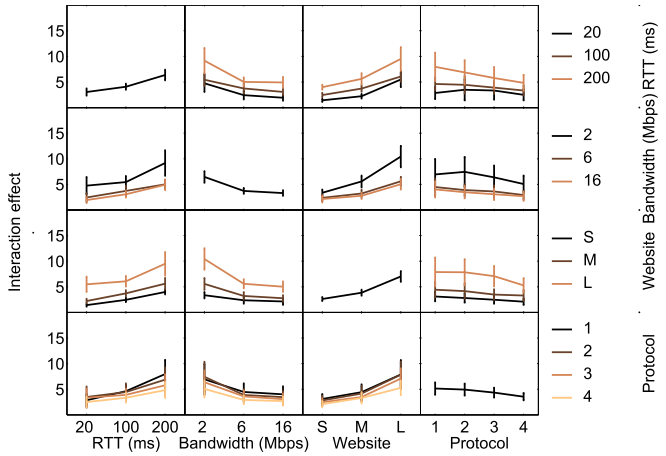

Fig. 3: Effects of Parameters on PLT for all Protocols. (1) HTTP/1 (2) HTTP/2 (3) QUIC (4) QUIC /w 0RTT

TABLE II: PLT Speedup Compared to HTTP/1.1 for 2 Mbps Bandwidth

	2 Mbps			
	20 ms RTT	50 ms RTT	100 ms RTT	200 ms RTT
Google Main Page				
HTTP/2	3.26 %	3.36 %	12.98 %	3.10 %
QUIC w/o 0RTT CE	−2.24 %	1.55 %	−3.60 %	14.54 %
QUIC w/ 0RTT CE	9.92 %	12.86 %	56.18 %	41.11 %
Google Drive				
HTTP/2	−1.88 %	2.55 %	9.23 %	1.79 %
QUIC w/o 0RTT CE	−1.31 %	16.12 %	18.99 %	43.56 %
QUIC w/ 0RTT CE	1.88 %	18.18 %	19.83 %	39.74 %
YouTube				
HTTP/2	5.22 %	−2.49 %	−3.47 %	14.96 %
QUIC w/o 0RTT CE	−1.54 %	−15.82 %	2.85 %	31.25 %
QUIC w/ 0RTT CE	14.97 %	6.30 %	25.26 %	73.52 %

no overlapping between 20 ms and 200 ms, nor between 100 ms and 200 ms. Hence, the RTT is definitely a major influence factor. For increasing bandwidth, the values again decrease, even though the reduction from 2 Mbps to 6 Mbps is bigger than from 6 Mbps to 16 Mbps. Also, the confidence intervals for 6 Mbps and 16 Mbps overlap. Therefore, an indication for the influence of the bandwidth is existent, however a clear statement can only be made for the change from 2 Mbps to a higher bandwidth. Considering the size of the web pages, it can clearly be seen that the trend follows the expectations and the previous separate evaluations. There is no overlapping of any of the confidence intervals. This clearly shows the major impact of the website size on the page load time. The last main effect is the influence of the individual protocols. Here, HTTP/1.1 has the highest average PLT, followed by HTTP/2 and QUIC without its 0RTT connection establishment. The protocol with the lowest value is QUIC using its fast establishment mechanism. As can be seen, the confidence intervals for all protocols overlap among each other. Due to this, a clear statement about the influence of the protocol on the PLT cannot be made. Nevertheless, the visible trend indicates the possible advantage of QUIC and its 0RTT connection establishment. Furthermore, the figure indicates that the interaction of the website size and the RTT has a real effect on the PLT for all protocols. Besides, the interaction of the website size and the bandwidth is noticeable.

The jump from 2 Mbps to 6 Mbps results in a significant reduction of the PLT, while the further increment to 16 Mbps only yields a small gain. This indicates that for 6 Mbps, the initial connection setup time is already the limiting factor when it comes to the PLT. Hence, in the following evaluation, we will only focus the first two bandwidth configurations. Since the particular PLT depends on the size of the website, its speedup with regard to the result of HTTP/1.1 is used for the rest of this evaluation, in order to assess the performance of the other protocols. Table II shows the speedups for all RTT scenarios and 2 Mbps bandwidth. The listed values are the results for the Google main page, Google Drive and YouTube, since these are the representatives chosen for the previous main effect evaluation. Looking at the table, some clear trends are emerging. One of these trends is, that QUIC with 0RTT CE always performs better than HTTP/2 for 2 Mbps bandwidth and regardless of the RTT. Further, it also performs best for the PLT metric in these scenarios except for Google Drive when experiencing 200 ms RTT. In this particular case, QUIC without 0RTT CE performs better. A closer look at the measured data revealed, that 65 % of the page loads performed with the 0RTT CE are faster than QUIC without the mechanism. However, page requests that take longer than 6.906 s to complete end up with a significant longer PLT, while the page loadings using QUIC without 0RTT CE do not perceive such a deterioration. Due to this, QUIC without 0RTT CE has a higher speedup for PLT with regard to HTTP/1.1 in this scenario. Another observable trend is that with increasing RTT, QUIC with 0RTT CE performs better. This behavior was expected, since the main effect evaluation indicated the influence of the RTTs, but also because the 0RTT CE of QUIC is thought to be the main performance boost. This mechanism eliminates the RTTs required for the connection establishment and should therefore give QUIC an advantage of three RTTs compared to HTTP. The determined trends can also be seen in Table III, which shows the corresponding speedup values for the 6 Mbps scenarios. There, QUIC with 0RTT CE is always faster than HTTP/2 regardless of the RTT, as well. Consequently, it also performs best for PLT considering these parameter combinations. Further, the performance gain of it tends to increase with growing RTT, too. When looking at the results of the remaining measurements, these two trends are throughout noticeable with a few exceptions.

Based on these observations, it can be said that the QUIC with 0RTT CE seems to perform best considering PLT regardless of the bandwidth. Moreover, the RTT really has an influence on the performance of QUIC, as the main and interaction effects indicated. Although the website size does influence the PLT as assumed due to the evaluated effects, this effect cannot be seen in the speedup for PLT.

V. Conclusion

Many AAL and ELE architectures and platforms proposed in the literature and prototyped in research projects rely on the HTTP-based Representational State Transfer (REST)-based

TABLE III: PLT Speedup Compared to HTTP/1.1 for 6 Mbps Bandwidth

	6 Mbps			
	20 ms RTT	50 ms RTT	100 ms RTT	200 ms RTT
Google Main Page				
HTTP/2	−10.68 %	13.44 %	24.72 %	4.10 %
QUIC w/o 0RTT CE	2.25 %	18.01 %	37.19 %	44.62 %
QUIC w/ 0RTT CE	21.41 %	36.34 %	58.99 %	63.02 %
Google Drive				
HTTP/2	2.27 %	14.80 %	15.94 %	16.35 %
QUIC w/o 0RTT CE	4.30 %	35.93 %	29.79 %	39.69 %
QUIC w/ 0RTT CE	10.42 %	46.27 %	29.48 %	70.68 %
YouTube				
HTTP/2	1.00 %	15.70 %	41.31 %	45.63 %
QUIC w/o 0RTT CE	−5.62 %	12.00 %	56.10 %	40.98 %
QUIC w/ 0RTT CE	1.21 %	33.42 %	68.73 %	88.98 %

APIs. In this context, different transport protocols can be used to realize the communication between AAL and ELE devices and their respective RESTful web service endpoints. Such protocols are standardized protocols like HTTP/1.1 and HTTP/2, but also the experimental QUIC protocol, which can already be used to access Google web services. This paper provides a measurement-based comparison of the three described protocols with regard to their performance in terms of web page loading times.

The measurement results reveal QUIC with its 0RTT Connection Establishment outperforms the other protocols with respect to the average page load times. Further, its performance also improves with regard to HTTP/1.1 with increasing RTT. Accordingly, we can conclude that the QUIC protocol, particularly with its 0RTT connection establishment feature, significantly reduces data exchange delays for RESTful web services. Nevertheless, this work constitutes only a first step towards a better understanding of the enhancements provided by the QUIC protocols. Particularly, its impact on the AAL ecosystem remains an open research question, mainly due to the huge variety of implementations possible for AAL and ELEs.

ACKNOWLEDGEMENT

This work has been supported/partially supported by the ICT COST Action IC1303 - Algorithms, Architectures and Platforms for Enhanced Living Environments (AAPELE).

REFERENCES

[1] A. Aztiria, G. Farhadi, and H. Aghajan, "User behavior shift detection in ambient assisted living environments," *JMIR Mhealth Uhealth*, vol. 1, no. 1, Jun 2013.

[2] K. Park, Y. Lin, V. Metsis, Z. Le, and F. Makedon, "Abnormal human behavioral pattern detection in assisted living environments," in *Proceedings of the 3rd International Conference on PErvasive Technologies Related to Assistive Environments*, ser. PETRA '10. New York, NY, USA: ACM, 2010, pp. 9:1–9:8. [Online]. Available: http://doi.acm.org/10.1145/1839294.1839305

[3] O. Gama and R. Simoes, "A platform to emulate ambient assisted living environments," in *e-Health Networking, Applications Services (Healthcom), 2013 IEEE 15th International Conference on*, Oct 2013, pp. 46–50.

[4] F. Palumbo, J. Ullberg, A. Stimec, F. Furfari, L. Karlsson, and S. Coradeschi, "Sensor network infrastructure for a home care monitoring system," *Sensors*, vol. 14, no. 3, p. 3833, 2014. [Online]. Available: http://www.mdpi.com/1424-8220/14/3/3833

[5] C. Nugent, L. Galway, L. Chen, M. Donnelly, S. McClean, S. Zhang, B. Scotney, and G. Parr, "Managing sensor data in ambient assisted living," *Journal of Computer Science and Engineering*, vol. 5, no. 3, pp. 237–245, 2011.

[6] F. Corno and F. Razzak, "Real-time monitoring of high-level states in smart environments," *Journal of Ambient Intelligence and Smart Environments*, vol. 7, no. 2, pp. 133–153, 2015.

[7] M. M., S. Wagner, C. Pedersen, F. Beevi, and F. Hansen, "Ambient assisted living healthcare frameworks, platforms, standards, and quality attributes," *Sensors*, vol. 14, no. 3, pp. 4312–4341, 2014.

[8] D. Rodrigues, E. Horta, B. Silva, F. Guedes, and J. Rodrigues, "A mobile healthcare solution for ambient assisted living environments," in *e-Health Networking, Applications and Services (Healthcom), 2014 IEEE 16th International Conference on*, Oct 2014, pp. 170–175.

[9] S. Gasparrini, E. Cippitelli, S. Spinsante, and E. Gambi, "A depth-based fall detection system using a kinect sensor," *Sensors*, vol. 14, no. 2, pp. 2756–2775, 2014. [Online]. Available: http://www.mdpi.com/1424-8220/14/2/2756

[10] L. Montanini, L. Raffaeli, A. D. Santis, A. D. Campo, C. Chiatti, G. Rascioni, E. Gambi, and S. Spinsante, "Overnight supervision of alzheimer's disease patients in nursing homes - system development and field trial," in *Proceedings of the International Conference on Information and Communication Technologies for Ageing Well and e-Health - Volume 1: ICT4AWE,*, 2016, pp. 15–25.

[11] S. Gasparrini, E. Cippitelli, E. Gambi, S. Spinsante, and F. Florez-Revuelta, "Performance analysis of self-organising neural networks tracking algorithms for intake monitoring using kinect," in *IET International Conference on Technologies for Active and Assisted Living (TechAAL)*, Nov 2015, pp. 1–6.

[12] L. H. and W. X., "Intervention strategies for improving patient adherence to follow-up in the era of mobile information technology: A systematic review and meta-analysis," *PLoS ONE*, vol. 9, no. 8, 2014.

[13] P. Porambage, A. Braeken, A. Gurtov, M. Ylianttila, and S. Spinsante, "Secure end-to-end communication for constrained devices in iot-enabled ambient assisted living systems," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, Dec 2015, pp. 711–714.

[14] S. Spinsante, E. Gambi, L. Montanini, and L. Raffaeli, "Data management in ambient assisted living platforms approaching iot: A case study," in *2015 IEEE Globecom Workshops (GC Wkshps)*, Dec 2015, pp. 1–7.

[15] A. D. Campo, E. Gambi, L. Montanini, D. Perla, L. Raffaeli, and S. Spinsante, "Mqtt in aal systems for home monitoring of people with dementia," in *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, Sept 2016, pp. 1–6.

[16] A. M. de Jesus Pereira, "Special issue: Ambient assisted living (aal): Sensors, architectures and applications," *Sensors*, 2014.

[17] Roskind, Jim, "QUIC: Design Document and Specification Rational," 2015. [Online]. Available: https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/edit

[18] I. Grigorik, *High Performance Browser Networking: What every web developer should know about networking and web performance*. "O'Reilly Media, Inc.", 2013.

[19] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, "Rfc 2068: Hypertext transfer protocol-HTTP1.1," *Status: PROPOSED STANDARD*, January 1997.

[20] M. Belshe, M. Thomson, and R. Peon, "Rfc 7540: Hypertext transfer protocol version 2 (HTTP/2)," 2015.

[21] Alex Gizis (Connectify), "Taking Google's QUIC For a Test Drive," 07. November 2013. [Online]. Available: http://www.connectify.me/blog/taking-google-quic-for-a-test-drive/

[22] A. Vernersson, "Analysis of udp-based reliable transport using network emulation."

[23] S. R. Das, "Evaluation of quic on web page performance," Ph.D. dissertation, Massachusetts Institute of Technology, 2014.

[24] G. Carlucci, L. De Cicco, and S. Mascolo, "Http over udp: an experimental investigation of quic," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 2015, pp. 609–614.

[25] P. Megyesi, Z. Krämer, and S. Molnár, "How quick is quic?"

[26] P. Isaas and K. Blashki, *Human-Computer Interfaces and Interactivity: Emergent Research and Applications*. IGI Global, 2014.