Federated Clock Synchronization in IIoT

Peter Waher, *Member, XMPP Standards Foundation*

***Abstract* — This paper presents a method to synchronize clocks between devices across the Internet using XMPP. This allows synchronization of clocks to occur across federated domains in global internetworks. Results from a study of clocks and clock synchronization between computers and servers in the Azure cloud is also presented.**

***Index Terms* — Industrial Internet of Things, IIoT, Timing, Synchronization, Clocks, XMPP.**

I. Introduction

Syncronization of clocks is important in Industrial applications. Clock synchronization methods for local area networks have existed for some time. While proprietary solutions exist with varying degree of efficiency, Global Industrial Internet of Things applications on the other hand lack standardized approaches for efficient clock synchronization. This paper presents an effort within IEEE to standardize a method of clock synchronization across federated global internetworks based on XMPP; *IoT Harmonization* [1].

Two kinds of clock synchronization are presented: Synchronizing absolute time across devices and synchronizing events. The first kind of synchronization is important, if devices have regular events occurring at given time periods, such as regular sampling. To make samples correlate across the network, samples should be taken at synchronized points in time, such as at the change of every hour. An accurate knowledge about absolute time is then required. The second kind of synchronization might be required, if distributed devices agree to execute a task at the same time, without regards for what the actual absolute time is. In this case, some form of internal time is sufficient. A higher level of accuracy can then be achieved.

II. XMPP

XMPP is a protocol that is standardized by the Internet Engineering Task Force IETF [2-4]. Entities are given strong global identities (*Jabber IDs*, or *JIDs*), and they are authenticated using SASL before given access to the network. Communication is done using XML fragments called *stanzas*. They form the building-blocks of more advanced telegrams. Three types exist: *message*, that allow for asynchronous messages between entities; *iq*, or information query, that allows for request/response exchange between entities; and *presence*, that implement a Publish/Subscribe pattern to distribute information about the entity to authorized recipients. The identities of all participants are forwarded in all stanzas sent across the network.

Clients connect to the network by connecting to a broker. Brokers on the Internet cooperate, forwarding stanzas between domains. All stanzas sent and received by a client are transmitted across the connection it has with the broker. This allows a client to communicate with any other client on the network, even if they reside behind different firewalls, and are connected to different brokers.

III. Method

The method used to estimate clock differences between two entities are divided into three parts.

A. Sampling

To synchronize its clock with another device, here called a *clock source*, the client begins by sending an *iq* with a clock synchronization request to the clock source. The clock source returns a response containing the internal clock with as high a precision as possible. It also optionally provides information about any high-resolution timer it has. Before sending the request, and when receiving the reply, the client samples its own clock and high-resolution timer. From these three samples, an estimate of the *latency* l in the network, and *difference* between the clocks Δt can be done, as is shown in Fig. 1.

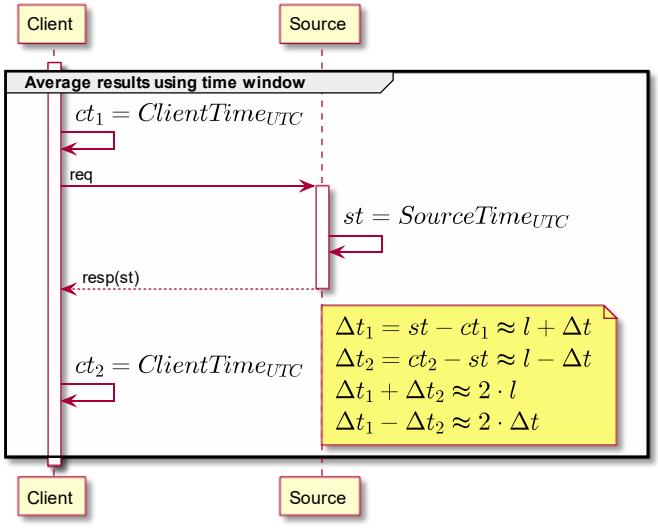

Fig. 1. Clock Synchronization communication pattern

B. Filtering

There are many sources of errors when estimating clock differences and latency on the Internet. Varying server loads, network bandwidth, process activity, processor shifts, server

shifts in clusters, locks, internal clock alignment algorithms, operating system specific operations, etc. are just a few examples. To reduce the error in the estimate, sampled values are first filtered, removing erroneous samples here called spikes.

To do this, a small window is defined (in our examples, containing 16 samples), and a rolling average formed. A specific position in the window is defined (in our study, it's the 6th latest value). The number of samples above the average are counted, as are the number below the average. In a normal situation, these should be around the same amount. In an error case, many samples are on one side, and few on the other. We define a *spike width* parameter (in our study, it's 3). If one side contains this number of samples, or less, we have a possible spike. To differ between an actual change in measurement, and an error measurement, we only consider spikes at the given spike position. If the sample at that position lies on that side of the average with a few samples, it is considered an error measurement and removed. A new average is calculated, this time only based on the remaining values after the spike position. This average is called the *filtered measurement*.

C. Averaging

The filtered measurements are in turn placed in a rolling window of a pre-defined size (in our case, it contains 100 filtered measurements). To reduce the variance of sampled values further, a rolling average of these filtered values are computed. If the client desires, the standard deviation across this window is also computed. This standard deviation can be used as a measurement of the amount of error in the final answer.

IV. Experiments

To test the pattern in a real-world scenario, two experiments were designed. In the first, one client application on one local laptop running Windows 10 try to synchronize its internal clock with the clock of an Integration Server running remotely in the cloud (Azure), using Windows Server 2016 [5]. The second experiment connects two different client applications running on the same local machine but interconnected through the same Integration Server in the cloud. Since they run on the same machine, we know they share the same internal clock. The validity of the method can thus be easily evaluated. In both experiments, 2000 samples, 5 seconds apart are made.

A. Results, synchronizing with virtual cloud server

In Fig. 2 we can see the difference between the clocks of the local machine and the server machine. Red shows raw samples, blue filtered samples and green averaged samples. There is a notable difference between the clocks. There is also a notable drift of the clock difference, as well discrete events occurring on the server, that makes the clock difference make a jump. This jump might be caused by a shift in physical server, a restart of a server, or some other event happening in the virtual server environment provided by Azure. Fig. 3 Displays the same data, except the raw values have been removed for clarity. Fig. 4 shows the standard deviation of the averaged values. Fig. 5-7 show the corresponding network latency.

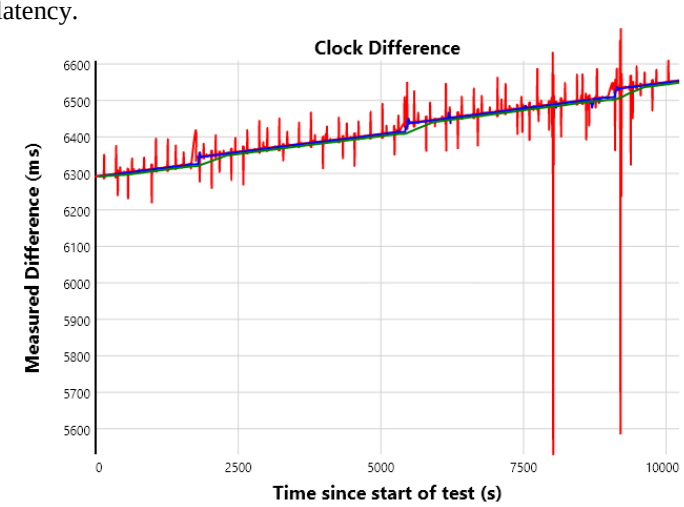

Fig. 2. Clock Difference between local machine and extas.is

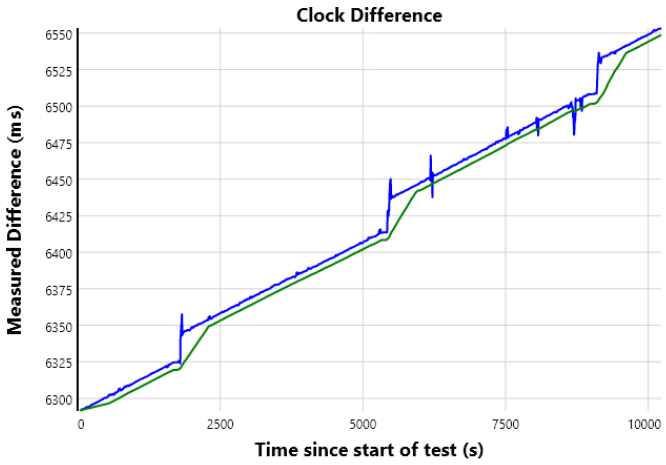

Fig. 3. Averaged clock difference values are delayed

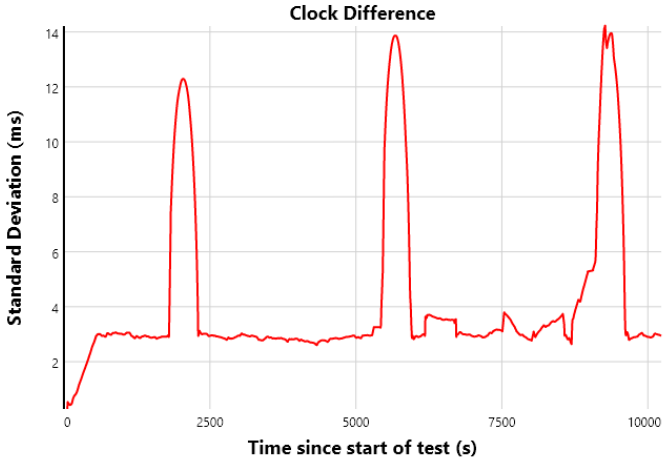

Fig. 4. Standard deviation of averaged clock difference

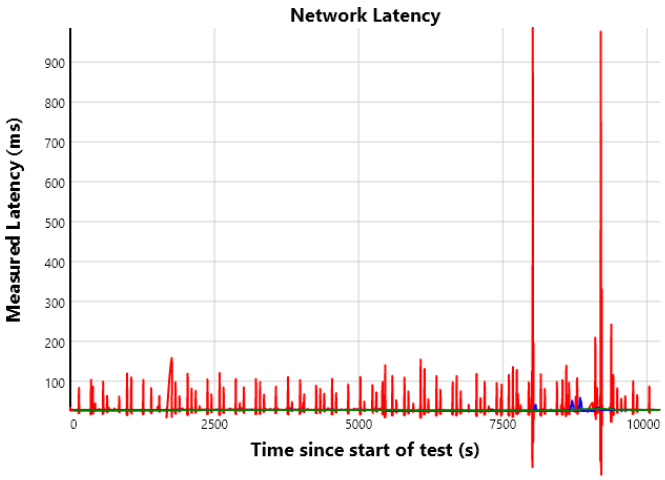

Fig. 5. Latency between local machine and extas.is

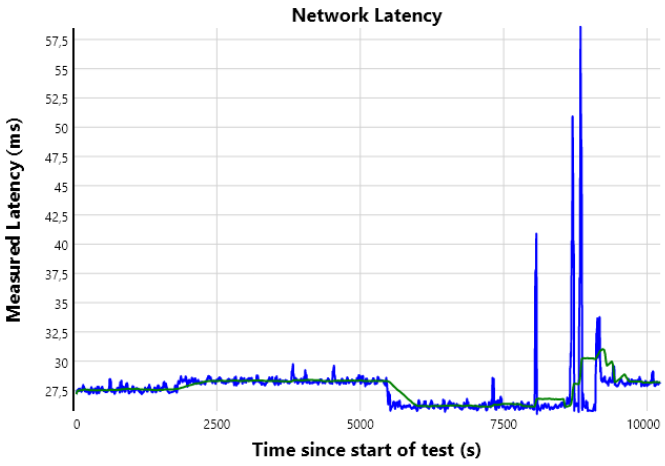

Fig. 6. Averaged latency values are delayed

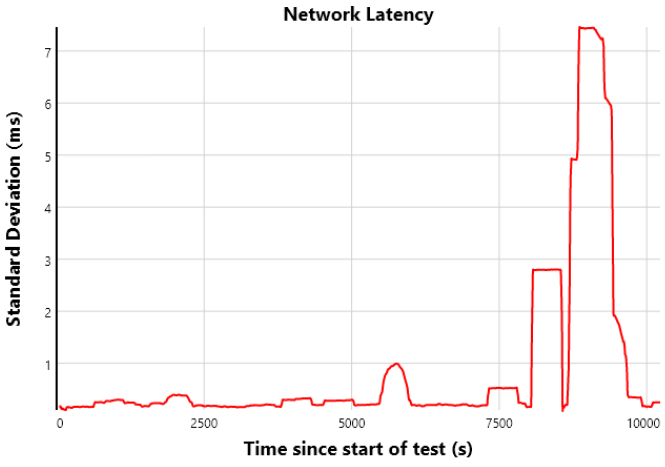

Fig. 7. Standard deviation of averaged latency

B. Results, synchronizing two clients

Now that we know that the broker we use in communication, as well as network performance is variable, we want to see how that affects clock synchronization between two controlled clients, communicating with each other over a variable and uncontrollable internetwork. The experiment connects two different client applications running on the same machine (to be able to control the actual clock difference parameter) and that same broker analyzed in the first experiment. Fig. 8-13 show the corresponding results.

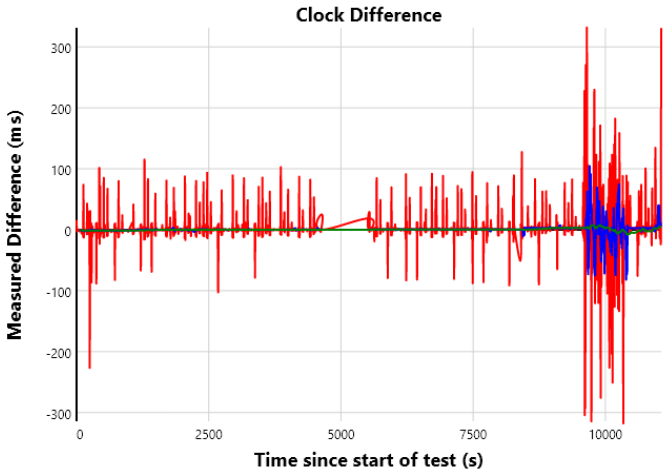

Fig. 2. Clock Difference between two machines interconnected over uncontrolled internetwork

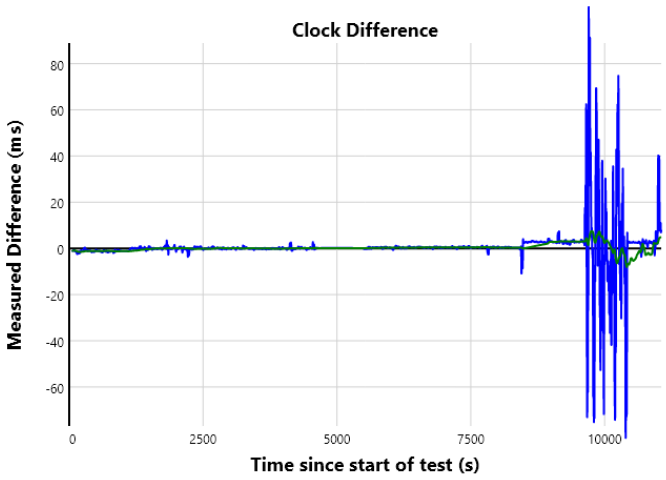

Fig. 3. Averaged clock difference values are delayed

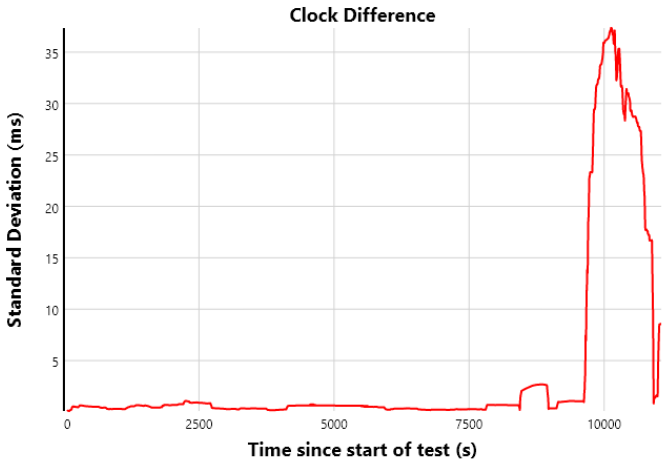

Fig. 4. Standard deviation of averaged clock difference

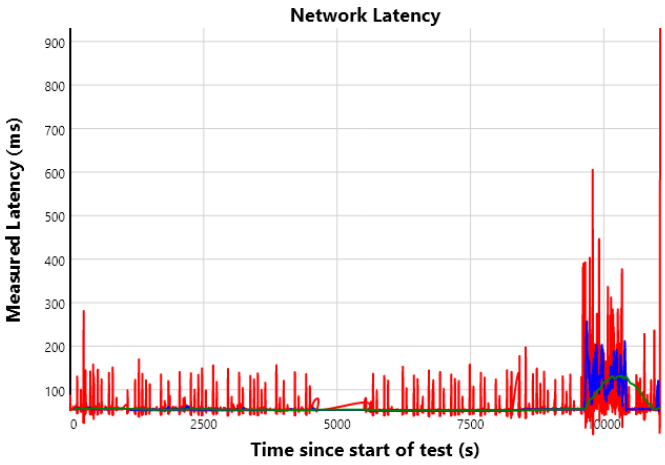

Fig. 5. Latency between two machines interconnected over uncontrolled internetwork

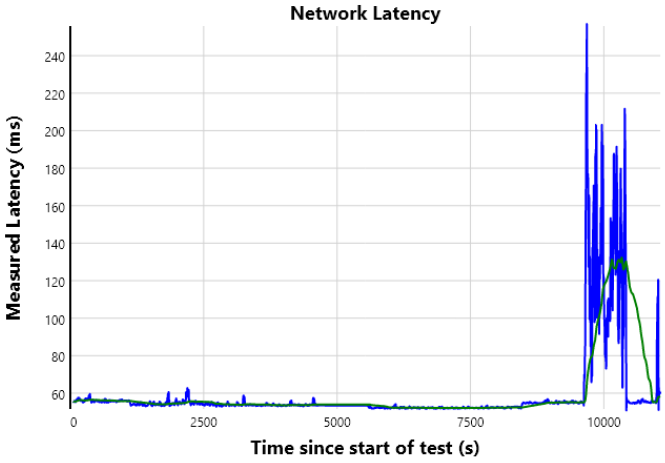

Fig. 6. Averaged latency values are delayed

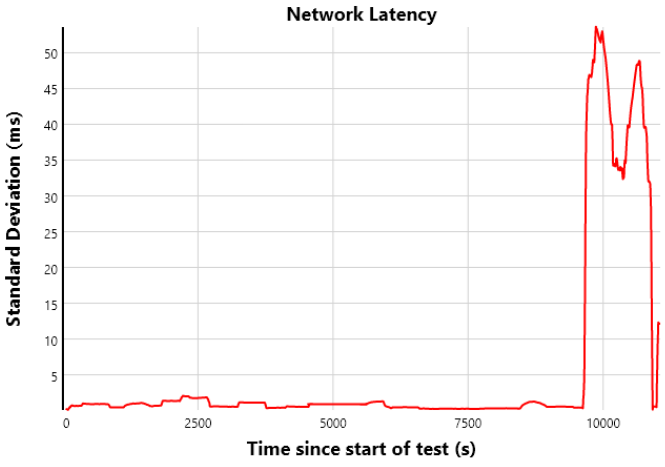

Fig. 7. Standard deviation of averaged latency

V. Conclusions

The method presented provides a mechanism whereby clients running on normal Windows operating systems, connected over a federated uncontrolled internetwork can synchronize their clocks to a relatively high level of accuracy (approximately 1 millisecond) using XMPP and the IEEE IoT Harmonization XMPP interfaces, even if the networks and servers used to interconnect them are variable. Clock synchronization accuracy in controlled networks, using clients running in real-time operating systems, can be expected to be much higher.

VI. Software

Following is a list of software used in the experiments. The Extasis Integration Server [5] is based on the IoT Gateway [7], which supports both XMPP and the Clock Synchronization extension [1]. The libraries providing these functionalities are available as NuGets [8, 9], with source code. Source code for the tool, coming with the IoT Gateway, used to monitor the clocks is also available [10].

VII. Acknowledgment

Financial support for this study was provided by The Internet Foundation in Sweden IIS [6].

VIII. References

[1] IEEE XMPP IoT Interfaces, IoT Harmonizaion. [Online]. Available: https://gitlab.com/IEEE-SA/XMPPI/IoT/blob/master/ClockSynchronization.md.

[2] P. Saint-André, “RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core,” [Online]. Available: http://xmpp.org/rfcs/rfc6120.html.

[3] P. Saint-André, “RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence,” [Online]. Available: http://xmpp.org/rfcs/rfc6121.html.

[4] P. Saint-André, “RFC 6122: Extensible Messaging and Presence Protocol (XMPP): Address Format,” [Online]. Available: http://xmpp.org/rfcs/rfc6122.html.

[5] P. Waher, “Extasis Integration Server” [Online]. Available: https://extas.is/

[6] The Internet Foundation of Sweden, IIS. [Online]. Available: https://iis.se/

[7] P. Waher, “IoT Gateway repository”, GitHub [Online] Available: https://github.com/PeterWaher/IoTGateway

[8] P. Waher, “Waher.Networking.XMPP”, NuGet [Online] Available: https://www.nuget.org/packages/Waher.Networking.XMPP/

[9] P. Waher, “Waher.Networking.XMPP.Synchronization”, NuGet [Online] Available: https://www.nuget.org/-packages/Waher.Networking.XMPP.Synchronization/

[10] P. Waher, “Waher.Utility.AnalyzeClock”, GitHub [Online] Available: https://github.com/PeterWaher/-IoTGateway/tree/master/Utilities/Waher.Utility.AnalyzeClock