3rd IFAC Symposium on Telematics Applications
The International Federation of Automatic Control
November 11-13, 2013. Seoul, Korea

Analysis of WebSockets as the New Age Protocol for Remote Robot Tele-operation

Lakshminarasimhan Srinivasan * **Julian Scharnagl** *
Klaus Schilling *

* *Department of Robotics and Telematics, University of Wuerzburg, 97074 Wuerzburg, Bavaria, Germany.*
E-mail: (srinivasan, scharnagl, schi)@informatik.uni-wuerzburg.de.

Abstract: This paper addresses the replacement of acknowledgement based protocols for teleoperation. The initial sections compare the currently used protocols like HTTP and TCP to WebSockets and examine the structural difference of using one over the other, for a real-time teleoperation scenario. A detailed analysis of HTTP and WebSockets is performed using live teleoperation by multiple users performing temporally seperated tests under variably delayed network conditions. The existing teleoperation laboratory at the University of Wuerzburg was used as the test bed to compare these protocols and final teleoperation results in terms of bandwidth, frame-rate, navigation efficiency and situational awareness are presented and discussed.

Keywords: Tele-operation, WebSockets, Low-bandwidth robotics, Remote robotics, Network technology adaptation, Teleoperation-Protocol;

1. INTRODUCTION

Protocols and user interafces play a significant role in teleoperating a robot. Significant strides have been made in the developemt of user interfaces like augmented reality (Chellali et al. (2013)), mixed-reality (Grasset et al. (2011)), 3-D (Ferland et al. (2009)), immersion (Labonte et al. (2010)) and haptic feedback. Many teleassistive systems have also been developed and implemented (Michaud et al. (2010), Driewer et al. (2007)) but in each case, the communication protocol used has been traditional, simple HTTP or propreitary applets using TCP. In those cases where specific protocols were developed, they have largely gone un-maintained or remain stuck in the era when they were developed. The advent of WebSockets promises to change all that and the following sections will elaborate on the protocol, its comparitive network performance and its specific performance as a robot teleoperation protocol.

2. TRADITIONAL PROTOCOLS

With the need for bidirectional real-time communication in web browsers due to the development of the Web 2.0 during the last years, different approaches have been developed to provide this functionality. These methods, used in modern teleoperation systems as well, are workarounds to use a protocol for something it was not designed for and thus show several disadvantages. Development of the WebSocket protocol adresses especially this need. The following sub-sections focus on the traditional methods and their attributes, describe their functionality and drawbacks.

2.1 HTTP (Polling, Long-Polling, Streaming)

The Hypertext Transfer Protocol (HTTP) is the foundation of all communication in the World Wide Web. It is layered onto a Transmission Control Protocol (TCP) connection and follows the request-response principle: a web browser acting as client requests a web page from a web server and the server responds with the requested information after which the connection is closed. Since HTTP is half duplex, the server is not able to initiate a data transmission to a client as long as it has not been specifically asked for. This protocol is not sufficient for current Web applications that require fast full-duplex communication between server and client. Workarounds based on HTTP that have been developed to extend its functionality, which are also used for most robot teleoperation functions at the moment, will be discussed below.

Polling The function principle of the polling approach, also known as AJAX (Asynchronous JavaScript and XML), can be described as follows: The client sends repeated, regularly timed HTTP requests in short time intervals. Whenever a request arrives, the server immediately responds and can enclose data in the response to the client. After the response, the connection is closed and the client starts a new request. Figure 1 illustrates this principle. This approach shows a few major disadvantages. First every request/response contains the full HTTP header and thus contains a huge overhead. Second to reduce the latency the client has to send requests as often as possible in as small intervals as possible. This is a good approach, if the intervals of new data available on the server are known, because then the polling requests can be synchronized with that (Lubbers and Greco (2013)). But since in many cases the server may collect and provide new data in irregular intervals, there might be many response transmissions containing no new data. Such unused transmissions generate another huge overhead (Hmlinen (2011)). Even if the polling requests are sent in very short intervals the server might still have to wait a short time after getting new data before the HTTP request arrives. This introduces a delay to the server-client communication.

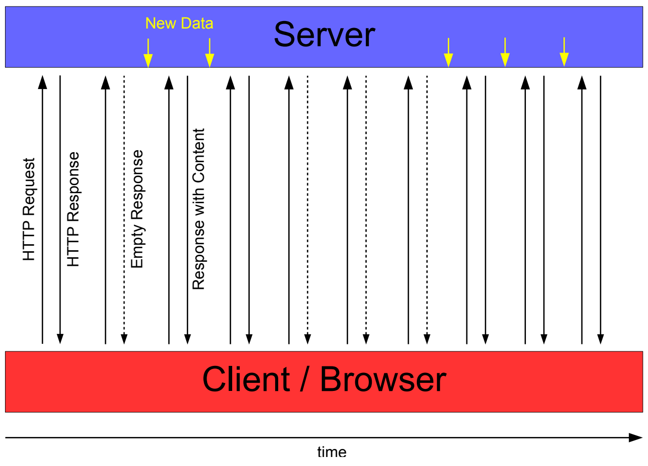

Fig. 1. Working principle and flow of HTTP Polling

Long-Polling Long-polling or also known as Comet uses a pending HTTP request (pending-POST or hanging-GET) to keep a connection between client and server alive. To establish a connection, the web browser sends a HTTP request, but contrary to polling, the server does not immediately respond to it and holds the pending request for a while. In the meantime it can wait for new information to be available and then respond to the request providing the new information immediately to the client. Figure 2 illustrates this functionality. This is a big advantage compared to polling, since the number of HTTP requests and responses is reduced to the number of data updates on the server (in best case). No unnecessary requests are performed. Still each request contains a timeout, so if there is no response from the server during the set period, the connection is closed and the client initiates a new request.

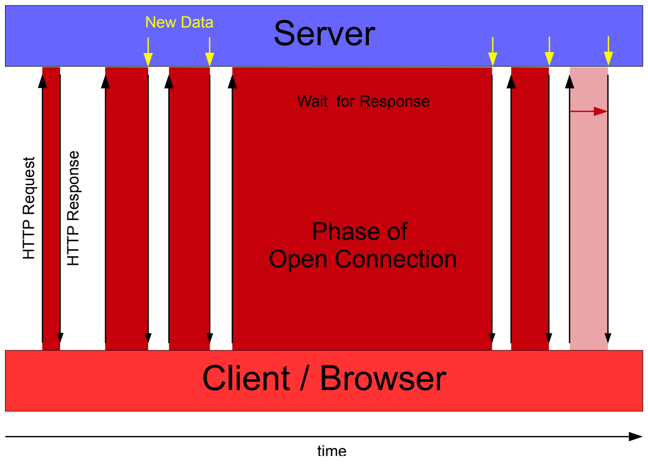

Fig. 2. Working principle and flow of HTTP Long-Polling

Although long-polling is in many cases an improvement to conventional polling, it still shows several disadvantages. If the number of data updates on the server and so message volume to the client is high, it shows the same network behavior as polling, since the client has to reconnect constantly to the server to get the new information (Wang et al. (2013)). In this case there is no performance improvement. It can even cause problems since the maximum number of HTTP requests is not fixed as it is in polling, but can increase to a continuous loop of requests and responses that may decrease the performance of both systems (Lubbers and Greco (2013)). The frequency of requests/responses in such a case is only limited by the network transit and processing time of the server and client. This defines also the maximal latency of the system. When the server gets two data updates at almost the same time, the second data transmission to the client has a delay of three times the network transit time (response containing first data update, client request, response containing second data update; Loreto et al. (2011)). Since there are still many HTTP requests and responses involved in the communication flow, they produce an additional overhead just as polling by sending unnecessary information (Loreto et al. (2011)). Another problem reducing performance arises from the fact that during a hanging request the client has to open a second connection and initiate a second HTTP request to provide information to the server. In addition to that the lack of standard implementations for long-polling affects the usability of this approach and may allow security problems to develop more easily.

Streaming The streaming approach also uses a hanging HTTP request to repeatedly send new data to the client. When the server receives the HTTP request from the client, it does not answer it, but it keeps the connection open. Whenever the server gets new data, it sends it as part of the HTTP response to the client, but it never completes the response. So the connection is kept alive and the server can always append new data to the response. Figure 3 illustrates this behavior. Streaming represents a further improvement compared to polling and long-polling, because it shows less overhead and delay introduced by the repeated HTTP requests and responses. Nevertheless the infinitely open request introduces other problems. Proxies or firewalls on the path between server and client can interfere with the transfer of the streamed response packages. They may buffer the packages, while waiting for the end of the request. This leads to an increased latency in message delivery (Loreto et al. (2011)). Thus many streaming solutions fall back to long-polling in that case (Lubbers and Greco (2013)). In addition to that streaming shows the same constraint on client-to-server transmission as long-polling. During the lifetime of the hanging HTTP request, the client needs to initiate a second request to send information to the server.

While these methods provide real-time (or almost real-time) communication they show disadvantages in terms of bandwidth consumption, latency and usability. These drawbacks mainly arise from the large headers used and the request-response model of the HTTP protocol which was developed to bring static web pages from a server to a browser and not for full-duplex, real-time communication.

Applets/Applications In addition to HTTP-based techniques, browser plug-ins (e.g. Java applets, Flash or Silverlight) or stand-alone applications have been used to provide real-time communication between client and server. Their main advantage is that they can use any network protocol and are not restricted to HTTP. Nev-

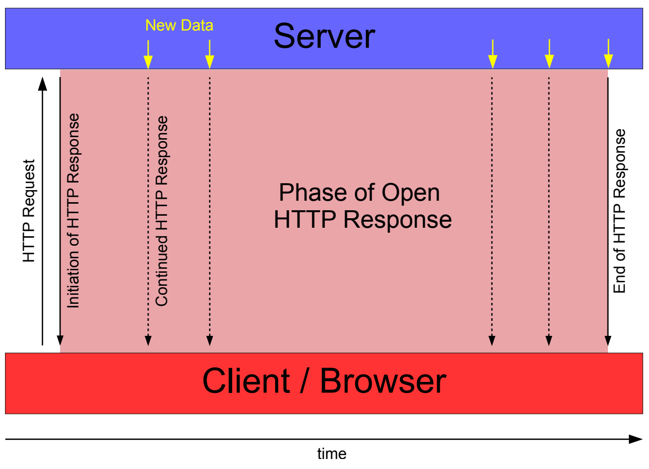

Fig. 3. Working principle and flow of HTTP Streaming

ertheless the commonly used protocol for internet applications is TCP. TCP shows several advantages compared to HTTP. An open connection is not closed after a transmission. It provides bi-directional communication, so both sides can send data at any time and it includes only small headers compared to HTTP so it produces only very little overhead. Still this approach shows several disadvantages. Applications have to be distributed together with plug-ins. Some browsers or operating systems may not support certain applications or plug-ins (e.g. there is no Java or Flash support in iOS or Android) (Gutwin et al. (2011)). Both issues are not present in pure browser-based solutions like HTTP or WebSocket. In addition to that, using a standard instead of a proprietary plug-in or application reduces the dependency on third-party software and their development. Particularly outdated versions of Flash and Java plug-ins are responsible for a huge amount of malware infections and attacks on browsers (Gutwin et al. (2011), Monsted (2013)). Using TCP provides more performance compared to HTTP, but it does not provide a standardized security mechanism like SSL in HTTPS or WebSockets. A developer has to create its own security solution and has to maintain it. Furthermore the browser-independent communication of an applet or application opens an additional port in the clients firewall that allows intruders to attack beneath the security model of a browser that is protecting other browser-based solutions.

3. WEBSOCKETS

WebSockets combine the strengths of all previously described solutions, displaying the performance of a TCP connection as used in a propreitary application as well as the safety and flexibility of a web standard used in a browser like the HTTP-based solutions. Usability is a result of an easy-to-use WebSocket API as well as the fact that both protocol and API are standardized. The fact that a security model is included in the WebSocket protocol standard together with the lack of additional weak points (like plug-ins or other third-party software) in a pure browser-based web application using WebSocket leads to the significant security enhancements. The performance improvements especially are lower latency, less network traffic or bandwidth consumption and less CPU usage in the server and client.

3.1 Principle and Functioning of the WebSocket Protocol

The WebSocket protocol was created to provide an effective and standardized solution for full-duplex / bidirectional communication between a web browser and a web server and to replace the workarounds that are currently widely used. It was designed for the use in web browsers, but it can also be used for other purposes. It was adopted by the Internet Engineering Task Force (IETF) in December 2011 as Request for Comments (RFC) 6455 (Fette and Melnikov (2011)) and is supported by all major web browsers.

A WebSocket connection is initialized from a HTTP request, but the connection itself is not based on HTTP. It is layered over the TCP. The connection is established by an opening handshake initiated by the client that is a simple HTTP request. It includes the special header “Upgrade: websocket” indicating that the client would like to upgrade the connection to the WebSocket protocol. An example HTTP request (from Wang et al. (2013)) to open a WebSocket connection is as follows:

```
GET /echo HTTP/1.1
Host: echo.websocket.org
Upgrade: websocket
Connection: Upgrade
Origin: http://www.websocket.org
Sec-WebSocket-Key: 7+C600xYybOv2zmJ
69RQsw==
Sec-WebSocket-Version: 13
```

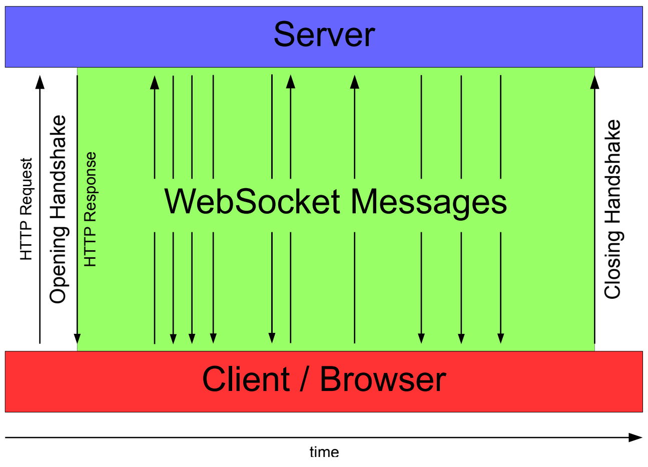

Fig. 4. Working principle and flow of WebSocket connection

If the server supports WebSocket and is willing to upgrade the connection, it sends a response (from Wang et al. (2013)) like below:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: fYoqiH14DgI+5 ylEMw
M2sOLzOi0
Date: Wed, 12 Jun 2013 01:33:37 GMT
Server: Kaazing Gateway
```

If a server does not support WebSockets, it still is able to understand the handshake, since it is a HTTP header, and can give a negative response to it. After the opening handshake server and client switch from HTTP to the WebSocket protocol. The connection remains open until a closing handshake is initiated. Both sides can do this and they can also provide a numerical code containing a reason for closing the connection. As long as the connection remains open, both sides can send messages at any time, even simultaneously. In contrast to HTTP, if there is a change on the server, it can send the changed information immediately to the client without the need for a request from the client. Figure 4 shows the functioning principle and flow of a WebSocket connection. After opening the connection both sides can send messages at any time. No request or response is required.

Messages can contain UTF-8 encoded text or binary data. The message header was designed to be as short as possible to avoid unnecessary overhead. This reduces the bandwidth consumption significantly. It is possible to split up a message in several frames and send them sequentially. This is especially useful for large messages or messages that are only partially available at the beginning of the sending (streaming). WebSocket connections can be established unencrypted or encrypted using Transport Layer Security (TLS) exactly the same way as HTTP connections. They are responsive to the uniform resource identifier (URI) *ws* and *wss* (following the same scheme as *http/https*) and use the same ports as *http* and *https* (80 and 443), which makes it particularly easy to handle for firewalls and proxies, since they do not require any changes to those existing for HTTP.

The WebSocket Application Programming Interface (API) defines the interface to use WebSocket connections in web applications. The World Wide Web Consortium (W3C) proposed it as standard in September 2013 as W3C candidate recommendation (Hickson (2012)). The API is completely event driven and handles the following four events:

- *onopen*: connection opened
- *onmessage*: message received
- *onerror*: error occurred
- *onclose*: connection closed

Because of its simplicity this API is very easy to use, while still supporting the full functionality. The code snippet in Listing 1 shows a typical implementation of the WebSocket API using JavaScript. Although the protocol itself supports additional functions like sending frames and handling incoming frames separately, this is not mandatory in the standardized WebSocket API.

Preliminary performance evaluation of the WebSocket protocol performed by other research groups are the first steps towards determining the usefullness of this protocol for remote robot teleoperation. Zhang et al. (2013) evaluated the usage of WebSocket for a medical real-time monitoring system using a smartphone browser as client and proved the applicability for low-volume, continuous, real-time traffic. Swamy and Mahadevan (2011) investigated WebSockets for Wireless sensor Networks. They especially showed the reduction of network traffic due to significantly smaller WebSocket headers compared to HTTP headers for large numbers of clients (e.g. 1,000 clients receive 1 message per second leads to a WebSocket header traffic of 0.230 % of the corresponding HTTP header volume).

Listing 1: WebSocket API example

```
var ws = new
    WebSocket("ws://127.0.0.1:80");

ws.onopen = function(e) {
    // WebSocket connection opened
};

ws.onerror = function(e) {
    // WebSocket error occurred
    console.log('error: ' + e.data);
};

ws.onclose = function(e) {
    // WebSocket connection closed
);

ws.onmessage = function(e) {
    // WebSocket message received
    console.log('message: ' + e.data);
);
```

Gutwin et al. (2011) analyzed web-based networking technologies for groupware in the browser. They performed measurements on message rates for different network speeds comparing HTTP long-polling, HTTP streaming, a Java applet and WebSocket. Their results show that WebSocket allows much higher message rates than the others (e.g. server to browser message rate for a message size of 500 bytes in a WAN: WebSocket 5113, applet 1245, streaming 328, long-polling 20 messages / sec.). Puranik et al. (2013) compared an AJAX server with a WebSocket server for real-time monitoring. Their results show that a WebSocket server consumes 50 % less network bandwidth, that a WebSocket client consumes memory at a constant, not at an increasing rate and that the WebSocket protocol is able to send up to 215 % more data samples while consuming the same amount of network bandwidth. Liu and Sun (2012) investigated web real-time communication comparing WebSocket and HTTP long-polling. They analyzed traffic and delay for a certain task (sorting columns and rows of a table and communicating the result) showing 10 times higher performance of WebSocket in terms of used transmitted bytes and time required. On the back of these strong results, our group decided to perform tests on teleoperation using WebSockets and the following section details the tests and their results.

4. TESTS AND RESULTS

In terms of teleoperation, four important aspects were tested during teleoperation of a mobile robot situated in our laboratory: bandwidth consumption during teleoperation, video refresh rate, navigation times and SAGAT scores (explained in the following subsection). The choice of these metrics is straightforward- bandwidth consumption and visual feedback rates are directly related to the performance of the system, and with all other parameters remaining the same, also reflect the performance of the indivudual protocols. A comparision of navigation times and SAGAT scores displays the effect of system performance on operator performance. The use of a system is dependant on operator performance. Irrespective of how well a system by itself performs, if user performance is not satisfactory, then the system cannot and will not be used. The following subsections describe results from 6 different users teleoperating a remote robot from a spatially seperated operation center. The operators had no idea about the remote environment and only possibility for feedback was through the network.

4.1 System Performance

The first set of tests clearly indicate the advantage of using WebSockets, and follows from theory that the overhead for WebSocket is negligible when compared to the request-response mechanisms of HTTP. Figure 5 shows the measurement of bandwidth consumption at various points of time during remote teleoperation. Average consumption of 190kbps when using WebSockets compared to 450kbps for HTTP is an almost three fold reduction. Although the result is along expected lines, the amount of difference was a startling revelation of the efficiency of the new protocol.

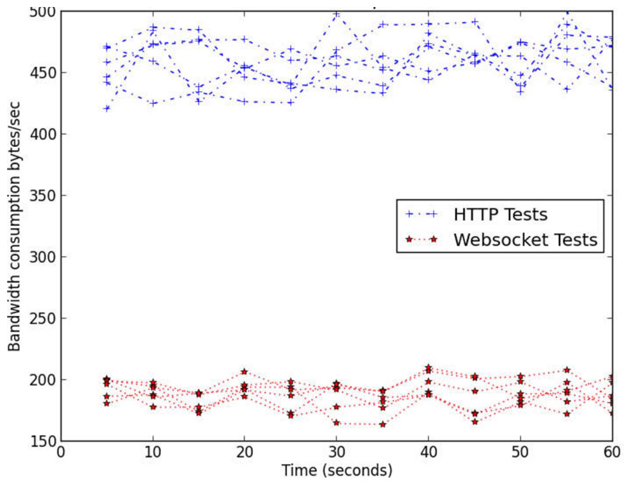

Fig. 5. Bandwidth comparison during teleoperation

Figure 6 shows the results from measuring 6 different operators performing 12 teleoperation attempts, each temporally seperated from one another. Although Figure 6 is a specific case of bandwidth consumption, it's significance lies in the fact that visual feedback is the single biggest influence on remote teleoperation and also indicates latencies in the communication system. An average client-side refresh rate of 57 ms compared to around 500 ms is again a 10 fold improvement, enabling extremely smooth visual feedback (video) from the remote site within the operator's browser.

4.2 Operator Performance

The previous section described the results that lay the foundation for operator performance. Given that the new protocol implementation provides extremely responsive user interfaces capable of being used across the Internet, user tests were performed to measure two important parameters, navigation times and SAGAT scores.

Figure 7 displays the navigation times during remote teleoperation. The efficiency of the system definitely improves operator efficiency as seen in the faster average navigation time (about 130 seconds) for teleoperation when using the WebSocket sytem as compared to the HTTP system (about 138 seconds).

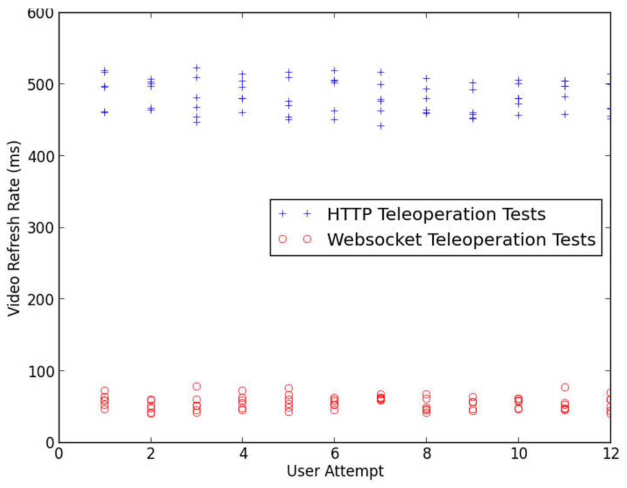

Fig. 6. Video refresh rate using HTTP and WebSocket

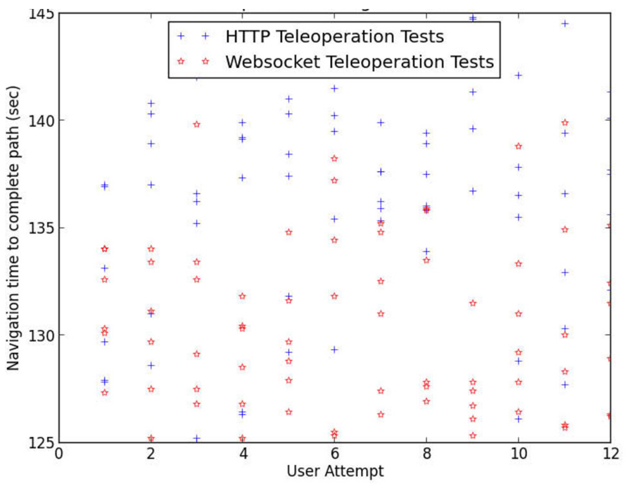

Fig. 7. Teleoperation navigation time performance

The final performance metric compared is the internationally accepted Situation Awareness Global Assessment Technique, commonly called SAGAT scores. This parameter is used to test operator situation awareness using intelligently poised questions during teleoperation and is a proven method to test user interfaces.

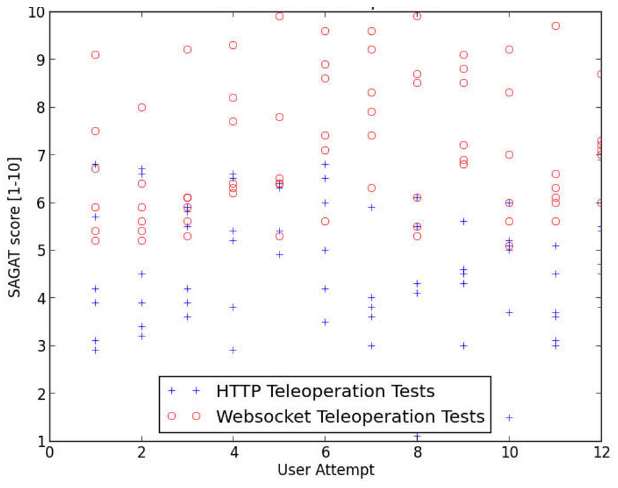

Fig. 8. Teleoperation performance using sagat tests

Figure 8 shows the SAGAT scores during each teleoperation of each user. Measured on a scale of [1-10] with 10 being the maximum possible, the average score for traditional-protocol teleoperation is around the 5 mark whereas, owing to improved video feedback and faster data aquisition from the remote site, the same user interface feels like an 8 on average on the SAGAT scale to the user perfoming teleoperation. This is a fairly important result that goes to establish the importance of providing a seamless operation using WebSockets, negating the latency effects and overhead of typical network communications that are associated with traditional teleoperation protocols.

5. CONCLUSIONS

This paper dealt with the implementation and testing of WebSockets as a teleoperation protocol and results indicate this is an extremely viable option. For typical scenarios like tele-maintenance of machines, tele-operation in manufacturing facilities using manipulators and educational courses, propreitary and hard to maintain communication protocols should definitely be replaced using the WebSocket standard. The test results presented in this paper certainly corraborate this conclusion. Although all tests here were performed in the presence of 'normal' variable network delays, an interesting extension would be to check performance metrics during a long-delay scenarios, especially useful in space operations.

ACKNOWLEDGEMENTS

The authors would like to specially thank the Virtual University of Bavaria for their constant support since 2005 and for their grant which partially funded our current implementation.

REFERENCES

Chellali, A., Milleville-Pennel, I., and Dumas, C. (2013). Influence of contextual objects on spatial interactions and viewpoints sharing in virtual environments. *Virtual Reality*, 17(1), 1–15. doi:10.1007/s10055-012-0214-5.

Driewer, F., Sauer, M., Leutert, F., and Schilling, K. (2007). A flexible framework for distributed three-dimensional models in telematic applications. *Proceeding (563) Robotics and Applications and Telematics*.

Ferland, F., Pomerleau, F., Tam, L.C., and Michaud, F. (2009). Egocentric and exocentric teleoperation interface using real-time, 3d video projection. In *Human-Robot Interaction (HRI), 2009 4th ACM/IEEE International Conference on*, 37–44.

Fette, I. and Melnikov, A. (2011). The websocket protocol. URL `http://tools.ietf.org/html/rfc6455`.

Grasset, R., Mulloni, A., Billinghurst, M., and Schmalstieg, D. (2011). *Navigation Techniques in Augmented and Mixed Reality: Crossing the Virtuality Continuum*. Springer New York. doi:10.1007/978-1-4614-0064-6_18.

Gutwin, C., Lippold, M., and Graham, T. (2011). Real-time groupware in the browser - testing the performance of web-based networking. In *In Proceedings of the ACM 2011 conference on Computer supported cooperative work, CSCW '11*, 167–176.

Hickson, I. (2012). The websocket api. W3C Candidate Recommendation.

Hmlinen, H. (2011). Html5: Websockets. Aalto University, Department of Media Technology. Helsinki, Finland.

Labonte, D., Boissy, P., and Michaud, F. (2010). Comparative analysis of 3-d robot teleoperation interfaces with novice users. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 40(5), 1331–1342. doi:10.1109/TSMCB.2009.2038357.

Liu, Q. and Sun, X. (2012). Research of web real-time communication based on web socket. *International Journal of Communications, Network and System Sciences*, 5(12), 797–801.

Loreto, S., Saint-Andre, P., Salsano, S., and Wilkins, G. (2011). Known issues and best practices for the use of long polling and streaming in bidirectional http. URL `http://www.hjp.at/doc/rfc/rfc6202.html`.

Lubbers, P. and Greco, F. (2013). A quantum leap in scalability for the web. http://www.websocket.org/quantum.html. URL `http://www.websocket.org/quantum.html`. Kaazing Corporation, Mountain View, USA.

Michaud, F., Boissy, P., Labonta, D., Briare, S., Perreault, K., Corriveau, H., Grant, A., Lauria, M., Cloutier, R., Roux, M.A., Iannuzzi, D., Royer, M.P., Ferland, F., Pomerleau, F., and Ltourneau, D. (2010). Exploratory design and evaluation of a homecare teleassistive mobile robotic system. *Mechatronics: Special Issue on Design and Control Methodologies in Telerobotics*, 20(7), 751 – 766. doi:10.1016/j.mechatronics.2010.01.010.

Monsted, J. (2013). How malware attacks on the windows platform. CSIS Security Group. URL `https://www.csis.dk/en/csis/news/3981/`. MW.

Puranik, D., Feiock, D., and Hill, J. (2013). Real-time monitoring using ajax and websockets. In *In Proceedings of the 20th Annual IEEE International Conference and Workshops on the Engineering of Computer Based Systems (ECBS)*.

Swamy, N. and Mahadevan, G. (2011). Event driven architecture using html5 web sockets for wireless sensor networks. White Paper. Planetary Scientific Research Center.

Wang, V., Salim, F., and Moskovits, P. (2013). *The Definitive Guide to HTML5 WebSocket*. Apress.

Zhang, W., Stoll, R., Stoll, N., and Thurow, K. (2013). An mhealth monitoring system for telemedicine based on websocket wireless communication. *Journal of Networks*, 8(4), 955–962.