A Comparison of current Web Protocols for usage in Cloud based Automation Systems

Dhavalkumar Shekhada, Michael Stiller, Aniket Salvi

Universität Augsburg
Lehrstuhl für Kommunikationssysteme Institut für Informatik
Universitätsstraße 2
86159, Augsburg

Fraunhofer ESK
Fraunhofer Institute for Communication System ESK
Hansastraße 32
80696,München
dhaval.shekhada@informatik.uni-augsburg.de
michael.stiller@esk.fraunhofer.de
ani9127@gmail.com

Abstract: Web protocols are critical factor in successful implementation of web based industrial applications. Nevertheless, the main issue that significantly affects the real-time performance of such application is Web latency. Traditional web communication technologies, such as HTTP1.1, provide a uni-directional link and a request/response message exchange model. This solution is not feasible in web based applications involving large number of different interconnected devices, such as Industrial Internet of things. In this paper, we focus on bi-directional web protocols WebSocket, WebRTC and HTTP/2.0. We compare the performance parameters like latency, reliability, and security of these three protocols. At the end of this paper, we will present some suggestions how to optimize the usage of web protocols in industrial process automation applications.[1]

Keywords- WebSocket, WebRTC, HTTP/2.0, Industry 4.0, IIOT

1 Introduction

There is a trend in todays Industrial automation to switch most of the communication structure across all levels of industry to Ethernet [Kap01]. This step allows embedded devices and PLCs to provide process data directly via TCP/IP and Web Services. Most of the latest PLC controllers contain a web server as well as special HTML pages on the device, these enabling a browser-based configuration and diagnosis of the controller. Process data or program variables form the control program and can also be read, and sometimes also written, with restrictions. This also encourages the use of Web based visualizations. One of the biggest hurdles to the expansion of IoT (Internet of things) is its proprietary and domain specific projects that lead to numbers of incompatible protocols[Hau11]. This makes the integration of data and services from different devices too complex and costly. To enable this vision of an open IIoT, a unique application layer protocol is required for devices and applications to talk to each other, regardless of how they are physically connected. The solutions are proprietary and adapted to the relevant controller. Open and consistent web interfaces are not available. The I40 (Industry4.0) requirements[FP16] cannot therefore be fulfilled. This issue can be solved by re-using the existing World Wide Web (Web) infrastructure. There is a difference between terms IoT and WoT. The term IoT has been focusing on lower layers and hardware infrastructure, while the WoT is realized exclusively on application level protocols. Therefore, this paper conducts a performance and process data latency investigation for light weighted, bidirectional, real-time web protocols. Based on better network performance, scalability and deployability, we end up with three well known Web protocols WebSocket, HTTP/2.0, WebRTC [Gri13].

This paper has carried out few tests to examine the data latency of WebSocket, WebRTC and HTTP/2.0 and their suitability for the Industry4.0.

[1]The IGF project CICS (18354 N) of the Forschungsvereinigung Elektrotechnik beim ZVEI e.V. FE, Lyoner Str. 9, D-60528 Frankfurt am Main is funded via the AiF within the framework of the programme for funding industrial community research and development (IGF) of the Federal Ministry of Economics and Technology based on the resolution of the German parliament

The remainder of this paper is organized as follows: Section 2 introduces the related work of global network of process data in Industrial Automation, Introduction to web protocols (WebSocket, WebRTC and HTTP/2.0), the classification of industrial automation application based on process data latency and Web protocols which are being used by OPC UA stack in industry. Section 3 shows the practical experiments of data latency in WebSocket, WebRTC and HTTP/2.0 web protocols. In section 4, we conclude with a proposal for the hybrid communication architecture for industrial automation by combining protocol architectures.

2 Related Work

2.1 Global Networking of Process Data in Automation

Web protocols are the key factor for web and web of things. Typical web services found on devices usually must be polled to get the latest value. Access via a web browser is via the HTTP 1.1 protocol and is hence query-based and relatively slow. Examples of this can be found in [Eva11]. Hence it introduced a huge overhead in communication which is very critical for Industrial Automation. For Industry, OPC UA could solve this problem but the required functionality is too complex to be implemented in constrained devices. OPC UA requires a large amount of resources during implementation in the browser [ISO08] [BM10]. In most automation systems, the important requirement is availability of the communication network. Especially in process automation systems where each hour of downtime may cost huge amount in lost production. The system recovery time requirements are from hundreds of milliseconds up to few seconds in process automation area [a7].

The PLC controllers can be integrated in supervisor, management and coordination systems (e.g. SCADA or MES systems), which are partly based on web technologies. The additional modules are required to integrate in the PLC controllers, that enabling a bidirectional and event-based process data transmission between the controller and supervisor & management system. This includes, for example, solutions such as the use of Java-Applets on websites for access to Siemens controllers [Kli05], the Web connector with MQTT broker in Bosch-Rexroth controllers [BR15] or also client(browser)-based access to controllers that already contain an OPC UA server [Gmb]. These solutions also involve proprietary and closed control-integrated modules. Although the modules utilize web Technologies, they cannot be transferred to other controllers. The global process data communication is used for HMIs (HMI - Human Machine Interface) and/or supervisor & coordination functions in the higher level of the automation hierarchy (e.g. plant management level). There is no availability of authoritative statements regarding the time response of the process data transmission. Nevertheless, different statements result from the latency times of $>$100 to 300 ms. The Open and consistent web interface is available in [FP16] by providing PLC as I40 component, which fulfilled the I40 requirements. In [FP16], the process data transmission between the virtual device (in any web browser) and device gateway (in PLC) is realized by WebSocket and it also shows the time response of bidirectional process data transmission over intranet and internet are 30 ms and 50 ms respectively. This encourages us to have a look in to different bidirectional web protocols.

2.2 Introduction to Web Protocols

In this part, we give a short introduction of three Web protocols and how they differs from each other [Gri13].

2.2.1 WebSocket

WebSocket is a set of multiple standards: the WebSocket API is defined by the W3C, and the WebSocket protocol (RFC 6455) and its extensions are defined by the HyBi Working Group (IETF). WebSocket enables bidirectional full duplex and long live communication over single TCP(Transmission Control Protocol) connection. WebSocket resources URL uses its own custom scheme: ws for plain text communication (e.g., ws://example.,com/socket) and wss encrypted channel. There are different schemes except HTTP because the WebSocket protocol can be used outside the browser and could be negotiated via non http exchange. Therefore, it requires different URL schemes. WebSocket doesnt need to worry about buffering, parsing and reconstructing the received data. After establishing the WebSocket connection, the client can send and receive UTF-8 encoded message, Array buffer or Blob used for binary transmission. WebSocket offers a bidirectional communication

channel which allows message delivery in both directions over same tcp connection.

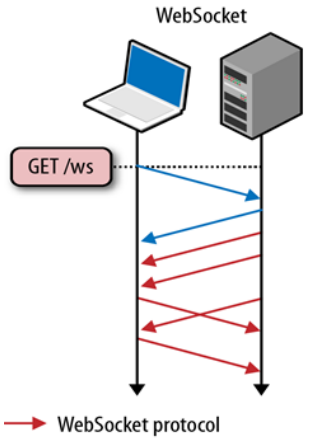

Abbildung 1: WebSocket Bidirectional communication

WebSocket does not decrease the roundtrip between the client and server. Regardless of the transport, the propagation latency of the data packet is the same. However, aside from propagation latency, there is also queuing latency: the time the message has to wait on the client or server before it can be routed to the other party. WebSocket is specifically referring to the elimination of message queuing latency. Fig 1 shows the client server communication over WebSocket. First handshake is done by HTTP protocol and after handshake protocol is switched to WebSocket. Once a WebSocket connection is established, the client and server exchange data via the WebSocket protocol: application messages are split into one or more frames, each of which adds from 2 to 14 bytes of overhead. Further, because the framing is done via a custom binary format, both UTF-8 and binary application data can be efficiently encoded via the same mechanism. WebSocket can transfer both text and binary data, and as a result it doesnt make sense to compress the entire session. The binary payloads may be compressed already! As a result, WebSocket must implement its own compression mechanism and selectively apply it to each message.

WebSocket Security

The browser is optimized for HTTP data transfers. It understands the protocol, and it provides a wide range of services, such as authentication, caching, compression, and much more. As a result, XHR (XMLHttpRequest) requests inherit all of this functionality for free. In WebSocket, streaming allows us to deliver custom protocols between client and server, but at the cost of bypassing many of the services provided by the browser: the initial HTTP handshake may be able to perform some negotiation of the parameters of the connection, but once the session is established, all further data streamed between the client and server is not transparent to the browser. As a result, the flexibility of delivering a custom protocol also has its downsides, and the application may have to implement its own logic to fill in the missing gaps: caching, state management, delivery of message metadata, etc.

Key exchange during handshake only confirms the Websocket communication that prevents cross protocol talks. It doesnt prove any data integrity or trust. So in simple words, it just identifying the WebSocket connection to prevent the cross protocol or text (for example it prevents to talk with SMTP) but not identifying server. The one aspect is that user-agent should not establish the plaintext (ws://) with secure resources(https://). Proxies and firewall cannot recognize WebSocket traffic. It is open to DOS(Denial of Service) Attack because of not limited concurrent WebSocket connections per origin. There is no way for server to authenticate the clients during the handshake. So most of security concerns are left for the web developers. Developers can handle this security issues by writing code for validating the users, verifying origin and so on [Lom].

2.2.2 WebRTC

Web Real-time Communication is a collection of standards, protocols, and JavaScript APIs. WebRTC breaks away from the familiar client-to-server communication model, which results in a full re-engineering of the networking layer in the browser, and also brings a whole new media stack, which is required to enable efficient, real-time processing of audio and video. Web Real-Time Communications (WEBRTC) W3C Working Group is responsible for defining the browser APIs. Real-Time Communication in Web-browsers (RTCWEB) is the IETF Working Group responsible for defining the protocols, data formats, security, and all other necessary aspects to enable peer-to-peer communication in the browser. This combination enables peer to peer audio, video and data sharing between browsers (peers). Delivering high quality, rich, Real time communication application such as audio and video teleconferencing and peer to peer data exchanges which is required lots of new functionalities in the browser. WebRTC abstracts these functionalities in three primary APIs.

- MediaStream: Acquisition of audio and video streams
- RTCPeerConnection: Communication of audio and video data
- RTCDataChannel: Communication of arbitrary application data.

WebRTC has performance characteristic such as low connection setup latency and less protocol overhead etc. WebRTC transports its data over UDP(User Datagram Protocol).
WebRTC provides two different paths for communication: the signaling path goes from the call initiator (caller) to the call receiver (callee) via HTTP or HTTPS connections to the calling sites. Then the media path is setup in a peer to peer fashion between caller and callee using network protocols such as session traversal utilities for NAT(Network Address Translation) and DTLS-SRTP (Datagram Transport layer Security- Secure Real-time Transport Protocols) for Real time communication. Both communication parties use an identity provider (IDP) for identity and authentication.

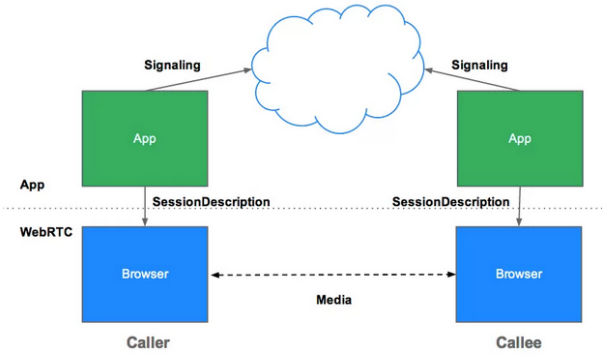

Abbildung 2: WebRTC Bidirectional communication

Fig 2 shows the WebRTC peer to peer communication. First WebRTC uses a signaling mechanism to establish peer to peer communication. Signaling methods and protocols are not specified by WebRTC. Signaling is also not a part of the RTCPeerConnection API. WebRTC app developers can choose whatever messaging protocol they prefer, such as SIP or XMPP, and any appropriate duplex (two-way) communication channel. In our test, we use WebSocket in our signaling server. Once the signaling process has completed successfully, data can be streamed directly peer to peer. Streaming is the job of RTCPeerConnection API. Audio and video streams are transported by SRTP over UDP, DTLS encryption is used in between.

WebRTC Security

In WebRTC, Signalling traffic protection between each browser and the server taking care by HTTPS and media traffic protection between browsers is offered by DTLS-SRTP but there is no end to end authentication for WebRTC. Unlike the standard public key infrastructure mechanism which is used in clientserver transection by HTTPS. In addition, web has no universal user credential system. But WebRTC controls end-to-end authentication in layers. First, the DTLS-SRTP layer associates the remote party with a public key. Then, the

identity layer maps that public key to a scoped identifier by way of JavaScript-based IdPs (Identity providers). At the end, the browser user interface can map the authenticated identifier to rich identity information from sources such as a users address book [Tho14]. Hence WebRTC provides better end to end security architecture compare to other real-time communication architecture.

2.2.3 HTTP/2.0

HTTP2.0 development was started in 2012 based on SPDY specifications which were developed at Google in mid-2009. HTTP/2.0 is a combination of RFC 7540 (HTTP/2.0) and RFC 7541(HPACK Header Compression). SPDY is not HTTP/2.0 but it was aimed to reduce the Page Load Time up to 50% compared to HTTP1.1 and make more efficient use of the underlying TCP connection by introducing new binary framing layer to enable request and response multiplexing, prioritization, and to minimize and eliminate unnecessary network latency. Similarly, HTTP/2.0 reduces latency by enabling the full request and response multiplexing; minimizing protocol overhead via efficient compression of header fields, and adds support for request prioritization and server push techniques [Gri13]. HTTP 2.0 doesnt change the semantics of HTTP1.1 in anyway. The core concepts, such as HTTP1.1 methods, status codes, URI and header fields, remain same.

Difference in HTTP and HTTP/2.0

- Multiplexed request - There is no limits to the number of concurrent request that can be sent over a single HTTP/2.0 connection.
- Prioritized requests - clients can request specific resources to be delivered first, this avoids congestion of network path with not useful resources when high priority request is there.
- Compressed Header - HTTP/2.0 uses hpack header compression. The only changed information in header or metadata will be sent for the same request so there is no overhead of header data over small bandwidth.
- Server pushed streams - this enables server to push the data to the client without any requests.

This protocol has shown the advantages in terrestrial network and it is used by Facebook, Twitter and Google servers as well [a14]. However HTTP2.0 only supports TCP. Unlike HTTP1.1, HTTP2.0 doesnt support WebSocket. It is compatible to old technologies like XHR (XMLHttpRequest) and SSE(Server Sent Event).

HTTP/2.0 Security

However, HTTP2.0 is vulnerable to four high-profile attacks like slow read (identical to DDoS attack), Hpack-bomb, Dependency cycle attack and stream multiplexing abuse [Imp]. The new mechanism of HTTP/2.0 provides network optimization but they also provide the loop holes for the attacks. Current version of HTTP2.0 is not useful for time sensitive data but it can be used for RPC (remote procedure call). The combination of (for ex. HTTP/2.0 over UDP or WebSocket over HTTP/2.0) would be great choice for future Industrial application.

2.3 Classification of Automation Applications with respect to Latency

As per IEC 62657-1 standard, Classification of Automation applications based on latency is shown below.

Process automation

- Time constants 100 ms-minutes
- Data rate 100 kbit/s (few bytes per packet)
- Periodic & event based data

Factory automation 1s -1min cycles

- I/O interval few ms (few bytes per packet)

- Low jitter (few microseconds timing accuracy)
- Periodic and event based data

Application	**Time in Second (S)**
Uncritical Automation, Enterprise System	20
Automation management, e.g manufacturing, factory automation	2
General Automation, e.g Process automation, power plants	0.2
Time-critical Automation, e.g Synchronized drives	0.002

Tabelle 1: Industrial applications and their latency

2.4 Web Protocolls used by OPC UA

OPC UA (OPC Unified architecture) is a well known interface in industry realizing interoperability in connecting machines and HMIs from different vendors using client server communication. It was standardized in 2010 as IEC 62541. The transport layer of OPC UA (V1.02) offers two possibilities for communication: HTTPS 1.1 based on TCP or pure TCP with an additional security layer. On top of HTTPS 1.1 there are also two choises: SOAP based web services or an optimzed OPC UA binary [U.S16]. To speed up communication the OPC foundation will introduce in 2017 UDP in combination with Pub Sub mechanisms based on AMQP. To get a better realtime approach for localy connected devices UDP Multicast will be used in combination with Time Senstive Network (TSN) for controller to controller communication [M.D16]. The roadmap of the opc foundation doesnt include any actions to integrate current web protocols. However J.Schmittt et.al. from ABB Research Center designed a prototype using websocket and XMPP based on OPC UA to provide asynchronous as well as bidirectional communication [et.14]. They could show that performance of OPC UA using websocket is about six times better than using HTTPS 1.1.

3 Test Bench

The underlying communication serves as the backbone of any dynamic system. The system needs to perform various tasks, each with its own restrictions. Selecting a particular type of communication depending on data-transfer requirements, might become a tedious task especially when designing with various platforms under consideration. We have created a TestBench to compare web browser based communication using protocols viz. WebSocket, HTTP/2.0 & Web-RTC. The TestBench server can be deployed within any network using NodeJS. The networked devices capable of browser based communication can then access this server from within the network. Various test cases for different kinds of data transfers can be carried out from these platforms. This can help us in analyzing the performance of that hardware platform with a particular communication protocol. These protocols are among the ones widely used in data communication. This Test Bench serves as a comprehensive benchmarking tool for communication parameters like latency, reliability, security and ease of deployment. In this paper, we use our Test Bench for analyzing the performance of these communication protocols with simple text data transfer. The analysis can be extended with other type of data, and is out of the scope of this paper. WebRTC has been divided into two types of communications. Its data channel has been configured to operate as Reliable (R) and Unreliable separately.

3.1 Test Bench Overview

The TestBench is built on NodeJS owing to its non-blocking asynchronous nature. NodeJS allows us to build highly scalable network applications and is capable of handling a huge number of simultaneous connections with high throughput. Test Bench Architecture consists of NodeJS servers running simultaneously for each individual communication protocol thus isolating them from each other. A common Monitor server takes care of the client requests and database management. These servers listening on different ports are responsible for processing incoming data, logging data as well as handling the database operations. Fig.3 gives us a structu-

ral overview of the Test Bed server. A client connects to the server using a web-browser. A fully functional TestBench HTML5 Application is provided to the client.

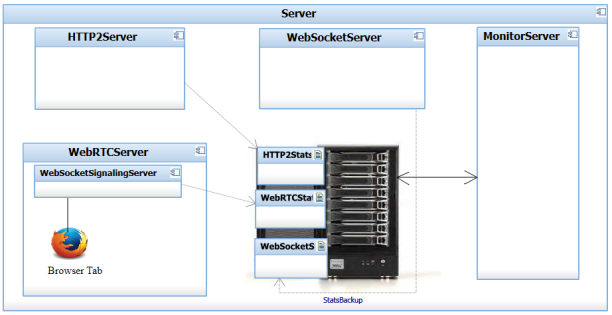

Abbildung 3: Test Bench Architecture

This application provides the user with various settings to test a particular communication individually from the device to the Test Bench Server. The users can have the options to carry out any of the specified communication with the Server. For testing purposes, the Server echo's returns the data received from client. The data transfer is monitored by the application to analyze quality of ongoing communication. At the end of communication, in this case after successful pings to echo server, the entire communication statistics are logged to the Monitor Server in its local database as shown in fig 3. Within the application, user can view these communication statistics and quality of Data transfer. The Roundtrip Time (RTT) is plotted against every ping in the application. Table 2 lists the parameters being monitored for a particular data transfer/ping.

Network	delay/bytes Sent/bytes Received
Application	delay/pings Sent/pings Received/RTT

Tabelle 2: Measurement Parameter

As this statistics data is being continuously logged by the Monitor server, it is possible for user to initiate a communication from their device and compare these results for different protocols. The RTT results are plotted on the same graph to compare results from all protocols done using the selected protocols. For every communication, a separate browser tab is being utilized at client end to avoid any interference with each other during communication. The Working model of the test bench has been demonstrated in Fig. 4 and 5.

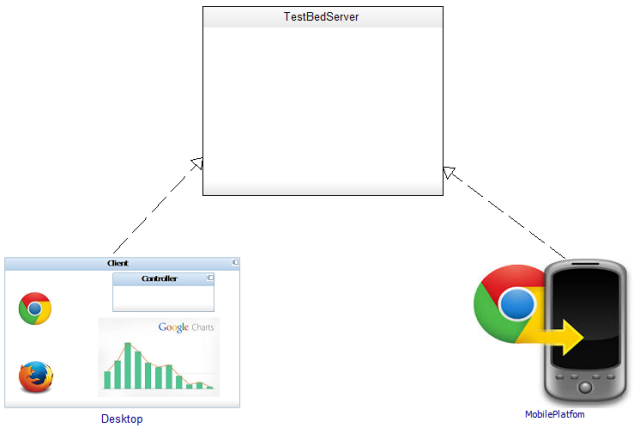

Abbildung 4: Test Bench Working Model

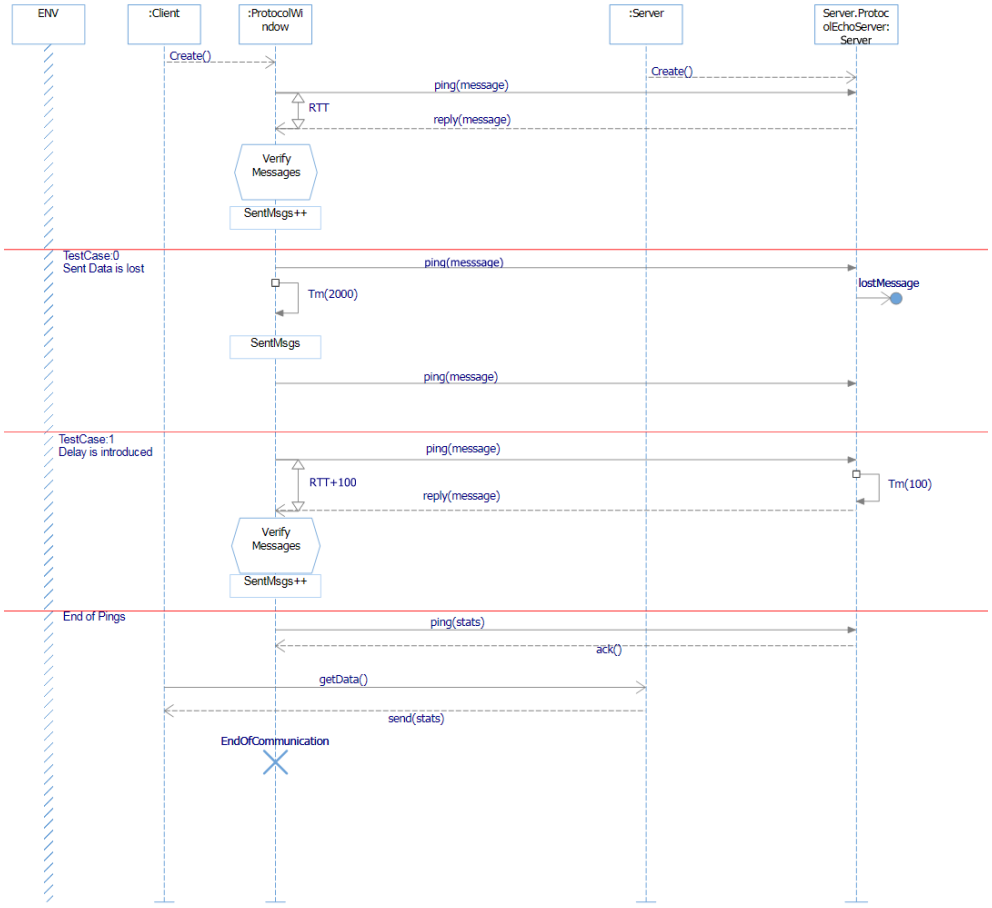

Abbildung 5: TestBench Operation

3.1.1 Test Bench Operation

The TestBench Server is to be hosted within the network. TestBench Operation and usage has been shown in Fig. 5. In this figure, the web-browser is shown as the Client object. When user wants to initiate communication for the selected protocol, it is to be done from Client Control Window. The client object creates a new Protocol window and redirects the user to the selected protocols communication web-page. Then, the communication takes place between Client side Protocol Window and TestBench Protocol Server. On successfully completing the selected number of ping requests, the communication statistics data is logged into TestBench Monitor Server. The communication statistics can be observed from within the Client Control Window. The TestBench also supports testing for mobile platform with Control window accessible from Android Chrome or Mozilla browser.

3.1.2 Performance Analysis

Using the TestBench we have performed benchmarking for WebRTC(Reliable), WebRTC(Unreliable), WebSocket, HTTP/2.0. A string data comprising of '0123456789' was transferred using each of the above mentioned protocols shown in fig 6 and table 3. The results obtained are for communication between two PCs with one running TestBench Server on Nodejs platform and other acting as client on whom the benchmarking tests are to be carried out. Both machines have internet connectivity and are connected to same local LAN network. Client PC is connected to network over wifi.

Here, Table 3 shows the avg RTT performance for 100 pings of given protocols over local area network.

Protocols	WebSocket	WebRTC	WebRTC R	HTTP/2.0
Avg. RTT in ms	4.1	1.54	1.61	6.1

Tabelle 3: Statistics for Chrome Browser (for 100 pings)

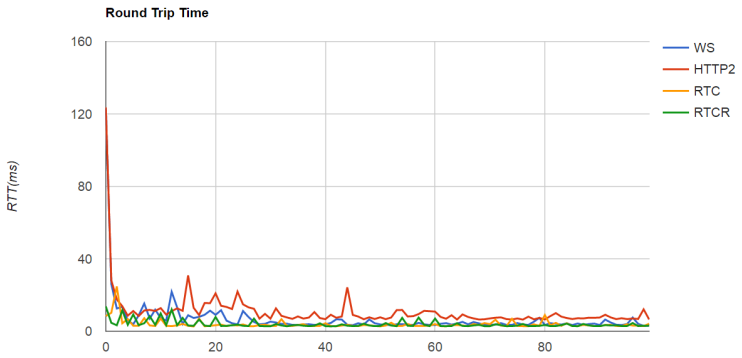

Abbildung 6: RTT comparison for chrome browser

3.1.3 Comparison of RTT in different browser for web protocols

To account for any processing overhead introduced at client end by the browser, a comparison between results obtained with Mozilla and Chrome browser are shown in Fig. 7.

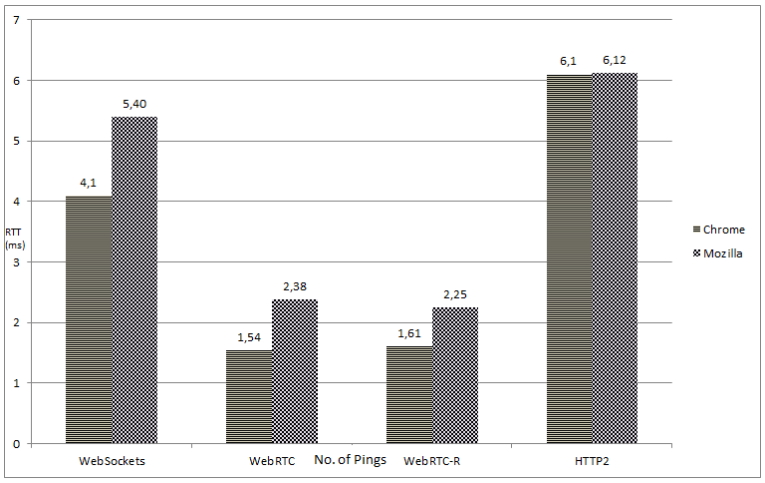

Abbildung 7: RTT comparison for different browser(for 100 pings over LAN)

It can be seen that in similar circumstances, Chrome gives better performance compared to Mozilla for these communication technologies. Also, it should be noted that WebRTC gets appreciable performance boost when operating with chrome browser. Note: In the case of WebRTC, initial signaling is done using WebSocket with a dedicated server and port reserved for this operation. The actual data communication being measured is done between client & WebRTC enabled browser tab running on the server machine.

3.2 Testcases

The Following test cases have been executed using this test bench. These test cases merely serve to demonstrate usage of the TestBench in platform benchmarking and protocol comparison. WANem software platform [Emu]

developed by TCS has been used to simulate network events such as delay, packet loss, etc. In this case, a WANem server running on separate machine is added to the network path of the communication between Client and TestBench Server.

Testcase: 1

TestSpecs: Delay=100ms, Pings = 20

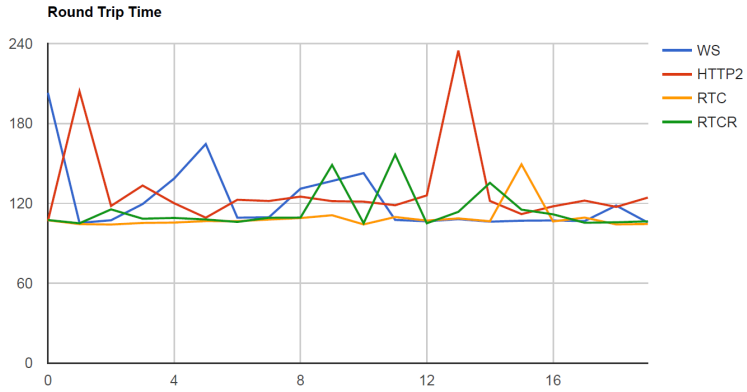

Abbildung 8: Delay=100ms & pings =20

In fig. 8, the entire graph has been shifted up by 100ms. The process was observed for duration of 20 pings.

Testcase: 2

TestSpecs: Bandwidth=2097kbps, Delay=10ms, PktLoss=2%, Jitter=10ms, Client Connection-wifi, Pings=100

As seen in Fig.9, except for HTTP2, all other protocols give steady performance.

Testcase: 3

TestSpecs: Bandwidth=2097kbps, Delay=10ms, PktLoss=2%, Jitter=10ms, Client Connection-wifi, Pings=1000

In order to observe the performance over longer duration, we select a similar test case but with larger ping requests. We consider these parameters for average packet loss [KG] on lan and wlan for small network but there are several other configurations for these parameters available in WAN emulator so one can set those parameters according to their requirement. Fig. 10 shows that the aberrations in HTTP/2.0 performance are more pronounced. WebSocket performance results very often show these departures from normal operation. On the other hand, WebRTC continues to give steady high performance i.e with low RTT timings.

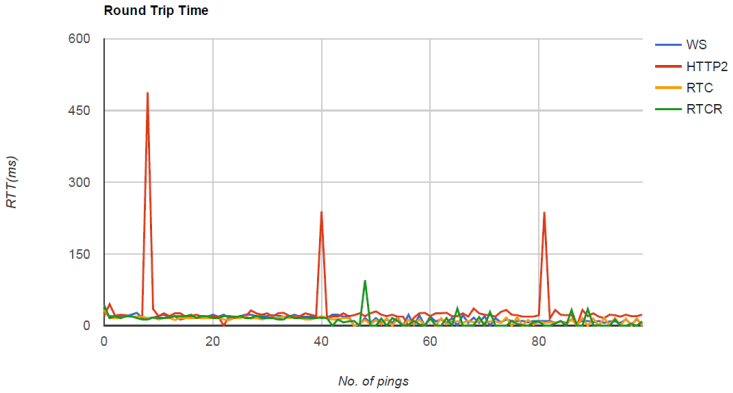

Abbildung 9: TestCase2 in chrome

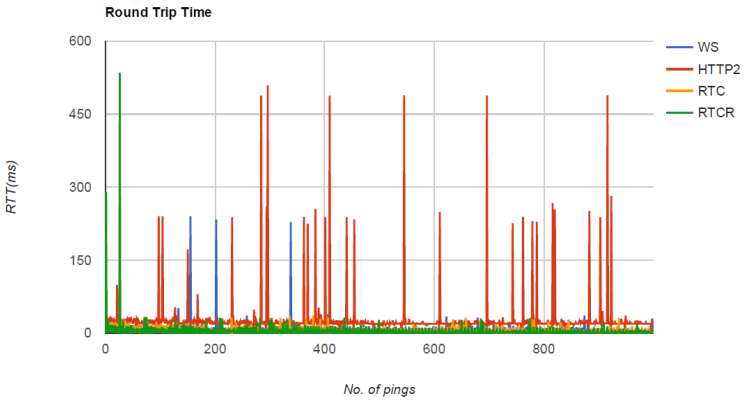

Abbildung 10: TestCase3 in chrome

We can evaluate that the results obtained are specific for the platform used for testing and network conditions at the time of testing. From the preliminary results obtained, we can see that in case of normal operation i.e no WAN emulation with normal network traffic, WebRTC gave relatively better performance with low RoundTrip Time(RTT) time. The influence of different browsers on performance is also shown in Fig.11.

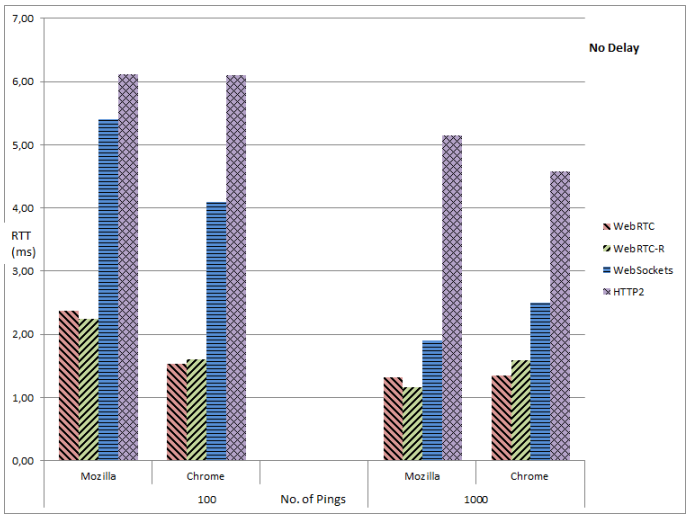

Abbildung 11: Comparison-Normal Operation

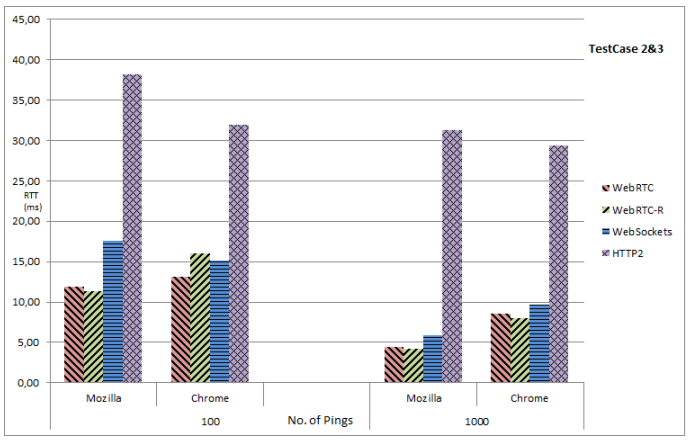

Abbildung 12: Comparison-WAN emulation

In case of WAN emulation, results obtained from the test bench have been summarized in Fig. 12. The network conditions have led to increased RTT. Although, there has been considerable increase in RTT timing for HTTP/2.0 protocol. WebRTC on other hand continues to give relatively better performance. Mozilla browser produced better RTT results over longer duration of time i.e 1000 pings.

3.3 WebRTC Analysis

WebRTC yields better results in most of the scenarios. Taking a closer look at the Mozilla webRTC internals log, we can demonstrate the inner workings of WebRTC which lead to WebRTCs superior performance. WebRTC communicates using network paths derived from use of Interactive Connectivity Establishment (ICE) candidates. Trickle ICE feature in WebRTC enables discovery of new candidates during ongoing communication. ICE candidates are constantly being discovered during ICE hole punching into a network. These candidates are exchanged between the communicating peers via signaling server. Candidate pairs are constantly tested for network path performance in order to maintain an effective communication and in almost all cases to keep session alive.

local candidate	**Remote candidate**	**ICE stats**
192.168.178.20:64826/udp(host)	192.168.178.22:53028/udp(host)	Succeeded
192.168.178.20:64826/udp(host)	188.174.79.187:53028/udp(serverreflexive)	failed
188.174.79.187:64826/udp(serverreflexive)		
2001:05ef5:79fd:1c78:cc2:4351:b044:63827/udp(host)		

Tabelle 4: ICE stats

Table 4 shows the ICE stats obtained from Mozilla log at the server end. The local & remote ICE candidate pairs were tested and the result is displayed in ICE stats column. Server machine was able to establish successful communication using first candidate pair and hence its ICE State was changed to succeeded. The Session Description Protocol (sdp) at Test Bench server end are shown in Fig. 13 & 14 below.

Local SDP

v=0
o=mozilla...THIS_IS_SDPARTA-49.0.1 1867711306094712400 0 IN IP4 0.0.0.0
s=-
t=0 0
a=sendrecv
a=fingerprint:sha-256
E1:F8:93:03:05:13:13:D1:CE:D9:F8:9B:3E:72:C4:2A:E1:88:10:80:4B:CE:BE:74:F1:26:87:51:93:B8:77:A6
a=ice-options:trickle
a=msid-semantic:WMS *
m=application 63826 DTLS/SCTP 5000
c=IN IP4 188.174.79.187
a=candidate:0 1 UDP 2122252543 192.168.178.20 63826 typ host
a=candidate:2 1 UDP 2122187007 2001:0:5ef5:79fd:1c78:cc2:4351:b044 63827 typ host
a=candidate:1 1 UDP 1686052863 188.174.79.187 63826 typ srflx raddr 192.168.178.20 rport 63826
a=sendrecv
a=end-of-candidates
a=ice-pwd:5d24973dfba5f71af77f20adec6099db
a=ice-ufrag:0ff9a1bb
a=mid:data
a=sctpmap:5000 webrtc-datachannel 256

Abbildung 13: Local Session Description Protocol

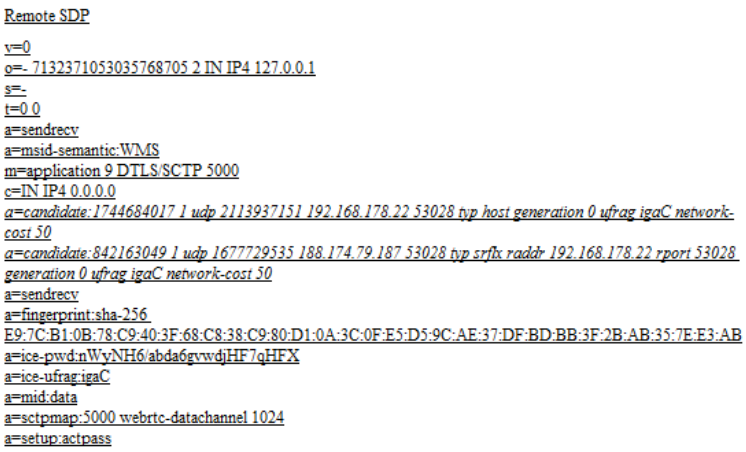

Abbildung 14: Remote Session Description Protocol

It can be seen from sdp for the server machine that it has discovered three ICE candidates for its local connection and also has acquired client ICE candidates as shown in remote sdp. Using these test cases, usage of the Test Bench in analyzing the network protocols has been successfully demonstrated. The results obtained from the Test Bench show us the viability of a platform in utilizing particular protocol for communication and these results should not be generalized. Users are encouraged to carry out test cases relevant to their requirements to find a suitable protocol for task at hand. In field of communication, often tradeoffs among different desirable parameters are to be made when considering speed of communication or reliability. Test Bench merely serves as a means to equip its user with sufficient data and aid him in choosing suitable communication protocol for a particular platform.

4 Conclusion & Future Scope

An objective comparison and analysis of communication technologies-WebRTC, WebSocket, and HTTP/2.0 has been successfully performed. HTTP/2.0 is providing binary and compressed bidirectional communication with some security aspects but it lags in performance in terms of network latency, while WebSocket is the easiest to deploy amongst all web protocols but it is based on TCP. WebRTC is interesting for real-time communication between multiple peers over UDP connection which has more advantage to scaling the cloud. As WebRTC API supports real-time video and audio streaming, it can be the useful choice for virtual reality concepts in industry 4.0. The optimization of webprotocols for industry 4.0 should be the combination of WebRTC and WebSocket. Where data communication would be taken placed over WebRTC secure channel and WebSocket would be the part of signaling mechanism and it can also provide a backup reliability for WebRTC channels if WebRTC communication is interrupted .

Literatur

[a14] Google Developer Webpage; 'Make the Web faster';https://developers.google.com/speed.

[a7] *Industrial Communication Technology Handbook, Second Edition*.

[BM10] S. Hennig; A. Braune; und M.Damm.JasUA. A JavaScript Stack enabling Web browsers to support OPC Unified Architectureâs Binary mapping natively. In Emerging Technologies and Factory Automation (ETFA). Seiten 1–4, sep 2010.

[BR15] Bosch-Rexroth. WebConnector: Automatisierungs- und Web-Umgebung einfach verbinden. Seite 50, 2015.

[Emu] WAN Emulator;http://wanem.sourceforge.net/.

[et.14] J.Schmitt et.al. Cloud-enabled Automation Systems using OPC UA. 7-8.2014.

[Eva11] Evans.Z. *Web Server Technology in Automation*. Siemens Industry, 2011.

[FP16] Richard Langmann; Leandro F.Rojas-Pena;. A PLC as an Industry 4.0 component. DOI: 10.1109/REV.2016.7444433:10–15, 2016.

[Gmb] Beckhoff Automation GmbH. From Sensor to IT Enterprise - Big Data & Analytics in the cloud.

[Gri13] Ilya Grigorik. *High-Performance Browser Networking*. O'reilly, sep 2013.

[Hau11] D. Pfisterer; K. Romer; D. Bimschas; O. Kleine; R. Mietz; C. Truong; H. Hasemann; A. Kroller; M. Pagel; M. Hauswirth. A Migration Approach towards a SOA-based Next Generation Process Control and Monitoring. 11:40–48, 2011.

[Imp] Imperva. HTTP/2:in-depth analysis of the top four flaws of the next generation webprotocol.

[ISO08] ISO/IEC. IEC 62541. 2008.

[Kap01] G. Kaplan. Ethernet's winning ways. Spectrum. Jgg. 38, Seiten 113–115. IEEE, Jan 2001.

[KG] Jianxia Ning; Shailendra Singh ; Konstantinos Pelechrinis; Bin Liu; Srikanth V. Krishnamurthy; und Ramesh Govindan. Forensic Analysis of Packet Losses in Wireless Networks. Seite 1.

[Kli05] R.B Klindt, C.J.Baker. Interface to a programmable logic controller. 8 feb 2005.

[Lom] Andrew Lombardi. *WebSocket lightweight client server communications*. O'reilly.

[M.D16] M.Damm. OPC UA Update Roadmap. June 2016.

[Tho14] Richard L. Barnes; Martin Thomson. Browser-to-Browser Security Assurances for WebRTC. 18:11–17, 2014.

[U.S16] U.Steinkraus. OPC UA Technical Introduction. June 2016.