A light-weight time protocol based on common web standards

M. Gutbrod, T. Klein, D. Sibold
Physikalisch-Technische Bundesanstalt
38116 Braunschweig, Germany
martin.gutbrod@ptb.de

***Abstract*—Distributed systems are an essential part of Industry 4.0 and IoT. In order to perform properly they depend on unambiguous time information while their stripped-down hardware prevents the use of extensive protocols and algorithms. We developed a light-weight protocol for time transmission aiming for simplicity, security and broad applicability by relying solely on common web standards. In this paper the new websocket time protocol (WST) will be presented.**

Keywords—time protocol, websocket, ntp, time transmission, WST

I. Introduction

The digital transformation is, among others, driven by the emergence of distributed systems which may for example consist of a large number of sensors performing measuring tasks. In order to correlate their data, it is important that all sensors, or more general all sub devices of the distributed system, use the same time scale.

Among the already existing time protocols, NTP (Network Time Protocol) is probably the most important one. It is well established and powerful; its binary packages carry a lot of information and it forms the basis for highly accurate system time by providing a hierarchical system for synchronizing system time and disciplining the system clock to UTC. NTP provides two security procedures to protect of integrity and authenticity: the pre-shared key scheme specified in RFC 5905 and the Autokey protocol [2]; however, neither of them fulfills state-of-the-art security requirements. The disadvantage of the shared key scheme is that it that it did not scale well enough for large network deployments or the global Internet. While Autokey does provide the required scalability by using public key infrastructure (PKI) mechanisms, its applied cryptographic primitives are too weak to withstand modern attacks [2,3]. Because of these shortcomings there is an ongoing effort in the IETF to standardize the Network Time Security (NTS) protocol for the protection of NTP [5,6].

Tlsdate was developed as a secure alternative to NTP and uses the timestamp provided by TLS servers during TLS handshake. The unexpectedly use of the TLS protocol might lead to inaccurate time information and will stop working if the timestamp is removed from the TLS protocol in future versions.

Highest accuracy can be reached by using PTP (Precision time Protocol) if the IT infrastructure fully complies to PTP requirements [4]. The current standard only provides an experimental annex for the integrity protection of PTP messages which was never well adopted and implemented. Thus, the current effort to revise the PTP specification includes a plan to provide a new security mechanism for PTP.

Our aim was to develop a light-weight and secure time protocol that is universally usable. Therefore, it is solely based on technologies that are available on virtually every IoT device communicating via internet.

II. Websocket

Websocket was developed to allow efficient bidirectional communication with low overhead over TCP connections and is designed to be compatible with the HTTP protocol. It is standardized by the IETF. By using TLS the integrity of the transmitted data is ensured. Because websocket uses the port 80 or 443 it is not affected by firewalls blocking UDP packages. Moreover, websocket allows to communicate through proxy servers, using encryption if necessary.

III. Websocket Time Protocol (WST)

To provide a timestamp via the websocket time protocol (WST) two packages are sufficient: request and response. A third package – acknowledge – is induced by the TCP protocol. All data exchanged via WST is in JSON format and UTF-8 encoding. All numbers are represented as strings in the decimal system.

A. Request

As shown in Table I, the request package send by the client consists of only one optional parameter, the *client time information* which is opaque to the server. This information may be used by the client to estimate package run times after receiving the response package without remembering the send timestamp or the request order on several requests. The server simply copies the *client time information* into the response package. Thus, its format may be client-specific and e.g. relative time designations are also possible. Furthermore, additional arbitrary information may be added as long as the request package does not grow above its maximum size of 1024 byte.

TABLE I. REQUEST FORMAT AND PARAMETER

		{ c: <client time information; optional> }
c	optional	Client time information. Returned by server as is. Necessary to calculate packet run time on client side. Additional information may be enclosed.

B. Response

TABLE II. RESPONSE FORMAT AND PARAMETER

		{
		c: <client time information; optional>,
		s: <server timestamp; required>,
		e: <error estimation; optional>,
		l: <leap information; optional>
		}

c	optional	Client time information. Returned by server as is.
s	required	Server timestamp [milliseconds since 01.01.1970 00:00:00 UTC; number as string representation in decimal system].
e	optional	Error estimation of server timestamp [milliseconds; number as string representation in decimal system].
l	optional	Leap information; syntax in line with RFC 5905. Values 0 (or empty), 1, 2, or 3 [Text in JSON].

There are four parameters in the response package send by the server, see Table II. The optional *client time information* is repeated without validation as described above.

The required main parameter, the *server timestamp*, is given as the number of milliseconds bygone since the Unix epoch 01.01.1970 00:00:00 in Coordinated Universal Time (UTC) neglecting time zones. It is accompanied by the optional parameter *error estimation* which is also given in milliseconds. The *error estimation* is an auxiliary means for the client to estimate the quality of the timestamp.

The *leap information* is given as a simple number between 0 and 3. The meaning of the numbers is equivalent to NTP: If there is no leap second pending, the *leap information* is set to 0. A *leap information* of 1 indicates that the last minute of the day has 61 seconds and it is set to 2 if the last minute has only 59 seconds. If the server is no able to inform about leap seconds the *leap information* is set to 3 or omitted.

IV. USE CASES

A. Time Server

A WST time server receives the request and sends a response including the time stamp. In our implementation a Linux server is used whose system time is managed by ntpd (NTP daemon), with stratum 2.

The time is determined using the operating system's interface *gettimeofday* which provides the number of microseconds past since the Unix epoch 1.1.1970 0:00. This value is converted into milliseconds. Thus, the dissemination of time is done with a resolution that is 1000 times lower than the resolution of time determination.

In order to estimate the error of the generated timestamp a parameter offered by NTP is used. The so called rootdelay denotes the total round trip time from our server to the reference clock (stratum 0). Assuming symmetric run times, the error estimation is set to one half of the rootdelay.

The execution of the program described above takes less than 1 ms. Websocket allows for performant server implementations that impose low requirements on server hardware.

B. Simple time request

The simplest time request conceivable goes without a client timestamp. The client sends a time request with an empty value of c. The server responses to the request and transmits its timestamp T_{STS} to the client. Upon receiving the response, all information is ignored except the timestamp T_{STS}, which is applied without applying any correction to the propagation delay of the request and response messages. This may be sufficient for IoT devices without demand for highly accurate time scales which maybe even lack a real time clock (RTC). This is especially true if package run times are comparatively short. One idea could be to operate a WST time server in the same local network which is pretty straightforward as described above.

C. Run time correction

There are numerous factors if timestamps are transmitted over networks. A fundamental treatment is offered by RFC 5905 but would go beyond the purpose of WST. Because usually the package run times are the most important influence factor WST allows to roughly estimate these.

In this case the request message contains the timestamp T_{CS} of its submission (see Fig. 1). The server returns the timestamp T_{STS} together with the timestamp T_{CS}. This enables the client to calculate network delay Δt after receiving the response package at time T_{CE}.

$$\Delta t = T_{CS} - T_{CE} \tag{1}$$

The individual propagation delay of the request and response message may thus be estimated by $\Delta t/2$. The offset between the client and the server clock Θ results in

$$\Theta = -T_{STS} + (T_{CS} + T_{CE})/2. \tag{2}$$

Strictly speaking, this assumption is only correct if the propagation delays are symmetric and if computation time of client and server may be neglected. In the worst case of maximum asymmetry, the maximum error of the offset Θ sums up to $\Delta t/2$. To keep this error low it may be reasonable to conduct three or more time inquiries and evaluate only the one with the smallest package run times.

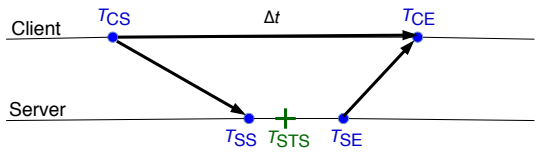

Fig. 1. Progress of time exchange

D. Use in HTML-Websites

All modern browsers support the websocket protocol. An example to use can be found at our web site https://uhr.ptb.de.

V. Conclusion

We presented a simple to use protocol for exchanging time information and some use cases. The protocol is light-weight and easy to employ. It is based solely on technologies that are available on virtually all IoT devices connected to the internet. Furthermore, it allows to protect the time information by using TLS.

VI. Outlook

There is ongoing work on implementing the use cases described in section IV which will serve as first examples of using the new WST protocol. We also plan on registering the websocket sub-protocol identifier “time” with IANA. Further standardization is conceivable.

References

[1] I. Fette, A. Melnikov, “The WebSocket Protocol,” Internet Engineering Task Force (IETF), RFC 6455, 2011. [Online]. Available: http://tools.ietf.org/html/rfc6455

[2] S. Röttger, "Analysis of the NTP Autokey Procedures," Technische Universität Braunschweig, Institute of Theoretical Information Technology, Braunschweig, 2012.

[3] D. L. Mills, “NTP security analysis”, May 2012 [Online] Available: https://www.eecis.udel.edu/~mills/security.html. [Accessed: 24-Jul-2017]

[4] IEEE. (2008). Precision clock synchronization protocol for networked measurement and control systems. In *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)* (Vol. 1588, pp. 271). New York: The Institute of Electrical and Electronics Engineers, Inc.

[5] Donoghue, K. O., Sibold, D., & Fries, S. (2017, Aug. 28 2017-Sept. 1 2017). *New security mechanisms for network time synchronization protocols.* Paper presented at the 2017 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS).

[6] Franke, D. F., Sibold, D., & Teichel, K. (2017). *Network time security for the network time protocol* (draft-ietf-ntp-using-nts-for-ntp-10). Retrieved from https://datatracker.ietf.org/doc/html/draft-ietf-ntp-using-nts-for-ntp-10