THE CHINESE UNIVERSITY OF HONG KONG

FINAL YEAR PROJECT REPORT (TERM 1)

A Time Synchronization Protocol based on Distributed Network

Author
Tsz Hin Leung
(1155079351)

Supervisor
Prof. LYU Rung Tsong Michael

LYU1804

Department of Computer Science and Engineering

Faculty of Engineering

Abstract

Network Time Protocol is one of the most important protocol in computer network history. Most systems rely on NTP to synchronize the system time. However, the design of this protocol is vulnerable in today's internet environment. To mitigate the risk of man-in-the-middle attack and distributed denial-of-service attack, using a distributed approach of blockchain can provide a more secure time keeping mechanism.

Timechain is the blockchain-based time keeping solution introduced in this project. It focuses on giving a more secure time instead of a more accurate time. The block structure and the consensus algorithm are modified to achieve the purpose of time keeping. Credibility is established when more and more blocks are appended to the chain. In the future, it can also take into consideration the network delay to offer a more accurate time.

Table of Contents

1. Introduction

In modern days computer networking, keeping the time synchronized across all machines on the network is critical. Without time synchronization, the local clock on each machine will slowly drift away from each other and eventually resulted in a significant time difference. The effect is especially crucial on time-critical systems. When these systems lost their frame of time reference across different machines, many security protocols will fail to work properly. For example, when you are establishing a secure communication with a remote server, it offers you a Transport Layer Security ("TLS") certificate to ensure its identity. Every certificate has an expiration date, and in fact an incorrect system time is one of the major causes for HTTPS certificate errors. A recent study of Google Chrome users shows that 6.75% of the client-reported times were having an error of over 24 hours. [1] The idea behind the expiration is to minimize the chance of the private key of the certificate owner being compromised. Whenever a new certificate is being generated after the old one expires, it ensures that the owner possess his new private key. However, if the time on the local machine is delayed, it may consider expired certificates as valid, and hence the certificate cannot serve the purpose of authenticating the server.

Nowadays, most systems rely on Network Time Protocol ("NTP") and its variants (such as Simple NTP) to get the time synchronized. These protocols are successful at providing a high precision time. However, they were developed before the 2000s and lack modern security features, which makes these protocols highly vulnerable to Distributed Denial-of-Service (DDoS) and man-in-the-middle (MITM) attacks.

With the increasing popularity of blockchain, it eliminates the central authority by a distributed network and maintain the operation by a consensus algorithm agreed by all nodes. This distributed model of networking mitigates the risk of single-point-of-failure and MITM attacks. It seems to be a solution in keeping the time safely. Credibility of time is being established when more and more blocks are being appended to the ledger, and gradually establishes a trustworthy time.

In this project, I will investigate the problems brought along by the current time synchronization approaches and suggest an alternative solution in blockchain. This report will be organized as follows: In chapter 2, I will investigate the design of NTP and analyze its security concerns. In chapter 3 and 4, I will propose the Timechain solution, an implementation of blockchain for time keeping in a distributed network. And finally, in chapter 5, I will propose the further development for the Timechain in the following semester.

2. Background

2.1 Network Time Protocol

2.1.1 History

NTP was originally developed by Professor David L. Mills at University of Delaware. It was first implemented as NTP version 0 (NTPv0) in 1985 and documented in RFC 958. However, that document only specifies the data representation and message formats and lacks synchronizing or filtering mechanisms. [2] These algorithms were implemented in the NTPv1, published in RFC 1059 in 1988. [3] The protocol was bought to wide attention in the engineering community with the publication of the article in the IEEE Transactions on Communications. [4] The latest version is NTPv4 which is specified in RFC 5905 published in 2010. It is modified to accommodate the Internet Protocol version 6 (IPv6) addresses and algorithms that increases the potential accuracy. [5] Following the retirement of David Mills, the NTP project is now maintained by the Network Time Foundation.

2.1.2 Clock strata

NTP uses a hierarchy structure for time sources. Each layer is called a stratum and is given a number starting from 0. The first layer, i.e. stratum 0, high-precision time-keeping devices, such as atomic clocks and satellites. They generate a very accurate pulse per second signal which are assumed that the clock skews are extremely small. They are used as the reference clocks for the whole system. And for the remaining stratums, the hosts at stratum n synchronize their time with other hosts in the stratum n-1. The number represents the distance of the distance from the reference clock and is used to prevent cyclic

synchronization. Within each stratum, there also exist peer-to-peer connections for sanity check and backup. The connection is illustrated in Figure 2.1 [6].

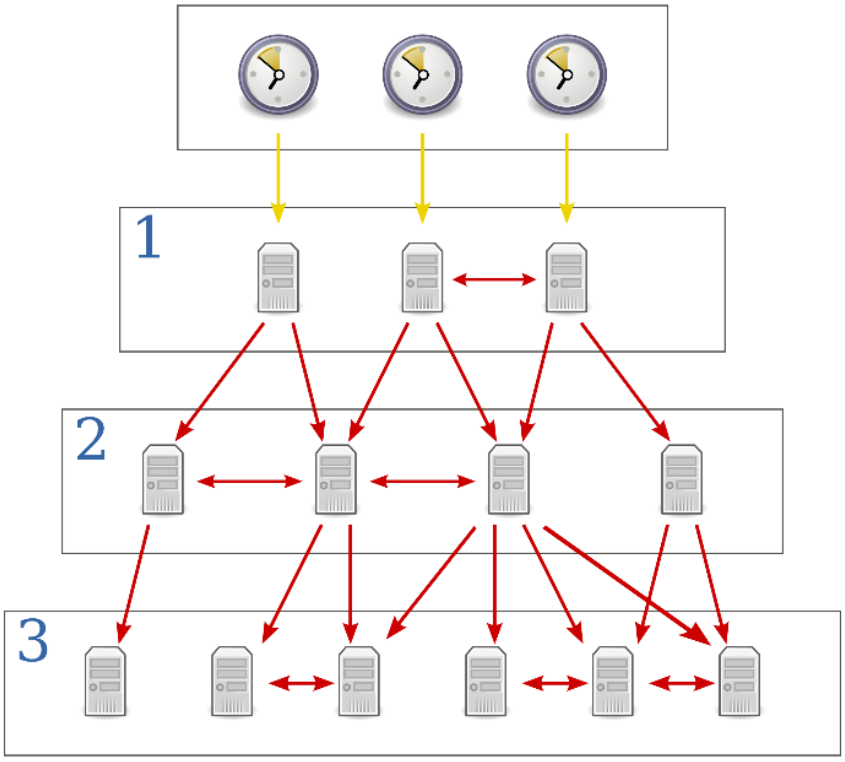

Figure 2.1Illustration of NTP Connections

2.1.3 NTP packet

NTP operates over the User Datagram Protocol (“UDP”). The NTP server listens for NTP client packets on port 123. Both NTP requests and responses share a common format, as shown in Figure 2.2 [7].

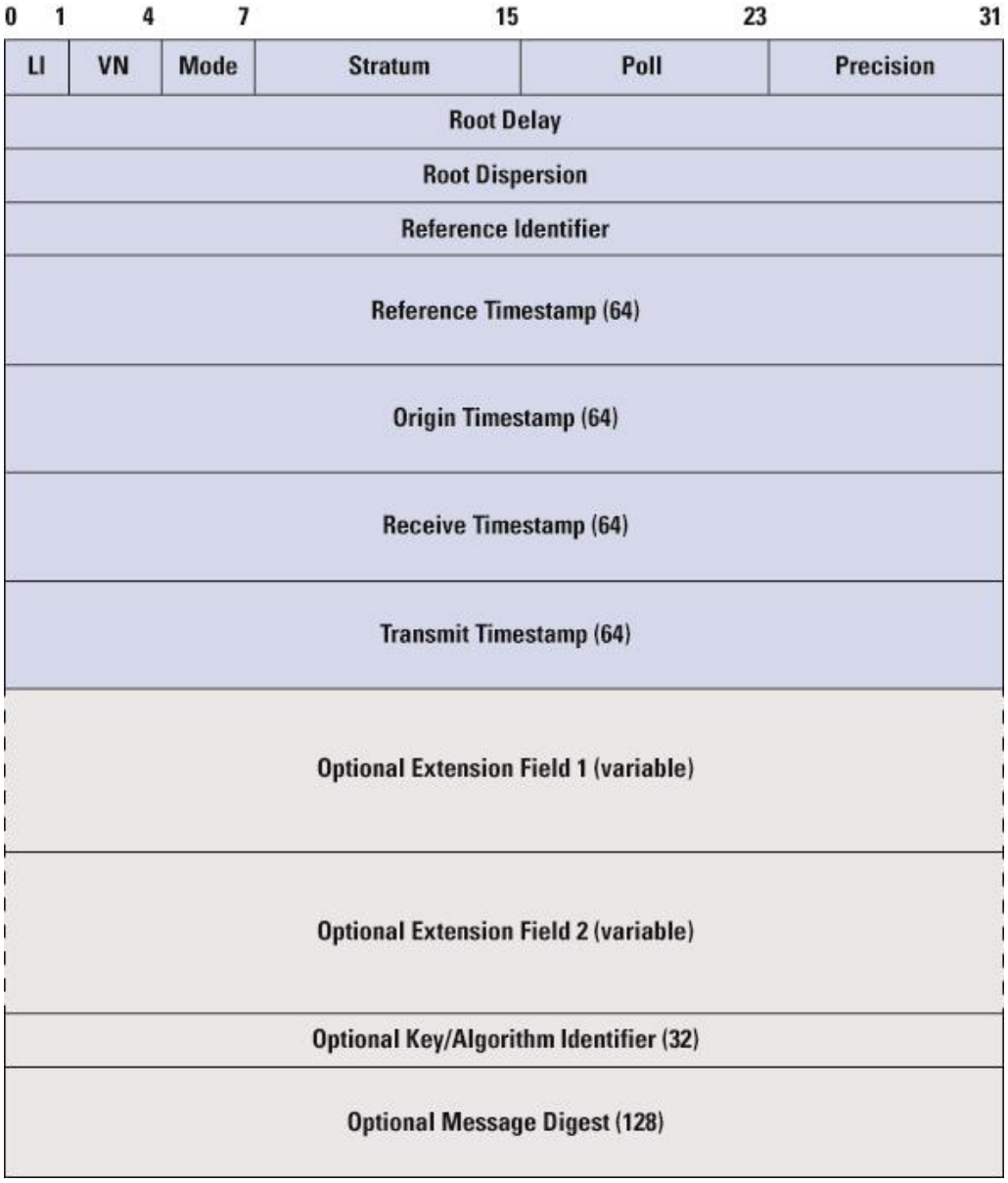

Figure 2.2 NTP Packet Format

2.1.4 Time synchronization

NTP uses the client/server model for exchange of time information. The communication between the client and server is illustrated as follows:

1. When the client requests the server for the current time, it sends the NTP packet with its own local timestamp t_1 in "Origin Timestamp". It is the time the packet is being transmitted to the server.
2. When the server receives the request, it adds the server time of receive t_2 in "Receive Timestamp" in the data packet.
3. The NTP server sends the response back to the client with the timestamp of the packet leaving the server t_3 appended in "Transmit Timestamp". The response is the whole NTP packet, including the "Origin Timestamp" and "Receive Timestamp".

When the client receives the response, it logs its local time of receive as t_4. By doing so, the client can derive both the server time and the round-trip network delay between the client and the server. The round-trip delay can be modelled by the time between the client sending the request and receiving the response minus the processing time of the server. It is given by the following formula:

$$\delta = (t_4 - t_1) - (t_3 - t_2)$$

And hence the offset of the client clock from the server clock can be calculated with the following formula. It is assumed that the upstream and downstream delay is symmetric:

$$\theta = \frac{1}{2}[(t_2 - t_1) + (t_3 - t_4)]$$

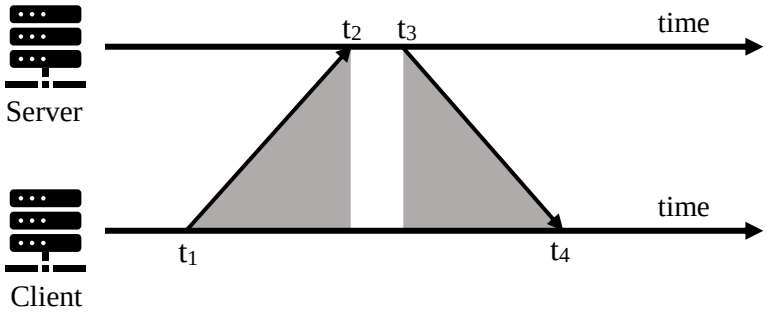

Figure 2.3 Round-trip delay δ

2.2 Security Concerns of NTP

2.2.1 Man-in-the-middle Attack ("MITM")

While NTP supports both symmetric and asymmetric cryptographic authentication, they are rarely used in practice. Symmetric cryptographic authentication requires the client to be pre-configured manually to accept packets hashed with the correct symmetric key, which makes this solution troublesome for public deployment. The secure distribution for symmetric key to only trustable users is a big challenge. For example, the internet time service provided by the National Institute of Standards and Technology ("NIST"), which is under the United States Department of Commerce, only accepts registered users to access their authenticated time service. The key is sent to the users only by mail or facsimile and it expires every year. [8] Hence, this approach is not suitable to deploy public NTP service which accepts arbitrary clients.

The asymmetric cryptographic authentication is provided by the Autokey protocol, which is specified in RFC 5906 [9]. However, RFC 5906 is an informational document instead of a standard requirement. Clients do not request Autokey associations by default, and most public NTP servers do not support Autokey. The major reason for the lack of implementation of Autokey is the performance drawback. Asymmetric cryptography authentication is considered as computationally expensive. While symmetric key authentication is believed to require almost constant CPU time, public key algorithms requires a significantly more and unpredictable CPU time. As NTP servers are designed to handle a huge number of requests, the computational overhead for asymmetric key authentication makes it impractical to be implemented. [10]

Unauthenticated network communications are highly vulnerable to man-in-the-middle attacks, and NTP is no exception. In fact, most operating systems take in unauthenticated NTP packet as their time reference. It becomes extremely easy to perform an on-path attacks. On-path attackers are attackers having a privileged position in the network. By exploiting various traffic hijacking techniques, it is possible to hijack the NTP communications between the client and the server. [11] For instance, NTP server is defined by either the host name or the IP address. It is configured manually and static. If the Domain Name System ("DNS") is being hijacked, the domain name can be resolved to another address which response to the NTP queries with an incorrect time.

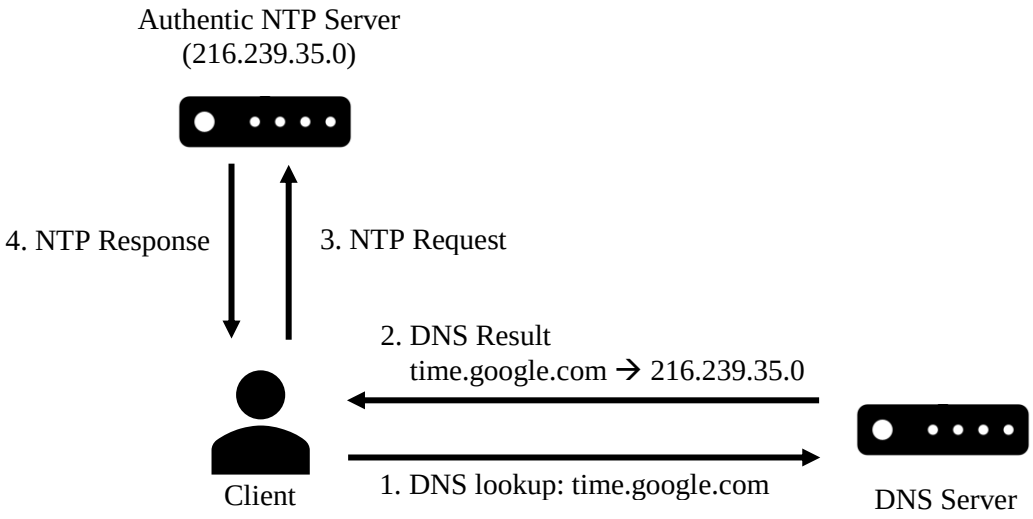

Figure 2.4 NTP Time synchronization in an expected environment. Client has NTP server configured to time.google.com is resolved correctly to the authentic NTP server.

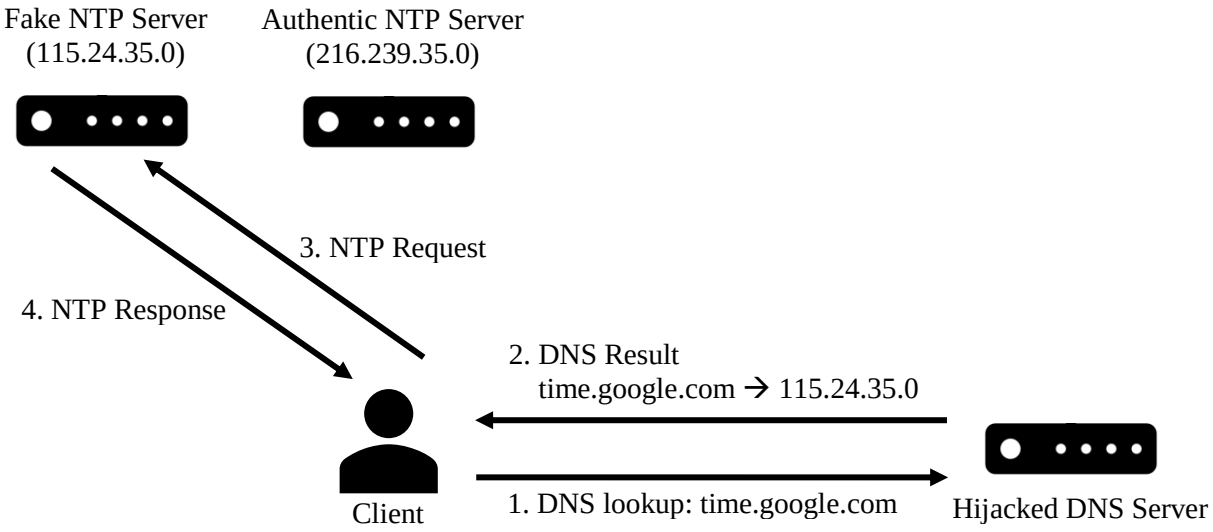

Figure 2.5 The DNS server is hijacked and time.google.com is resolved to another address. The client hence requests the NTP time from the fake server without authenticating and update its own time accordingly.

The implementation of NTP defines a panic threshold of 1000 seconds (16 minutes 40 seconds) to mitigate the effect of the system time being influenced by an incorrect time. If the NTP time received instruct the local clock to alter by over the panic threshold, no adjustment to the clock will be made. NTP also defines a step threshold of 125 milliseconds. If the timestep is larger than the step threshold and smaller than the panic threshold, the clock will be altered if the predefined time duration called "stepout" have elapsed since the last clock update. [5] In the most recent implementation of the NTP daemon (ntpd v4.2.8), the stepout value is 300 seconds. Therefore, to shift the client clock for 1 year, it requires at least 114 days. [11]

However, there exist another method to shift the time by a great step by exploiting reboot. In the implementation of NTP, most operating systems allow any time shift when the system starts. This is a reasonable implementation as on-board clocks shift significantly when powered down. Yet whenever rebooting the system due to manual intervention, system

upgrade, or power cycling, it creates a window for a fake NTP time to be injected into the system. [11]

2.2.2 Single Point of Failure and Distributed Denial-of-Service Attack

All centralized servers are the single points of failure within a system. Their failure produces an outage of the whole time synchronization mechanism. In the implementation of NTP client, it is allowed to have set one or multiple sources of time as the NTP server, but by default usually only one server is being set. Also, all centralized servers are subject to DDoS attacks. The attacker can generate a huge amount of traffic from different sources and flood them against the NTP server, thus disabling the ability for the server to handle proper requests. If that single server fails or the time is tampered with, all the clients polling this server will be unable to obtain an authentic server time.

The risk of single point of failure can be mitigated by setting multiple time servers for reference and use NTP server pools instead of a single server. The server pool creates redundancy for the timing service. In case a single server breaks down, other services in the cluster can still share the load of the down server. For instance, pool.ntp.org is a project of a big cluster of time servers. It is estimated to be serving around 5-15 million systems worldwide. [12] However, they are also worry about the rapid increase of internet-connected devices and the need of more timing capacity for these devices.

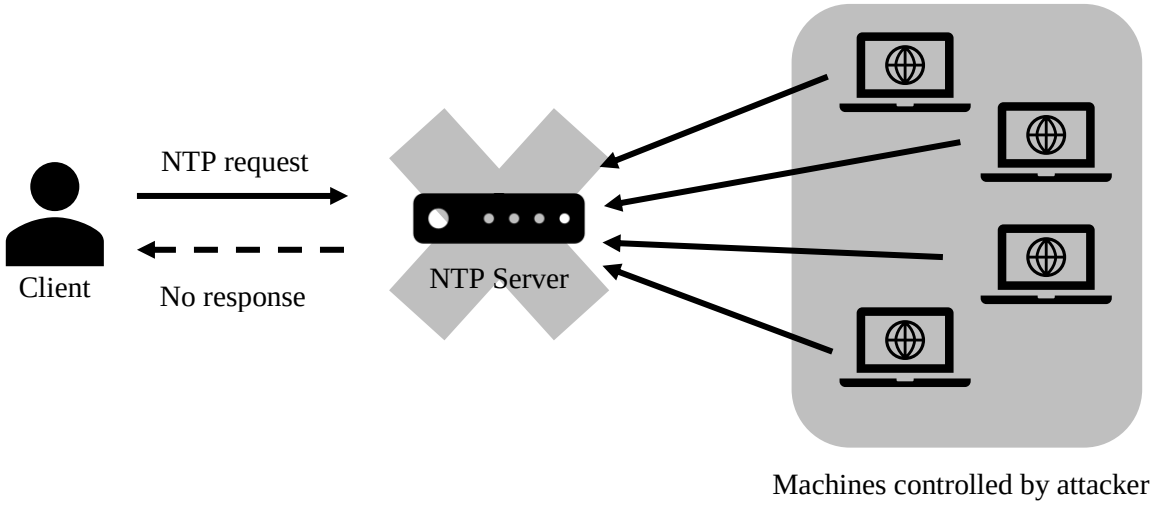

Figure 2.6 Illustration of attacker disabling NTP server with big volume of requests.

3. Solution Design

3.1 Blockchain

Blockchain is a decentralized, distributed and public digital ledger. The ledger is maintained by the participants of the network without a centralized authority. It allows data to be appended to the blockchain by different nodes and each node participants in verifying and auditing the data involved. Each block is being built on top of previous blocks such that the blocks cannot be modified easily. Due to the distributed nature of blockchain, it eliminates the weakness of single point of failure.

3.1.1 History

In 2008, a whitepaper titled “Bitcoin: A Peer-to-Peer Electronic Cash System” was published by a person (or a group of people) named Satoshi Nakamoto to solve the existing financial market problems. This paper focuses on developing a decentralized online payment method such that no central authority is required to process the transaction. [13] He implemented the system in the follow year. That is where the word “block chain”, as two separate words, first appeared in a comment of his source code.

After the success of Bitcoin in recent years, the public realized the importance of blockchain technology, not only in financial industry, in keeping that data safe from being altered. Other blockchain platforms are being developed, such as Ethereum and Hyperledger. These

blockchains may not include a native currency but enabled other possibilities such as smart contracts.

3.1.2 A block

The distributed ledger is represented by chain of blocks stored in a database. It is maintained by a network of machines called nodes. Unlike traditional database models, the entire database is stored on every node and each node is allowed to write to its own copy with validated blocks. The moment a node joins the network, it downloads the most updated chain of blocks and work with other nodes to keep the chain continuing.

A block mainly consists of the following fields:

- Block index: A block index of 0 indicates a genesis block.
- Timestamp: The timestamp of the block creation.
- Hash: The hash value of the current block. A fixed-length string is generated according to the content of the block.
- Previous hash: The hash value of the previous block. It links the current block to the previous block.
- Nonce: A value used to track the Proof of Work ("PoW") algorithm counter. It will be explained later.
- Block body: The data to be stored.

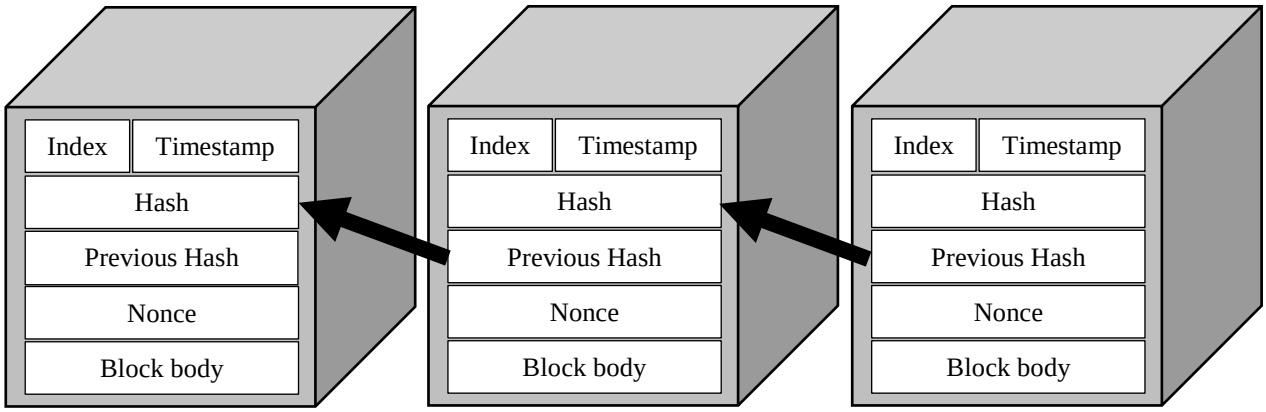

Figure 3.1 Illustration of relationship between blocks

Hashing is the backbone of the immutability of the blockchain. It ensures no blocks are being tampered with. As the hash of the block is computed with the content of the whole block, it also takes into the previous hash for calculation. In other words, the hash of a block is dependable on the hash of the previous block. In order to add a block, it has to be approved and verified by every other node in the network. If a block in the chain is altered, the hash will be changed and the whole the blockchain will become disconnected. To connect the blockchain again, all hashes of the subsequent blocks have to be updated, and the resulted blockchain will also be different. In fact, the process of updated all the blocks is extremely difficult and computational-intensive. This will be explained in the following parts.

3.1.3 Consensus

Consensus algorithm is an important part of the blockchain system. As anyone can participate in maintaining and updating the blockchain, it is important to setup a set of rules such that every node adheres to to maintain the blockchain. Whenever new block is being generated by a node, it is broadcasted to other nodes. The other nodes will verify the block

with the consensus algorithm and only if the block is verified, it will be added to the blockchain.

The most widely used consensus algorithm is the Proof of Work ("PoW"). It was invented by Satoshi Nakamoto and implemented in the Bitcoin. In order to create a valid block, a node has to solve a complicated mathematical puzzle which takes computational power. The mathematical puzzle is nothing but to achieve a desired hash. In computing the hash, the algorithm takes in the block content and reaches a hash of a desired pattern. The only varying field in a block is the nonce. By taking in the content of the block with varying nonce, eventually, a block hash of a desired pattern is reached. The block, together with the hash and the nonce, is then being broadcasted to the network. Other nodes receiving the block will verify if the combination of block content and nonce can come up with the desired hash.

	Nonce	Hash
Block content	0001	888B19A43B151683C87895F6211D9F8640F97BDC8EF……
	0002	4FAC6DBE26E823ED6EDF999C63FAB3507119CF3CB……
	0003	446E21F212AB200933C4C9A0802E1FF0C410BBD75F……
	……	
	1234	03AC674216F3E15C761EE1A5E255F067953623C8B38……

Figure 3.2 Illustration of the Proof of Work mathematical puzzle. In this example, an arbitrary hashing algorithm is used. Block content is kept constant and only the nonce is changing. The block is accepted if the hash starts with "0".

As the process of hashing is believed to generate a string which the probability of a pattern is equally distributed, the process of solving this mathematical puzzle can only be achieved by brute-force approach. The computational power required is the barrier to update a block,

or in the case of an attacker, after altering of a certain block, it requires a huge amount of computational power to modify the hashes of all subsequent blocks. The difficulty is defined by the complexity of the matching pattern. For instance, requiring the hash to start with two zeros ("00……") is more difficult of starting with one zero ("0……").

However, in order to verify the correctness of the proof of work, it only takes constant time. Upon receiving of a block, a node simply calculates the hash, which includes the nonce and the block content, of it. If the resulted hash matches the hash as indicated in the block and the desired pattern, the block can be deemed valid. This also shows that the block created have worked on creating the block, hence the method is called Proof of Work.

Other methods of consensus also exist, such as Proof of Stake, Byzantine fault tolerance algorithm, and the delegated Proof of Stake algorithm. In this project, I will focus on the Proof of Work.

3.1.4 Network

The blockchain runs in the following sequence:

1. Each node collects the data and prepare its own block
2. Each node works on finding the proof of work for its block
3. If the proof of work is found, the node broadcast the block to all nodes reachable
4. Nodes that receives a block verify the PoW result and its data (if necessary)
5. If the block is being accepted, it appends the block to the end of its chain and use its hash to work on the next block. Otherwise, it continues to work on the current block and wait if some other nodes broadcast other solutions.

In case of multiple nodes broadcasting different versions of the next valid block at the same time, different nodes may receive different blocks. Nodes always consider the longest chain of blocks as the valid one. Hence, they work on the first node they received. Eventually, when then next PoW is found and a new block is arrived, the tie will be broken. If the new block contributed to the other chain, that chain will become longer and the chain the node is working on will be disposed.

3.2 Timechain

Inspired by the blockchain, I have designed Timechain: a more secure time synchronization mechanism. It is designed mainly to mitigate the security risks of NTP:

- It is designed mainly to mitigate the risk of man-in-the-middle attack. By nature of blockchain, the creditability of data is established by the consensus of the whole network. The chain is immutable and there is no single node the hacker can compromise to get the fake data being spread in the network.
- As blockchain works in a distributed manner and every node holds a copy of the ledger, the single point of failure is eliminated and almost impossible to perform a denial of service attack. Even if the attacker compromised a number of nodes in the network, the time synchronization mechanism can still function.

However, compared to NTP, this design of time synchronization is mainly targeted on security applications, such as certificate expiration. In such scenario, the precision of time is less important. It is acceptable to have a certain error, say, within 1 minute. This is sufficient to reject a certificate which is expired by 1 minute or more. Hence, is this implementation, the time security is more important than the time precision. It can be worked together with NTP to achieve redundancy for time synchronization.

3.2.1 Block

The block structure of the Timechain is similar to that of a blockchain. However, in the original implementation of the blockchain, the timestamp is recorded at the time of block creation. As it takes time for the proof of work to be completed, by the time the block is

being broadcasted to other nodes, the time is already delayed, hence the timestamp cannot be used directly for time synchronization.

To take into consideration the time of the block being broadcasted to the network, instead of using a nonce value for PoW, the timestamp itself will be used as the nonce. In other words, for every time a node calculates the hash, the timestamp on the node is being updated. This ensures that the block comes with the timestamp being broadcasted to the network (and the processing time for computing the hash value, but it is negligible as the time required to compute a single hash value is constant). After subsequent blocks are being appended to the chain, their hashes are also affected by the hash of preceding blocks that includes an updated time, hence altering any of the blocks will break the chain.

	Timestamp	Hash
Block content	13:24:01.89392 24/12/2018	888B19A43B151683C87895F6211D……
	13:24:01.89393 24/12/2018	4FAC6DBE26E823ED6EDF999C63……
	13:24:01.89394 24/12/2018	446E21F212AB200933C4C9A0802……
	……	
	13:24:05.29348 24/12/2018	03AC674216F3E15C761EE1A5E25……

Figure 3.3 Illustration of the Proof of Work mathematical puzzle. In this example, an arbitrary hashing algorithm is used. The timestamp is used as the nonce.

3.2.2 Consensus

The proof of work consensus algorithm is being used in the Timechain implementation. However, apart from validating the mathematical puzzle solution as shown above, upon receiving a new block, each node also checks the timestamp for an acceptable range of error (mainly due to network latency). If the time difference between the timestamp on the block and the local time is above a certain threshold, say 15 seconds (meaning an error of 15

seconds), the block is considered having an inaccurate timestamp instead of a large delay, and will be discarded by the node.

3.2.3 Network

The blockchain runs in the following sequence:

1. Each node repeats the following until the PoW is solved:
    a. Create a block with the current timestamp
    b. Calculate the hash of the block
2. If the PoW is solved, the block is broadcasted to other reachable nodes
3. Nodes that receives a block log the local time of receive and verify the PoW result
4. If the PoW is correct, a node then compare the logged time to the timestamp on the block. Accept the block if the time difference is smaller than the threshold
5. If the block is being accepted, it appends the block to the end of its chain and use its hash to work on the next block. Otherwise, it continues to work on the current block and wait if some other nodes broadcast other solutions
6. For every certain amount of time, say 5 minutes, each node will broadcast the chain of nodes on their local machines to its peers. By doing this each of the nodes will check if they are having the same chain. If no, the longest, and yet valid chain will replace the existing local chain.

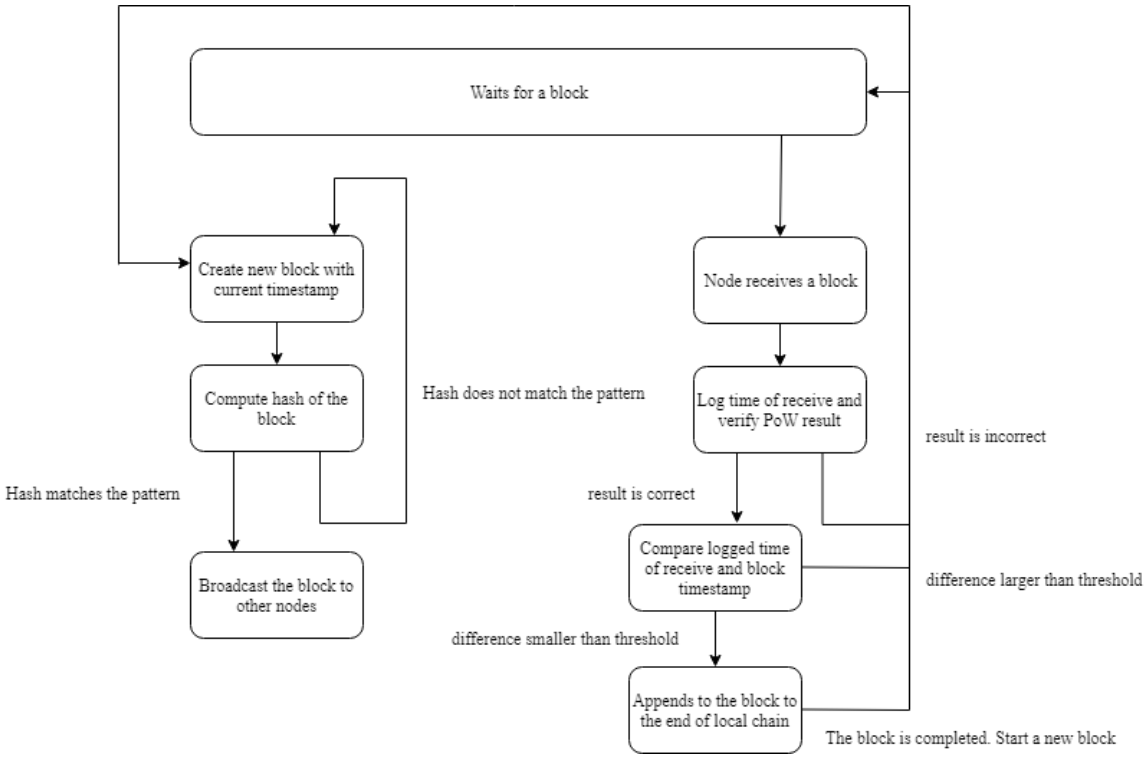

Figure 3.4 Flow of Timechain

As a node is more likely to receive new blocks and chains announcements from nearby nodes (nearby here refers to low network delay) first, the delay for each message is small, but the total time delay propagates from the writer of the time blocks. The system works well in a small scale without caring about the time delay, but when it is deployed in a wide area network, the delay should be taken into consideration. This will be further explained in the part of "Further Development".

3.2.4 Credibility

As it takes time for the PoW to come up with a solution, in order to taper with a certain block in a blockchain, all the PoW of the subsequent blocks have to be recalculated. This

requires a huge amount of computing power and makes it almost impossible to catch up with the development of the blockchain.

For instance, an attacker attempts to alter the n-5 block, where n is the total number of blocks currently exist in the longest chain. Suppose generating the PoW for each block requires approximately 10 minutes. (As PoW is a brute force approach, the time required for computation is unpredictable. This number here is just an average value.) In order to alter the n-5 block and keeping the whole chain valid, the attacker also has to modify blocks n-4, n-2, n-1 and n. To modify 5 blocks in total, it takes 50 minutes. By then, the whole chain is still progressing by other nodes in the network, and probably reach the n+5 blocks already. The attacker is unable to catch up with the longest chain.

Hence, by the PoW, when more and more blocks are being appended to a chain, the more credible and previous blocks are. The Timechain algorithm will determine the current clock with reference to blocks few levels before the latest block. The time is more trustworthy compared to consider only the timestamp of the latest block.

4. Implementation

4.1 Blockchain Platforms

In order to implement the Timechain blockchain, major blockchain platforms are considered.

4.1.1 Bitcoin

Bitcoin is the first peer-to-peer payment method which operates with no central authority. It is decentralized and every participant on the network manage transactions and issue bitcoins collectively. [13]

4.1.2 Ethereum

Ethereum is a public, block-chained based distributed environment which is equipped with the capability of running applications on top of it. It consists of a native currency Ether and the decentralized virtual machine, the Ethereum Virtual Machine, which can execute scripts on the network. The idea is to avoid a complete dependency on a single server for data storage and manipulation. The smart contract is the programmable interface which is mainly used to manage agreements between users. [14]

4.1.3 Hyperledger

Hyperledger is a project started by the Linux Foundation. It is designed for cross-industry business usage. It aims to create enterprise grade distributed ledger framework and code bases to support business transactions. It also supports smart contracts and membership

services, such that the blockchain can operated as a permissioned network. Nodes can be configured for read/write permissions, allowing certain nodes to write to the blockchain while some other nodes can only view what is in the blockchain. [15]

4.1.4 Comparison

	Bitcoin	Ethereum	Hyperledger
Governance	Bitcoin Developers	Ethereum Developers	The Linux Foundation
Permission restrictions	Permissionless	Permissionless	Permissioned
Consensus algorithm	Proof of Work	Proof of Work	Byzantine fault tolerance
Native currency	Bitcoin	Ether	N/A
Scripting ability	Limited	High High level language support: Solidity	High High level language support: Go
Mining reward	Yes	Yes	No

Figure 4.1 Comparison between Bitcoin, Ethereum and Hyperledger

After comparing the three major options of blockchains available, Bitcoin has a limited customization ability, hence firstly rejected in this project. Ethereum and Hyperledger both allows a higher level of customization. Hyperledger, designed for implementing cross-industry business logic, is more suitable in our situation as there are less limitations compared to Ethereum. However, the consensus algorithm in both platforms cannot be

modified easily. To modify the consensus algorithm, an extensive editing on the source code of the consensus algorithm is required. In this project, the main purpose is to demonstrate the ability of using the blockchain for time-keeping. To streamline the development process, I have chosen to deploy my own blockchain.

4.2 Golang

In this project, I have chosen Golang as the language of development. It is developed by Google in 2009 and is designed to improve programming efficiency. The language is static typed (like C++) and designed with ease of use (like Python). It is also a great language for system design as support for various web frameworks and concurrency is built in. [16]

4.3 Implemented functions

The following are the major functions included in the implementation:

- `func newBlock(prevBlock Block) Block`

  It takes in the previous block and extracts the index number and the hash value. It then creates a new block by filling in the corresponding values and hash the block to attempt to match the Proof of Work pattern.

- `func checkHash(hash string) bool`

  It takes in the hash of the block as a string and compare the string to the required pattern. A Boolean value is returned.

- `getHash(block Block) string`

  It is called by the `newBlock` function and generates the hash value of the block. SHA256 is used here and the string of the hash value is returned.

- `checkValid(generatedBlock Block, prevBlock Block) bool`

  When a new block is being received, this function is being called to check if the block is valid. It checks the validity of the block by 4 requirements: 1. The index of the block should be 1 larger than the previous block; 2. The previous hash of the block should match the hash of the previous block; 3. `getHash` of the block should

be the same as the hash field of the block; and 4. The timestamp of the block is within the acceptable delay range. In this implementation for demonstration purpose, this value is configured to 5 seconds.

- `func replaceChain(newBlocks []Block)`

  It checks for the length of the blockchain. If the chain is longer than the Timechain record in the machine, the local chain will be replaced by the longest chain.

In order to test the functionality of consensus based on the timestamp, I have added support for a manual clock deviation. It accepts a manual input of deviation value and adds it to the timestamp being broadcasted to other nodes. If the deviation is larger than 5 seconds, other nodes will not accept this block.

5. Future Development

5.1 Round-trip delay calculation

In the current implementation, it is assumed that all nodes are very close to each other, and hence the network latency is negligible. However, in reality, the network delay in the worst scenario can reach tens of seconds. Not only the time is unable to get synchronized, valid blocks are also being rejected, creating unnecessary computations and network communications. In order to take the delay into consideration, I will be utilizing the blockchain announcement in the network to achieve a delay calculation.

The idea is whenever a new block is generated and being announced, it includes the timestamp of the node time of the block leaving the node (t_0). When another node receives the block, it logs the local time of receive (t_1). It then updates the local time accordingly. After some time, the node announcing the block broadcasts its whole chain of blocks to all other nodes. The broadcast includes the timestamp of the broadcast leaving the node (t_3). When another node receives the broadcast, it logs its local time again (t_4). Notice that during the first synchronizing, t_1 is corrected to t_0 already. It is assumed that the time on the two nodes are synchronized. The resulting difference between t_3 and t_4 is solely due to the network delay.

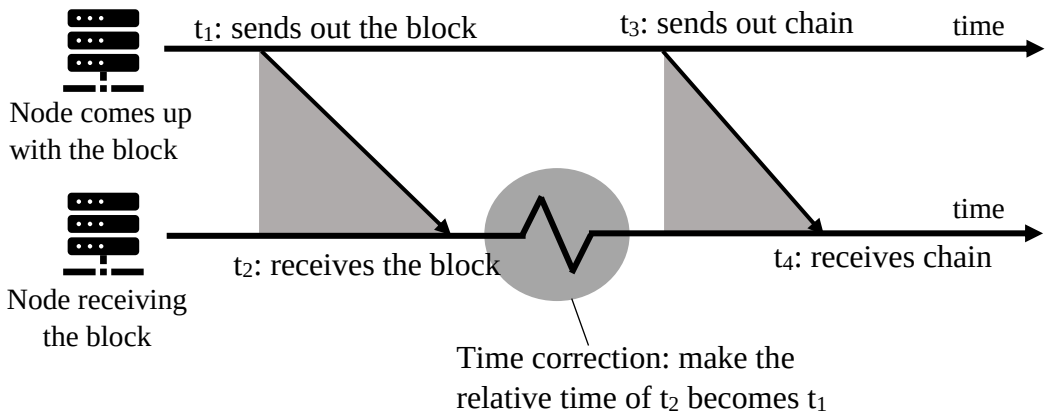

Figure 5.1 Calculation for network delay.

In the calculation of delay, instead of sending a request for time, the node only receives block and chain broadcasts from other nodes. Only down stream network communication is involved. Hence, the delay is given by the difference between t_3 and t_4. Also, as the time is already synchronized without taking care of the delay, t_4 is strictly larger than t_3.

$$\theta = t_4 - t_3$$

As a result, even though multiple time sources can publish their time to the blockchain and being accepted within the allowed threshold, the calculation of network latency is simpler than NTP. Therefore, when the chain grows, the time in all nodes will converge.

5.2 Browser Plugin

Another future development for this project is a browser plugin which checks the validity of the certificates. In the modern browsers, a warning is often displayed if the certificate of a site is expired. This detection is based on the local clock of the machine. If there is a huge error on the local clock, say over a day, expired certificates may still be deemed valid. Therefore, the plugin works independently from the local machine clock. It acts as another layer of validity check but instead of checking the validity according to the local clock, it gets the time from the Timechain. This is a scenario where a safe, but less accurate time is more important than a super-accurate time.

5.3 2nd Term Schedule

December 2018 – January 2019	Investigation on how to calculate the network delay; test if the suggested method is accurate enough in the scenario.
January – Mid-February 2019	Implement the delay calculation.
Mid-February to Mid-March 2019	Development of browser plug-in.
Mid-March to April 2019	Testing on the functionality on the whole implementation.

6. Conclusion

In this final year project, through a set of research, I am aware that the widely-used protocol in time synchronization, the network time protocol, is indeed highly vulnerable to various attacks. With the increasing popularity on blockchain, it seems to be a great idea of providing timing service in a distributed network. I examined the idea behind the blockchain, with a more in-depth research on how the blocks are being generated and the consensus algorithm. I also examined the different blockchain frameworks available and end up decided to write up my own blockchain to achieve the time keeping purpose.

In the Timechain, the time is being kept on a distributed ledger across the network. All nodes can participate in maintaining the chain by providing its own time, as well as benefit from the chain of ensuring the local machine time is in sync with other nodes in the network. There are difficulties as the original idea of blockchain is not designed to be real-time, hence using the idea of blockchain for time keeping requires modifying the generation of the block and the consensus algorithm.

The implementation, although not completed, but demonstrates the ability of Timechain keeping an updated time and rejecting some nodes from appending an inaccurate time to the chain. Eventually, when the chain among all the nodes are in sync, a synchronized time in the whole network can be achieved.

7. Bibliography

[1] M. E. Acer, E. Stark, A. P. Felt, S. Fahl, R. Bhargava, B. Dev, M. Braithwaite, R. Sleevi and P. Tabriz, "Where the Wild Warnings Are: Root Causes of Chrome HTTPS Certificate Errors," October 2017. [Online]. Available: https://acmccs.github.io/papers/p1407-acerA.pdf.

[2] D. Mills, "Network Time Protocol (NTP), RFC 958," September 1985. [Online]. Available: https://tools.ietf.org/html/rfc958.

[3] D. Mills, "Network Time Protocol (Version 1) Specification and Implementation, RFC 1059," July 1988. [Online]. Available: https://tools.ietf.org/html/rfc1059.

[4] D. Mills, "Internet Time Synchronization: The Network Time Protocol," *IEEE Transactions on Communications,* vol. 39, no. 10, pp. 1482-1493, October 1991.

[5] D. Mills, E. J. Martin, J. Burbank and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification, RFC 5905," June 2010. [Online]. Available: https://tools.ietf.org/html/rfc5905.

[6] B. D. Esham, "Network Time Protocol servers and clients.svg," Wim, September 2013. [Online]. Available: https://commons.wikimedia.org/wiki/File:Network_Time_Protocol_servers_and_clients.svg.

[7] A. Geoff Huston, "Protocol Basics: The Network Time Protocol," *The Internet Protocol Journal,* vol. 15, no. 4, pp. 2-11, 2012.

[8] National Institute of Standards and Technology, "NIST Authenticated NTP Service," October 2017. [Online]. Available: https://www.nist.gov/pml/time-and-frequency-division/time-services/nist-authenticated-ntp-service.

[9] E. B. Haberman and D. Mills, "Network Time Protocol Version 4: Autokey Specification, RFC 5906," June 2010. [Online]. Available: https://tools.ietf.org/html/rfc5906.

[10] Network Time Foundation, "Encryption," August 2013. [Online]. Available: http://www.ntp.org/ntpfaq/NTP-s-algo-crypt.htm#AEN2590.

[11] A. Malhotra, I. E. Cohen, E. Brakke and S. Goldberg, "Attacking the Network Time Protocol," October 2015. [Online]. Available: http://www.cs.bu.edu/~goldbe/papers/NTPattack.pdf.

[12] NTP Pool Project, "The NTP Pool for vendors," November 2018. [Online]. Available: https://www.ntppool.org/en/vendors.html#ntp-pool-offer.

[13] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf.

[14] V. Buterin, "Ethereum White Paper," [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper.

[15] The Linux Foundation Projects, "About Hyperledger," [Online]. Available: https://www.hyperledger.org/about.

[16] The Go Programming Language, "Documentation," [Online]. Available: https://golang.org/doc/.