Taylor Manivanh
Dr. Li Feng
CS 481
September 17, 2018

The Byzantine Problem

Abstract

In order for a distributing system to be reliable, the system must be able to handle failures caused by conflicting information. Erroneous information can be traced back to differing systems that have been overlooked or malfunctioned. The basics of the problem can be illustrated by the Byzantine Generals Problem. Imagine there are several generals and their respective armies surrounding a city. All the generals must reach a consensus on deciding whether to attack or retreat. A common decision must be made between all generals. The Byzantine Problem, also known as the Byzantine Fault Tolerance (BFT), was proposed in 1984 along with solutions of how a computer can cope with the failure of one or more components. Over time, the Byzantine Fault Tolerance has been revised, modified, and transformed an innumerable about of times to be implemented into different softwares and programs. And as a result, there are several applications of the fault tolerance in today's world that are integrated into everyday life.

Outline

The Byzantine Generals Problem has paved the way for future advancements and continues to be one of the building blocks behind countless of technologies to this day. This paper recounts the discovery and resolution to the original Byzantine Generals Problem. As well as, the ongoing role and importance of Byzantine Fault Tolerant protocols in our lives.

The paper is organized as follows. Section 1 introduces Byzantine Fault Tolerant protocols. Section 2 establishes background information about the Byzantine Generals Problem. Section 3 describes the multiple ways Byzantine Fault Tolerant protocols can be implemented for differing reasons. Section 4 reveals how BFT protocols are still being used in today's technology. Section 5 evaluates the effect of the protocols and concludes the paper.

1. Introduction

In terms of a distributed computing system, the Byzantine Problem posed as a difficult obstacle when trying to reach a consensus between the machines in the system. With support from NASA, Leslie Lamport, Robert Shostak and Marshall Pease of SRI International introduced the Byzantine Generals Problem in an academic journal in 1982.[13] The journal explored the main issues of the BFT as well as the several subproblems involved with the original issue. Lamport, Shostak, and Pease included solutions and proofs specific to the constraints in each of

the different situations. Therefore, successfully resolving the Byzantine Fault Problem and demonstrating how these solutions can be implemented into a reliable distributing system.

Since the emergence of the Byzantine Fault Problem, there have been a countless number of programs created with the focus of effectively handling Byzantine Faults. The rise of these programs began with Miguel Castro and Barbara Liskov in 1999, who created the Practical Byzantine Fault Tolerance (PBFT).[5] After which, more BFT protocols were introduced, each looking for ways of improving and optimizing how computers handle component failures. Some of the most notable BFT protocols include: Query/Update (Q/U)[1] and Hybrid Quorum (HQ), which deals with the enhancement of performance and cost issues; Aardvark[6] and Redundant Byzantine Fault Tolerance (RBFT)[3], which concentrates on robust issues; and Adapt[2] which focuses on creating an adaptive BFT protocol. Outside of BFT programs, there are additional advancements such as UpRight[7], which is a library for fault tolerant replication.

With the introduction of various programs and softwares comes the implementation of Byzantine Fault Tolerance protocols in everyday life. The most notable use of BFT protocols in today's climate can be seen within the use of Bitcoin and Blockchain. In addition to playing a role within cryptocurrency, BFT protocols have also been put to use in aircraft systems. Decades after the original Byzantine Fault problem was pioneered, new protocols are still being developed and integrated into modern day technology.

The original Byzantine Generals Problem posed a great obstacle in terms of a distributed system. However, a solution was also presented with the problem. Even though the solution was not the most efficient nor optimal, the problem was still resolved. As a result, several different protocols were developed to strive for a better solution for tolerating Byzantine Faults. And to this day, new methods are being developed and continually incorporated into today's world. The Byzantine Problem greatly influenced distributed systems in 1984, and has continued to play a part in the technology being used today.

2. The Byzantine Generals Problem

The Byzantine Problem occurs within distributed computer systems, or systems that are comprised of several computers with several software components running on them. Distributing computing was established to make a network of computers function as a single computer. Distributed systems allow several computers to work in sync; and when one computer malfunctions, productivity does not stop due to the fact that other computers can complete the task of the downed computer. Even though the system possesses these capabilities, the complexity of the Byzantine fault or failure, can still cause significant errors within a network of computers.[12]

First and foremost, the difference between a Byzantine fault and a Byzantine failure must be established. Any fault that transmits information to one observer, then transmits differing information to another observer, is to be considered a Byzantine fault. In turn, a Byzantine failure occurs when a system cannot perform its tasks until a consensus is reached in said

failure.[9] When working within a distributing system, Byzantine faults and failures are among the most difficult faults to detect. Traditional failure detection systems can be tricked by Byzantine failures. This occurs when a corrupted node can appear to be functioning by presenting arbitrary information. And as a result, a Byzantine Fault Tolerance cannot be achieved until a majority of the computers within a system can reach an agreement. Furthermore, these errors are not solely caused by human interaction; errors can be a consequence of an electrical shortage or inaccuracy.

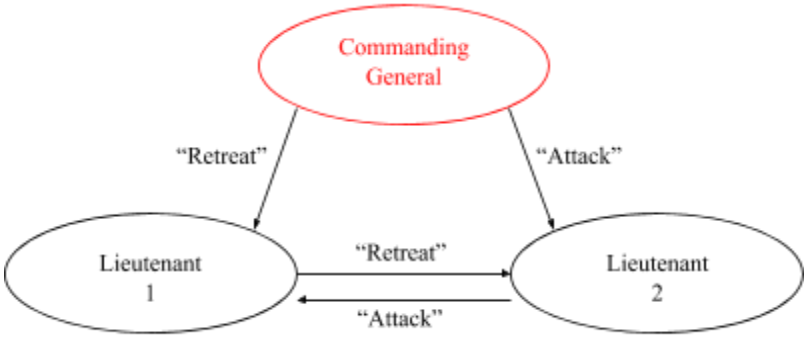
2.1 First Solutions to the Byzantine Generals Problem

Returning to Lamport, Shostak, and Pease's original definition of the Byzantine failure, the problem involves two conditions: oral messages and written signed messages. The paper revealed the following characterizations of oral messages: "A1. Every message that is sent is delivered correctly. A2. The receiver of a message knows who sent it. A3. The absence of a message can be detected."[13] These assumptions can handle the hypothesis of a "traitorous general", or a general who will instruct one lieutenant to attack and then instruct another lieutenant to retreat. Assumptions one and two will resolve a traitorous general while assumption three will establish a default order - which in this case would be to retreat. For this particular hypothesis, the solution states that for m number of traitorous generals, there is a required $3m+1$ generals who are exchanging information for $m+1$ rounds. Due to the fact that a consensus cannot be reached with only three generals, the solution demands for $3m+1$ generals to be in communication.

In contrast, with written, signed messages, the first three assumptions, A1-A3, apply, as well as: "A4. A loyal general's signature cannot be forged, and any alteration of the contents of his signed messages can be detected. A5. Anyone can verify the authenticity of a general's signature."[13] With these given conditions and assumptions, a consensus can be reached with only three generals. For instance if two of three generals are loyal, they will come to a consensus of which plan of action should be taken. Therefore, for m traitors, there needs to be $2m+1$ generals with $m+1$ rounds to exchange information.

Conversely, if the assumption A4 - that states a general's signature cannot be forged - were to be ignored, an additional issue of forged messages arises. Given that the commanding general is loyal, the hypothesis can be solved so long as there are m number of generals and t number of traitors in m, and $m>3t$. The communication between these generals must also be synchronous, or involve real-time communication. However, if the commanding general were to be a traitor (traitorous commanding general marked in red in Figure 1), then finding a solution with only one commanding general and two lieutenants proves to be impossible. If a general were to tell one lieutenant to attack, and tell the other to retreat, then when the two lieutenants exchange information, they will be unable to determine who is the traitor. The lieutenants are unsure whether the commander is a traitor or if the other lieutenants is the traitor and is forging the commander's signature. [Figure 1]

Figure 1 **Byzantine Generals Problem: Traitorous General**



Lamport, Shostak, and Pease posed the question of the Byzantine Generals Problem as well as provided a feasible solution. There is no doubt that the solutions presented satisfied the problem, however the aforementioned methods were not the most efficient. Therefore, a new wave of programs and protocols emerge to remedy this fact.

3. The Expansion of Byzantine Fault Tolerance Protocols

When the Byzantine Generals Problem was initially introduced, there several techniques that attempted to create improved protocols, but few made significant progress. Eventually in 1999, Miguel Castro and Barbara Liskov shared the algorithm for Practical Byzantine Fault Tolerance (PBFT). Castro and Liskov published "Practical Byzantine Fault Tolerance and Proactive Recovery" in the *ACM Transactions on Computer Systems*[5], and explained the goal and algorithm behind the PBFT. The "Practical" BFT was created due to the fact the previous techniques that implemented agreement and replication, were inefficient or assumed synchrony. The PBFT was notable because the algorithm can tolerate Byzantine faults by utilizing state machine replication (SMR). According to their journal, state machine replication is the replication of servers, and the coordination of client interactions with said replicated servers.[5] Therefore, the Practical Byzantine Fault Tolerance efficiently tolerates Byzantine faults within asynchronous networks with the aid of the SMR and thus can circumvent denial-of-service attacks.

3.1 Query/Update and Hybrid Quorum

Following the emergence of the PBFT, several varying tolerance protocols were created. The next two protocols: Query/Update and Hybrid Quorum, were both introduced to boost performance of the BFT. Query/Update is a quorum based algorithm that provides better fault scalability, as opposed to the SMR that relies on an agreement-based system. The algorithm calls for $5f+1$ servers in order to tolerate f number of Byzantine faulty servers.[1] Although the Q/U requires multiple servers, this is a minor trade off because the cost of server failures will not decline, but the cost of servers will. When the number of fault tolerated increases, the fault

scalability achieved by Q/U allows performance to fall at a gradual rate rather than immediately. Similarly, Hybrid Quorum is a lightweight quorum based protocol when there is no contention - or disagreement - between computers in a system. But, when there is contention, HQ implements the Byzantine Fault Tolerance. For *f* number of Byzantine faults, HQ uses *3f+1* replicas to tolerate the faults; in turn, allowing for scalable performance when *f* increases.

3.2 Aardvark and Redundant Byzantine Fault Tolerance

Next, the BFT protocols: Aardvark and Redundant Byzantine Fault Tolerance, were established to handle robust issues. In order to be robust towards failures, Aardvark designed and implemented three stages that include: client request transmission, replica agreement, and primary view change.[6] During execution, Aardvark does sacrifice some performance, but makes significant improvements during uncivil executions, or executions where the network remains functioning, but the clients or servers may be faulty. Correspondingly, RBFT was also created with the intention of combatting robust issues within the BFT. The RBFT algorithm executes multiple instances of the BFT protocol in parallel. During these executions, a fairness mechanism is implemented between clients, latency of requests are monitored, and malicious primaries are detected.[6] As a result, the Redundant Byzantine Fault Tolerance can remain robust against faults with only a small loss in performance.

3.3 Adapt and UpRight

Additionally, the protocol Adapt was formed with the objective of creating an adaptive and abortable Byzantine Fault Tolerance protocol. Adapt applied a dynamic switching method that utilized machine learning predictions.[2] At each execution, an evaluation score is given to each running protocol. Adapt will then switch to the protocol with the highest evaluation score. With this method, Adapt outperforms other protocols under dynamic workloads.

One of the most notable Byzantine Fault related derivations is called UpRight. UpRight is a new library for fault tolerant replication, and was created to make a Byzantine Fault Tolerant alternative for crash failures that occur within cluster services.[7] The library's main concern is its ability to be adopted and integrated by already existing application, leaving performance as a less important goal. Objectives of UpRight include adopting "safe-guard availability" and "safe-guard correctness"[7]. In simplified terms, the objectives work towards keeping systems up and running with accuracy and correctness. As a library, UpRight can be utilized to tolerate Byzantine Faults in cluster systems that already exist. On the other hand, Upright also makes implementing Byzantine Fault Tolerant protocols easy if the situation calls for building a completely new service. Although UpRight stated that performance was a secondary concern to implementation, the performance of the library is comparable to that of original BFT protocols with added robustness.

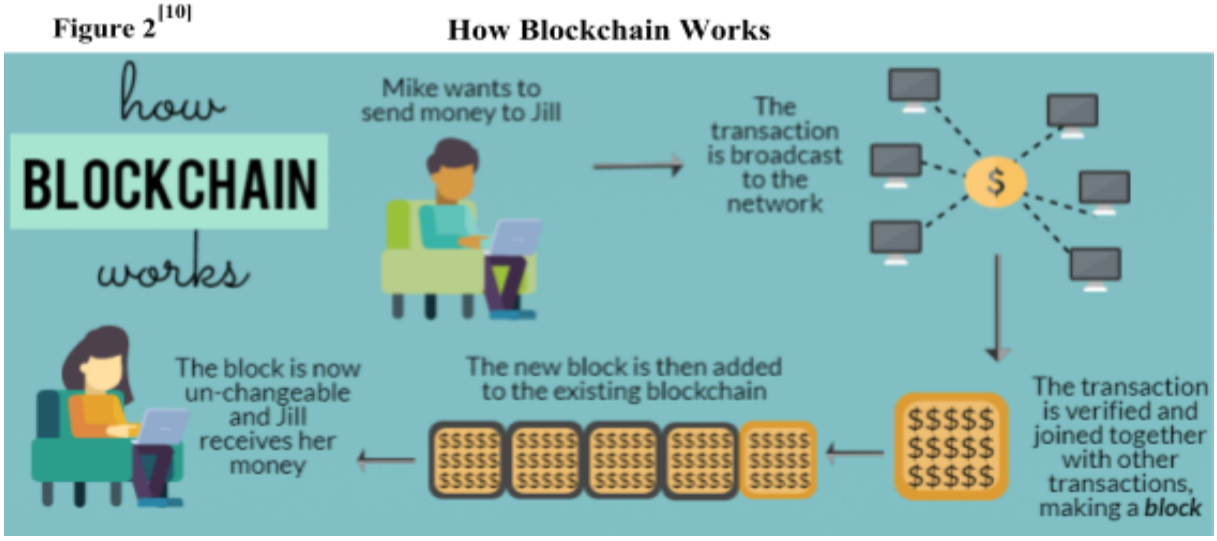
The previously mentioned programs and protocols are the most prominently known fault tolerant systems. A countless number of additional protocols exist not including the ones already

mentioned, and each of them accomplishing a wide variety of tasks. Thus, the next section will cover the breadth and impact of Byzantine Fault Tolerant protocols in our day to day lives.

4. Relevance and Implementation of Byzantine Fault Tolerant Protocols

The various implementations of Byzantine Fault Tolerant protocols have expanded beyond the original use within distributed computing systems. One of the most prevailing uses of Byzantine Fault Tolerance exists in Blockchain. A blockchain is a component of state machine replication; where a service has the ability to maintain a state and allow clients to send operations.[4] These client sent operations change the state and is capable of producing an output. Through a distributed protocol, Blockchain imitates a trusted service, allowing this interaction to occur. Nodes that are linked on the Internet are responsible for running the distributed protocol. Then, an asset will be created by the service, where all of the nodes will have a stake. At this point the BFT concept of the consensus reappears. All of the nodes will work alongside each other to keep the circuit running, but the nodes do not trust one another. With all of these components, a basic Blockchain is established. [Figure 2]

Figure 2[10]



4.1 Permissionless and Permissioned Blockchain

However, the complexity of a Blockchain can increase when dealing with the concept of permissions. When a Blockchain is permissionless, new opportunities can arise, such as the development of cryptocurrency. A prominent example is the introduction and proliferation of Bitcoin. Bitcoin and cryptocurrencies in general, operate on a "proof-of-work" system. Any individual is able to operate a node and spend CPU cycles so long as they are able to show this "proof-of-work".[14] Bitcoin is based on a unorthodox method of the Byzantine consensus protocol. "Cryptographic puzzles keep a computationally bounded adversary from gaining too much influence".[14] The ideals behind the Byzantine consensus is a major aspect in the success of Bitcoin today.

Contrastingly, when a Blockchain is permissioned, the circumstances are vastly different from a permissionless Blockchain. With permission, the Blockchain model has the authority to control which nodes are granted access to participate in validation.[4] Additionally, all of the involved nodes have established identities and form a consortium[4], association or conglomerate, that collaborate in order to reach a consensus.

In recent years, progress has been made to work on improving the traditional Blockchain. For instance, the Hyperledger Project is an effort to develop an open-source ledger framework and code base. The goal of the project focuses on allowing the ledger to be distributed at an enterprise-level. The Linux Foundation concentrated on this objective in 2016 when the project was launched. And in the last two years, the Linux Foundation have worked on identifying and generating a standard platform for distributed ledgers that are cross-industry.[4]

4.2 Byzantine Fault Tolerant Protocols in Aircrafts

With a wide range of versatility, Byzantine Fault Tolerant protocols also make appearances outside of traditional computer systems. In fact, these fault protocols can even been seen within aircrafts and flight systems. One example includes the PVS verification system developed at SRI International. The organization wanted to create a system that developed formal methods powerful enough to be applied on an industrial scale.[15] As a basis, digital flight control systems must be "extremely improbable" for passenger aircrafts; the rate of failure must be less than 10^{-9} per hour. Therefore, a great deal of fault tolerance and redundancy management is needed. In order to reach this objective of ultra-high reliability, the system must be able to organize this redundancy and fault tolerance. But, this can be challenging and if done even remotely wrong, this failure can become the primary source of failure within the whole system.

In attempts to solve this issue, the SIFT general approach was developed. The approach begins with several independent channels that are able to make computations; while operating in approximate synchrony. Then, with one source of data, information is distributed to all channels in such a way that the system in resistant to Byzantine faults. All of the valid, functioning channels will receive the same data around the same time. Each channel possesses "sensor-conditioning and diagnosis" and are able to deal with failed sensors when they occur. And, if a failed channel were to appear in the system, the channel would be masked by majority voting.[15] The SIFT method is a functioning approach to solving the organization of redundancy and Byzantine faults within systems; however, the SIFT method suffered great performance issues.

Failure of the SIFT approach performance-wise is a contributing factor to the creation of the PVS verification system. Researchers at SRI International found that for the PVS to be successful, the system must have fault tolerant architecture to withstand simultaneous Byzantine faults. In turn, the organization achieved this by reconfiguration and removing faulty channels; causing redundancy to reduce. The act of reconfiguration is a monumental feat due to the complexity of the problem and the potential to cause even more design faults if implementation

is done incorrectly. However, by the end of the project, the PVS was established as a powerful verification system that: embed multiple constraints into types, kept the main specification uncluttered, and ran an efficient consistency check.

All in all, the influence of BFT protocols is increasingly prominent in the modern world. And with new ways of implementation being developed, the use of these protocols will continue to persist.

5. Conclusion

Introduction of the Byzantine Generals Problem, paved the way for opportunities and advancements far beyond the original goal of the problem. When Leslie Lamport, Robert Shostak, and Marshall Pease introduced the idea of the Byzantine fault and failures, this idea of consensus within a distributed computing system became a template to deal with similar issues within other systems. Although Lamport, Shostak, and Pease provided a solution to the Byzantine Generals Problem, the method to reaching this consensus was not the only way to solve the problem. By presenting the original problem, solution, and explanation for Byzantine faults and failures, the men at SRI International opened the issue for further interpretation and expansion.

As a result, numerous programs, libraries, softwares, etcetera were created with the intention of improving the original Byzantine Fault problem. Modifications began with the Practical Byzantine Fault Tolerant protocol. The PBFT aimed for efficiency while trying reach a consensus within an asynchronous system. In order to achieve this, developers Castro and Liskov utilized state machine replication to combat software attacks and protect information on servers. Then, there were programs such as Query/Update and Hybrid Quorum that strived for increased performance. Q/U was created with the objective of increasing the tolerance of faults without sacrificing performance. With an agreement based system, Q/U was able to provide a fault scalable service with minimal tradeoffs in the number of servers required. Similarly HQ is also fault scalable to increase overall performance. But, HQ only provides a new implementation when the number of faults increases. Next, Aardvark and Redundant Byzantine Fault Tolerance both focused on robustness. Through replicated agreement and parallel executions respectively, Aardvark and RBFT are able make systems robust to failures without sacrificing a great deal of performance. Adapt is another program that centers on the implementation of Byzantine Fault Tolerant protocols. The program dynamically switches the BFT system to make the system more adoptive within a dynamic environment. Lastly, UpRight was a library developed to favor the integration of Byzantine Fault Tolerant protocols into existing or even in completely new systems. All of these programs, and countless others are examples of the wide range that the original Byzantine Fault Tolerance problem has grown into.

Beyond the growth of Byzantine Fault Tolerant protocols, the application and implementations of these protocols have also broadened and diversified. Utilization of BFT protocols can be seen within Blockchain and the ideas of consensus over nodes on the Internet.

Through Blockchain new opportunities arise such as cryptocurrency. Bitcoin is one of the leading forms of cryptocurrency today and a "proof-of-work" based consensus is derived from Blockchain. There are also steps towards improving Blockchain such as the Hyperledger Project which strives to create a distributed ledger at an enterprise level. Furthermore, Byzantine Fault Tolerance protocols can also been seen within aircrafts and flight systems. The PVS verification system in commercial passenger aircrafts utilize BFT protocols to reduce redundancy and withstand more simultaneous Byzantine faults.

5.1 Advantages and Disadvantages of Byzantine Fault Tolerant Protocols

Since the establishment of the Byzantine Generals Problem, there have been numerous progressive steps towards the advancing of new methods and protocols. Byzantine Fault Tolerant protocols are useful in today's world due to the fact that now these protocols can be easily applied and implemented. Regardless of the environment or circumstance, with libraries such as UpRight, fault tolerant protocols can be integrated into different types of systems with little conflicts. Additionally, the Byzantine Fault Tolerant protocols can also be modified to meet specific requirements. With Aardvark, RBFT, HQ, Q/U, and other programs, certain aspects can have priority over others. For instance, robustness, performance and adoption. BFT protocols also continue to remain relevant with a role in cryptocurrency and electronic commerce in the economy today.

While Byzantine Fault Tolerant protocols provide multiple positives, there are also downfalls. None of the solutions or current protocols were developed overnight. Often times, the methods to achieving and polishing these BFT protocols were exceptionally complex and intricate. Even though the protocols can tackle issues such as robustness and performance, there is also no one program that can satisfy all criteria. In order to improve one facet, there are tradeoff in other areas of the protocol. The evolution of Byzantine Fault Tolerant protocols from the original Byzantine Generals Problem has been substantial, but there are constantly ways to improve and enhance current methods and designs.

5.2 Final Remarks

Finally, the presentation of the Byzantine Generals problem was revolutionary for computers within a distributed system. And in a matter of years, an increased amount of Byzantine Fault Tolerant protocols were popularized and implemented in clustered, dynamic, and other systems. The basic idea of a consensus and a system coming to an agreement, has now become instrumental in countless aspects in society. Fault tolerances can be seen ranging anywhere from economics to transportation. Thus, the mutual agreement of Byzantine Fault Tolerance protocols are more prominent than they appear and will continue to play an instrumental role as technological advances are made.

Bibliography

[1] Abd-El-Malek, Michael, et al. “Fault-Scalable Byzantine Fault-Tolerant Services.” *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles - SOSP '05*, Oct. 2005, doi:10.1145/1095810.1095817.

[2]Bahsoun, Jean-Paul, et al. “Making BFT Protocols Really Adaptive.” *Making BFT Protocols Really Adaptive - IEEE Conference Publication*, IEEE Xplore Digital Library, 20 July 2015, ieeexplore.ieee.org/document/7161576.

[3]Aublin, Pierre-Louis, Sonia Ben Mokhtar, and Vivien Quéma. "RBFT: Redundant Byzantine Fault Tolerance." *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*. IEEE, 2013.

[4]Cachin, Christian. "Architecture of the Hyperledger Blockchain Fabric." *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*. Vol. 310. 2016.

[5] Castro, Miguel, and Barbara Liskov. “Practical Byzantine Fault Tolerance and Proactive Recovery.” *ACM Transactions on Computer Systems*, vol. 20, no. 4, 2002, pp. 398–461., doi:10.1145/571637.571640.

[6]Clement, Allen, et al. “Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults.” *USENIX Association*, NSDI ’09: 6th USENIX Symposium on Networked Systems Design and Implementation, 2009.

[7]Clement, Allen, et al. “Upright Cluster Services.” *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles - SOSP '09*, 11 Oct. 2009, doi:10.1145/1629575.1629602.

[8] Cowling, James, et al. “HQ Replication: A Hybrid Quorum Protocol for Byzantine Fault Tolerance.” *Proceedings of the Twenty-First ACM Symposium on Operating Systems Principles - SOSP '07*, 2007.

[9] Driscoll, Kevin, et al. “Byzantine Fault Tolerance, from Theory to Reality.” *Lecture Notes in Computer Science Computer Safety, Reliability, and Security*, 2003, pp. 235–248., doi:10.1007/978-3-540-39878-3_19.

[10]“Get Your Head Around Blockchain & Consensus.” *UtterCrypto*, 20 Jan. 2018, *www.uttercrypto.nz/blog/blockchain-consensus.*

[11] Ghosh, Debraj. “How the Byzantine General Sacked the Castle: A Look Into Blockchain.” *Medium*, Medium, 5 Apr. 2016, *medium.com/@DebrajG/how-the-byzantine-general-sacked-the-castle-a-look-into-blockchain-370fe637502c*.

[12] “IBM Knowledge Center.” *The Analytics Maturity Model (IT Best Kept Secret Is Optimization)*, IBM Corporation, www.ibm.com/support/knowledgecenter/en/SSAL2T_7.1.0/com.ibm.cics.tx.doc/concepts/c_wht_is_distd_comptg.html.

[13] Lamport, Leslie, et al. “The Byzantine Generals Problem.” *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, 1 July 1982, pp. 382–401., doi:10.1145/357172.357176.

[14]Miller, Andrew, and Joseph J. LaViola Jr. "Anonymous Byzantine Consensus from Moderately-Hard Puzzles: A Model for Bitcoin.", *http://nakamotoinstitute.org/research/anonymous-byzantine-consensus* (2014).

[15]Owre, Sam, et al. "Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS." *IEEE Transactions on Software Engineering* 21.2 (1995): 107-125.