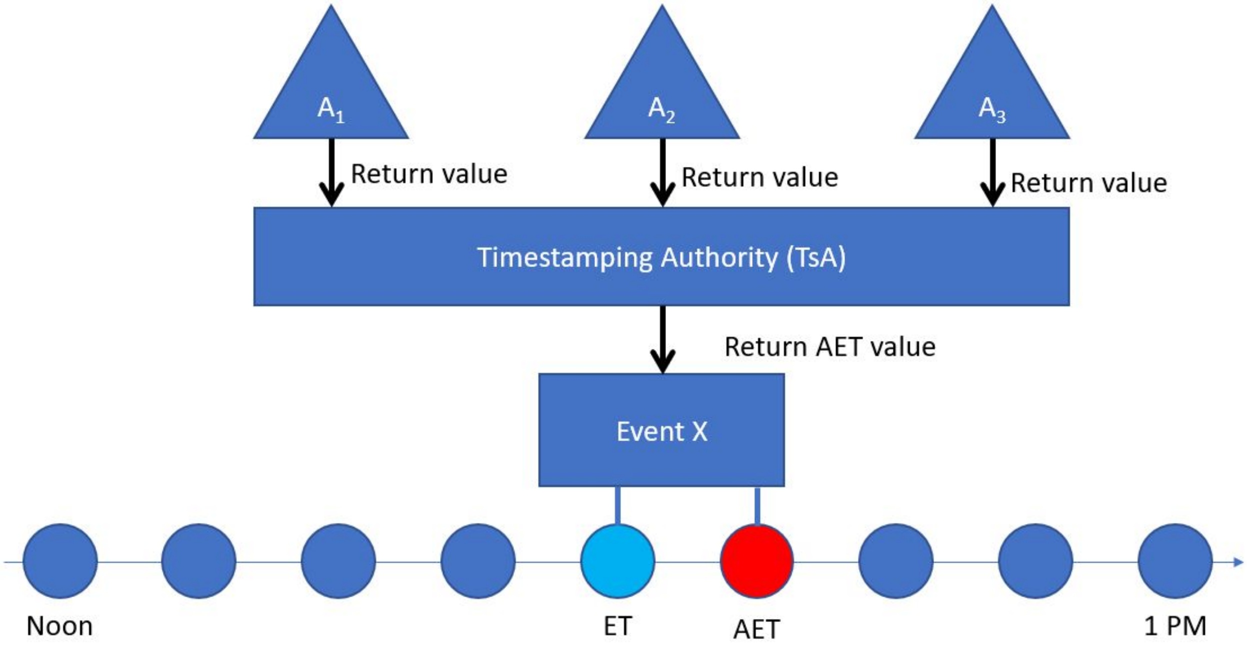
Timestamping Authority Server Prototype

Introduction

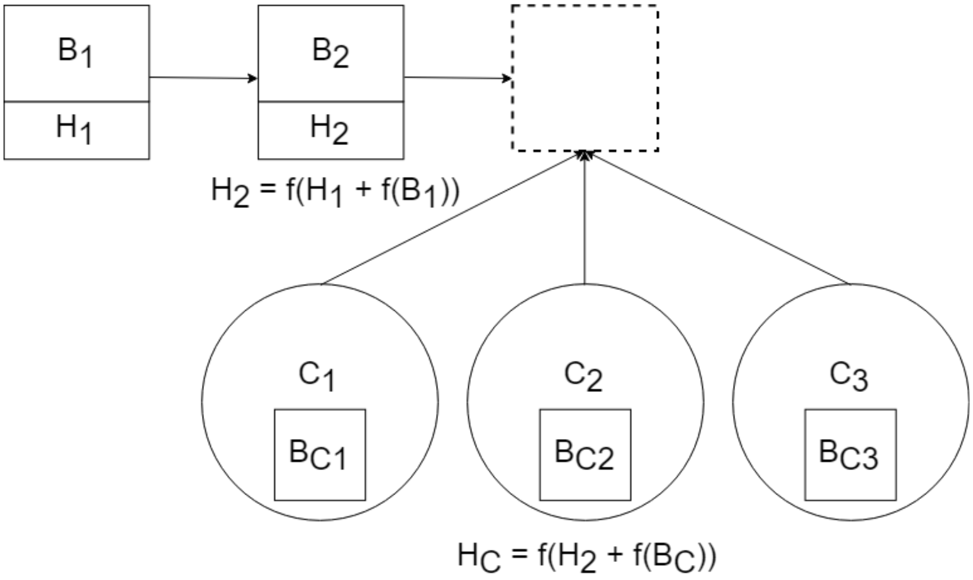
Timestamping has been used throughout human history and refers to marking the time when a certain event happened. In the modern world it usually means the digital date and/or time attached to the data. It has applications in and outside of computing. For example, it can be used to mark any kind of event, such as when the message was sent, or it can be used to tackle race conditions emerging in concurrent programming. The purpose of this project is to suggest a system called Timestamping Authority (TsA), that would be able to provide reliable real-time high precision timestamps preserving the order of events regardless of their number and time when they were processed.



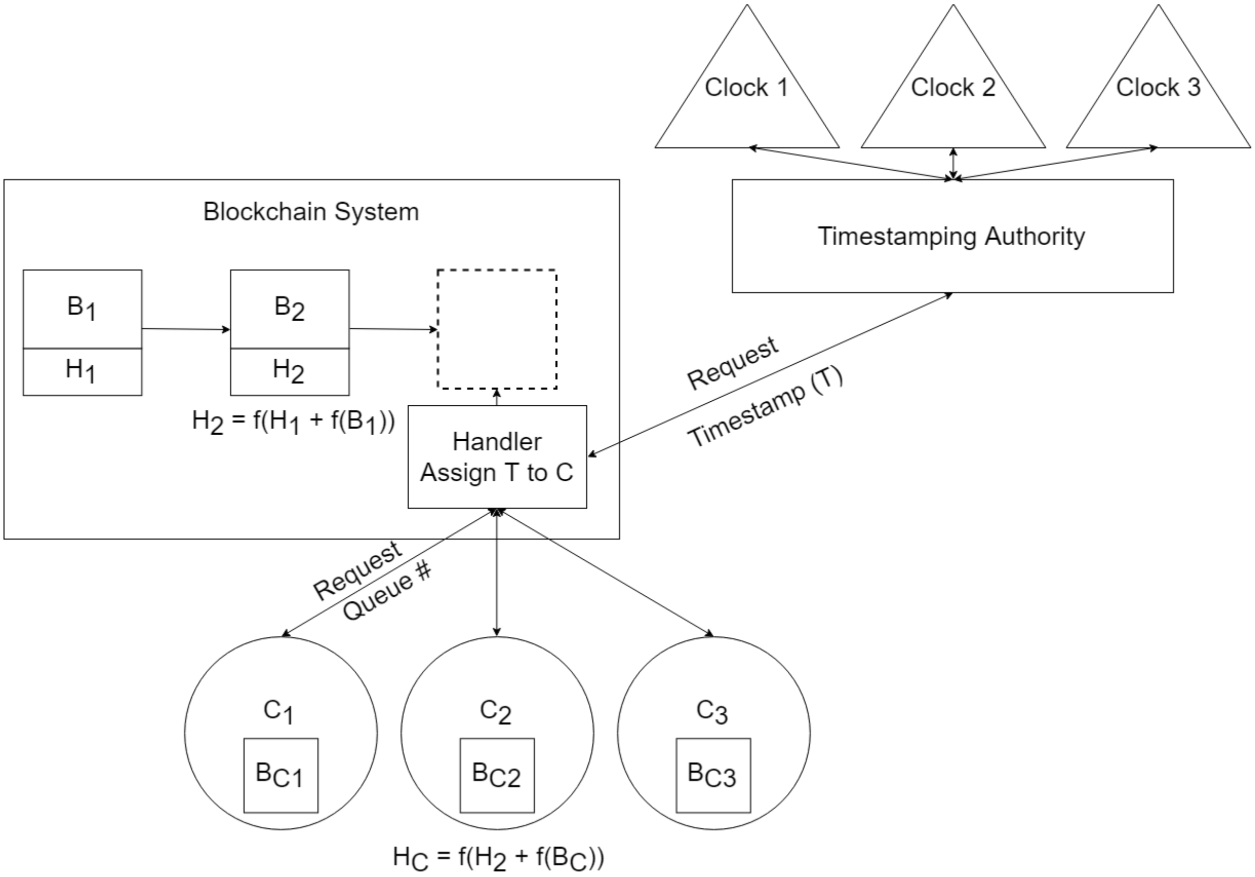
Scheme of TsA operation. ET - exact time of the event. AET - approximate exact time.

Motivation

Timestamps could be useful in a variety of situations. One of them is adding a block into a blockchain. In order to add a new block, the candidate that wants to insert the block needs to calculate hash using the last block in the chain, and submit it to the system. However, if there are multiple candidates willing to add their blocks, only the one who finishes the calculation first will be added. All other candidates have to start over with the new last block. Timestamping service could be integrated into the system, and it could assign timestamps to the candidates and form a queue, giving some time to the first-comer and telling everyone else to wait. This could reduce CPU time spent by the candidates. Timestamps will serve as a proof that the candidate was indeed first to come. This would be especially useful if there are multiple servers accepting blocks and they need to be coordinated. In this case, they could query the TsA and receive timestamps that could be used across the servers.



Adding a block to the blockchain without a TsA. Only one candidate will succeed and all others will have to start over the calculation. B - block, H - hash, f - hash function, C - candidate.



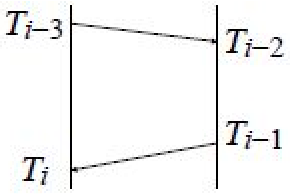
Adding a block to the blockchain with a TsA. The queue is formed among the candidates and each one is given some time to do the computation. If the client fails, the opportunity is given to the next candidate. B - block, H - hash, f - hash function, C - candidate.

Clients cannot be allowed to use their local time because it might be inaccurate. Maintaining accurate UTC time is difficult, and TsA does not claim that the time provided is accurate. Instead, it establishes a common timescale for the system that would preserve order of timestamped events. This project tries to achieve this by aggregating time from several official reliable atomic clocks and use it to produce the common timescale. It currently takes time from three atomic clocks nearby and takes the average of their times. More effective formula can be used after the aggregation. Tests with local server and client show that the order of timestamps is preserved.

Means of Operation

The accepted protocol for time synchronization on the internet, which is used by countless devices all over the world is Network Time Protocol (NTP). This protocol operates over User Datagram Protocol (UDP), which is a low-level network protocol. NTP version 3, which is used in this implementation, was standardized as RFC-1305 (https://tools.ietf.org/pdf/rfc1305.pdf), and is compatible with the latest version 4, RFC-7822 (https://tools.ietf.org/pdf/rfc7822.pdf). Many atomic clocks used as time sources for this project have publicly available NTP servers, which are synchronized to the clocks, thus providing accurate time to the clients. The communication with the client also works over NTP. NTP can provide up to 1 millisecond accuracy under ideal network conditions, and the accuracy deteriorates as asymmetric network routes are introduced (see RFC-1305, Appendix F The NTP Clock-Combining Algorithm). When a network route is asymmetric, time for a request to get to the server is different from the time to get the response. This usually happens when large-distance requests are made or when the network condition is unsatisfactory. Since only the nearest atomic clocks are used, the accuracy is preserved.

NTP also accounts for latency automatically (see RFC-1305, H.3. Measurement Errors). It achieves that by transmitting 4 timepoints in the UDP message that allow to calculate correct network delay and local clock offset from the server clock (server is the atomic clock).

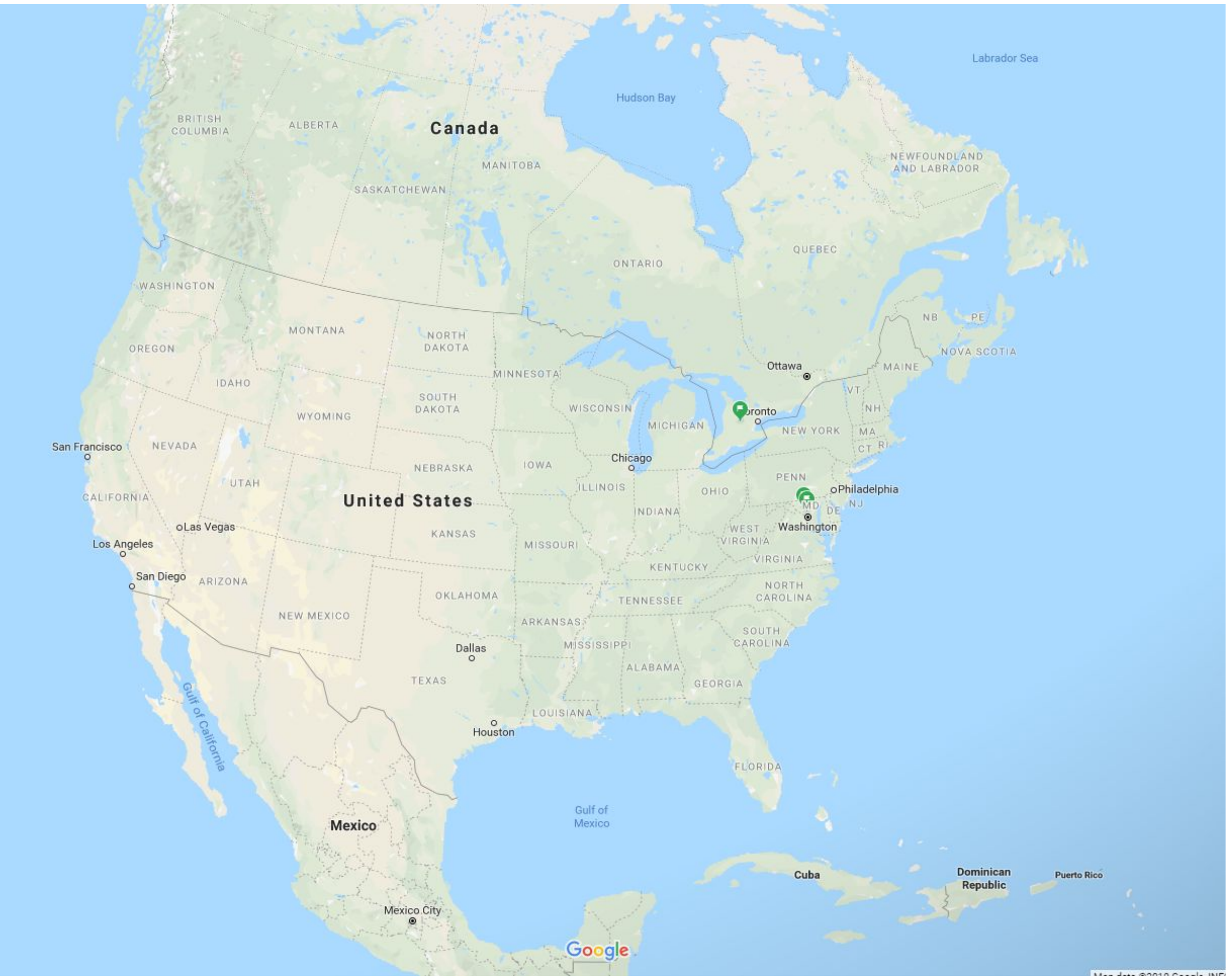


$$\delta = (T_i - T_{i-3}) - (T_{i-1} - T_{i-2})$$
$$\theta = \frac{(T_{i-2} - T_{i-3}) + (T_{i-1} - T_i)}{2}$$

(RFC-1305, 3.4.4 Packet Procedure)

Clocks Used and Their Reliability

International Bureau of Weights and Measures (IBWM or BIPM in French), the organization that makes SI units, creates UTC time standard (https://www.bipm.org). This standard is created based on over 400 participating laboratories measuring time with their atomic clocks and reporting the measured time every 5 days to IBWM, which then calculates the weighted average based on the clock precision and produces UTC. The problem is that it is difficult to transmit clock data in real-time without losing accuracy, hence all the calculated time points are in the past. Nevertheless, the data from the atomic clocks suggests that most of them are synchronized up to hundreds of nanoseconds (https://www.bipm.org/en/bipm-services/timescales/time-ftp/Circular-T.html#nohref), which is more than enough for the problem at hand. IBWM also publishes Annual Report on Time Activities (https://www.bipm.org/en/bipm-services/timescales/time-ftp/annual-reports.html), where they list laboratories participating in creation of the time standard, and are known to be reliable. Some of those laboratories have publicly available NTP servers linked to the clocks, which are used in this project as time sources. For the North American region possible clock servers are the National Institute of Standards and Technology NTP (https://tf.nist.gov/tf-cgi/servers.cgi), the United States Naval Observatory NTP (https://tycho.usno.navy.mil/NTP/) and the National Research Council of Canada NTP (https://nrc.canada.ca/en/certifications-evaluations-standards/canadas-official-time/network-time-protocol-ntp). One problem with using these clocks is that the public servers are vulnerable to DDoS attacks, and therefore all clients making too frequent requests to them are banned (e.g. NIST clock encourages one request every 4 seconds). This problem has been solved by maintaining offset from local time to each of the used clocks, and updating it every 10 seconds (this interval can be configured). Even though local clock is imprecise, it is very unlikely that any significant deviation will happen in 10 seconds.



Locations of Atomic Clocks used in the Implementation

Implementation Description

NTP utilities library from Apache Commons Net was used in the project (https://commons.apache.org/proper/commons-net/). Some code was modified, some was just used as is. The project currently consists of 4 executables written in Java. The first and the primary executable is the server. It can be configured and started via command line, the NTP servers to be used can be provided in a text file. The server periodically updates local time offsets of the used clocks and services client requests. There is a class TimeStampingAuthorityServerRunner with a main method that is compiled to the executable server. There are also other two classes that can be used as library code. One of them (TimeStampingAuthority) just requests time from the clocks, maintains clock offsets and can provide aggregate time via the API, and another one (TimeStampingAuthorityServer) runs an actual NTP server on specified port in addition to this. There is a separate Clock class that contains logic for maintaining the clock offsets and network communication with the atomic clock servers. The second executable is the client for testing. It can be configured and started via command line, and it logs the time points received from the server to a .txt file. The client makes requests to the server at the specified address with a given frequency. Interval of 10 ms or more should preserve events ordering on the TsA side. The third executable takes the log file produced by the client and checks if timestamps the client received are in the ascending order. It can check multiple files at the same and is also usable from a command line. The fourth executable takes several log files and combines them. This is used when several clients were run simultaneously to test how well TsA preserves order of events with its timestamps. All the executables are configurable, please refer to Javadocs of corresponding classes for reference.

The project uses Maven framework for dependency management and build automation. It also uses Git for tracking changes. The project is to be published on Github soon. Executables can be used on their own, but it is suggested that an IDE supporting Maven and Git is used in development, e.g. IntelliJ IDEA or Eclipse (both are free). There are four Maven modules corresponding to four executables in the project. 3 contain just one class with a main method, and one contains several classes mentioned above. There is a directory called “testing” which contains all the executables, file with the clock data and a script for testing. Maven is configured in such a way that when “package” command is run, all the modules are assembled into .jar files. When “verify” command is run after that, .jar files are moved into correct directories for convenience. “out” directory contains all 4 executables bundled with their dependencies, and a library .jar, containing non-executable server-side classes without dependencies. This library jar can be used in other projects as a dependency, assuming that Apache Commons Net library is included.

Limitations

1. The order-preserving property of current TsA implementation has been tested only locally. More tests are needed for the technology to be reliable.

2. Sometimes several consequent timestamps get “merged” and have the same value, even though it is known that the events were supposed to have a time interval between them. For example, there could be several timestamps with values 1, 1, 1, 4, 5, while the events were supposed to happen at times 1, 2, 3, 4, 5. It is assumed that this happens due to low computing power of the development machine used, and as a consequence of running several processes for actual testing of the program and other processes running in the background. More tests are needed.
3. NTP assumes that the offset from the client clock is to be calculated by the client, so the TsA has no way to know if the client completed the timestamp calculation correctly.
4. When requesting time from the clocks and serving time to the client, only one NTP-request is made, since decent network conditions are assumed. It is suggested to make several NTP-requests and take one with the lowest round trip time to exclude asymmetric route errors.
5. RFC-1305 suggests a formula for combining several clock times to increase accuracy and precision. This could be used instead of a simple average of all the clocks, which is currently used.

Installation

This project uses Java Development Kit (JDK) 12.0.1 as it is the latest at the moment. None of the new features are used, so it should not be a problem to run the project on the older JDK. The project has been tested to compile and run successfully with JDK 1.8.0_211.

If used as a .jar, the executables do not have any dependencies except for Java runtime.

If used as Java classes, you will need to download Apache Commons Net library, which provides implementation of many network protocols, including NTP, along with utilities that make it easier to use these protocols in the program. You can include this library as a dependency in a Maven project using the information on https://mvnrepository.com/artifact/commons-net/commons-net/3.6 or download the libraries as source code or as binaries on https://commons.apache.org/proper/commons-net/download_net.cgi. The libraries are open source and are available under the Apache License, Version 2.0. After downloading, add the library to your project in your IDE and you are ready to go.

It is suggested that you use and IDE with support of Maven and Git if you would like to use them in the development of the project, because Maven downloads the dependencies for you and makes building and packaging easier and Git helps revert any changes and keep track of the history of the project. If not, a simple solution would be to rip out the classes of all 4 modules and put them in a new project together. Do not forget to add Apache Commons Net library as a dependency.

This file and the Javadoc can be found in “docs” directory of the project.