

國立臺灣大學電機資訊學院資訊網路與多媒體研究所

碩士論文

Institute of Networking and Multimedia

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

以視域為基底的聯合拜占庭容錯共識系統

View-Based Federated Byzantine Agreement System

廖彥智

Yen-Chih Liao

指導教授：廖世偉 博士

Advisor: Shih-Wei Liao Ph.D.

中華民國 110 年 1 月

January, 2021





Acknowledgements

本篇論文之所以能夠完成，最需要感謝的就是廖世偉教授，在我就讀碩士的期間引導了我所研究的方向，尤其是在最後一年當中幫助我把最後的研究收斂至最終結果，除此之外也介紹了許多了解此方面的專家給了許多寶貴的建議，其中包含了對我論文影響最大的郭博士、黃博士、以及在 DeepQ 實習時的上司張博士，同時還有在口試之後指出我論文撰寫中的盲點的口試委員們；當然也不能忘記實驗室的學長以及同學和 DeepQ 同事們一直以來在面對研究感到徬徨時給予的鼓勵與支持；最後我要感謝我的家人、王睦榕和我的朋友們一路走來始終默默地支持我，讓我能有堅持完成這一切。





摘要

共識演算法已經經歷了數十年的研究，且被廣泛的使用在各種分散式系統當中，他在區塊鏈中也就想當然地佔據了一個關鍵的環節。不同的共識演算法都有不盡相同的優點以及缺點，也因此適用的範疇也不盡相同。由恆星發展基金會所提出的聯合拜占庭容錯共識系統便是一種較為獨特的共識設定。這個共識系統賦予了相較於一般其他拜占庭容錯的複製狀態機任意選擇所信任的節點的彈性，但是也因為這個彈性的選擇權，實作聯合拜占庭容錯共識系統的恆星共識協定需要採用以投票為基的協定經由額外的幾輪訊息交換作為犧牲才能達成共識。

在本篇研究中，藉由引入以視域為基的拜占庭共識演算法以及恆星共識協定當中的部分功能，達到能在不存在所有人共同直接信任的領袖的情況下藉由推舉領袖來產稱提案的方法。藉此，該以視域為基底的聯合拜占庭容錯共識系統就成為了一個更加簡單且容易理解的聯合拜占庭容錯共識系統。

我們展示了本研究的做法只要作惡的節點不超過總結點數的 7% 就能比同為聯合拜占庭容錯共識系統實作的恆星共識協定更有效率，儘管 7% 出錯比率下才有比較好的效能在具有高達三分之一容錯的演算法當中顯得較為嚴苛，具有高公信力節點的系統，好比政府部門或是知名公司等比較不會違規的單位就能採用本篇研究的演算法以獲得更高的吞吐量；相反的，如果是預期會變得更多元的系統，或只是系統內的互信基礎較為薄弱的話，就應該優先採用原本的恆星共識協定。

關鍵字：區塊鏈，恆星共識協定，聯合拜占庭容錯共識系統，共識演算法



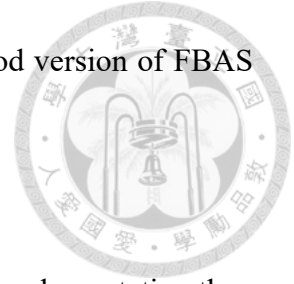


Abstract

Consensus algorithms have been researched for decades and they are crucial to a variety of distributed systems. Needless to say, consensus algorithms are also a critical part of the blockchain. Each of them has its advantages and disadvantages and is suitable for different applications. The Federated Byzantine Agreement System (FBAS)[15] introduced by Stellar foundation is one of the consensus algorithm framework that has a rather unique setting. It is a Byzantine Fault Tolerant (BFT) state machine replica (SMR) which allows nodes to choose whoever they want to trust compared to its counterparts. Due to this flexibility, the Stellar Consensus Protocol (SCP), which is an implementation of FBAS by Stellar foundation, requires a few more rounds of message exchange by adopting a ballot-based protocol as a trade-off.

This study introduces a novel methodology, view-based FBAS (vFBAS), which adopts certain functions from both view-based BFT algorithms and SCP in order to generate a faster proposal by electing a leader when there is no unanimously trusted leader. By doing

so, this work turns out to be less complex and easier to be understood version of FBAS implementation.



We show that vFBAS can be a more efficient version of FBAS implementation than SCP when the faulty nodes consist of less than 7% of the network. Although the 7% mark for this work to perform better is rather strict considering the algorithm is designed to hold up to less than a third of faulty behaviors. Networks with nodes that hold high credibility such as government or renowned companies, which are less likely to break the rules, could adapt vFBAS for higher throughput. On the other hand, networks that expect to have more variety or networks that have less mutual trust should adopt the original SCP.

Keywords: Blockchain, SCP, FBAS, consensus algorithm



Contents

	Page
Acknowledgements	i
摘要	iii
Abstract	v
Contents	vii
List of Figures	ix
List of Tables	xi
Chapter 1 Introduction	1
Chapter 2 Related Work	5
2.1 PBFT	6
2.2 Tendermint	6
2.3 HotStuff	7
2.4 FBAS	8
2.5 SCP	9
Chapter 3 View-Based FBAS	13
3.1 Modified nomination protocol	13
3.1.1 Original nomination protocol	14
3.1.2 The modification	15

3.1.3	Differences	16
3.2	Adopting view-based variants	17
3.2.1	View-based variants	17
3.2.2	Adopting	19
Chapter 4	Experiments	23
Chapter 5	Conclusions	29
References		31





List of Figures

Figure 3.1	Failed nomination	14
Figure 3.2	Successful nomination	14
Figure 3.3	Loop of trust	15
Figure 3.4	PBFT process flow	19
Figure 3.5	Tendermint process flow	19
Figure 3.6	HotStuff process flow	19
Figure 3.7	SCP process flow	19
Figure 4.1	Timeout per 10000 success	26
Figure 4.2	Timeout with Byzantine behavior	26
Figure 4.3	Death cross node count	27





List of Tables

Table 3.1	Variables for view-change	19
Table 3.2	View-based algorithms adopting to FBAS	21





Chapter 1 Introduction

Consensus algorithms for distributed system has been developed for a few decades already from simpler Crash-Fault Tolerant (CFT) algorithms such as Paxos[12] to more complex Byzantine Fault Tolerant (BFT) ones like Practical Byzantine Fault Tolerant (PBFT). In the past decade, because of the booming development of Distributed Ledger Technology (DLT) after the birth of Bitcoin[16], different kinds of consensus algorithms under various kinds of assumptions have been adopted as the backbone of different chains based on their purposes. Each of those algorithms has to sacrifice either their synchrony or correctness proven by FLP theorem[9]. The term "correctness" here can be further divided into safety and liveness by adopting a partial synchronous algorithm[8]. The PBFT algorithm is one of those partial synchronous algorithms that guarantees its correctness.

Among BFT consensus variants, FBAS has a unique setting. In a regular BFT consensus protocol, every node directly trusts more than two-thirds of all the other nodes regardless of the actual composition within the system. Therefore, whomever wants to join in a BFT consensus system needs to be directly trusted by all the other nodes. Besides, the criterion of accepting new nodes to participate in the system should be strict enough to prevent sibyl attack from happening. As a consequence, a regular BFT consensus protocol is quite hard to scale. FBAS, however, allows every node to directly trust whichever composition within the system. In such manner, the system size expands as the

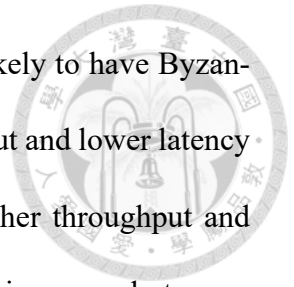
chain of direct trusts grows. SCP with the restricted flexibility of FBAS allows it to be more flexible than its counterparts. It will be further explained in Section 2.

SCP's flexibility comes with a cost, though. To reach consensus, SCP adopts a ballot-based consensus protocol which requires six rounds of message exchange. Meanwhile, view-based consensus protocols such as PBFT, Tendermint, and HotStuff requires only three, three, and four rounds respectively. The extra rounds in SCP is there to compensate for the lack of an unanimous leader due to the flexible trust.

To increase the scalability of SCP, we survey consensus algorithms which shares similar setting with SCP in Section 2. We then change the way how SCP works without unanimous leader such that our FBAS can be compatible with conventional BFT algorithms. This part can also be viewed as changing the way of electing leader from a round-robin one to ones that suits FBAS setting. Therefore, vFBAS obtains shorter protocol as a view-based BFT variant while preserving FBAS' flexible trait. In the next section, we give our opinion about which consensus algorithm is more suitable for FBAS to adopt from given different situations. Later on, we conduct an experiment to show that our work can always outperform SCP when less than 7% of nodes are Byzantine. However the performance of our work drops rapidly after there are more than 7% of adversaries whilst node count in the system increased.

As a consensus that can tolerate up to a third of Byzantine nodes, a 7% mark for this work to perform better is strict. Besides, the usage of SCP is to handle remittance, which throughput is not the major issue, between banks. However, cases such as Ukraine[3] government that tries to adopt SCP to handle their payment system, vFBAS could be a great candidate. Firstly, a government promoted distributed system should have nodes

which have a high credibility already. Those nodes should be less likely to have Byzantine behavior. Secondly, handling payments requires higher throughput and lower latency comparing to handling remittances. The vFBAS offers exactly higher throughput and lower latency under the first condition. After all, the usage of vFBAS is narrow but never useless.







Chapter 2 Related Work

As proven in FLP theorem, asynchronous with only one faulty process can not be deterministic, different compromises has to be made. For asynchronous consensus algorithms such as proof of work, zyzzyva[10], they have to sacrifice their safety and liveness respectively. Partial synchronous protocols, on the other hand, ensure both their safety and liveness by relying its safety on asynchronous assumption and its liveness on a synchronous one. Aside from synchronous assumptions, how the protocol deal with timeout is another way to categorize consensus algorithms. There are view-base protocols like PBFT that attempts to reach consensus on a single proposal from a leader and jumps to next view if timeout happens. The other one is ballot-based protocols like Paxos and SCP that replicas vote to lock on certain ballot among multiple proposals and jump to next ballot if timeout happens. Furthermore, based on the pattern how message is passed to reach consensus, BFT consensus algorithms can be further categorized into replica-based ones like PBFT, a quorum-based ones like Q/U[4], or even a hybrid approach such as HQ[7]. This study mainly focuses on consensus algorithms that share similar setting with SCP which are partial synchronous and replica-based ones. Representative ones that fit our restriction would be PBFT, Tendermint, and HotStuff.



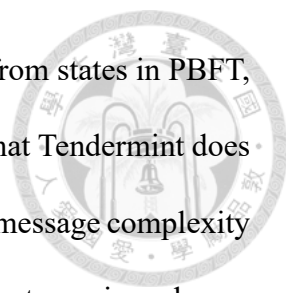
2.1 PBFT

The request for a consensus to tolerate Byzantine failure which malicious nodes can send arbitrary message to others regardless of the protocol has long existed. Practical Byzantine fault tolerance (PBFT) is the first implementation that has a reasonable message overhead. To reach a consensus, nodes go through states of pre-prepare, prepare, and commit to finally externalize the output. This process takes three times of message exchanges to accomplish. The first exchange ensures the message validity from the leader. The second one ensures the majority is working on the same proposal for nodes to lock on the value. The third one ensures the majority has locked on the value in case of the timeout so that a node is safe to commit. One thing worth mentioning is that PBFT only initiate a view-change to change the leader when timeout happens. Also, the message complexity of view-change in PBFT is $O(n^3)$ where n is the total number of nodes in the system. This is result from the fact that every node has to send a proof of size of n to all the others.

2.2 Tendermint

Tendermint[6, 11], inspired by PBFT, has a similar structure to PBFT. Originally, Tendermint suffers from a livelock[1]. It does not enjoy the property of optimistic responsiveness, where nodes confirm upon actual network delay, mentioned in thunderella[18] pointed out by HotStuff paper[19]. Then, Yackolley Amoussou-Guenou et al.[5] analyzed the former bugged version and introduced a variant of it. As for now, the issues mentioned before is fixed.

To reach a consensus, nodes of Tendermint go through states of propose, pre-vote,

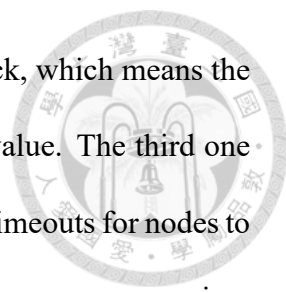


pre-commit, and commit. Although each states is named differently from states in PBFT, there functionality is actually comparable. The biggest difference is that Tendermint does not have to send anything when view-change happens. Therefore, the message complexity of Tendermint is $O(n^2)$ comparing to $O(n^3)$ of PBFT. The lower cost on view-change allows Tendermint to add view-change into the critical path of its algorithm. Having the feature of a frequent rotating leader is actually considered healthier for the system as no node can always be the one who proposes. As a trade-off, nodes can not jump straight into view-change in any step of the protocol but to keep sending message of each state. Since the voting in each step of Tendermint can be interpreted as voting for a proposal and voting for skipping the current round at the same time. The main differences are listed as the following:

- No additional message needed when changing view.
- View-change happens even when the consensus is reached successfully.
- Can not jump to view-change when timeout happens.

2.3 HotStuff

HotStuff[19], similar to Tendermint, also perform view-change every round. Unwilling to perform the high-cost view-change as PBFT does, HotStuff takes a different approach. HotStuff requires one more message exchange. Instead of giving each state a name, HotStuff gives each state change a name, which are prepare, pre-commit, commit, and decide respectively. The second and the third message exchange here is similar to the second message exchange in both PBFT and Tendermint. The second message exchange



ensures the majority is working on the same proposal for nodes to lock, which means the node guarantees to re-propose the value in case of timeouts, on the value. The third one ensures the majority has guarantees to re-propose the value in case of timeouts for nodes to lock such that the node guarantees to only accept the value when others are re-proposing. Note that both locks from second and third message exchange are kept instead of overwriting each others. By this additional message exchange, nodes can actually perform view-change directly when their timers fire as PBFT does. Besides, the HotStuff takes the advantage of threshold signature. Therefore, nodes no longer has to send messages in an all-to-all pattern but only communicating with the leader. This decreases the message complexity further to $O(N)$. The main differences are listed as the following:

- $O(n)$ complexity view-change. ($O(n^2)$ without threshold signature)
- View-change happens even when the consensus is reached successfully.
- Requires one more message exchange to reach consensus
- Replicas only communicate with the leader using threshold signature with $O(n)$ message complexity.

2.4 FBAS

Stellar proposed FBAS which enables flexible trust to regular BFT consensus protocols. Later on, G. Losa et al.[14] described a Personal Byzantine Quorum System which used a view-based partial synchronous protocol with a relaxed restriction.

To define 'trust', a set of trusted nodes enough to convince another node into executing a state change is considered as trusted. The union of all the possible combination of

these kind of sets is the set of nodes trusted the the node.

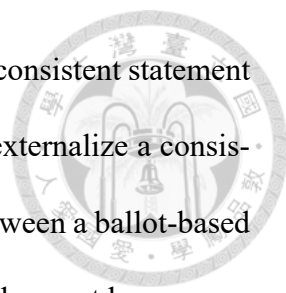
A node in regular BFT consensus has to trust every other node in the system. Nodes have to act accordingly when a unanimously pre-defined $\lfloor \frac{2N}{3} + 1 \rfloor$ portion of nodes having a consensus given N total nodes in the system. FBAS on the other hand allows nodes to choose the set of nodes they trust freely. To explain such flexible system, SCP needs to define a bunch of terms that is originally straightforward in other BFT counterparts. The followings are the explanation of the most important ones:

- Quorum slice/slice: Set of nodes enough to convince a node into making any state transition. A node can select multiple quorum slices at the same time. A quorum slice can also be nested.
- Quorum: Set of nodes that is enough to convince each others in the set. That is, every node in the set has at least a quorum slice as a subset.
- v-blocking set: Set of nodes that covers every quorum slice of a node.

As a quorum slice is set of nodes that is enough to convince a node to perform any state change. This opens up to about 2^{2^N} combinations, which is not ideal for implementation.

2.5 SCP

SCP, the implementation of FBAS, slightly constrains the freedom of FBAS. Nodes in SCP can only choose a fraction of a selected set of nodes. This reduce the possibility of $O(2^{2^N})$ to $O(2^N)$. Despite this constraint, SCP remains much flexible than its BFT counterparts. Nodes can select there own set regardless of other nodes' decisions. The



SCP, inspired by Paxos, uses a ballot-based approach to externalize a consistent statement per slot. On the other hand, regular BFT consensus protocols try to externalize a consistent statement per view. As mentioned before, the main difference between a ballot-based approach and a view-based approach is that a ballot-based approach does not have a pre-decided leader for each ballot while a view-based one does. As a result, a ballot-based approach has to spare some message exchanges to lock on a specific proposal, in other words, abandoning all the other proposals, on a ballot number to reach consensus. Based on this setting, the protocol is composed of nomination protocol and ballot protocol. The ballot protocol can be further divide into ballot-prepare and ballot-confirm protocol. Nomination protocol, ballot-prepare protocol, and ballot-confirm protocol are all based on a three phase commit (3PC) protocol called voting scheme. The process of voting scheme is listed below:

- Vote: Nodes vote for some proposals.
- Accept: Node accepts when a quorum voted for the same proposal.
- Confirm: Node confirms when a quorum accepts the same proposal.

These protocols sum up to six message exchanges in total. Comparing to three and four message exchanges from previously mentioned protocols, SCP has some trade-off for this flexibility. The main goal of each protocol is listed below:

- Nomination protocol: Try to converge the proposals from different nodes without an unanimous leader.
- Ballot-prepare protocol: Lock on a certain proposal at a ballot from nomination protocol and abandon incompatible ones.

- Ballot-confirm protocol: Confirm the result from ballot-prepare protocol.





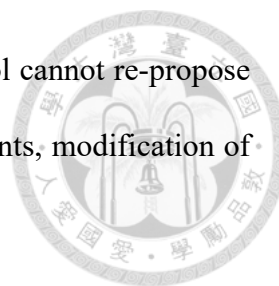


Chapter 3 View-Based FBAS

In this section, we introduced a way for FBAS requirements to adopt a view-based BFT protocol. Firstly, we take a deeper look at how SCP deal with the lack of leader resulted from its flexibility. We then modify its functionality such that a leader can be generated without using up too many timeouts. Secondly, we analyze regular view-base protocols to see which approach is more suitable for a FBAS setting under different situations.

3.1 Modified nomination protocol

Before modifying anything, one question might be asked. Can we adopt PBFT variants in setting of FBAS by simply replacing a leader's job with nomination protocol, which converge a proposal? The answer is no, although it might seem reasonable at the first glance. A leader from PBFT variants not only proposes a deterministic proposal but also takes the responsibility of recovering from the previously unsuccessful commits if there is any. Nomination protocol from SCP, on the other hand, merges different proposals from different nodes into one. A node initiates ballot protocol with a subset of proposals that have gone through their 3PC voting scheme. As a result, the system might end up having multiple nodes having different legal proposals. Also, in the case which only some



of nodes from the system commit on a proposal, nomination protocol cannot re-propose the committed proposal as required. Therefore, to adopt PBFT variants, modification of nomination protocol is needed.

The first section covers more detail about how original nomination protocol works. The second one describes how we modify it into vFBAS. The third one compares the differences in between.

3.1.1 Original nomination protocol

Instead of letting every node propose at their own will, SCP introduced a priority function in order to merge proposal from different nodes faster. The function takes local states into a hash function, which we can see it as a random function, for a node to select its own leader from its slices so that the node can echo whatever the leader offered as long as the proposal is valid. A node can only propose when it is the leader of itself. As shown in the following Figure 3.1, where nodes have an arrow pointing toward its temporal leader, nodes failed to collect majority vote on the same proposal. This would end up node *A* and *D* voting for {*A*}, node *B* and *C* voting for {*B*}, and timeouts. In the next round shown in Figure 3.2, each node adds a new leader. With each of them having two leader, node *A*, *B*, and *D* could end up voting for {*A*, *B*, *D*} while node *C* votes for {*B*, *D*}.

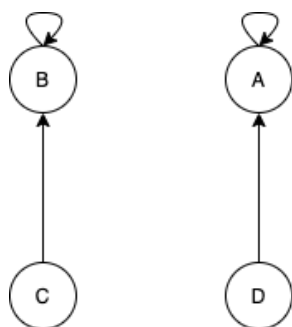


Figure 3.1: Failed nomination

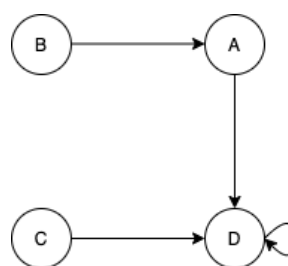


Figure 3.2: Successful nomination

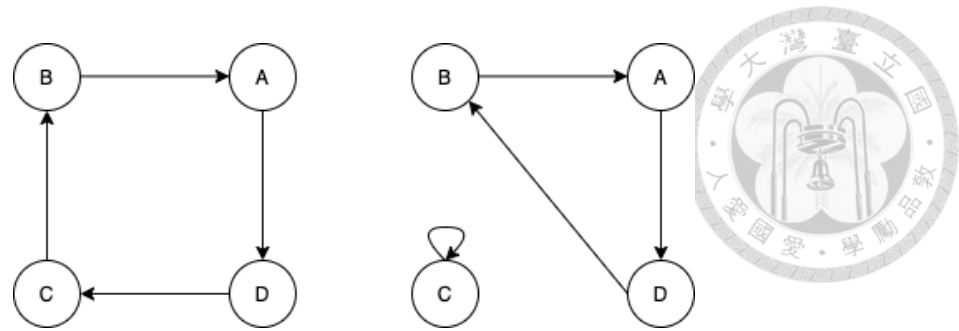


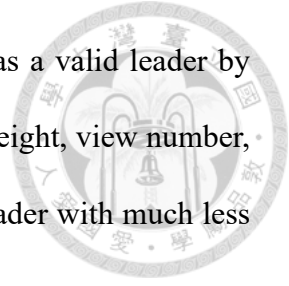
Figure 3.3: Loop of trust

Even in cases that the system does not split, nodes could also end up with no one broadcasting anything as shown in Figure 3.3. In these cases, after timer fired, nodes simply add another leader to its list while also keeping the values from the previous rounds. Upon 3PC being accomplished, a node could possess multiple proposals from different nodes in the state of confirmed. The node then combines all the confirmed proposals into the ballot protocol. Needless to say, the combined value going into ballot protocol from different nodes could differ from each other due to the message ordering.

3.1.2 The modification

In our approach, we want an unanimous leader giving a deterministic proposal and taking the responsibility of re-proposing proposals that could have been committed. Therefore, instead of combining different proposals from different rounds, we utilized the same priority function to figure out the leader. Different from the original approach, each node now broadcast their own proposal along with its own leader. Nodes will then recursively treat the leader of their leader as an indirectly trusted leader. As long as nodes in the chain of trust formed a quorum, within only a message exchange, nodes can bring the proposal of the final leader to the next step of the protocol. As per original nomination protocol, our approach moves to a higher round if timer fires.

To accelerate, vFBAS treat a loop in trust chain in figure 3.3 as a valid leader by further compute the node with the highest hash value that takes the height, view number, and node ID into account. By doing so, the system can produce a leader with much less rounds. The result will be shown in the experiment section.



3.1.3 Differences

The way SCP produces a propose needs two message exchange with no guarantee it is deterministic. The SCP accumulates the valid proposals from multiple rounds until the consensus is finally reached. This process grant SCP the power of producing a final proposal that contains proposals from multiple different nodes. Even a node not trusted by a lot of other nodes can get a much higher chance to get its proposal into rounds of consensus more frequently. Nevertheless, a shorter view-based BFT consensus algorithm can not be adopted because it needs a single leader producing a single proposal instead of a accumulated proposals from accumulated leaders.

On the other hand, vFBAS use only one message exchange to produce deterministic proposal without accumulating anything in order to adopt a view-based BFT consensus algorithm. Although the number of message exchange is lower, nodes that are not trusted by a lot of other nodes will have less chance to propose anything.

Another notable difference is that vFBAS treat sufficient circular leader nomination as valid while SCP does not. Although the FBAS papers does not state any reason, here is our speculation. The SCP intentionally does it to incur timeout so that the final proposal is able to collect more proposals from nodes. Having a higher chance to encounter timeout at the phase of generating proposals in SCP is not as harmful as vFBAS does. By

accumulating both nominated leaders and proposed values from previously failed rounds, the chance of encounter timeout only gets lower for SCP if network condition remains the same. The chance for vFBAS to encounter timeout, however, always remains the same. This is also the reason why vFBAS performs worse when faulty behavior gets higher later shown in Section 4.

Note that the way both SCP and vFBAS produce proposal can suffer from timeout even in a synchronous environment.

3.2 Adopting view-based variants

Given our unique way of selecting leader out off nodes that does not trust all the others, it is like we are choosing leaders randomly while others uses round-robin to do so. However, there is one caveat remaining. Nodes in FBAS explores the system by learning from slices of their counterparts whereas nodes in conventional BFT knows every other nodes in advance.

In this section, view-based BFT consensus algorithms will be broken down to analyze. Afterwards, we gives our advice on how to adopt view-based variants under different conditions.

3.2.1 View-based variants

Since the use of terms different from paper to paper, this section will use the following terms to describe the algorithms instead of their original names.

- Phase: The local state of a node.

- Step: State changes.
- $lock_r$: The value used to recover from timeouts if there are potential commits.
- $lock_c$: The value used to check if incoming leader proposal is valid.



For consensus of N phases, it always has $N - 1$ steps. Normally, aside from the first step which nodes take the proposal only from the leader, other steps all requires nodes to act upon majority votes. For conventional view-based variants, it would be more than two-thirds of votes; for FBAS, it would be a quorum of votes. To prevent forking, algorithms need to lock on the proposal before the step that externalize it. With this value kept, nodes are able to use this lock, $lock_r$ to "recover" if only a part of the system externalized. Also, nodes would only accept proposal from leaders that is same as its lock, $lock_c$, for "checking".

The use of both $lock_r$ and $lock_c$ is redundant in some algorithms. PBFT does not has to keep $lock_c$, instead, all the replicas do the same computation as the leader does from the all-to-all view-change message they get. On the contrary, Tendermint does not broadcast message when view change happens. Therefore, it has its $lock_c$ equivalent to $lock_r$ but needs to finish all the message exchanges as a trade-off. As for HotStuff, it has view change message but only sends it to the leader. Given that other nodes are not able to do the computation as the leader does similar to PBFT, nodes have to have one additional step in the process to keep $lock_c$. In comparison with Tendermint, the additional step in HotStuff allows nodes to skip the rest of the protocol when the majority disagree with the proposal or when the timer fires. Table 3.1 displays their main differences when it comes to view-change.

Inspired by HotStuff, the Figures 3.4, 3.5 and 3.6 shows the crucial operations in each

Consensus	Step	$lock_r$	$lock_c$	timeout message	timeout proposal
PBFT	3	step2	recompute	proof of $lock_r$ to all	recompute
Tendermint	3	step2	step2	N/A	local $lock_r$
HotStuff	4	step2	step3	proof of $lock_r$ to leader	highest $lock_r$

Table 3.1: Variables for view-change

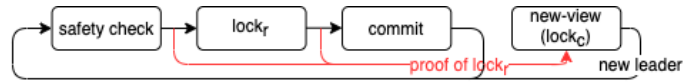


Figure 3.4: PBFT process flow

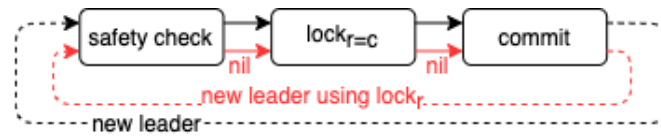


Figure 3.5: Tendermint process flow

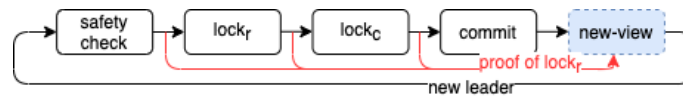


Figure 3.6: HotStuff process flow

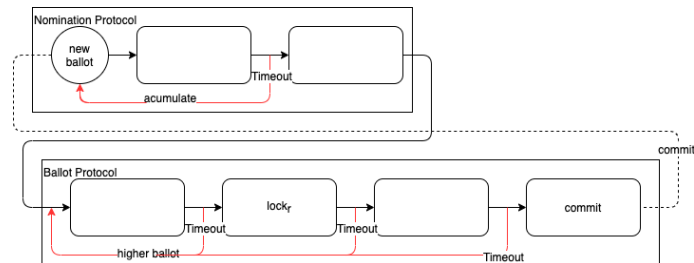
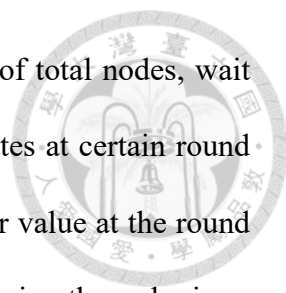


Figure 3.7: SCP process flow

step of the algorithms including SCP. In comparison, SCP is also represented in the same manner in Figure 3.7.

3.2.2 Adopting

Before getting into too much details of advantages and disadvantages, we first substitute some straight forward predicates in view-based variants into an FBAS ones. For each



step in the algorithms that needs to wait until more than two-thirds of total nodes, wait until votes forms a quorum instead. Votes forming a quorum indicates at certain round indicates that there is no other quorum can be formed to vote another value at the round as long as the quorum structure is safe. Change the condition of skipping through views when a third of total nodes have already on the higher view into skipping through views when nodes having higher view is a v -blocking set. A v -blocking set voting for the same value in certain round indicates that it could either be the value getting a quorum of vote or no consensus is reached at all. Note that a quorum in FBAS and $2f + 1$ nodes out of $3f + 1$ nodes in conventional algorithms is equivalent. On the contrary, $f + 1$ nodes out of $3f + 1$ blocks nodes in conventional algorithms safety-wisely but a v -blocking set in FBAS only blocks a node liveness-wisely. It is because of the fact that the amount of faulty nodes that causes safety and liveness issues can be discussed separately even though for conventional algorithms the amount is the same shown by J. Li et al.[13]

Using nomination mechanism for v FBAS, nodes can not predict the next leader. As a result, they have to do a all-to-all message exchange. Also, nodes in FBAS explores the system without knowing all of their peers in advance. Therefore, using threshold signature is unreasonable, either. Using HotStuff with all these handicaps would become a PBFT with one more step. Besides, our nomination mechanism is similar to randomly choosing a leader. Using PBFT which changes leader even without timeout is costly. All-to-all message exchange each containing a proof of size proportional to the total node count every view should be avoid. Hence, were we going to adopt a existing algorithm with the least effort, Tendermint would be the choice.

However, considering v FBAS incurs timeout even when messages all reaches timely if the chain of trust does not forms a quorum, having to do the rest of message exchange

in Tendermint can hurt. Shown later in Section 4, as there are more Byzantine node in the system, the timeout rate grows in a non-linear pattern. Adding one more step suggested in HotStuff allows us jump to the next view directly instead of finishing every step left. The following Table 3.2 compares the differences between view-based BFT variants after adopted by vFBAS.

Consensus	Step	$lock_r/lock_c$	timeout message/proposal	timing
SCP	6	step5/5	N/A / local $lock_r$	nominate/ballot
vFBAS-PBFT	3	step2/recompute	$lock_r$ to all / recompute	timeout
vFBAS-Tendermint	3	step2/2	N/A / local $lock_r$	last timeout
vFBAS-HotStuff	4	step2/3	$lock_r$ to all / recompute	timeout
vFBAS-mix	4	step2/3	N/A / local $lock_r$	timeout

Table 3.2: View-based algorithms adopting to FBAS

orange indicates the downsides; red indicates the change due to FBAS' setting

Note that not adopting any of above existing consensus algorithms does not imply they are inefficient in other cases but just less suitable under the FBAS setting.

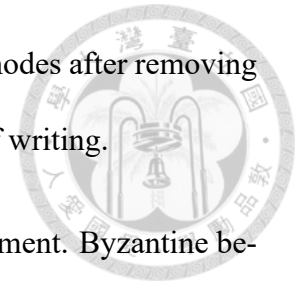




Chapter 4 Experiments

As mentioned in the previous section, nomination mechanism for vFBAS, similar to SCP, incurs timeout even when all the message reaches timely. Although vFBAS does not accumulates leaders and the votes across different rounds, a loop in the trust chain is considered valid for us. Curious about how these changes could affect the timeout rate, the experiment is conducted under a uniformly distributed system. Uniformly distributed system here means that if people calculate the PageRank[17] of the system, they would end up getting every node with the same weight. The experiment is also conducted as a synchronous system to rule out the effect of default timer and the network condition, especially when checking if multiple accumulated proposals reach the threshold one by one can be slow sometimes. This experiment measures the timeout nodes encountered before reaching consensus 10000 times, in other words, reaching height of 10000. Both nomination mechanisms collecting a quorum of votes for the first time count as reaching consensus of producing proposal once, although nomination protocol from SCP requires twice as much. This does not affect the outcome in such synchronous environment. Besides, the axis measuring total node count in this experiment is not continuous integers. Instead, the total node count is reversely calculated from continuous integer of faulty nodes and rounded up. This is why some data points is sparser than others. Lastly, the experiment is only conducted up to the system size of 160 nodes. It is more than twice as much

as the node count in current Stellar network[2], which has 67 active nodes after removing nodes that trusts no one and nodes that are not trusted, by the time of writing.

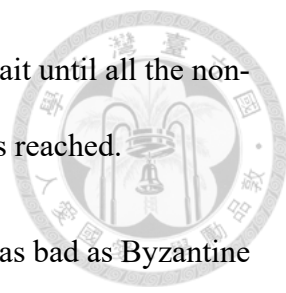


Byzantine behavior is also an important parameter in the experiment. Byzantine behaviors can cause a variety of trouble such as livelock[1]. The vFBAS, however, is only about replacing the leader election part of the algorithms. The one additional weak spot that can be attacked is to stop the leader from ever successfully elected. To do so, byzantine nodes could either mislead nodes trusting in them into believing in different leaders or herd nodes trusting them into multiple separated small groups. Any of these attempts even when Byzantine node are working closely collaborated are just as bad as not sending any message as if it is a crashed node.

To show that collaborated Byzantine nodes in leader election is just as bad as crashed ones, we can categorize there actions into two. First, they can treat themselves as leaders. By doing so, they can broadcast invalid proposals or even different proposals to different nodes having them as leader at the round so that there is no majority consensus on the same proposal. Second, they can forward the chain of trust to other nodes such that the system remains splitting by making sure each chain of trust is short enough or even making nodes having them as leader at the round echoing proposals from different leaders. Byzantine can achieve both cases by not sending anything.

To be more specific, these experiment use the following settings:

- Uniformly distributed trust: The quorum slices of node i in a system of totally N nodes is set as at least $\frac{5}{6}$ nodes from nodes ranging from $i \bmod N$ to $i + \frac{4}{5}N \bmod N$.
- Iteration: Attempts to reach consensus. Stop once the consensus is reached for 10000 times.

- 
- Synchronous environment: every iteration of the experiment wait until all the non-crashing nodes send their vote to check if majority consensus is reached.
 - Byzantine behavior: implemented as crashed nodes, which are as bad as Byzantine nodes shown above, which does not sent message at all.
 - Success criterion: In each iteration, both vFBAS and SCP count as a success if the chain of trust consisting a quorum is formed. vFBAS also count a loop of trust consisting a quorum as success.
 - Leader nomination: For simplicity, node randomly choose a node from their quorum slices as the leader they vote for.
 - Node behavior: vFBAS reset leader and votes every iteration whereas SCP accumulates if the previous iteration is not successful.
 - Data point: Total node count of $\{4 + 3n | n \in \mathbb{N}, 0 \leq 4n + 3 \leq 160\}$ is measured when no faulty node is in effect; when there are f faulty nodes, total node count of $\{nf | n \in \mathbb{N}, 1 \leq nf \leq 160\}$ is measured.

The first experiment is conducted without Byzantine nodes in figure 4.1. The result shows that accepting loops in the trust chain converges the result faster than accumulating leaders and votes with Byzantine behaviors. Another surprising finding is that our approach is not affected by the total number of nodes in the system as much as an SCP ones does.

The second experiment demonstrates that both nomination mechanisms encounter more timeout when more Byzantine nodes is in the system in Figure 4.2. Besides, when Byzantine nodes surpassed 10% in the system, the impact gets even higher with more nodes in the system.

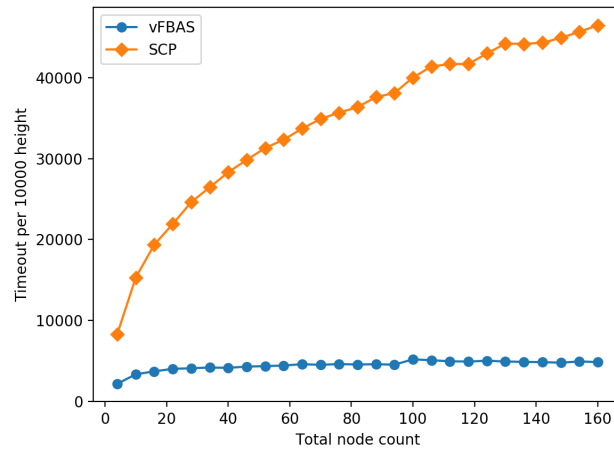


Figure 4.1: Timeout per 10000 success

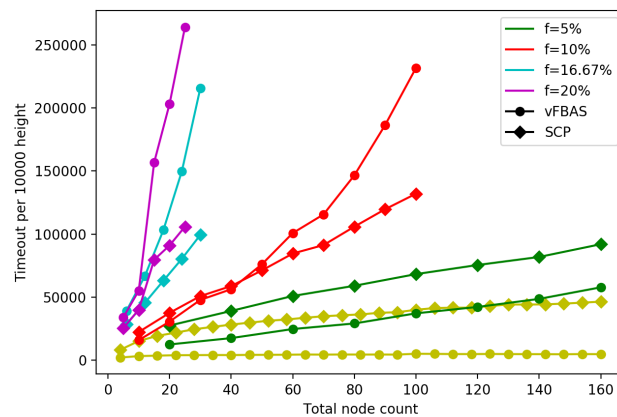


Figure 4.2: Timeout with Byzantine behavior

Each color represents different ratio of faulty nodes in the system. The vFBAS still has lower timeout rate than SCP when faulty rate < 5% up to 160 nodes.

The third experiment aims to find out the number of nodes in the system for original nomination mechanism to surpass our approach under different percentage of Byzantine nodes in Figure 4.3.

The experimental results might not seem promising at the first glance. Nevertheless, nodes can take the advantage of flexibility granted by FBAS by removing Byzantine nodes manually themselves. Conversely, conventional BFT algorithms can not remove a peer without every other node agreeing. A node removing a peer locally does not cost either safety or liveness shown in Theorem 12 and 14 from Stellar whitepaper[?] as long as the

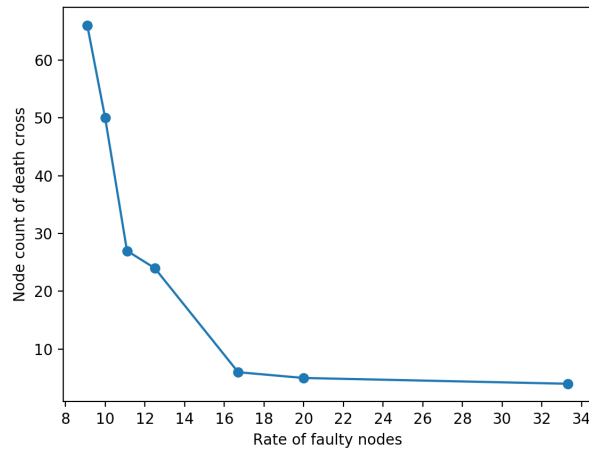


Figure 4.3: Death cross node count

The lowest node count for SCP to outperform vFBAS for different faulty rates.

quorum intersection is held. Moreover, instead of choosing exactly one leader per round that can suffer from network splitting even in synchrony, nodes can broadcast a series of leaders at once along with their proposal. By the time nodes find out the first leader from other nodes could not forms a quorum already, they can use the second leader in messages it received. This method not only could negate the timeout frequency resulting from the split of network, but also decreases the effect of crashed nodes. However, such method can still suffer from Byzantine behaviors. Besides, nodes are not be able to find out that the network has already split all the time given that the network of FBAS is discovered in the process of message exchanges but not known in advance. Additionally, this method grants Byzantine nodes more information such that they could minimize the possibility of some nodes being elected as a leader.



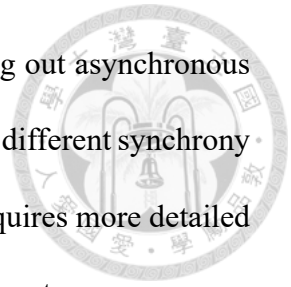


Chapter 5 Conclusions

The FBAS has introduced a lot of flexibility into BFT consensus in terms of trust. A threefold 3PC protocol, six steps in total, used by SCP might stop applications requiring very high throughput from using it. As a partial synchronous BFT consensus protocol, all the additional steps in the protocol are due to the lack of unanimous leader. With the way to elect a unanimous leader we introduced, an FBAS no longer has to spend steps just to converge proposal from nodes. By adopting to a view-based BFT variant, the latency drops from six round trip time to at most four. Despite the fact that adopting PBFT, Tendermint, and HotStuff are all doable for vFBAS, a Tendermint and HotStuff mix which need four step to commit without a view-change message fits the FBAS setting the most. On the flip side, nodes not gaining much trust from peers would have less chance to propose proposals they prefer. Also, the performance of vFBAS drop rapidly when there are more faulty nodes in the system. As a result, we suggest using vFBAS when nodes have high credibility in advance.

Further researches can be done by further exploring ways to reduce the chance of timeout even under synchrony at the phase of generating leaders. By doing so, the mark of faulty behavior proportion for vFBAS to outperform can be higher. Although the latency of vFBAS should still remains more vulnerable to behavior of faulty nodes compared to SCP, it can still increase its applicability by a large margin. Aside from increasing

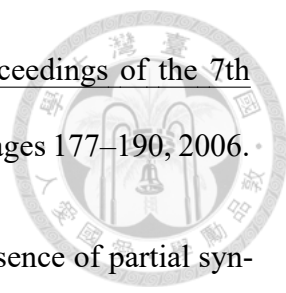
the applicability by reducing the chance of timeout occurrence, trying out asynchronous algorithms or more complex ones could be also a interesting aspect. A different synchrony assumption makes the algorithm less comparable to SCP, and thus requires more detailed proofs. It may turn out to be applicable for some fields that SCP does not.

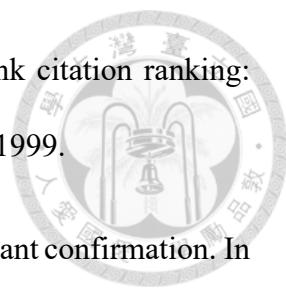




References

- [1] livelock issue. <https://github.com/tendermint/tendermint/issues/1047>.
Accessed: 2021-01-07.
- [2] Stellar network api. <https://api.stellarbeat.io/v1/network/stellar-public>. Accessed: 2021-01-15.
- [3] Ukraine government picks stellar development foundation to help build national digital currency. <https://finance.yahoo.com/news/ukraine-government-picks-stellar-help-140028973.html>.
- [4] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie. Fault-scalable byzantine fault-tolerant services. ACM SIGOPS Operating Systems Review, 39(5):59–74, 2005.
- [5] Y. Amoussou-Guenou, A. Del Pozzo, M. Potop-Butucaru, and S. Tucci-Piergiovanni. Dissecting tendermint. In International Conference on Networked Systems, pages 166–182. Springer, 2019.
- [6] E. Buchman. Tendermint: Byzantine fault tolerance in the age of blockchains. PhD thesis, 2016.
- [7] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shrira. Hq replication: A

- 
- hybrid quorum protocol for byzantine fault tolerance. In Proceedings of the 7th symposium on Operating systems design and implementation, pages 177–190, 2006.
- [8] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. Journal of the ACM (JACM), 35(2):288–323, 1988.
- [9] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. Journal of the ACM (JACM), 32(2):374–382, 1985.
- [10] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzyva: speculative byzantine fault tolerance. In Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles, pages 45–58, 2007.
- [11] J. Kwon. Tendermint: Consensus without mining. Draft v. 0.6, fall, 1(11), 2014.
- [12] L. Lamport. The part-time parliament. In Concurrency: the Works of Leslie Lamport, pages 277–317, 2019.
- [13] J. Li and D. Mazières. Beyond one-third faulty replicas in byzantine fault tolerant systems. In NSDI, 2007.
- [14] G. Losa, E. Gafni, and D. Mazières. Stellar consensus by instantiation. In 33rd International Symposium on Distributed Computing (DISC 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
- [15] D. Mazières. The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus, 2015.
- [16] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

- 
- [17] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.
- [18] R. Pass and E. Shi. Thunderella: Blockchains with optimistic instant confirmation. In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pages 3–33. Springer, 2018.
- [19] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, pages 347–356, 2019.