Leaderless Byzantine Fault Tolerance

Tian Qin

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2020-121
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-121.html

May 29, 2020

Acknowledgement

I would like to express my appreciation to Jian Liu and Peng Gao for their guidance, advice, and feedbacks during this research work.

Leaderless Byzantine Fault Tolerance

by Tian Qin

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Dawn Song
Research Advisor

05/29/2020

(Date)

* * * * * * *

Professor Sylvia Ratnasamy
Second Reader

Leaderless Byzantine Fault Tolerance

ABSTRACT

In this work, we propose Leaderless Byzantine Fault Tolerance (LBFT), a novel consensus algorithm that combines Snowball algorithm and Practical Byzantine Fault Tolerance (pBFT) algorithm, two existing consensus algorithms. Achieving consensus in decentralized systems has been difficult as they lack certain properties that many algorithms assume. Our approach looks to take advantage of the decentralized aspect of Snowball and the deterministic property of pBFT so that the weaknesses of Snowball's probabilistic property and pBFT's reliance on "leader" are eliminated. The algorithm we propose is applicable to decentralized systems such as blockchain systems even though Snowball and pBFT both make stronger assumptions than decentralized systems allow. By simulating real-world environments and comparing with pBFT performances, we show that LBFT is feasible in the context of real-world decentralized systems and has sufficient performances that are close to those of pBFT.

1 INTRODUCTION

A distributed system is simply a group of computers working together. Distributing a system brings many advantages such as horizontal scalability, fault tolerance, low latency, etc. A core feature of a distributed system is that it should appear as a single computer during all interactions with users. The scalability, parallelism, and communications among different machines in a distributed system shall be abstracted away. The power of distributed systems comes with a downside: achieving consensus is difficult due to faults, malicious actors, and many other factors and without agreement/coordination among individual nodes, a distributed system fails.

Practical Byzantine Fault Tolerance [4] is one consensus algorithm that carries out a three-phase procedure: pre-prepare, prepare, and commit. pBFT is robust for distributed systems with the weakest assumptions: asynchronous networks, a type of networks where consensus is very difficult due to a complete lack of understanding of timing of message transmissions.

Snowball [1] is a consensus algorithm that takes majority votes from random subsets of the network to achieve network-wide agreement in a probabilistic way. It is robust in partially synchronous networks.

Both pBFT and Snowball are robust in presence of Byzantine faults, which are conditions where there's incomplete information on whether a node has failed. However, each has its downsides. During each round of consensus execution, pBFT relies heavily on a node being the "leader" and such a centralized concept renders the algorithm vulnerable to numerous attacks. Unlike pBFT, Snowball is decentralized, but its consensus is probabilistic and is not robust in asynchronous networks.

We propose a new consensus algorithm that doesn't rely on any notion of leaders and works in asynchronous environments - Leaderless Byzantine Fault Tolerance (LBFT). It is applicable to decentralized systems, in particular blockchains; it is leaderless and deterministic: there is no differentiation among network nodes and transaction commitment is deterministic; it is built upon a gossip protocol to relay messages and data across the whole network, practical Byzantine Fault Tolerance algorithm to commit transactions in a deterministic manner, and Snowball to realize decentralization of nodes (i.e. leaderless).

In this paper, we identified the requirements of a leaderless consensus protocol, designed algorithmic architecture, built a gossip protocol, the Snowball protocol, the pBFT protocol, and the LBFT protocol, integrated optimization techniques such as transaction pipeline to mitigate loss in performance of LBFT compared to pBFT, and tested, evaluated, and cross-compared each of the four protocols.

Thus, this paper makes the following contributions:

- Design and Analysis of LBFT, which assumes asynchronous network
- System implementation and testing of LBFT

2 BACKGROUND

Machines in a distributed system have shared states. Consensus is unanimous agreement on state values across a network. A consensus algorithm is a procedure that achieves such states in a network. In other words, consensus algorithms coordinate network nodes in a distributed setting. The ability to achieve consensus is at the core of every distributed system.

To create formal proofs of consensus algorithms, distributed systems are generally assumed to be synchronous, partially synchronous, or asynchronous. These three timing models each have different properties regarding latency of message transmissions among network nodes. In a synchronous model, which has the strongest assumptions, message transmissions delays have a known upper bound, message transmissions follow FIFO (first in, first out) rule, and local clocks of all nodes in the network are synchronized; in a partially synchronous model, there exists an upper bound on transmission delay but it's unknown; in an asynchronous model, nodes' clocks are not synchronized and message transmissions can be delayed for an arbitrary amount of time and there's no order in message arrival. In The choice of timing model assumption is important to correctness and liveness of consensus algorithms as a distributed system's timing model determines whether a node can differentiate between a peer that has failed and one that is just taking time to respond. Assumption of a more friendly timing model makes designing consensus algorithms much easier.

The asynchronous model has the weakest assumptions and resembles real-world systems the most, where nodes fail and messages get dropped all the time. Consequently, reaching consensus in a real-world distributed system is difficult in the presence of process failures and communication failures. These challenges are laid out in many theoretical scenarios such as The Byzantine Generals' Problem, which describes a situation where consensus is very difficult with presence of corrupt parties and false information. Many distributed consensus algorithms have a notion of leaders, which is a subset of all network nodes that make core decisions on how the system should move forward. One example practical Byzantine Fault Tolerance (Castro and Liskov, 1999). It proposes a pre-prepare,

prepare, commit process Not to mention that such a notion of leader renders the system more vulnerable, these consensus algorithms are not applicable to decentralized systems such as blockchains, which are more robust than traditional distributed systems and thus can uphold fewer assumptions that many consensus algorithms rely on.

Practical Byzantine Fault Tolerance [4] is a consensus algorithm that was introduced in the late 90s and its consensus procedure is carried out in three phases: *pre-prepare*, *prepare*, and *commit*. Nodes move through a succession of configurations called views and in each view, one node is assigned as the primary. The *pre-prepare* and *prepare* phases guarantee that non-faulty nodes agree on a total order of requests within a view and the *commit* phase guarantees a total order of requests across views. In *pre-prepare* phase, the primary assigns a sequence number to the request and multicasts the request, digest of the request, the sequence number, and the view number to all other nodes, who then transition to *prepare* phase if message and signature are correct and sequence number and view number are valid. In *prepare* phase, each node multicasts to every other network node a *prepare* message that includes view number, sequence number, digest, and its own node ID. A node accepts a *prepare* message if sequence numbers, view numbers, and digests match. If a node accepts at least $2f$ *prepare* in a network of size $3f + 1$, it enters *commit* phase. In *commit* phase, a node multicasts view number, sequence number, digest, and node ID to other nodes. After receiving $2f + 1$ commit messages, a node executes the request. Since a total order of requests is agreed upon, non-faulty nodes execute requests in the same order.

Several attacks on pBFT have been carried out that exploited the fact that pBFT consensus execution relies on certain nodes being leaders/primaries that take on additional responsibilities. One example was introduced in paper "Honey Badger" [5]: the attack causes pBFT to halt by introducing a network scheduler that breaks the weakly synchronous hypothesis of pBFT. Specifically, when one faulty node becomes the leader, it holds the *PRE_PREPARE* message longer than the timeout to trigger a view change of the entire network. Then, the network scheduler delays the delivery of *VIEW_CHANGE* messages to all potential next leader nodes. Since the new leader doesn't receive *VIEW_CHANGE* messages from other nodes, it will not broadcast *NEW_VIEW* message, leading to timeout of view change of other nodes. So all other nodes will choose a new leader and broadcast *VIEW_CHANGE* messages. With the loop continuing, the leader will never be settled and the pBFT network halts.

Snowball [1] is a leaderless consensus protocol proposed by Team Rocket. It repeatedly does the following:

(1) take a random subset of the network
(2) get vote from each node in the subset
(3) determine majority vote and increase its count by 1
(4) take the majority vote with maximum counts, if the count passes a high enough threshold, adopt the value in that majority vote

Snowball protocol reaches consensus in a probabilistic way: the more rounds each node samples and takes majority votes, the more likely network is in consensus.

Snowball's probabilistic nature and pBFT's vulnerability to attack vectors due to centralization are the two properties we want to eliminate in our algorithm.

On the other hand, blockchain systems have shown potential in revolutionizing many fields such as finance, supply chain, and internet of things. However, current widely deployed consensus algorithms have different drawbacks: inefficiency, unfairness, etc. As consensus algorithms have been a bottleneck for adoption of blockchain systems, an efficient and secure consensus algorithm for blockchain systems has been a widely pursued area of research and our proposed algorithm also looks to contribute to it. By combining pBFT and Snowball, we aim to arrive at an algorithm that's robust in decentralized systems.

3 OVERVIEW

The protocol consists of two layers of implementation:

- a peer-to-peer (p2p) base layer
- a consensus layer on top of the p2p layer that implements a deterministic and leaderless consensus algorithm built upon Snowball and practical Byzantine Fault Tolerance

The peer-to-peer layer utilizes a gossip protocol to route data to all members of network. A gossip protocol specifies a method of peer-to-peer communication in a similar manner to how epidemics/gossips spread. When new information arrives at a machine, it picks a neighbor at random to share that piece of news.

The consensus layer implements an algorithm that combines two existing algorithms: • Practical Byzantine Fault Tolerance: a deterministic algorithm that relies upon nodes taking on "leader" roles for consensus processes to carry forward • Snowball: a probabilistic algorithm that does not assign certain nodes as "leaders", i.e. does not differentiate among nodes

pBFT being deterministic means that during each consensus step, we know with certainty whether network has reached consensus or not. Similarly, Snowball algorithm being probabilistic means that we only have a probabilistic notion on whether consensus has been achieved.

We propose substituting the "leader/primary" notion in pBFT with Snowball, thus achieving a deterministic and leaderless protocol. In other words, the "pre-prepare, prepare, commit" phase in pBFT is appended to Snowball execution. When a new client request comes in, instead of one single node (the primary) proposing a sequence number, the entire network will first agree on one by executing Snowball algorithm and then carry out the rest of pBFT steps. In other words, the leader's tasks are carried out by every node in the network and there's no single node that every other node relies upon for consensus to move forward. Such an approach does not have as strong assumptions as pBFT does and thus is more robust. We expect system throughput to fall but security level to rise.

4 DESIGN OF LEADERLESSBFT

We took a layered approach to our algorithm design. As outlined in Figure 1, the bottom layer is a peer-to-peer communication layer that relays messages based on a gossip protocol (resembling how an epidemic spreads). On top of that we implement the Snowball protocol and the pBFT protocol that use the gossip layer to transmit protocol messages. Finally, on top of Snowball and pBFT, we implemented LBFT, which combines Snowball procedure with pBFT procedure without interacting directly with the Gossip layer.

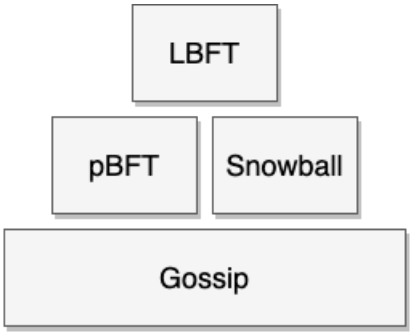

Figure 1: Layered Approach to LBFT

4.1 Gossip

The peer-to-peer layer provides basic communication API by implementing a general gossip protocol that spreads a piece of information quickly across a network.

A node can be in the following three states for a message

- unknown: node does not know about the message
- on-route: node has been poked about the message and will receive it soon
- known: node knows about the message

A node stores its own IP address, IP addresses of its peers/neighbors, whether it has stored a given request, and a list of known requests. Node X follows the following steps to transmit message M to node Y: poke Y to see whether Y knows about or is about to receive M and if Y is in state unknown regarding M, transmit M to Y, otherwise don't transmit.

Upon receiving a request, a node appends it to its list of requests and the node is considered as "knowing" the request, i.e. gossip has reached this node.

4.2 Snowball

Snowball protocol is implemented on top of the Gossip layer. Each Snowball node stores for each request counts of votes for each proposed sequence number. Upon receiving a new request, a node initiates snowball protocol: sample a random subset of the network, multicast a Snowball query to get each node's vote, store majority, and repeat. The number of rounds we run Snowball query to determine the sequence number of each request θ is $\frac{4}{3} * E[R]$, where $E[R]$ is the expected number of rounds consensus is reached, which can be calculated by the formula proved in [1]. Such θs are experimentally stable in reaching consensus without too big a loss in performance. Finally, when a node counts one identical majority votes more than the threshold number of times [1], the node adopts this majority vote.

4.3 pBFT

The design of pBFT is identical to the one described before [4]: a three-phase process - *pre-prepare*, *prepare*, and *commit*. Upon a client request, the primary node assigns a sequence number and sends out *pre-prepare* messages. Once nodes enter *prepare* phase, they multicast *prepare* messages and once enough *prepare* messages are received, nodes enter *commit* phase. In *commit* phase, nodes wait for $2f + 1$ valid commit messages in a network of size $3f + 1$ to execute client request.

All protocol message transmissions take place on the Gossip layer. Each node has a buffered channel that stores incoming protocol messages and processes them as soon as possible, similar to how a reader/writer queue works.

4.4 LBFT

The functionality of an LBFT node consists entirely of Snowball and pBFT. When an LBFT node receives a new client request request, it first relays the request to the Snowball layer for the entire network to achieve a consensus on the sequence number of that request by calling the gRPC method provided by Snowball layer. After the network has reached a consensus on the sequence number for this transaction, the regular phases of *pre-prepare*, *prepare*, and *commit* carry on just like in pBFT except now that sequence number has been agreed upon across network, nodes can skip sequence number verification.

In our design, we use Snowball to replace functionalities of a "leader" node in pBFT. The aforementioned attack in Honey Badger will no longer work because our design is a leaderless consensus. There is no distinction of nodes in our design and an attacker cannot figure out whose messages they should withhold. Moreover, the attacker cannot thwart consensus by delaying a small number of nodes because transactions (clients' requests) are distributed in a way of "network-wide random querying" and a small number of "dead" nodes will not have influence on the overall process.

More concisely, LBFT executes as follows:

(1) On a new client request, initiate Snowball protocol across entire network for nodes to reach consensus on its sequence number
(2) Run the three pBFT phases of *pre-prepare*, *prepare*, and *commit*, after which a total order of requests has been achieved
(3) Execute the request and reply to client

4.5 Optimization

Due to its decentralized nature, LBFT is slower than pBFT in execution and to account for such loss of performance, we implemented transaction pipeline as an optimization. Pipeline is the idea in computer architecture of processing requests in a time-sliced and overlapped fashion. Without pipeline, a following request cannot enter consensus process until the previous one is all the way through the entire consensus process. With pipeline, nodes can process a second request while first request is in second stage of consensus execution, and a third request while first request is in third stage and second request in second stage, and so on.

5 IMPLEMENTATION

Our implementations of gossip, Snowball, pBFT, and LBFT protocols are all in Go and inter-process communications are facilitated by gRPC, an open source remote procedure call system. Protocol Buffers are used to succinctly define gRPC services/APIs.

Generation of network graph configuration is written in Python 3.7. Deployment and testing are carried out through bash scripts.

6 EVALUATION

6.1 Experimental Setup

Testing framework is deployed on Savio cluster, a UC Berkeley high performance research computing cluster. Savio cluster consists of 600 nodes, has over 15,300 cores, and can achieve a peak performance of 540 teraFLOPS (CPU). The machines where deployment and testing are carried out run Scientific Linux 7 as software and each have 96 GB RAM and Skylake processor (2x16 @ 2.1 GHz).

We measured latency for gossip, Snowball, pBFT, and LBFT protocols. For each one, we test in networks of sizes 10, 50, 100, 200, 300, 400, and 500. As transaction pipeline optimization is implemented in pBFT (and thus LBFT), we test pBFT and LBFT with and without pipeline as well. In each test, we conducted the following procedure 10 times: send 100 requests in a row to the network, measure total processing time, and calculate average processing time. Then we further average the 10 average processing times to obtain a more trustworthy latency measure.

We also measured throughput for pBFT and LBFT in networks of size 10, 50, 100, 200, 300, 400, and 500. Again we tested each with and without transaction pipeline optimization. For each setup, we continuously send requests to the network until number of requests processed per minute converges to a fixed number.

6.2 Latency

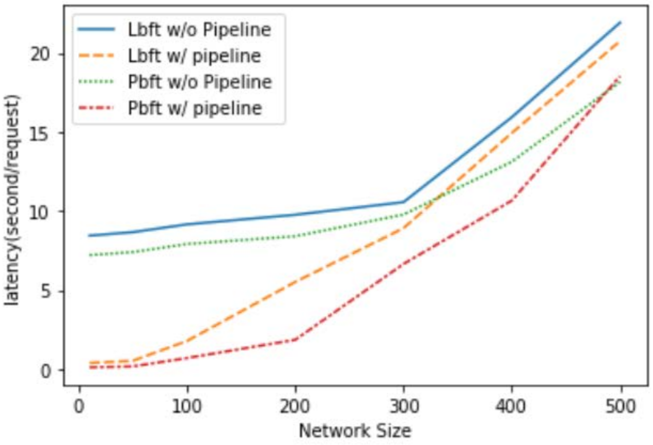

Figure 2: Latency of LBFT and PBFT

As we can see from Table 1, the gossip protocol tends to increase very slowly in latency as network size increases while latency of Snowball protocol increases more sharply. From Table 2, LBFT and pBFT have very similar latency measures across all network sizes, with LBFT having slightly larger results than pBFT, a trend confirmed by Figure 2. As for pipelined LBFT and pBFT, latency tends to increase more sharply as network sizes increase. When network size is small, pipeline reduces latency much more than when network size is large, the reasoning for which is that when network is large, more messages are sent during each consensus step and the network's maximum capacity and each node's maximum capacity are much more quickly reached with just a few transactions than when network is small, rendering pipelining ineffective. However, this effectiveness is a result of constraints in our testing environments and with a more powerful testing environment more lenient on number of open sockets, pipelining should still have significant performance boosting.

6.3 Throughput

From Table 3, we can see that throughput decreases as network size increases. For LBFT and pBFT without pipeline, the decrease is very gradual while it's very sharp for those with pipeline. In our testing environment, pipeline has at least an order of magnitude throughput boosting for small network sizes and for larger networks, we can't measure pipeline as effectively due to testing constraints, similar to testing latency. Expectedly, throughput of LBFT is lower than that of pBFT across all network sizes. Again, with a more powerful testing environment, we can realize more accurate numbers for large network sizes. However, our results on small network sizes demonstrate the performance of our algorithm and significance of pipeline optimization.

7 DISCUSSION

From both latency and throughput results, we see that pipelining improves performance much more for small network sizes than it does for large network sizes. This decreasing effect can probably be migitated by further optimizations of the implementation. Set-up of the testing automatically tests for correctness and we see that LBFT does achieve correctness with latency similar to that of pBFT. In fact, from the graph, we can see LBFT and pBFT have latency plots right next to each other's.

Due to limitation of maximum number of sockets in testing environment, we had to restrict network connectivity, which increases latency of message propagation. With more powerful testing environments that simulate real-world systems better, the performances can be further improved.

As we have shown, LBFT is a feasible algorithm that can be deployed in an asynchronous environment and achieve safety and liveness, properties carried from pBFT and Snowball. It has shown great potential for small network sizes which don't reach our testing constraints. Thus, further studies should be done with stronger testing environments to fully show its capability for large network sizes.

8 RELATED WORK

Bitcoin [6] is a cryptocurrency that uses Proof of Work to reach consensus on a ledger of UTXO transactions. Unlike many traditional Byzantine Fault Tolerance protocols, Bitcoin assumes an honest majority and gives a probabilistic guarantee. Bitcoin has been struggling with its low throughput.

Similar to Bitcoin's PoW algorithm, in the paper Avalanche [1], the author(s) presented a family of probabilistic consensus protocols, including Snowball. It outlines Snowball algorithm, presents a simple scenario to illustrate its execution, and analyzes its security properties. This paper also lists a comprehensive set of consensus algorithms, some being modifications of pBFT, and what their assumptions and properties are. The paper tests Snowball in a partially synchronous network but analyzes it in a synchronous setting.

In the paper Practical Byzantine Fault Tolerance [4], pBFT was proposed and a detailed execution of pBFT is described and tested in an asynchronous environment. The paper also includes correctness and liveness analyses and several optimizations.

Network Size	Gossip	Snowball
10	0.01	0.04
50	0.06	0.72
100	0.17	2.05
200	0.51	4.08
300	0.9	7.1
400	1.46	10.39
500	1.87	16.34

Table 1: Latency of Gossip and Snowball (second/request)

Network Size	LBFT w/o Pipeline	LBFT w/ Pipeline	pBFT w/o Pipeline	pBFT w/ Pipeline
10	8.439	0.389	7.21	0.116
50	8.66	0.53	7.4	0.178
100	9.153	1.782	7.911	0.702
200	9.758	5.513	8.4	1.857
300	10.556	8.913	9.771	6.65
400	15.926	14.927	13.112	10.655
500	21.904	20.737	18.157	18.526

Table 2: Latency of LBFT and PBFT (second/request)

Network Size	LBFT w/o Pipeline	LBFT w/ Pipeline	pBFT w/o Pipeline	pBFT w/ Pipeline
10	11.5	2040	22	2230
50	11	150	19.5	220
100	10	42	16	60
200	7.5	12	12	18
300	6	6	8.5	10
400	4	4	5	5
500	3	3	4	4

Table 3: Throughput of LBFT and pBFT (number of requests per minute)

Other works have been building on top of pBFT as well. Large-scale BFT [7] allows for arbitrary number of replicas and failure threshold, resulting in a probabilistic guarantee of liveness for some failure ratio while protecting safety with high probability.

In the paper Honey Badger of BFT protocols [5], the authors described a detailed attack on the leader notion of pBFT (described in Section 2) that causes consensus to halt. Other protocols that have leaders include Tendermint [3], which rotates the leader for each block.

Another leaderless protocol that also takes advantage of gossip is Hashgraph [2]. It builds directed acyclic graph via randomized gossip and is essentially also a variant of pBFT.

9 CONCLUSION

We have presented Leaderless Byzantine Fault Tolerance, a novel consensus algorithm applicable to any asynchronous system (e.g. decentralized systems) or any that has stronger properties (e.g. partially synchronous). We extracted pBFT's deterministic property and Snowball's leaderless property and achieved a robust deterministic Byzantine fault tolerance protocol whose latency and throughput are close to those of pBFT, thanks to high performance of Snowball protocol. By essentially making pBFT leaderless, we constructed an algorithm suitable for real-world blockchain systems.

Future work can dive into security properties of the new algorithm, further improve its performance through optimizations such as block compression and message aggregation, and compare the algorithm with state-of-art blockchain consensus algorithms such as Proof of Work, Proof of Stake, etc.

REFERENCES

[1] Snowflake to avalanche : A novel metastable consensus protocol family for cryptocurrencies team rocket. 2018.

[2] Leemon Baird, Mance Harmon, and Paul Madsen. Hedera: a public hashgraph network & governing council.

[3] Ethan Buchman. Tendermint: Byzantine fault tolerance in the age of blockchains. 2016.

[4] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, page 173–186, USA, 1999. USENIX Association.

[5] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 31–42, New York, NY, USA, 2016. Association for Computing Machinery.

[6] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Cryptography Mailing list at https://metzdowd.com*, 03 2009.

[7] Rodrigo Rodrigues, Petr Kuznetsov, and Bobby Bhattacharjee. Large-scale byzantine fault tolerance: safe but not always live. page 17, 06 2007.