

# ByzGame: Byzantine Generals Game

James R. Clavin, Pradeep M. Prakash, and Sisi Duan  
{jclavin, pmargas1, sduan}@umbc.edu  
University of Maryland, Baltimore County

## ABSTRACT

Byzantine Fault Tolerance (BFT) has gained renewed interest due to its usage as the core primitive in building consensus in blockchains. One of the primary challenges with BFT is understanding the theory behind it. Numerous BFT protocols have been proposed; unfortunately some of them have had correctness issues. We present ByzGame, a web application that uniquely connects a frontend visualization to a backend BFT implementation, and makes both BFT consensus theory and implementation more understandable. Our evaluation among two groups of students demonstrates that ByzGame can greatly increase the effectiveness in teaching and learning both fundamental and advanced topics related to BFT.

## CCS CONCEPTS

• **Computer systems organization** → **Reliability; Availability; • Human-centered computing** → **Visualization design and evaluation methods;**

## KEYWORDS

Byzantine generals problem, Byzantine fault tolerance, consensus

### ACM Reference Format:

James R. Clavin, Pradeep M. Prakash, and Sisi Duan. 2020. ByzGame: Byzantine Generals Game. In *The 14th ACM International Conference on Distributed and Event-based Systems (DEBS '20)*, July 13–17, 2020, Virtual Event, QC, Canada. ACM, New York, NY, USA, 4 pages. <https://doi.org/10.1145/3401025.3401739>

## 1 INTRODUCTION

Byzantine fault-tolerant (BFT) is the only generic approach for masking arbitrary failures and malicious attacks. In recent years, with the emergence of blockchains, BFT has gained renewed interest. Specifically, BFT is the core primitive for building consensus in different types of blockchains [10, 15, 19].

Consensus protocols are known to be difficult to understand and implement, as observed by both industry [6] and academia [16]. In 2007 the authors of [6] commented that ‘the fault-tolerance computing community has not developed the tools to make it easy to implement their algorithms.’ Seven years later, the Raft protocol [16] was created, with the motivation of being more understandable than Paxos [13], the state-of-the-art crash fault-tolerant (CFT) protocol

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).  
*DEBS '20, July 13–17, 2020, Virtual Event, QC, Canada*

© 2020 Association for Computing Machinery.  
ACM ISBN 978-1-4503-8028-7/20/07...\$15.00  
<https://doi.org/10.1145/3401025.3401739>

that tolerates only benign failures. Raft’s enhanced understandability has made it popular enough that some blockchain systems have adopted it, or a BFT version of it, despite the fact that Raft is only a CFT protocol, and that the BFT version of it is not correct [4, 10].

Compared with CFT, BFT is more difficult to understand and implement due to the additional steps that are necessary to tolerate arbitrary failures. Despite numerous efforts to make it more practical, a lot of existing systems (e.g., blockchains) are still using incorrect protocols [4]. Due to the complexity in reasoning about the correctness, some prior state-of-the-art protocols had correctness issues [1].

BFT protocols are typically represented as time-space diagrams [12] to illustrate and ‘visualize’ the workflow. After working with students and developers with different expertise levels, we find that a lot of details are not captured well and cannot be easily visualized, e.g., view change in partially synchronous BFT protocols [9]. As a result, the correctness of such protocols cannot be easily ensured even for researchers with deep expertise [1]. Therefore, having a way to visualize such protocols would be beneficial. There have been a few blockchain or consensus protocols that provide certain forms of visualization [2, 16, 18]. Unfortunately, all of the known ones either focus on cryptocurrency or are simple web-based visualizations. Furthermore, none of them is easily ‘understandable’.

To cope with these challenges, we developed ByzGame, a visualizable and understandable BFT. Our motivation is not to propose a new protocol that is more intuitive, but rather to use a novel approach to provide an interface that makes the underlying concepts and implementation understandable. ByzGame is a web application that connects a visualization to a backend BFT consensus implementation. ByzGame visualizes the classic *Byzantine Generals Problem* [14], and displays the generals as *cities* on a global map. The cities’ purpose is to reach agreement to *attack a target city*. We view ByzGame as a unique component in the BFT and distributed systems areas which can help students and developers of various BFT expertise learn both the theory behind BFT and how to correctly implement it. We have deployed ByzGame on Amazon EC2 and used it both in classes and in the lab. Our evaluation results validated that ByzGame can help students grasp BFT concepts.

## 2 RELATED WORK

Numerous BFT protocols have been proposed [3, 5, 7, 8, 17, 20]. Most BFT protocols assume partial synchrony [7, 17, 20]. Such protocols may have zero throughput in an asynchronous environment [11]. In comparison, asynchronous protocols [3, 8] do not suffer from this problem. In ByzGame, we build the interface based on BFT-SMaRt [17], a protocol that assumes partial synchrony. Our system can also be adapted to use asynchronous protocols.

Partially synchronous protocols are usually leader-based where a designated replica is the leader, and usually includes two major

components: *normal operation* and *view change*. During normal operation, the leader *proposes* the sequence of certain client request(s) and replicas agree on the sequence. When the leader is suspected to be faulty, a new one needs to be elected via *view change*.

To the best of our knowledge, no BFT protocol has been created with understandability as a first class property. The comprehensibility of BFT consensus is challenging, not only for beginners, but also for experienced researchers. As a result, some state-of-the-art BFT protocols were later on found to have correctness issues [1]. Therefore, a way to achieve better understanding of how the components of a BFT protocol function from a distributed systems perspective is needed, especially by beginners.

### 3 BYZGAME: THE BYZANTINE GENERALS GAME

We have built ByzGame, a web-based application that connects a web frontend with a BFT implementation, so that users can directly configure the replicas, run the protocol, and visualize the message flow. Users can do real world tests of the theory of BFT consensus, learn the basic concepts about BFT, examine the BFT implementation, and identify issues with the protocol. As illustrated in Figure 1, the main interface has three components: *user configuration*, *visualization*, and *system log*. The user configuration (left panel) allows users to configure the system and manage the replicas. The visualization (top right panel) visualizes a global map where replicas are cities. Correct cities need to agree on a target city to *attack*. The system log (bottom right) presents the log from the underlying BFT implementation and shows the workflow of the consensus.

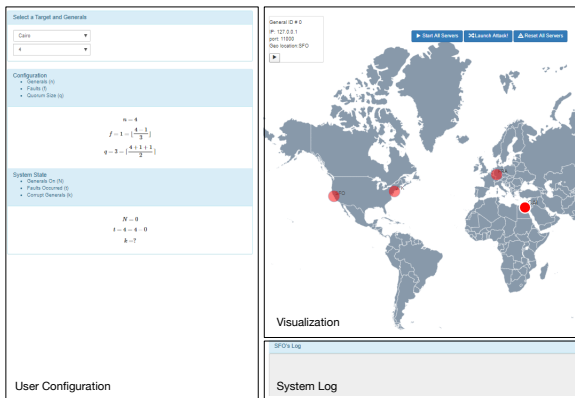


Figure 1: The main ByzGame interface.

#### 3.1 User Configuration

ByzGame presents two system configuration inputs to users:  $n$ , the number of replicas - or *generals*; and the *target* - or the city against which the generals launch their attack. Upon selection of  $n$ , the system state variables for faults allowed, or  $f$ , and quorum size, or  $Q$ , are calculated and shown to the user, as shown in Figure 2. The user also needs to select the target city where replicas will run the consensus protocol to agree on whether they will attack the city.

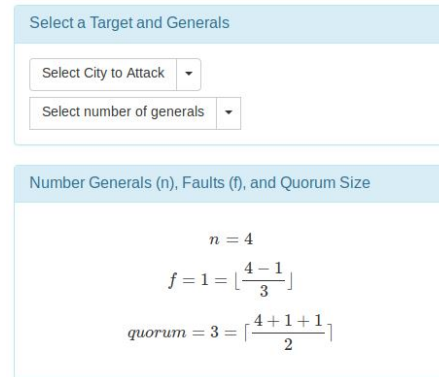


Figure 2: ByzGame after user selects  $n = 4$  for the number of generals.  $f$  and  $quorum$  automatically recalculate when  $n$  changes.



Figure 3: ByzGame mimicking replicas as cities. One city is the target, as stated in the Byzantine Generals Problem

#### 3.2 Visualization

A user can select up to  $n = 38$  replicas. After  $n$  is selected, the replicas are placed on a global map, as shown in Figure 3. The global map will be re-generated each time a user chooses a different  $n$  from the user configuration. After the global map is generated, the user can then issue the following commands.

- **START** initializes all servers in the system. The backend codebase automatically changes the configuration, clears all the logs, and starts all consensus nodes.
- **STOP** stops all servers in the system. The backend codebase automatically stops all consensus nodes. System configuration and logs remain the same and will not be deleted, unless a user restarts the servers.
- **LAUNCH** gives the order to "Launch Attack". Replicas start to agree on *whether they will launch an attack on the same city*.
- **RESET** stops all the running servers and puts the system state to the default setting where  $n = 4$  and  $f = 1$ .

A user can also configure the replicas to run on other machines as illustrated in Figure 4. To run ByzGame on other machines, all the dependencies have to be installed and the nodes must have opened the corresponding ports for communication.

After configuration, a user can select START to initialize all the replicas via the BFT codebase in the system. The leader, if one exists, shows on the map as a yellow dot. If the user selects a city to attack, the LAUNCH command sends a client request to the replicas, who return either a 1 indicating they will *attack*, a 0 if they will not, or are unresponsive. If the replicas reach a consensus, the interface animates either an attack against, or retreat from, the target city, as in the original Byzantine generals problem [14]. An animation of message flow is visualized in the map as shown in Figure 5.

System logs at the nodes are accessible to illustrate message flow in the protocol. When failures occur, a user can search the logs to identify the cause. The user can monitor each replica's progress by clicking on their node on the map to see their log, as in Figure 6.

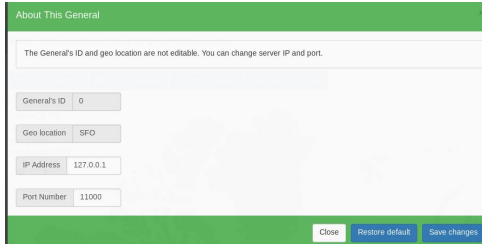


Figure 4: Update-able configurations for replicas



Figure 5: ATTACK animation

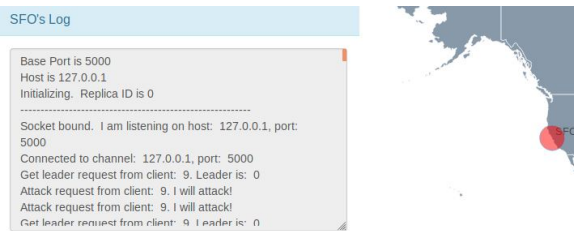


Figure 6: Replica log example

### 3.3 General Start / Stop

If a general is stopped, all the animation from and to the corresponding city is ceased and the configuration panel shows that the the number of running generals is zero, and whether the number of failures has exceeded  $f$ . At the system level the animated attack also ceases if a valid quorum no longer exists (i.e., there do not exist at least  $\lfloor \frac{n+f+1}{2} \rfloor$  running nodes in the backend) and continues if one does. In other words, when there are more than  $f$  failures in the system, a user will not be able to successfully *launch an attack* against any city.

### 3.4 Client Requests

Clients are started on demand from the web application in response to orders sent by the user. A user can instantiate a client class that can issue one of three command types.

- **LEADER** learns the id of the current leader in the system. The leader is highlighted on the map as yellow.
- **ATTACK** requests replicas to attack the target city. An attack animation is visualized on the map.
- **SHUTDOWN** stops the selected replica(s) on the backend.

Upon receiving a LEADER request, a replica obtains the local view number  $v$ , gets the id of the current leader, and sends the id to the client. In response to an ATTACK request to the system, replicas will agree on whether they will *attack* or *retreat*, similar with the LAUNCH command.

## 4 IMPLEMENTATION

Excluding the BFT consensus library, the ByzGame project has about 5,000 lines of code written in Python, JavaScript, and HTML. We use the Python library Tornado to build the web framework, javascript library D3 to present the global map, and LevelDB as the database for maintaining system logs. We utilize BFT-SMaRt, an open source java-based BFT library [17]. We extend the BFT-SMaRt library to create a service for ByzGame that includes ByzGame client requests and insert numerous system logs for the frontend.

## 5 EVALUATION

Our evaluation focuses on how ByzGame can be a useful educational tool to enhance the understandability of BFT. Our experiments among students with different expertise levels show that ByzGame can greatly help students understand BFT concepts.

### 5.1 Understandability

Inspired by Raft [16], we use in-class experiments to evaluate how ByzGame can enhance the understandability of BFT. We have tested ByzGame among two groups of students. The first group was a graduate-level Distributed Systems course with 46 students with little to no knowledge about consensus theory. A class lesson introduced BFT, and included the Byzantine generals problem, state machine replication, PBFT, and a brief proof. We then asked the students to take a quiz and did not provide the correct answers. After that we introduced ByzGame, and asked students to play the game for half an hour. Then we asked students to take the same quiz again. After explaining the answers, we asked the students to take the second quiz, and provide feedback.

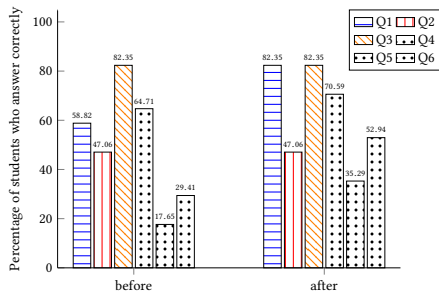
For the second group, we studied among five PhD students in our research lab where each student had 1-2 years of research experience in BFT and distributed systems. We skipped the lecture about BFT since they were already familiar with the concepts. We combined the results from the two groups and obtained the percentage of students who answered the questions correctly.

### 5.2 Results

The first quiz is designed to see what, if any, effect playing the ByzGame had on students' answers. The second quiz includes a few new questions and our purpose is to show whether with certain intervention and explanation, ByzGame can help students understand the core concepts of BFT and consensus in general.

**Quiz One.** We collected 34 effective answers for student teams who took quiz one twice, once before demonstrating ByzGame and once after the demonstration. We compare the two groups of results, show the difference for the results, and demonstrate the use of ByzGame can help students better understand the concepts.

The results for multiple choice questions of quiz one are presented in Figure 7. Q1 to Q4 are fundamental questions and Q5 to Q6 are more advanced questions. For fundamental questions,



**Figure 7: Percentage of student teams who answer the questions correctly before and after showing ByzGame.**

student answers for two of the questions improved significantly while two others remain unchanged. The question where student answers improve the most is *Q1: In BFT what is the threshold for faulty nodes?*

The two questions where student answers remain unchanged are Q2 and Q3 which are questions related to view changes. The results for Q2 is expected since ByzGame does not demonstrate any concepts *directly* related to PBFT. The results for Q3 is unexpected since ByzGame demonstrates view changes and leader election process. We believe this unchanged result may have been caused by students having not played with the view change functions. Adjustments to the lecture and procedures during the group discussion time may be necessary to address this.

In comparison, the percentage of students who answered Q5 and Q6 correctly have increased significantly compared with the fundamental questions. Since these questions are not as straightforward as the other basic questions, more than half of the students still did not answer them correctly. The low percentage is caused mainly because the question is less straightforward. Therefore, compared with the fundamental questions, ByzGame is even more effective in teaching the advanced topics.

*Overall grade.* We also grade the quiz one before and after showing ByzGame where the maximum is 7 points. The results are summarized in Table 1. As expected, the overall grade has improved significantly for all the standards, especially mean and median grade. To conclude, we believe that ByzGame can greatly help enhance the understandability of BFT consensus.

	Min	Max	Mean	Median
Before	2	6	3.76	4
After	2	7	4.6	5

**Table 1: Grade of quiz one before and after ByzGame demo.**

**Quiz Two.** We collected 33 useful respondents to quiz two. Quiz two includes two types of questions: *advanced questions* and *user feedback*. Advanced questions are similar with those in quiz one, where we include three more questions related to quorum sizes and the number of failures a given system can tolerate. User feedback includes a few questions from the user about the learning experience of ByzGame.

Quiz two results show that for advanced questions, 87.88%, 69.7%, and 69.7% student teams answered the questions correctly. The average scores for the other two questions are also higher than those similar questions in quiz one even after showing ByzGame.

*User feedback.* We obtain mostly positive feedback for user and learning experience for ByzGame. 96.88% of respondents indicated they agreed or strongly agreed that ByzGame was helpful in teaching about BFT general concepts. 93.75% agreed or strongly agreed that ByzGame helped them learn the technical concepts. 100% stated they understood how to configure the ByzGame interface. In addition, we also noticed informally that most students performed much better than prior years in the exams for the questions related to BFT and consensus in general.

## 6 CONCLUSION

We present ByzGame, a visualizable and understandable Byzantine fault-tolerant system that uniquely connects frontend web visualization with a backend BFT implementation. ByzGame enables users to apply BFT theory in a real system, and initial assessment demonstrates ByzGame improves BFT understanding by individuals with different expertise levels. ByzGame helped students learn and understand the concepts of BFT consensus. We have used the ByzGame with a second cohort of students, and initial results hold. We plan to make the game more widely available; ByzGame access can be requested via <https://www.byzgame.com>.

## REFERENCES

- [1] Ittai Abraham, Guy Gueta, Dahlia Malkhi, Lorenzo Alvisi, Rama Kotla, and Jean-Philippe Martin. 2017. Revisiting fast practical byzantine fault tolerance. *arXiv preprint arXiv:1712.01367* (2017).
- [2] Giuseppe Di Battista, Valentino Di Donato, Maurizio Patrignani, Maurizio Pizzonia, and Roberto Tamassia. 2016. BitConeView: Visualization of Flows in the Bitcoin Transaction Graph. In *VizSec*.
- [3] Christian Cachin, Klaus Kursawe, and Victor Shoup. 2005. Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography. *Journal of Cryptology* 18, 3 (2005), 219–246.
- [4] Christian Cachin and Marko Vukolić. 2017. Blockchain consensus protocols in the wild. In *DISC*. 1:1–1:16.
- [5] Miguel Castro and Barbara Liskov. 2002. Practical Byzantine fault tolerance and proactive recovery. *TOCS* 20, 4 (2002), 398–461.
- [6] Tushar D Chandra, Robert Griesemer, and Joshua Redstone. 2007. Paxos made live: an engineering perspective. In *PODC*. ACM.
- [7] Sisi Duan, Hein Meling, Sean Peisert, and Haibin Zhang. 2014. BChain: Byzantine Replication with High Throughput and Embedded Reconfiguration. In *OPODIS*. 91–106.
- [8] Sisi Duan, Michael K Reiter, and Haibin Zhang. 2018. BEAT: Asynchronous BFT Made Practical. In *CCS*. ACM, 2028–2041.
- [9] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1988. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)* 35, 2 (1988), 288–323.
- [10] Elli Androulaki et al. 2018. Hyperledger fabric: A distributed operating system for permissioned blockchains. In *EuroSys*.
- [11] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. 1985. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)* 32, 2 (1985), 374–382.
- [12] Leslie Lamport. 1978. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM* 11, 7 (1978), 558–565.
- [13] Leslie Lamport. 1998. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)* 16, 2 (1998), 133–169.
- [14] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine generals problem. *ACM TOPLAS* 4, 3 (1982), 382–401.
- [15] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. 2016. A secure sharding protocol for open blockchains. In *CCS*. ACM, 17–30.
- [16] Diego Ongaro and John Ousterhout. 2014. In search of an understandable consensus algorithm. In *ATC*. 305–319.
- [17] João Sousa, Eduardo Alchieri, and Alysson Bessani. 2014. State machine replication for the masses with BFT-SMaRt. In *DSN*. 355–362.
- [18] Tri A. Sundara, Ideva Gaputra, and Siska Aulia. 2017. Study on Blockchain Visualization. *International Journal on Informatics Visualization* (2017).
- [19] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper* 151 (2014).
- [20] Maofan Yin, Dahlia Malkhi, MK Reiterand, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT consensus with linearity and responsiveness. In *PODC*.