Stable Byzantine Fault Tolerance in Wide Area Networks With Unreliable Links

Sitong Ling, Zhuotao Liu, Qi Li, *Senior Member, IEEE*, Xinle Du, Jing Chen, and Ke Xu, *Fellow, IEEE*

***Abstract*— With the increasing demand for blockchain technology in various industry sectors, there has been a growing interest in the Byzantine Fault Tolerance (BFT) consensus that is the backbone of most of these blockchains. However, many state-of-the-art algorithms that require reliable connections can only offer limited throughput in wide-area networks (WANs), where participants are connected over long distances and may experience unpredictable network failures. The partially-connected BFTs are designed for unreliable and highly dynamic networks yet impose exponential communication complexity. This paper proposes Stable Byzantine Fault Tolerance (SBFT), a BFT communication abstraction that can sustain high throughput and low latency in WAN. SBFT separates the leader from consensus in pipelined BFT consensus and uses an adaptive consensus mechanism to resist dynamic faulty links, maintaining consensus efficiency when network connectivity is high while adapting to dynamic networks with low connectivity. We implemented a prototype of SBFT and tested it on the WAN. The results demonstrate that SBFT has a throughput similar to HotStuff in a fault-free environment but can reduce about 80% of consensus latency. Besides, SBFT retains 40% of the original throughput when the link failure probability is 0.4, while the baseline HotStuff retains less than 40% when the link failure probability is only 0.1.**

***Index Terms*— Byzantine fault tolerance, stable consensus, adaptive consensus, dynamic network, partially connected network.**

I. Introduction

RECENT advances in permissioned blockchains [2], data sharing [22], [40], [48] and network security [26], [33], [47] have shown that Byzantine Fault Tolerant (BFT) state-machine replication is not a concept of solely academic interest but a problem of high relevance in practice. In particular, the ability to withstand arbitrary component failures makes BFT replication a powerful tool for building critical systems that must meet high reliability and availability requirements. In order to achieve high-throughput and low-latency consensus, some arts focus on scaling BFT consensus via mechanisms such as committee elections [15], sharding [25], layering [28], and tree-structured communication [24], [36].

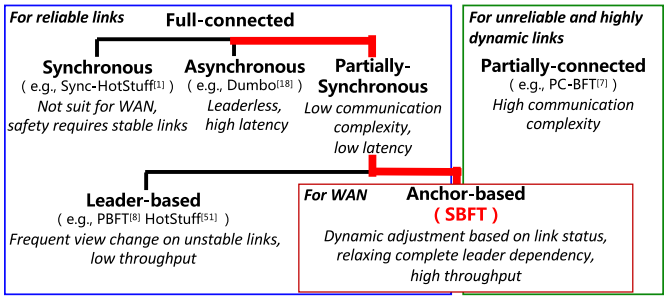


Fig. 1. Design space in BFT.

These new consensuses demonstrate tremendous improvements; however, deployment in Wide Area Networks (WAN) is a challenging issue [22]. The most critical problem for distributed systems running in the WAN network is link unreliability, which may cause the sudden increment of message delay or even link disconnection [5], [13], [37], [50]. Link unreliability happens all the time in WAN, due to a variety of reasons such as national level security audits [5], Distributed Denial of Service (DDoS) network attacks [37], hijacking attacks against TCP [13], and even network failures [50].

Broadly speaking, the prior work can be classified as full-connected BFT, and partially-connected BFT protocols. Unreliable and unstable WAN links can make the full-connected consensus protocols [1], [8], [9], [12], [18], [19], [34] defective or even unusable because they designed for reliable links. For instance, Synchronous BFT protocols (such as Sync-HotStuff [1]) assume bounded communication latency to achieve safety, which may not hold when facing unreliable links. Asynchronous BFT protocols (such as Dombo [18]) require more rounds of interaction and have to wait for the link recovery to meet the minimum network connectivity requirement, thus increasing the consensus latency. Leader-based BFT protocols (such as PBFT [8], HotStuff [51]) will cause frequent view changes with unreliable links, thus reducing the consensus throughput.

Received 24 November 2023; revised 2 June 2024; accepted 6 September 2024; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor G. Fanti. Date of publication 26 September 2024; date of current version 19 December 2024. This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFB2701700, in part by the National Science Foundation for Distinguished Young Scholars of China under Grant 62425201, in part by the Science Fund for Creative Research Groups of the National Natural Science Foundation of China under Grant 6222100, in part by the Key Program of the National Natural Science Foundation of China under Grant 61932016 and Grant 62132011, and in part by the National Natural Science Foundation of China under Grant U22B2031. *(Corresponding author: Ke Xu.)*

Sitong Ling, Jing Chen, and Ke Xu are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: lingst21@mails.tsinghua.edu.cn; jchencs@tsinghua.edu.cn; xuke@tsinghua.edu.cn).

Zhuotao Liu and Qi Li are with the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China (e-mail: zhuotaoliu@tsinghua.edu.cn; qli01@tsinghua.edu.cn).

Xinle Du is with Huawei, Shenzhen 518129, China (e-mail: duxinle1@huawei.com).

Digital Object Identifier 10.1109/TNET.2024.3461872

TABLE I
THE SUMMARY OF KEY PROPERTIES OF SBFT AND SEVERAL KINDS OF RELATED BFT PROTOCOLS

Protocols	*Liveness Condition*			Network	Efficiency in synchronous network	
	Message Delay	*Network Topology*	*Leader Dependency*		*Communication Complexity*	*Consensus latency*
BP-BFT [8]	$< \Delta$	Full-connected	Strong	Static	$O(n\|B\| + n^2)$	3
CP-BFT [51]	$< \Delta$	Full-connected	Strong	Static	$O(n\|B\| + n)$	5
S-BFT [1]	$< \Delta$	Full-connected	Strong	Static	$O(n\|B\| + n^2)$	3
A-BFT [18]	–	Full-connected	N/A	Static	$O(n^2\|B\| + n^3)$	18
PC-BFT [7]	$< \Delta$	Partial-connected	Strong	Static	$O(n^2\|B\| + n!)$	3
SBFT	$< \Delta$	Anchor-based	Weak	Dynamic	$O(n\|B\| + n^2)$	3

Prior works on partially connected networks (such as [6], [7], [41], [46]) also have several key limitations. First, they primarily focus on static consensus, achieving high link fault tolerance but introducing exponential communication complexity. Second, these works mainly study the one-shot consensus, which cannot be directly applied to replicated services that require consecutive consensus (see details in § II).

Thus from a philosophical standpoint, it is worth asking: Can BFT ensure high throughput and low latency with unreliable links? Can we simply enhance the traditional BFT consensus to WAN? How to distinguish partial-connected networks from node errors when consensus stops running? How to adapt to network dynamics? In this paper, we seek to answer these questions with a new scheme Stable Byzantine Fault Tolerance (SBFT).

Figure 1 surveys the design space for BFT and places SBFT in context by following the thick red lines through the design tree. At the highest level, SBFT is designed for WAN with unreliable links. Next, SBFT is a partially-synchronous BFT, which has low communication complexity and low latency. As shown in the lowest branch of the design tree, SBFT falls into the category of a high throughput *anchor*-based approach that relaxes complete leader dependency and dynamically adjusts the acceptance-rules based on link status.

In SBFT, we have two key designs, a) giving a new network model, with which replicas can use a lightweight local state notification mechanism to acquire liveness when the network is not synchronous, and b) adjusting the consensus strategy dynamically. First, we relax the complete dependency on the leader without sacrificing consensus speed in SBFT and propose to use *anchor* nodes instead of leaders to drive consensus. Through the *anchor*-driven consensus design, SBFT can make stable progress on producing blocks *even when the leader node has very limited connectivities* (in fact, the leader node only needs a single *anchor* neighbor to transmit blocks) (detailed in § III). Second, SBFT allows nodes to dynamically adjust their consensus strategies based on their perceived status of neighbors. This ensures that SBFT achieves high link fault tolerance for running stably when the network connectivity is low, and meanwhile achieves the same throughput as the pipelined HotStuff [51] given rich network connectivities (detailed in § IV-B).

We use a system model to theoretically prove the safety and liveness of SBFT. We implemented a prototype of SBFT with more than 3000+ lines of code and tested it on 16 machines in 4 regions. Our testbed experiments show that SBFT can maintain high throughput and low latency in WAN with unreliable links. For instance, SBFT has a similar throughput as HotStuff in a fault-free environment but can reduce more than 80% of consensus latency. Besides, when link failure probability reaches 0.1, SBFT can retain at least 70% of the original throughput and constantly low latency in different experiment configurations, while the baseline HotStuff can retain less than 40% of the original throughput. We further perform the faulty-probability-determined (FPD) model [28] simulation to evaluate the link fault tolerance of SBFT. The results show that existing protocols like PBFT and HotStuff start to consistently trigger leader election (even if the leader is correct) when link failure probability exceeds 0.2, while SBFT can run stably even when link failure probability reaches 0.4.

II. BACKGROUND AND MOTIVATION

A. Replicated Service

Replicated service is built based on state machine replication (SMR), which aims to make a set of possibly faulty distributed replicas execute the commands in the same order, maintaining a consistent state.

When designing a Byzantine-fault-tolerant (BFT) protocol for the replicated service, ensuring the following two properties is critical.

- **Safety.** All correct replicas commit and execute commands in the same order.
- **Liveness.** All correct replicas will eventually commit and execute new commands.

When less than $1/3$ byzantine replicas exist, a well-designed BFT protocol with asynchronous commit-rules (i.e., commit a command after collecting enough messages instead of waiting for a sufficiently long time) can achieve safety regardless of network conditions. Yet, it achieves liveness only if the assumed network condition is satisfied, as discussed below.

B. Synchronous Liveness Condition

Synchronous liveness condition refers to the assumption that the message delay between two correct nodes is bounded by an (unknown) value Δ, i.e., the network is stable. BFT protocols, depending on the synchronous liveness condition, often elect a leader to drive the consensus, and such protocols probably achieve the fastest consensus [27] in theory. Considering that the leader may fail, these protocols set a timer to monitor the

process of consensus. Upon timeout, the protocol considers the leader failed, and a view-change protocol is triggered to select a new leader, during which consensus cannot proceed.

Although synchronous liveness condition is beneficial to achieve fast consensus, it may hurt the stability of BFT protocols deployed on WAN, on which network jitters, faults, attacks (such as eclipse [21], [42], [44], BGP hijacks [3], [45], and DDoS [30], [31].) appeared frequently. When the message delay on a link exceeds the assumed bounded value Δ, the link is considered as temporal faulty. Accumulated faulty links will trigger frequent view changes, which may eventually hurt liveness. In Fig. 2, we simulate a replicated service with 61 nodes to demonstrate this.

Broadcast-based partial synchronous BFT (BP-BFT). BP-BFT protocols [4], [8], [9], [20], [52] require all nodes to broadcast their votes on the leader's proposal and accept it when collecting votes from $2/3$ neighbors. Therefore, these protocols assume a fully connected network for correct nodes, i.e., all correct nodes can communicate with an authenticated communication channel. Only when all messages sent on these channels at t can arrive at $t + \Delta$ will BP-BFT protocols achieve liveness. Therefore, BP-BFT has a low tolerance for link failures. As shown in Fig 2, given 20% faulty nodes (i.e., $p_n = 0.2$), 10% more faulty links (i.e., $p_l = 0.1$) will cause a 90% probability of leader failure (i.e., $p_{lf} = 0.9$). However, BP-BFT relies on all replicas to collect votes by themselves, thus can achieve higher link fault tolerance when achieving a simple announcement mechanism and $f + 1$-acceptance rule, i.e., Passive-SBFT.

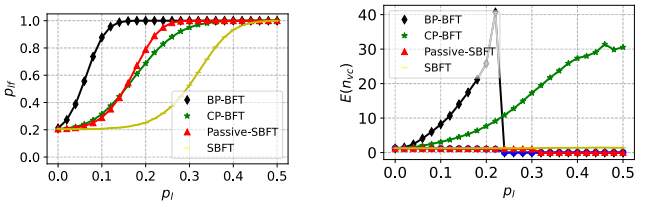
Collector-based partial synchronous BFT (CP-BFT). CP-BFT protocols [16], [23], [43], [51] let the leader (or some other fixed nodes [16]) to be the collector, which is responsible for collecting votes to create a certificate and send it to other nodes. Therefore, synchronous communication between the leader and correct nodes is sufficient to achieve liveness, yielding a higher link-fault tolerance than BP-BFT protocols. However, these protocols still fully rely on a well-connected leader to achieve liveness. As shown in Fig 2b, when $p_n = 0.2$ and $p_l = 0.1$, 8 view changes are expected ($E(n_{vc})$) for CP-BFT protocols to find a new leader.

Synchronous BFT (S-BFT). Synchronous BFT protocols [1], [19], [29], [39] rely on complete synchronization to achieve both liveness and safety. Thus, they achieve $1/2$ Byzantine fault tolerance. However, when deployed in a network with unreliable links, synchronous protocols may not achieve safety, i.e., correct nodes may commit different commands for the same sequence.

In summary, although the synchronous liveness condition enables fast BFT protocols given a synchronous network, it is ill-suited for achieving stable consensus in WAN with unreliable links.

C. Relaxed Liveness Conditions

Asynchronous BFT (A-BFT). Asynchronous BFT protocols [10], [11], [12], [18], [34], [49] relax the liveness condition by assuming no maximum message delays, thus can tolerate unstable network links. However, these protocols still require reliable links between all correct nodes, and the sent messages must arrive eventually to achieve liveness. Followed by the pioneering FLP impossibility [14] (i.e., no deterministic consensus is reachable in an asynchronous network), asynchronous BFT protocols require all nodes to propose a block just as a leader, and then sort these committed blocks (a total order) with a randomized consensus. Therefore, compared with leader-based protocols requiring 3 message latency to achieve consensus, asynchronous BFT protocols will incur more latency caused by the process of achieving a total order among blocks. For instance, state-of-the-art total-order protocol needs an average of 12 message latency to terminate [17].



(a) Probability of leader failure (b) Expected view changes

Fig. 2. p_{lf} and $E(n_{vc})$ with respect to p_l when $p_n = 0.2$.

Partially connected BFT (PC-BFT). Prior art (such as [6], [7], [41], [46]) on partially connected networks usually design the protocols that can run stably on an assumed topology, which the network must satisfy to achieve liveness. However, existing works all study static consensus, which may tolerate more link faults, but introduce high communication overhead when the network is synchronous. For example, the protocols in [41] assume an arbitrary dynamic network but introduce exponential communication complexity; Bonomi et al. [7] avoids exponential transmission of message payload in the traditional reliable broadcast (RBC) protocol with the assumption of $2f+1$-connected topology (i.e., any two nodes have $2f+1$ disjoint paths), yet it still introduces $O(n^2|B|+n!)$ communication complexity. Besides, these works mainly study the one-shot consensus, which cannot be easily applied to replicated services that require consecutive consensus.

D. Motivation

The motivation of our SBFT is to find a relaxed liveness condition (a prerequisite for achieving stable consensus in WAN), and meanwhile, retain consensus efficiency (i.e., low communication complexity and consensus latency) that is usually only possible under the synchronous liveness condition. To this end, we propose the following key designs in SBFT:

i) SBFT relaxes the strong dependency on the leader and lets the *anchor* drive fast consensus, thus can proceed in a relaxed *anchor*-based topology.
ii) SBFT assumes a dynamic network and allows nodes to adjust their consensus strategies based on their perceived network connectivity. Thus, it can achieve the same communication complexity as BP-BFT, though on a relaxed *anchor*-based topology.

We analytically compare SBFT with the related BFT protocols in Table I.

III. System Model

We assume a system consisting of $n = 3f + 1$ replicas, and index each replica with an integer in $\{0, \ldots, n-1\}$. We use a static byzantine failure model [1], [8], [32], [51], i.e., the adversary decides a fixed byzantine replica set B ($|B| \leq f$) before starting the consensus, and the remaining nodes are correct. We assume the cryptography techniques (such as collision-resistant hashing, message authentication codes (MACs), and signatures) are secure.

Network. We assume an unreliable dynamic network, on which replicas communicate with a point-to-point and authenticated bi-direction communication channel (e.g., the TLS channel). Unreliable means that all channels can be correct or faulty, r_1 and r_2 are neighbors to each other only when the channel between them is correct. Thus all correct channels and related replicas construct the network topology. Dynamic means that the network topology can change over time. Authenticated means that a replica r receives a message from its correct neighbor only if the neighbor sent the message to r, i.e., no replicas can disguise a correct replica to send a spoof message. We denote a channel to be correct when all messages sent on it at t will be received before $t + \Delta$, and other channels are denoted as faulty.

Goals. Like the traditional BFT protocols, safety and liveness are two primary goals of SBFT. SBFT achieves safety without any assumption on the network topology but achieves liveness during the anchor-based interval, i.e., the network presents an anchor-based topology. We give three related definitions as follows.

Definition 1 (Anchor): An anchor *is a correct replica, which has at least $2f$ correct neighbors.*

Definition 2 (Anchor-based topology): There is at least one anchor *in the anchor-based topology, and all correct replicas have at least one* anchor *neighbor.*

Definition 3 (T_s-anchor-based interval): A time interval T_s, during which the network is anchor-based topology.

Another goal of SBFT is to adapt to the dynamic topology, which can help SBFT run stably on WAN. We assume that the *anchor* nodes are fixed between any two topology-change, then we give a supplement definition as follows.

Definition 4 ((T_s, T_d)-anchor-based interval): A time interval with length T_s, during which the network exhibits anchor-based topology, and the length of the time interval between any two network topology changes greater than T_d.

Based on the definitions, we describe the safety and liveness properties of SBFT as follows. With the description of liveness, the smaller the value T_d, the stronger the ability to adapt to the dynamic changes of the network.

- **Safety.** For any two correct replicas r_1 and r_2, suppose that r_1 and r_2 commit command c and c' respectively at sequence s, then $c = c'$.
- **Liveness.** There exist two limited values T_s and T_d, all correct replicas will commit at least one new command during a (T_s, T_d)-anchor-based interval.

IV. The SBFT Protocol

SBFT uses a pipeline mechanism [23], [51] and relies on a leader to order the commands. The leader packages the ordered commands as a block (also including a hash of its predecessor block) and pre-proposes the block to all other nodes. Correct replicas accept blocks based on some acceptance-rules (detailed in § IV-B). Once a block is accepted, replicas can commit its predecessor block, and propose its successor block (i.e., broadcast a vote on it). Before diving into the details of SBFT, we first introduce some essential building blocks. We will explain how these technologies help improve stability.

A. Relaxing Complete Leader Dependency

Traditional pipelined BFT protocols require the leader to be an *anchor* to drive the consensus, and the leader will pre-propose a new block only after it has accepted its predecessor block. Such an interaction mode may affect the liveness of BFT protocols in a dynamic *anchor*-based topology, because every network topology change may cause the existing leader to be a non-*anchor*.

Considering an illustrative example of a replicated service with 4 replicas and $2f+1 = 3$. As shown in Fig. 3, the leader can only communicate with its *anchor* neighbor r_2 correctly, r_1 and r_3 will not output any vote since they cannot receive the pre-proposed block from the leader. Therefore, the consensus stalls and is forced to enter the view change process.

To address the limitation in the traditional pipelined BFT protocols, We first use a simple announcement mechanism to ensure the blocks arrive at all replicas. As shown in Fig. 4a, all replicas will announce the blocks they received, and reply to replicas who require missed blocks from them. Therefore, at most 10Δ is enough for a block arriving at $2f + 1$ correct replicas when the leader has an *anchor* neighbor. Besides, we give the claim 1 and prove it in Appendix A.

Claim 1: In anchor*-based topology, when the correct leader broadcasts a message, and all nodes help transmit it with the announcement mechanism, it takes at most 10Δ for all correct replicas to get the message.*

Based on the announcement mechanism, we further introduce the *pipelined pre-proposing mechanism*: the leader pre-proposes new blocks at a fixed rate without waiting for consensus (shown in Fig. 4b). Replicas will add all received blocks to a consensus-waiting queue (denoted as *cw-queue*), and vote them sequentially (shown in Fig. 4c). With such a consensus mode, all *anchor* replicas can collect 3 votes and accept a block although the leader is non-*anchor*. When the *cw-queue* is not empty, all *anchor* replicas accept a new block, on average, in one round interaction. Considering that at least one round of interaction is necessary for nodes to collect $2f + 1$ votes, accept a block, and propose a new block, the improved interaction model can achieve the theoretical optimal block throughput in the pipelined BFT protocols.

B. Anchor Based Acceptance-Rules

Traditional BFT protocols mainly use the $2f + 1$-*anchor* acceptance-rule, which requires the correct replicas to collect

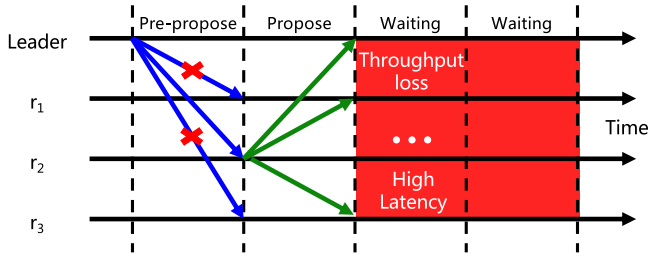


Fig. 3. The traditional pipelined interaction model with a non-anchor leader.

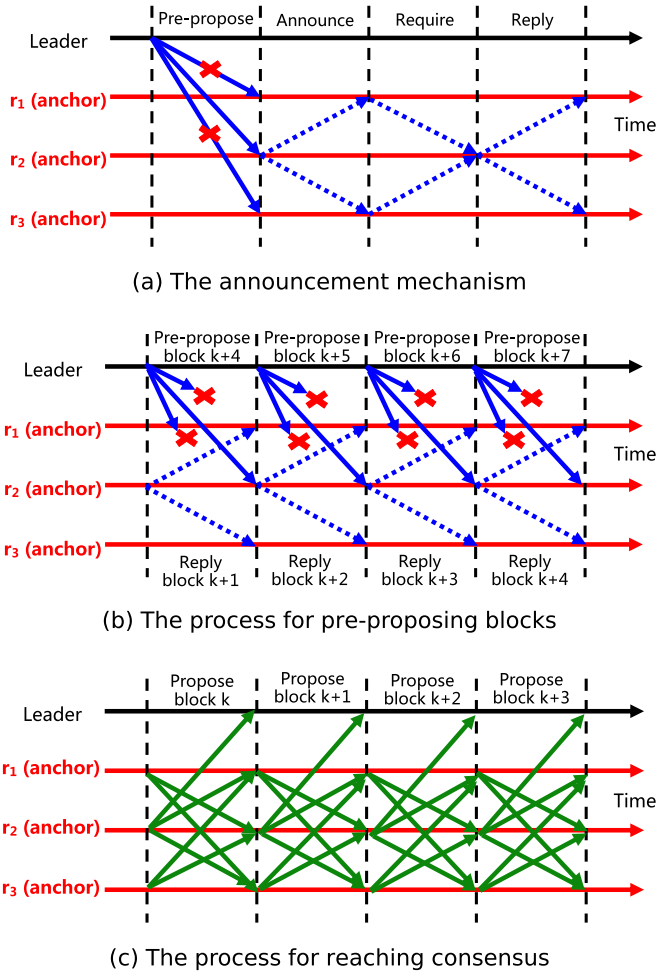


Fig. 4. Improved pipelined interaction model built on separation of the leader and consensus.

$2f+1$ votes ($2f$ votes from their neighbors and 1 vote from themselves) to accept a block. Therefore, at least $2f+1$ *anchor* nodes are needed for these protocols to achieve liveness. While in *anchor*-based topology, the number of *anchor* replicas varies from 1 to $2f+1$. Thus, SBFT introduces another two *anchor*-based acceptance-rules.

f+1-anchor Acceptance-Rule. In SBFT, when $f+1$ replicas claim to have accepted a block, at least one of them is correct and indeed receives $2f+1$ votes. Based on such an idea, we define the $f+1$-*anchor* acceptance-rule as follows.

- **Accept.** Upon collecting $2f+1$ votes for the same block, replica r accepts the block; Upon collecting $f+1$ votes for the same block, if a replica r has not accepted its predecessor block, then r will accept the predecessor block.

Based on the $f+1$-*anchor* acceptance-rule, *anchor* replicas accept a block by collecting $2f+1$ votes and further broadcast the vote on the next block. Therefore, correct replicas can proceed to achieve consensus as long as they have at least $f+1$ *anchor* neighbors. Comparing with $2f+1$-*anchor* acceptance-rule that requires at least $2f+1$ *anchor* replicas, $f+1$-*anchor* acceptance-rule reduces the topology requirement while retaining the $O(n^2)$ communication complexity.

1-anchor Acceptance-Rule. 1-*anchor* acceptance-rule relies on the *anchor* replicas to additionally prove they indeed received $2f+1$ votes by broadcasting quorum certificates (QC, a message constructed by $2f+1$ vote messages) upon accepting a block. In particular, the definition of 1-*anchor* acceptance-rule is defined as follows.

- **Accept.** Upon collecting $2f+1$ votes for the same block, replica r accepts it; upon collecting $f+1$ votes for the same successor block, or receiving a QC for a block that has not been accepted, replica r accepts it.

Based on the 1-*anchor* acceptance-rule, SBFT can always achieve liveness in *anchor*-based topology. However, due to the extra broadcasting of the QC by the *anchor* nodes, the communication complexity is $O(n^3)$ when there are $2f+1$ *anchor* exist. We will discuss how to achieve $O(n^2)$ communication complexity via adaptive consensus in § V-C.

C. Adaptive Consensus Strategies

The final critical building block is adaptive consensus, aiming to achieve efficiency when using 1-*anchor* acceptance-rule while also adapting to dynamic *anchor*-based topology, achieving fast recovery when the network topology changes.

In the traditional leader-based BFT protocols, replicas cannot distinguish network failure from leader failure because of the FLP impossibility. Therefore, replicas can only trigger the view change when consensus fails to work out. Only when the network meets the synchronization assumption and a correct leader is elected can liveness be achieved. The core idea for adaptive consensus is to use a more lightweight local-state notification to replace the view change when the network meets the requirement of *anchor*-based topology but not the synchronous network. Based on the local-state notification mechanism, replicas can subscribe to QC from the potential *anchor* neighbor and acquire liveness in *anchor*-based topology. However, one challenge in achieving the idea is to decide when to trigger the local state notification instead of the view change. To resolve such a challenge, we first define the stable state of SBFT as follows.

Definition 5 (Stable state): The stable state means that the network is anchor*-based topology, no conflict blocks from the leader are observed, and all correct replicas have subscribed from at least* 1 anchor *neighbor for the QC.*

When in *anchor*-based topology, a legitimate leader can consistently pre-propose new blocks, and the *anchor* will help to transmit them to fill *cw-queue* of all correct nodes.

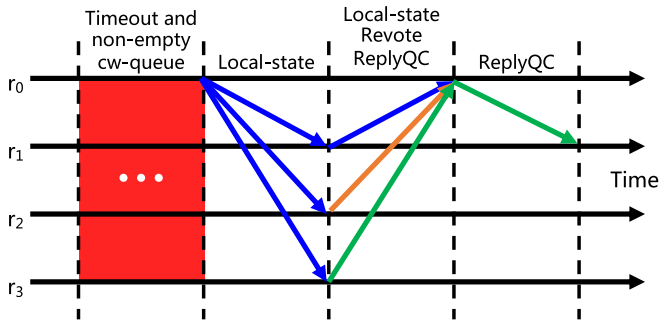


Fig. 5. The local state notification mechanism.

When the *cw-queue* of all correct replicas are non-empty, we can prove that the disruption of the stable state is regardless of the leader. We give the claim 2 and prove it in Appendix A.

Claim 2: When consensus is stable, and the cw-queue *of all correct replicas are non-empty. Suppose that all correct nodes have accepted the same block at T, these nodes accept no more than k new blocks at $T + k\Delta$, and no less than k new blocks at $T + 2k\Delta$.*

Therefore, when consensus fails to work out but the *cw-queue* is non-empty, replicas can attempt to use the local state notification mechanism to acquire liveness. As shown in Fig. 5, a replica can broadcast the local-state message to tell the basic information of its latest accepted block. Such behavior is an attempt to subscribe to QC from neighbors who have the QC for a higher block. replicas who received a local-state message from r could have three behaviors. The first is to reply to r a local-state message when only having QC for a lower block. The second is to revote on the next block by replying to r a vote message when having QC for the same block. The third is to reply to r the QC for a higher block, which means a successful subscription, i.e., the replica will send the QC to replica r when accepting a new block. Based on such a mechanism, we give the claim 3 and prove it in Appendix A.

Claim 3: In anchor*-based topology and the* cw-queue *of all correct replicas are non-empty, at most* 5 *rounds of state notification can ensure that SBFT enters a stable state.*

Considering that the byzantine leader can pre-propose different blocks at the same height (i.e., conflicted blocks), and all these conflict blocks will be relayed to correct replicas, replicas should trigger view changes (detail in § V-D) upon detecting such Byzantine activities.

V. Protocol Details

This section describes the details of SBFT, which include three parts: passive-SBFT, adaptive consensus, and view change protocol. Passive-SBFT is the normal case of SBFT, which builds on the pipelined interaction model and $f+1$-*anchor* acceptance-rule and relaxes complete dependency on the leader. Adaptive consensus uses a dynamic subscription mechanism and the 1-*anchor* acceptance-rule to tolerate more link failures without sacrificing efficiency.

A. Notations

Running of SBFT can be divided into continuous views, and every view is denoted by a number v. The leader in view v is replica i such that $i = v \pmod N$ [8], [19], [32], and it is responsible for coordinating the acceptance of new blocks in the steady-state protocol, which is designed in a pipelined version. All replicas will monitor the leader. When replicas suspect leader failures, they will change the leader based on the view change protocol. In the following description, we use $H(m)$ to denote hash digest of the message m, $\sigma_r(m)$ to denote a signature of replica r on $H(m)$, and $< m >_r$ to denote the message $(m, \sigma_r(m))$.

Block Format. Block has the format as follows:

$$B_k := (b, k, v, H(B_{k-1}), H(B_k), \sigma_L) \tag{1}$$

In block B_k, b includes the packaged commands, k is the height of block, $H(B_{k-1})$ is hash digest of the prior block B_{k-1}, v is the view number, $H(B_k)$ can be get by computing the Hash of $(b||k||v||H(B_{k-1}))$, σ_L is the signature of the leader L on $H(B_k)$. The first block B_0 has the format $(b, 0, 0, \perp, H(B_0), \perp)$.

Block Relationships. If a block B_k contains hash of another block B_{k-1}, we say B_k extends B_{k-1}, then B_k is successor of B_{k-1}, and B_{k-1} is predecessor of B_k. For any two blocks B_k and $B_{k'}$, if there exists blocks $B_{k_0}, B_{k_1}, \ldots, B_{k_n}$, where B_{k_i} extends $B_{k_{i-1}}$ when $1 \le i \le n$, B_k extends B_{k_n}, and B_{k_0} extends $B_{k'}$, then we say B_k is before $B_{k'}$, $B_{k'}$ is after B_k. if two blocks do not have any of the above relationships, we say they are in conflict with each other. In addition, for block B_k which is proposed in view v, and block $B_{k'}$ which is proposed in view v', we say B_k is higher than $B_{k'}$ if $k > k'$, B_k is newer than $B_{k'}$ if either (i) $k > k'$ or (ii) $k = k'$ and $v > v'$.

Quorum Certificate. We use $\sigma_r(H(B_k))$ to denote replica r's proposal signature for block B_k. We use $QC(B_k)$ to denote the quorum certificate (QC) for block B_k, which is either q different proposal signatures for B_k or $f+1$ different proposal signatures for B_{k+1} that extends B_k. When a replica receives $QC(B_k)$, it will accept B_k, and commit all blocks before B_k to its local committed chain. Replicas will execute the committed block B_k when all blocks before B_k are executed.

Block Status. We say block B_k is safe if it does not conflict with committed blocks from any correct replica. Every replica maintains four state variables. $B_{proposed}$ records the latest proposed block, $B_{committed}$ records the highest committed block, B_{locked} records the safe block, which is typically the latest accepted block or the decision block in view change protocol. B_{pre} records the latest received pre-proposed block.

B. Passive-SBFT

Passive-SBFT uses the $f+1$-*anchor* acceptance-rule, thus requiring correct nodes to have at least $f+1$ *anchor* neighbors. Passive-SBFT separates the leader from consensus, thus a non-*anchor* leader can also pre-propose new blocks, and other replicas will help to relay the pre-proposed blocks. We will explain the details of passive-SBFT according to Fig. 6.

Let v be the current view number and replica L be the leader of the current view.

- **Pre-propose.** Leader L constructs B_k that extends B_{pre}, then broadcasts $(\textit{pre-propose}, B_k)$ and sets B_{pre} to B_k.
- **Announce.** When replica r receives a valid *pre-propose* message for B_k that extends B_{pre}, it announces the $H(B_k)$ to other replicas and sets B_{pre} to B_k. When receiving an announcement of $H(B_k)$ from r' and not stored Block B_k, replica r will require from r' for the B_k.
- **Propose.** When $B_{proposed}$ is accepted or $\perp$, replica r gets B_k that extends $B_{proposed}$ in *cw-queue*, broadcasts $(\textit{propose}, H(B_{k-1}), H(B_k), k, v, \sigma_r(H(B_k)))$ and sets $B_{proposed}$ to B_k.
- **Accept.** When replica r collects (i) q valid *propose* messages for B_k, or (ii) $f+1$ valid *propose* messages for B_{k+1} that extends B_k in view v, it stores them as $C(B_k)$, sets B_{locked} to B_k, and executes $Learn(B_k)$. (i.e., the $f+1$-*anchor* acceptance-rule) Besides, replica r resets the *timer* to 2Δ.
- **Learn.** When executing $Learn(B_k)$, there are three cases:
  - If B_k extends B_{locked}, replica r sets $B_{committed}$ to B_{locked}, B_{locked} to B_k, then adds $B_{committed}$ to committed chain.
  - ElseIf B_k has the same height of B_{locked}, replica r sets B_{locked} and $B_{proposed}$ to B_k.
  - Else, replica r sets B_{locked} to B_k, and requests all missed blocks from other replicas.

Fig. 6. Passive-SBFT.

Pre-Propose. The leader L pre-proposes new blocks by broadcasting a *pre-propose* message, which includes the successor block of B_{pre} (we denote the block to be B_k, in which k equals to 1 plus the height of B_{pre}). Then, the leader changes the value of B_{pre} to record the latest pre-proposed block and adds it to *cw-queue*. When replica r receives a valid *pre-propose* message for B_k, it changes the value of B_{pre} to record the latest pre-proposed block and adds it to *cw-queue*.

To achieve the optimal throughput and avoid being suspected, a legitimate leader can pre-propose blocks faster at the beginning of the new view and then slow down when several blocks have accumulated in *cw-queue*. Besides, SBFT uses a simple announcement mechanism for transmitting blocks (details in Fig. 6). When deployed, SBFT can also use other existing protocols [35], [38] for saving network bandwidth.

Propose. Since the leader pre-proposes new blocks consistently, the *cw-queue* is nonempty in the steady state. Therefore, when accepting $B_{proposed}$, i.e., the latest proposed block in *cw-queue*, a replica r could simultaneously propose the next block that extends $B_{proposed}$. The block is denoted to be B_k, in which k equals to 1 plus the height of $B_{proposed}$, r proposes B_k by broadcasting a *propose* message and set $B_{proposed}$ to B_k to record the latest proposed block.

Accept. The QC is proof for accepting the related block. Therefore, when receiving $QC(B_k)$ and B_k has not been accepted, replicas will accept and execute the commands in it. The exact block execution process depends on the local state of the replica, as discussed below.

In passive-SBFT, there is no need for replicas to send *accept* messages. The *propose* message for a block is equivalent to the *accept* message for its predecessor block. Therefore, $QC(B_k)$ has two possible formats: one is q *propose* messages for B_k, and the other one is $f+1$ *propose* messages for B_{k+1} that extends B_k.

Learn. There are three cases for executing $learn(B_k)$ based on the local state of replicas. In the case where the accepted block B_k extends B_{locked}, B_{locked} can be safely committed by replicas to their local committed chains. Replicas then change the value of B_{locked} to B_k, which is safe to be extended in the next proposal.

In the case where B_k has the same height of B_{locked}, B_k must be a block decided in view change protocol (see details in § V-D), which means that B_k is safe to be extended by next proposal. Then, replicas only need to change the value of B_{locked} to B_k, because the blocks before B_k should have been committed in previous views.

If B_k is higher than B_{locked}, then the replicas have just recovered from failures and missed several accepted blocks during its failure. Therefore, the replicas change the value of B_{locked} to ensure the participation of proposing the next block, then request all missed blocks from other replicas.

Predict the latency and set the timer. As stated in Claim 5, when consensus is stable, and the *cw-queue* of all correct replicas are non-empty, SBFT can acquire throughput of no more than $1/\Delta$ and no less than $1/2\Delta$. Thus, SBFT can record the average time interval t_{avg} of accepting new blocks and predict Δ as t_{avg}. When accepting a new block, replica r reset the *timer* to 2Δ, the bounded value for SBFT accepts a new block in the stable state.

C. Adaptive Consensus

For the adaptive consensus to run successfully, every replica r uses a *timer* to monitor the consensus process and maintains a local set *To*, which is initialized to be an empty set and includes all replicas who have subscribed from r for QC. When r acquires a QC by collecting $2f+1$ propose signatures, it will send the QC to all replicas in *To*. We will explain the protocol details according to Fig. 7.

OnCertificate. With the 1-*anchor* acceptance-rule used in adaptive consensus, nodes may receive the QC from their neighbors. When receiving a QC that can accelerate the consensus, nodes will accept the block directly without waiting to collect q *propose* messages. However, When receiving a QC that cannot promote consensus, nodes can send the *unsubscribe* message to the neighbor who sent the QC to

- **OnCertificate.** When replica r receives a $QC(B_k)$ from r' in view v, if has not accepted B_k, it stores the $QC(B_k)$ and executes $Learn(B_k)$; if replica r has accepted B_k, it sends $(\textit{unsubscribe}, v)$ to r'.
- **Status notification.** When *timer* reaches to 0 and *cw-queue* is not empty, replica r broadcasts $(\textit{local-status}, v, H(B_{locked}), k)$, and resets *timer* to 2Δ. Whenever a replica r' receives a *local-status* message from r, there are three cases:
  - If B_r is after B_{locked}, replica r' constructs and sends a *local-status* message to r.
  - ElseIf B_r has the same height of B_{locked}, replica r' sends a *propose* message for $B_{proposed}$ to r.
  - Else, replica r' sends the local stored $QC(B_{locked})$ and a *propose* message for $B_{proposed}$ to r, and adds r to *To*.
- **Unsubscribe.** When replica r receives a valid *unsubscribe* message from r', it removes r' from *To*.

Fig. 7. Adaptive consensus.

cancel the subscription to avoid unnecessary communication overhead.

Status Notification. The network topology may have changed when the timer *timer* expires while *cw-queue* is non-empty. Then, replica r will join the status notification process instead of suspecting the leader.

When joining into the status notification process, replica r will broadcast a *local-status* message mainly for two goals. One is to attempt to subscribe QC from potential *anchor* neighbors. The other is to tell the poor-connected neighbors to subscribe to QC from r itself.

When a replica r' receives a valid *local-status* message from r, which includes hash of block B_k at height k, it proceeds based on its local state as follows:

- If B_k is after B_{locked} (indicating that r' is slow), replica r' constructs and sends a *local-status* message to r to subscribe the QC.
- If B_k has the same height of B_{locked} (indicating that r and r' are accepting blocks at the same rate), replica r' just sends a *propose* message for $B_{proposed}$ to r.
- If B_k is before B_{locked} (indicating that the connectivity of r is poor and cannot accept new blocks in time), replica r' sends the local stored $QC(B_{locked})$ to r and adds r to *To*. Considering that r could be a recovered *anchor* without the latest status, r' will also send a *propose* message for $B_{proposed}$ to r, help replica r to collect q *propose* messages and confirm its identity of *anchor* node.

Considering that a byzantine node can first disguise as *anchor* node and then stop issuing certificates, in which case, the victim nodes would only need to find another *anchor* neighbor by joining the status notification process when timeout. When completing the broadcast of the *local-status* message, replica r has subscribed QC from all its potential *anchor* neighbors. Only after providing QC for accepting new blocks will these neighbors not be unsubscribed, thus the malicious behaviors of byzantine nodes will not affect protocol liveness.

Set the timer. After broadcasting a *local-status* message, replica r will reset *timer* to 2Δ, for the reason that r will receive q *propose* messages or a *local-status* message in 2Δ. In both cases, r can proceed to accept a new block.

D. View Change Protocol

The view change protocol is triggered when replicas observe abnormal behaviors of the current leader. The goal of view change is to elect a new leader and decide the block that is safe to be extended in the next view. We explain the protocol using Fig. 8.

Leader Failure Suspicion. During the execution of the steady-state protocol, all replicas monitor the leader to detect possible malicious behaviors. Whenever replica r suspects the leader in view v, it will broadcasts $\langle \textit{suspect}, v \rangle_r$. There are two possible triggers for raising leader failure suspicions. The first one is the timeout. Every replica maintains *timer* to monitor the rate of accepting new blocks. Different from traditional BFT protocols, replicas will proceed to confirm the status of *cw-queue* when *timer* reaches 0. When *cw-queue* is non-empty, replicas will join the status notification process (detailed in § V-C) and reset the *timer*. When *cw-queue* is empty, there are two possible reasons:

- the leader fails, indicating a legitimate suspicion;
- the predicted value Δ is smaller than actual message delay δ, which means the network is unstable, nodes will increase the predicted value Δ to accelerate consensus recovery;

The other suspicion case is observing two conflict blocks with a valid signature of the leader. This is definitive proof that the leader is byzantine faulty.

Noted that replicas will proceed consensus process in view v after broadcasting the *suspect* message. Besides, whenever replica r receives a *suspect* message with an older view from replica r', r will send the latest *new-view* message to r' to bring it back to the latest view.

Status and New-view. The Status and New-view processes are similar to traditional BFT consensus. The core difference is that replicas will forward the received *status* message to the new leader, and forward the *new-view* message to other replicas. we will describe the details as follows.

Upon collecting q valid *suspect* messages for the same view v from different replicas, a replica r will broadcast them, enter view $v+1$, and stop to propose any blocks in view v. Then, replica r broadcasts a *status* message, which includes the new view number $v+1$, B_{locked}, view number v', signatures on $H(v+1, B_{locked}, v')$, and the certificate for B_{locked} in view v'. Then it resets *timer* to 10Δ, the derived maximum latency for receiving the *new-view* message (see details below). Considering that the new leader may not be an *anchor*, all replicas will forward the received *status* messages to it (note that the expected leader for each view is predetermined based on its ID).

Let L and $L^{'}$ be the leaders of view v and $v+1$, respectively.

- **Suspect.** There are two situations for suspecting the current leader L
  - When *timer* reaches 0 and *cw-queue* is empty, replica r broadcasts $\langle suspect, v\rangle_r$, and reset *timer* to 2Δ.
  - When replica r observes conflict blocks with valid signature of leader L, it broadcasts the blocks and signatures, then broadcasts $\langle suspect, v\rangle_r$.
- **Status.** On collecting q valid *suspect* messages for view v, replica r broadcasts them, enters view $v+1$, and resets *timer* to 10Δ. After that, replica r broadcasts $\langle status, v+1, H(B_{locked}), v', k'\rangle_r$ and $C_{v'}(B_{locked})$, then forwards all received *status* messages and certificates to $L^{'}$.
- **New-view (leader $L^{'}$).** On collecting q valid *status* messages, $L^{'}$ finds the newest block $B_{k'}$, constructs $B_{k'+1}$ that extends it, and broadcasts $\langle new\text{-}view, v+1, B_{k'+1}, S, C_{v'}(B_{k'})\rangle_{L'}$. Then, $L^{'}$ sets B_{pre} to $B_{k'+1}$, and excutes $Learn(B_{k'})$.
- **New-view (replica).** When replica r receives a *new-view* message for $v+1$, it verifies that S includes q valid *status* messages, and $B_{k'+1}$ extends the newest block in these messages, then forwards it to other replicas. After that, replica r resets *timer* to 2Δ, sets B_{pre} to $B_{k'+1}$, and executes $Learn(B_{k'})$.

Fig. 8. View change protocol.

When the expected leader L' has received q *status* messages for view $v+1$ from different replicas, it finds the newest block $B_{k'}$ in these messages, and gets $C_{v'}(B_{k'})$. Then L' constructs S, which includes the q *status* messages without certificates, to prove that $B_{k'}$ is safe to be extended in the next view. Besides, L' will constructs a new block $B_{k'+1}$ that extends $B_{k'}$, and broadcasts *new-view* message.

For any replica r, upon receiving a *new-view* message, it will verify that S includes q valid *status* messages and $C(B_{k'})$ is valid for the newest block in these messages. It further verifies that $B_{k'+1}$ indeed extends $B_{k'}$. Upon verification, replica r will forward the *new-view* message to other replicas, and reset *timer* to 6Δ, which is the predicted maximum latency for accepting the next block.

We will prove in Appendix B that the newest block $B_{k'}$ from any $f+1$ correct replicas (using their local B_{locked}) is safe to be extended in the new view. Thus, after broadcasting the valid *new-view* message, all replicas will enter steady-state protocol and execute $Learn(B_{k'})$ and set B_{pre} to $B_{k'+1}$ to record the latest pre-proposed block.

Set the timer. Before entering the new view, replica r should have broadcasted $2f+1$ *suspect* messages. According to Claim 4 (the proof is in Appendix A), it will take at most 4Δ for all correct replicas to receive these messages and enter view $v+1$. Further, it will take at most 2Δ for all correct replicas to broadcast *status* messages and forward them to the new leader. Finally, it will take at most another 4Δ for all correct replicas to receive *new-view* messages. Therefore, when entering view $v+1$, replica r resets *timer* to 10Δ, and waits for the *new-view* message.

Claim 4: In anchor*-based topology, when the correct leader broadcasts a message, and all nodes help transmit it directly, it takes at most 4Δ for all correct replicas to get the message.*

E. Efficiency

We analyze the message complexity, communication complexity, and throughput for our consensus protocol in a steady state. We use δ to denote the message delay, which is smaller than a predicted value Δ.

Block Throughput. We consider the maximum block throughput SBFT can achieve, i.e., the throughput when the *cw-queue* of all correct replicas are nonempty. According to Claim 2 (the proof is in Appendix A), SBFT needs average at most 2 message latency to accept a new block. When the network connectivity is high, i.e., all correct nodes can collect $2f+1$ *propose* messages from their neighbors, SBFT only needs average 1 message latency to accept a new block (According to Claim 5, which is proved in Appendix A). In comparison, HotStuff needs an average 2 message latency to accept a new block.

Communication Complexity. We can let all nodes send a hash before relaying the $pre-propose$ message. Therefore, when a block has the size of $O(|B|)$, SBFT can achieve $O(n|B|+n^2)$ of communication complexity, the same as PBFT, and higher than HotStuff of $O(n|B|+n)$.

Consensus latency. SBFT uses the two-chain commit rule, and the consensus latency equals the latency for a block transmitted to all correct replicas plus two block acceptance intervals. Therefore, the consensus latency is three message delays in a synchronous network, the same as in a PBFT.

Claim 5: In a synchronous network with fully connected topology, when the cw-queue *of all* anchor *nodes are non-empty. Suppose all* anchor *nodes have accepted the same block B at T; they will accept at least k new blocks with heights higher than B at $T+k\Delta$.*

F. Security Analysis

We mainly prove the properties of safety and liveness in Appendix B, and the three theorems are shown below.

Theorem 1 (Safety): Correct nodes can't commit two different blocks at the same height.

Theorem 2 (Liveness): There exist two limited values T_s and T_d; all correct replicas will commit at least one new command during a (T_s, T_d)-anchor-based interval.

VI. Evaluation on Link Fault Tolerance

This section evaluates the link fault tolerance of SBFT with many other BFT protocols.

Experiment setup. We use a boolean matrix G to simulate the topology of a replicated service. $G[i][j]$ will be set to false when the link between r_i and r_j is faulty and be set to true when the link between r_i and r_j is correct. We use the faulty-probability-determined (FPD) model [28] and assume that the failures among different nodes and links are independent. The first replica is the leader in the initial state, and we generate the topology based on the preset node failure probability (p_n) and link failure probability (p_l). Then, we decide the state of the generated topology and three cases that could result in leader failures:

- The leader is faulty.
- The leader is correct but fails because of link failures.
- The network topology does not meet the requirement of the liveness condition.

When the leader fails, the consensus achieves liveness only if another effective leader exists. We examine the nodes in increasing order and use n_{vc} to record the number of view changes required to find an effective leader. Our evaluation results only count cases where n_{vc} is not infinity (i.e., the leader fails while an effective leader exists).

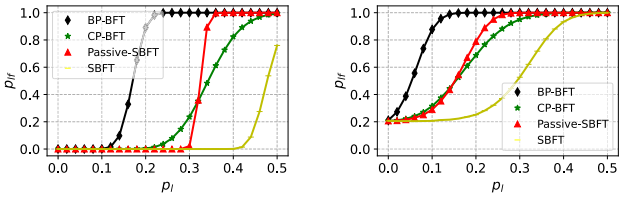
Baselines. SBFT aims to relax the synchronous liveness condition without sacrificing efficiency. Therefore, we mainly consider the partial synchronous BFT protocols as the baselines. There are two interaction modes in the existing partial synchronous BFT protocols. One is BP-BFT, such as PBFT [8], where all replicas broadcast *propose* messages. The other CP-BFT, such as HotStuff [51], where all replicas only communicate with the leader.

Probability of Leader Failure. We simulate a replicated service consisting of 61 nodes and consider two scenarios with different node failure probability, i.e., $p_n = 0$ and $p_n = 0.2$. For each scenario, we vary the p_l (link failure probability) from 0 to 0.5 and plot the p_{lf} (probability of leader failure) with respect to p_l. The results are shown in Fig 9.

When $p_l = 0$, all three consensuses have same p_{lf} (equal to p_n). When $p_n = 0$, SBFT can ensure stability even when $p_l = 0.4$, while CP-BFT can be stable when $p_l = 0.2$, BP-BFT can only be stable when $p_l = 0.1$. Considering that $p_n = 0.2$, SBFT can still ensure stability when $p_l = 0.15$, while BP-BFT and CP-BFT may trigger view change when there are any link failure exists.

To further demonstrate the stability of SBFT, we study the scenario where the leader is experiencing attacks by only considering the link failures associated with the leader. As shown in Fig 10, BP-BFT and CP-BFT are stable only if p_{ll} is smaller than 0.2, while passive-SBFT and SBFT are stable even when p_{ll} is 0.9.

Cost for recovering from leader failure. We further evaluate the cost (i.e., the number of view changes $E(n_{vc})$) for recovering from leader failure. According to the results shown in Fig 11, When the leader fails, about one view change is needed for SBFT to find a new leader and recover, while BP-BFT and CP-BFT all have a much higher $E(n_{vc})$ to find a new leader. The reason is that BP-BFT and CP-BFT fail to relax the dependency on the leader and require the leader to have higher network connectivity. Therefore, when cumulative faulty links trigger the view change, the probability of finding an *anchor* as a new leader is low.



(a) $p_n = 0$ (b) $p_n = 0.2$

Fig. 9. The leader failure probability, i.e., p_{lf} with respect to p_l under varying p_n.

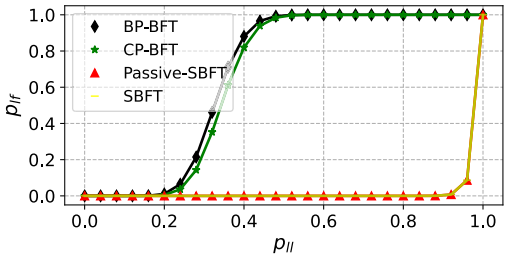
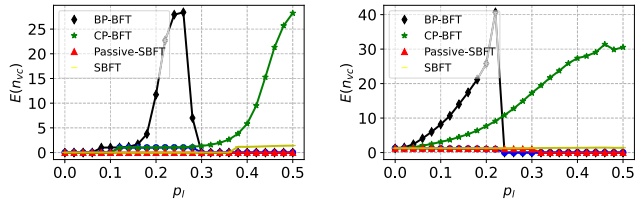


Fig. 10. p_{lf} with respect to p_{ll}.



(a) $p_n = 0$ (b) $p_n = 0.2$

Fig. 11. Cost for recovering from leader failure, i.e., $E(n_{vc})$ respect to p_l under varying p_n.

Influence of node size on stability. The node size will usually affect the stability of consensus. We choose SBFT, SBFT without adaptive consensus (i.e., passive-SBFT), BP-BFT, and CP-BFT as the experimental objects and analyze the leader failure probability p_{lf} with respect to the link failure probability p_l in different node sizes, and get the results shown in Fig 12. We can find that for all four protocols, and there exists a limit value p_{lim}, when $p_l < p_{lim}$, the consensus becomes more stable with the expansion of node size, and the p_{lf} will eventually approach to 0. When $p_l > p_{lim}$, with the expansion of node scale, the p_{lf} will eventually approach 1. Therefore, higher p_{lim} is equivalent to higher link fault tolerance when $n \to \infty$.

In the actual environment of BFT protocols, the node size is limited. Therefore, according to the results in Fig 12, SBFT can achieve the highest stability in the actual deployment environment.

VII. Evaluation on Performance

This section evaluates the performance of SBFT compared to HotStuff.

Implementation Details. We implement SBFT based on the HotStuff codebase. We rewrite the consensus and synchronizer

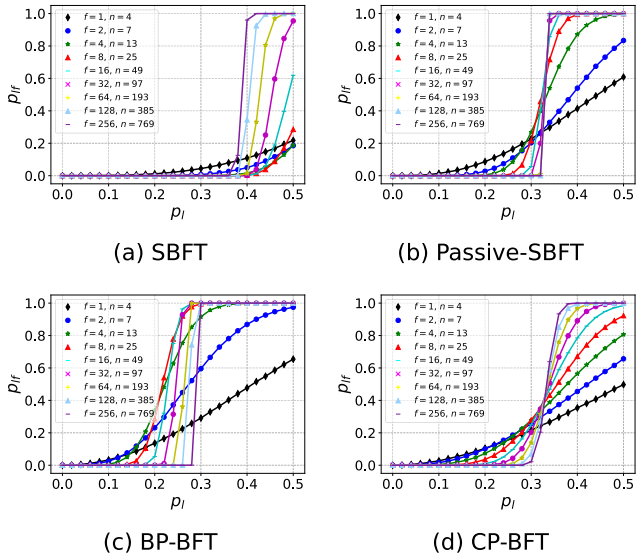


Fig. 12. p_{lf} with respect to p_l and node size given $p_n = 0$.

package because SBFT has a different consensus and view-change protocol from HotStuff. Besides, we add a relations package to support the maintenance of set *To* and the simulated fault network links. At the beginning of the program, nodes establish communication channels through TLS sockets. We will tag all communication links as correct or faulty according to the preset probability of faulty links and fresh the distribution of faulty links based on predefined fresh-duration. When receiving messages from tagged faulty links, replicas will ignore them. Overall, implementing the functionalities described above required the addition/adaptation of about 3000 locs to the HotStuff codebase.

Experiment Setup. All the replicas and clients were installed on ECS SA5.2XLARGE32 instances. Each instance has 8vCPUs supported by AMD EPYC Bergamo(-/3.1GHz) and has a maximum network bandwidth of 100Mbps. All the instances were distributed across 4 regions, i.e., Silicon Valley, Singapore, Frankfurt, São Paulo, and the RTTs between these datacenters are tested and shown in Tab. II. We deploy 16 ECS instances evenly distributed on the four regions, and each instance will run one or two replicas. Besides, we deploy another 4 ECS instance that is evenly distributed in the four regions, and each instance will run one client. When running the experiments, all commands carry 20B payloads, making the command size to be 32B.

When running the experiments, we set the Δ to $180ms$, i.e., $RTT/2$ between São Paulo and Singapore. Besides, we only consider the transmission delay for broadcasting blocks and *new-view* messages and add the computed transmission delay when resetting the *timer*. The transmission delay is computed using $\Delta_{td} = CommunicationBits/Bandwidth$. The $Bandwidth$ is the leader's bandwidth; $CommunicationBits$ can be obtained by computing $(N-1)|m|$, in which N is the number of replicas and $|m|$ is the size of the broadcasted message m, i.e., block or *new-view* message.

Baselines. SBFT aims to relax the synchronous liveness condition without sacrificing the efficiency of partial synchronous BFT protocols. SBFT is implemented based on the HotStuff codebase and aims to optimize the throughput and latency of pipelined BFT consensus, thus we use HotStuff [51] as the baseline, including HotStuff-ecdsa which uses ecdsa as the cryptographic algorithm and Hotstuff-bls which uses bls12 as the cryptographic algorithm.

TABLE II
THE RTT BETWEEN DIFFERENT DATACENTERS (MS)

Regions	*silicon*	*Singapore*	*Frankfurt*	*São Paulo*
silicon	–	175	155	181
Singapore	175	–	182	360
Frankfurt	155	182	–	207
São Paulo	181	360	207	–

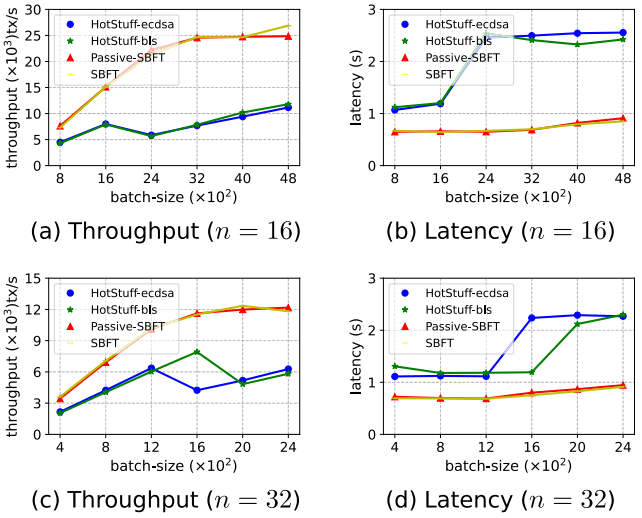


Fig. 13. Performance with respect to different batch sizes (small batch size).

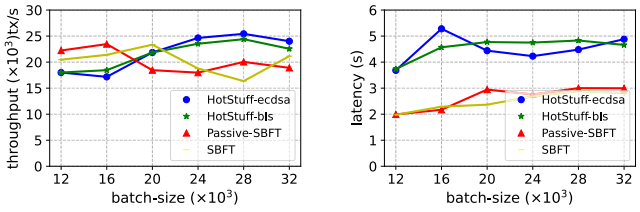
TABLE III
THE ADDITIONAL OVERHEAD CAUSED BY EACH BLOCK (SMALL BATCH SIZE)

Batch size		**Number of transmitted QC**		**Number of retransmitted block**	
$n = 16$	$n = 32$	$n = 16$	$n = 32$	$n = 32$	$n = 32$
800	400	1.12	6.03	0.96	1.62
1600	800	1.32	5.99	0.49	6.19
2400	1200	2.11	4.87	1.45	2.81
3200	**1600**	2.86	20.13	0.97	2.15
4000	2000	1.5	15.05	2.5	9.81
4800	2400	5.51	37.31	1.71	7.83
Baseline		240	992	210	930

A. Performance in a Fault-Free Environment

We study the performance of SBFT in a fault-free environment, i.e., set the link failure probability as 0. All experiments in this subsection are running 60 seconds, and the results are obtained from the statistical data from the 11th to 50th seconds.

Performance with small batch size. The throughput and latency are shown in Fig.13; SBFT and Passive-SBFT can achieve lower latency than HotStuff because of relaxed



(a) Throughput ($n = 16$) (b) Latency ($n = 16$)

Fig. 14. Performance with respect to different batch sizes (large batch size).

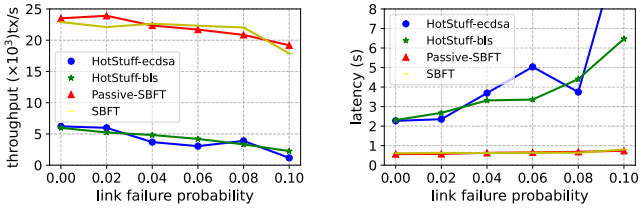
dependency on the leader. Besides, SBFT can achieve the optimal throughput without a large batch size because the leader can propose blocks sequentially instead of waiting for the consensus of the previous block to be finished. Thus, the leader's bandwidth can be fully used. While in HotStuff, the leader's bandwidth is wasted when waiting for the votes from replicas. In addition, when the bandwidth usage of the leader is high, we find that the performance of Hotstuff is easily affected by the worst links, i.e., the link between São Paulo and Singapore. This could be a potential drawback of collector-based BFT consensus.

In addition, we study the additional cost introduced by SBFT, i.e., the number of transmitted QC and retransmitted blocks, we record related data during the experiment and obtain the results shown in Tab. III. The baseline is calculated when all replicas broadcast QC and transmit the blocks. When we don't pursue the ultimate throughput (which leads to a meaningless increase in consensus delay), but instead choose a batch size that can acquire throughput close to the optimal throughput, i.e., the fourth line in Tab. III, the additional cost can be negligible compared to the baseline.

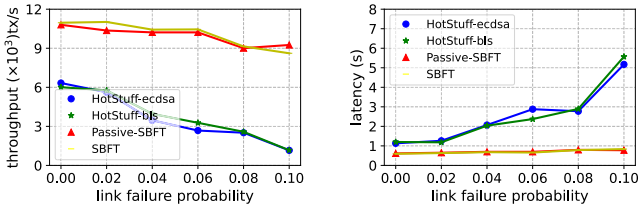
Performance with large batch size. To further study the impact of large batch size on the performance of SBFT compared to HotStuff, we run the experiments in 16 nodes scale and varying batch size from 12000 to 32000. As shown in Fig. 14, using a large batch size will not improve the throughput but improve the consensus latency of SBFT, thus using a small batch size i.e., 3200, is optimal for SBFT. HotStuff can acquire throughput similar to SBFT when using a large batch size. For example, when batch size is 28000, HotStuff can acquire the throughput of about $25000tx/s$ but latency approach to $5s$. While SBFT and Passive-SBFT can acquire a throughput of about $25000tx/s$ and latency of about $0.8s$ when using a batch size of only 3200.

B. Performance With Simulated Faulty Links

All experiments in this subsection are running 160 seconds, and the results are obtained from the statistical data from the 11th to 110th seconds. The batch size is 3200 for 16 replicas and 1600 for 32 replicas. The distribution of faulty links is refreshed every 20 seconds, and faulty links are generated according to the "math/rand" Golang package. In ith link refresh, we use i as the seed to rand to generate the random numbers. For the kth link, if the kth random number is less than the predefined link failure probability (p_l), we tag the kth link as faulty.

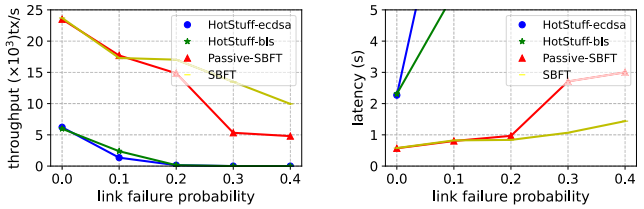


(a) Throughput ($n = 16$) (b) Latency ($n = 16$)

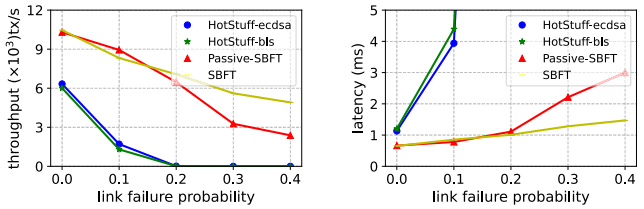


(c) Throughput ($n = 32$) (d) Latency ($n = 32$)

Fig. 15. Performance with low link failure probability.



(a) Throughput ($n = 16$) (b) Latency ($n = 16$)



(c) Throughput ($n = 32$) (d) Latency ($n = 32$)

Fig. 16. Performance with high link failure probability.

Performance with low link failure probability. We first consider the more common scenarios, where the network crash only causes a small percentage of faulty links between consensus nodes. As shown in Fig 15, when varying the link failure probability p_l from 0 to 0.1, SBFT and Passive-SBFT only reduced less than 30% of throughput while HotStuff reduced more than 80% of throughput.

Performance with high link failure probability. We further consider a relatively rare scenario where more than 10% of faulty links appear, which can be caused by network attacks such as BGP hijacks. The study on such a scenario is also meaningful because it can reflect the ability of consensus to resist network attacks. As shown in Fig 16, the throughput of HotStuff degrades to nearly 0 when $p_l = 0.2$, while SBFT maintains about 70% of the throughput and Passive-SBFT maintains about 60% of the throughput. Considering a scenario where the network is extremely terrible, where 40% of the

communication links between consensus nodes crash, SBFT still gets 40% of the throughput and gets a latency of no more than 1.5s.

VIII. Conclusion

This paper presents SBFT, a new BFT consensus with high link fault tolerance to be stable under network attack while ensuring efficiency on a well-connected network. SBFT uses 1-*anchor* acceptance-rule and relaxes strong dependency on the leader, achieving high link fault tolerance and optimal block throughput of pipelined BFT protocols. Besides, SBFT implements adaptive consensus to achieve communication complexity of $O(n^2)$. Theoretical and simulation results show that SBFT can simultaneously achieve efficiency and stability. The results of experiments in real-world deployment further demonstrate the feasibility and high link fault tolerance of SBFT.

Appendix A
Theoretical Proof for the Claims

Theorem 3 (Anchor distance): For any two anchor *replicas r_1 and r_2, then (i) r_1 and r_2 are neighbors and have at least other $f-1$ common correct neighbors, or (ii) r_1 and r_2 have at least $f+1$ common correct neighbors.*

Proof 1: Based on the definition of *anchor* and the quorum of the generalized BFT model, each *anchor* replica has at least $2f$ correct neighbors. Considering two anchor replicas r_1 and r_2, we use V_1 as the nodes set, which contains r_1 and all its correct neighbors, then $|V_1| \geq 2f+1$. We use V_2 as the nodes set, which contains r_2 and all its correct neighbors, then $|V_2| \geq 2f+1$. Considering that the total number of nodes is $3f+1$, the $|V_1 \bigcap V_2|$ must be no less than $f+1$. Therefore, the theorem is proved.

Corollary 1: In anchor*-based topology, when the correct leader broadcasts a message, and all nodes help transmit it with the announcement mechanism, it takes at most 10Δ for all correct replicas to get the message. When the correct leader broadcasts a message, and all nodes help transmit it directly, it takes at most 4Δ for all correct nodes to get the message.*

Proof 2: According to theorem 3, the distance between any two *anchor* nodes is no more than 2. Therefore, in *anchor*-based topology, every correct node has at least one *anchor* neighbor, and the distance between any two correct nodes is no more than 4. When the correct leader broadcasts a message, it takes at most 3 times of transmission for the message to arrive at all correct nodes. When all nodes transmit the received message directly, it takes at most $\Delta + 3 * \Delta$ for all correct nodes to get the message. When all nodes transmit the received message with the announcement mechanism, it takes at most $\Delta + 3 * 3\Delta$ for all correct replicas to get the message. Therefore, the Corollary is proven.

Corollary 2: In a synchronous network with fully connected topology, when the cw-queue *of all* anchor *nodes are non-empty. Suppose all* anchor *nodes have accepted the same block B at T; they will accept at least k new blocks with heights higher than B at $T + k\Delta$.*

Proof: Suppose that at $T = 0$, all *anchor* replicas have accepted the block B_k; they will propose the block B_{k+1} in *cw-queue* immediately. Therefore, at $T = \Delta$, all *anchor* replicas will receive $2f+1$ *propose* message for B_{k+1} and accept it. Suppose that at $T = m\Delta$, where $m > 1$, all *anchor* replicas have accepted block B_{k+m}, and proposed the block B_{k+m+1} in *cw-queue*. At $T = (m+1)\Delta$, all *anchor* replicas will receive $2f+1$ *propose* message for B_{k+m+1} and accept it. Therefore, the Corollary is proven. □

Corollary 3: In anchor*-based topology and the* cw-queue *of all correct replicas are non-empty, at most 5 rounds of state notification can ensure that SBFT enters a stable state and commits a new block.*

Proof: Suppose the stable state is disrupted, and replica r has the QC for the newest block B_k. According to theorem 3, it requires at most 3 rounds of state notification for all *anchor* nodes to get the QC. Then, another round of state notification can ensure that these *anchor* nodes collect $2f+1$ votes for the next block B_{k+1}, i.e., get the QC for a newer block B_{k+1}. Therefore, in the fifth round of state notification, all non-*anchor* replicas will get the QC of B_{k+1}, commit B_k, and subscribe to QC from at least one *anchor* neighbor. Therefore, the Corollary is proven. □

Corollary 4: When consensus is stable, and the cw-queue *of all correct nodes are non-empty. Suppose all correct nodes have accepted the same block B at T; these nodes accept at least k new blocks with heights higher than B at $T + 2k\Delta$.*

Proof: Suppose that at $T = 0$, all correct nodes have accepted block B_k; they will propose the block B_{k+1} in *cw-queue* immediately. At $T = \Delta$, all $anchor$ nodes will receive q *propose* message for B_{k+1} and accept it, then propose B_{k+2} and send the certificate for B_{k+1} to all nodes in To. At $T = 2\Delta$, all correct nodes will accept B_{k+1} by collecting $f+1$ *propose* messages for B_{k+2}, or receiving a certificate for B_{k+1}.

Suppose that at $T = 2m\Delta$, where $m > 1$, all correct nodes have accepted block B_{k+m}, and proposed the block B_{k+m+1}. At $T = (2m+1)\Delta$, all *anchor* nodes will receive $2f+1$ *propose* message for B_{k+m+1} and accept it, then propose B_{k+m+2} and send the certificate for B_{k+m+1} to all nodes in To. At $T = 2(m+1)\Delta$, all correct nodes will accept B_{k+m+1} by collecting $f+1$ *propose* messages for B_{k+m+2}, or receiving a certificate for B_{k+m+1}. Therefore, the Corollary is proven. □

Appendix B
Theoretical Proof for Security of SBFT

We prove that SBFT can achieve the properties of safety (i.e., Theorem 4), and liveness (i.e., Theorem 5).

Proof of safety includes two parts: one is the safety in the steady-state protocol under the same view, i.e., Lemma 1; The other is that SBFT can ensure safety after the view changes, i.e., Lemma 2.

Lemma 1: Correct nodes cannot commit two different blocks in the same view v.

Proof: Suppose two correct replicas r and r' exist and commit two conflict blocks B_k and B'_k in view v. r and r' must have accepted blocks B_{k+1} and B'_{k+1}, respectively. Therefore, at least $2q$ *propose* messages are generated for B_{k+1} and B'_{k+1}. In the steady-state protocol of SBFT, only

Byzantine replicas may propose two conflict blocks, and N nodes output at most $N+f$ *propose* messages for the two blocks. According to the assumptions $N=3f+1$ and $q=2f+1$, we can get $N+f<2q$, a contradiction occurs. Therefore, correct nodes cannot commit two conflict blocks in the same view v. □

Lemma 2: If a correct replica commits B_k, for any $B_{k'}$ ($k'>k$) that accepted by correct replicas, $B_{k'}$ must be after B_k.

Proof: Suppose a correct replica r commits B_k in view v. Based on Lemma 1, when $k'>k$, the $B_{k'}$ accepted by any correct replica in view v is after B_k. Therefore, we assume that B_k is the highest block that r commit in view v, and B_{k+1} is the highest block that r accept in view v, and block $B_{k'}$ is the first block accepted by correct replicas after view v, now we prove that $B_{k'}$ is after B_k.

Since r accept B_{k+1} in view v, at least $2f+1$ replicas have proposed B_{k+1}. When view v ends, among these $2f+1$ nodes, there are at least $f+1$ correct replicas whose local B_{locked} is set to B_k or B_{k+1}. Therefore, during the view change from v to $v+1$, the newest block in any $2f+1$ *status* messages must be B_k or B_{k+1}, which will be decided in view $v+1$. Since no new blocks are accepted from the end of v to entering view v', there are still at least $f+1$ correct replicas with local B_{locked} set to B_k or B_{k+1} at the end of view $v'-1$. Therefore, the global safety block decided in view v' must be either B_k or B_{k+1}. Since the first new block $B_{k'}$ accepted in view v' must extend the decided block in *new-view* message, i.e., B_k or B_{k+1}, $B_{k'}$ must be after B_k. Thus, Lemma 2 is proved. □

Theorem 4 (Safety): Correct nodes cannot commit two different blocks at the same height.

Proof: Suppose there exist two correct replicas r and r' who commit two different blocks B_k and B'_k respectively at the same height, according to the Lemma 1, these two blocks must be committed in different views. Suppose B_k is committed in view v and $B_{k'}$ is committed in view v' and $v'>v$. Then, there must exist a block $B_{k'+1}$ that is accepted in view v', and by Lemma 1, $B_{k'+1}$ must be after B_k, contradiction appears. Therefore, Theorem 4 is proved. □

For liveness, we first prove that byzantine nodes cannot lead to the change of a correct leader in a *anchor*-based topology.

Lemma 3 (Leader safety): In the anchor*-based topology, byzantine replicas will not lead to the change of a correct leader despite the network topology change.*

Proof: According to the view-change protocol in Fig 8, all correct replicas set *timer* to 2Δ when receiving a *new-view* message in which a new block exists. Since the leader is correct, it pre-proposes at least one new block in every interval of Δ at the beginning of the new view, and *anchor* replicas can help relay them (despite the network topology change, there is at least one *anchor* exists), thus control the *cw-queue* to be non-empty. Therefore, correct replicas will not send suspicion messages, though they fail to achieve consensus due to the network topology change. Byzantine nodes can generate no more than f *suspect* messages, and $f<q=2f+1$; thus, byzantine nodes will not lead to the change of a correct leader. □

Then, we prove another two lemma as follows.

Lemma 4: In the anchor*-based topology, there exists a limited value T_{d1}, when the network topology changes at time T and the current leader is faulty, SBFT will change the faulty leader before $T+T_{d1}$.*

Proof: In the *anchor*-based topology, when the leader is faulty and the network topology changes at T. Then, at $T+10\Delta$, without receiving a *new-view* message, all correct replicas must have broadcast a *suspect* message. At $T+11\Delta$, an *anchor* will collect and broadcast $2f+1$ *suspect* messages. At $T+12\Delta$, all correct nodes will change the faulty leader and enter a new view. Therefore, we can get $T_{d1}=12\Delta$ □

Lemma 5: In the anchor*-based topology, there exists a limited value T_{d2}, when the network topology changes at time T and the current leader is correct, SBFT will commit at least one new block at $GST+T_{d2}$.*

Proof: According to the leader safety, i.e., in the *anchor*-based topology, byzantine replicas will not lead to the change of a correct leader despite the network topology change. Then, based on the corollaries 4, after 4Δ, at least 2 consecutive blocks are accepted, and thus, at least one new block is committed. Therefore, we can get $T_{d2}=4\Delta$. □

Theorem 5 (Liveness): There exist two limited values T_s and T_d, all correct replicas will commit at least one new command during a (T_s,T_d)-anchor-based interval.

Proof: According to the above two lemmas, we can get $T_d=max\{T_{d1},T_{d2}\}$, and $f*Td$ is enough to change f consecutive faulty leaders. Based on the lemma of leader safety, another T_{d2} is enough for all correct replicas to commit a new block. Therefore, $T_d=max\{T_{d1},T_{d2}\}$, and $T_s=f*Td+T_{d2}$ □

REFERENCES

[1] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and M. Yin, "Sync HotStuff: Simple and practical synchronous state machine replication," in *Proc. IEEE Symp. Secur. Privacy*, May 2020, pp. 106–118.

[2] E. Androulaki et al., "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proc. 13th EuroSys Conf.*, 2018, pp. 1–15.

[3] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking Bitcoin: Routing attacks on cryptocurrencies," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 375–392.

[4] A. Bessani, J. Sousa, and E. E. Alchieri, "State machine replication for the masses with BFT-SMART," in *Proc. 44th Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw.*, Jun. 2014, pp. 355–362.

[5] A. B. Carol, J. Beznazwy, and A. Houmansadr, "How China detects and blocks shadowsocks," in *Proc. ACM Internet Meas. Conf.*, Oct. 2020, pp. 111–124.

[6] M. Biely, U. Schmid, and B. Weiss, "Synchronous consensus under hybrid process and link failures," *Theor. Comput. Sci.*, vol. 412, no. 40, pp. 5602–5630, Sep. 2011.

[7] S. Bonomi, J. Decouchant, G. Farina, V. Rahli, and S. Tixeuil, "Practical Byzantine reliable broadcast on partially connected networks," in *Proc. IEEE 41st Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2021, pp. 506–516.

[8] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proc. 3rd USENIX Symp. Operating Syst. Design Implement.*, 1999, pp. 173–186.

[9] A. Clement, E. L. Wong, L. Alvisi, M. Dahlin, and M. Marchetti, "Making Byzantine fault tolerant systems tolerate Byzantine faults," in *Proc. NSDI*, vol. 9, 2009, pp. 153–168.

[10] T. Crain, V. Gramoli, M. Larrea, and M. Raynal, "DBFT: Efficient leaderless Byzantine consensus and its application to blockchains," in *Proc. IEEE 17th Int. Symp. Netw. Comput. Appl. (NCA)*, Nov. 2018, pp. 1–8.

[11] T. Crain, C. Natoli, and V. Gramoli, "Red belly: A secure, fair and scalable open blockchain," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2021, pp. 466–483.

[12] S. Duan, M. K. Reiter, and H. Zhang, "BEAT: Asynchronous BFT made practical," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 2028–2041.

[13] X. Feng et al., "PMTUD is not panacea: Revisiting IP fragmentation attacks against TCP," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2022, pp. 1–18.

[14] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, pp. 374–382, Apr. 1985.

[15] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling Byzantine agreements for cryptocurrencies," in *Proc. 26th Symp. Oper. Syst. Princ.*, 2017, pp. 51–68.

[16] G. Golan Gueta et al., "SBFT: A scalable and decentralized trust infrastructure," in *Proc. 49th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2019, pp. 568–580.

[17] B. Guo, Y. Lu, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Speeding dumbo: Pushing asynchronous BFT closer to practice," in *Proc. 29th Netw. Distrib. Syst. Secur. Symp.* San Diego, CA, USA: The Internet Society, Apr. 2022, pp. 1–18.

[18] B. Guo, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Dumbo: Faster asynchronous BFT protocols," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 803–818.

[19] Y. Guo, R. Pass, and E. Shi, "Synchronous, with a chance of partition tolerance," in *Advances in Cryptology—CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part I 39*. Springer, 2019, pp. 499–529.

[20] S. Gupta, J. Hellings, S. Rahnama, and M. Sadoghi, "Proof-of-execution: Reaching consensus through fault-tolerant speculation," 2019, *arXiv:1911.00838*.

[21] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on Bitcoin's peer-to-peer network," in *Proc. 24th USENIX Secur. Symp.*, 2015, pp. 129–144.

[22] Y. Hu, S. Kumar, and R. A. Popa, "Ghostor: Toward a secure data-sharing system from decentralized trust," in *Proc. 17th USENIX Symp. Networked Syst. Design Implement.*, 2020, pp. 851–877. [Online]. Available: https://www.usenix.org/legacy/event/osdi02/tech/waldspurger/waldspurger.pdf

[23] M. M. Jalalzai, J. Niu, C. Feng, and F. Gai, "Fast-HotStuff: A fast and resilient HotStuff protocol," 2020, *arXiv:2010.11454*.

[24] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing Bitcoin security and performance with strong consistency via collective signing," in *Proc. 25th USENIX Secur. Symp.*, 2016, pp. 279–296.

[25] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "OmniLedger: A secure, scale-out, decentralized ledger via sharding," in *Proc. IEEE Symp. Security Privacy (SP)*, May 2018, pp. 583–598.

[26] M. Y. Kubilay, M. S. Kiraz, and H. A. Mantar, "CertLedger: A new PKI model with certificate transparency based on blockchain," *Comput. Security*, vol. 85, pp. 333–352, Aug. 2019.

[27] L. Lamport, "Generalized consensus and Paxos," Microsoft Res., Mountain View, CA, USA, Tech. Rep. MSR-TR-2005-33, 2005.

[28] W. Li, C. Feng, L. Zhang, H. Xu, B. Cao, and M. A. Imran, "A scalable multi-layer PBFT consensus for blockchain," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 5, pp. 1146–1160, May 2021.

[29] S. Liu, P. Viotti, C. Cachin, V. Quéma, and M. Vukolić, "XFT: Practical fault tolerance beyond crashes," in *Proc. 12th USENIX Symp. Operating Syst. Design Implement.*, 2016, pp. 485–500.

[30] Z. Liu et al., "Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric DDoS attacks with programmable switches," in *Proc. 30th USENIX Secur. Symp.*, 2021, pp. 3829–3846.

[31] Z. Liu, H. Jin, Y.-C. Hu, and M. Bailey, "MiddlePolice: Toward enforcing destination-defined policies in the middle of the internet," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 1268–1279.

[32] D. Malkhi, K. Nayak, and L. Ren, "Flexible Byzantine fault tolerance," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 1041–1053.

[33] S. Matsumoto and R. M. Reischuk, "IKP: Turning a PKI around with decentralized automated incentives," in *Proc. IEEE Symp. Security Privacy (SP)*, May 2017, pp. 410–426.

[34] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of BFT protocols," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 31–42.

[35] G. Naumenko, G. Maxwell, P. Wuille, A. Fedorova, and I. Beschastnikh, "Erlay: Efficient transaction relay for Bitcoin," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 817–831.

[36] R. Neiheiser, M. Matos, and L. Rodrigues, "Kauri: Scalable BFT consensus with pipelined tree-based dissemination and aggregation," in *Proc. ACM SIGOPS 28th Symp. Operating Syst. Princ.*, Oct. 2021, pp. 35–48.

[37] Nexuguard. (2021). *DDoS Threat Report 2020 Q3*. [Online]. Available: https://blog.nexusguard.com/threat-report/ddos-threat-report-2020-q3

[38] A. P. Ozisik, G. Andresen, B. N. Levine, D. Tapp, G. Bissias, and S. Katkuri, "Graphene: Efficient interactive set reconciliation applied to blockchain propagation," in *Proc. ACM Special Interest Group Data Commun.*, Aug. 2019, pp. 303–317.

[39] R. Pass and E. Shi, "Thunderella: Blockchains with optimistic instant confirmation," in *Advances in Cryptology—EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29–May 3, 2018 Proceedings*. Springer, 2018, pp. 3–33.

[40] Y. Peng, M. Du, F. Li, R. Cheng, and D. Song, "FalconDB: Blockchain-based collaborative database," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, May 2020, pp. 637–652.

[41] U. Schmid, B. Weiss, and I. Keidar, "Impossibility results and lower bounds for consensus under link failures," *SIAM J. Comput.*, vol. 38, no. 5, pp. 1912–1951, Jan. 2009.

[42] A. Singh, T.-W. Ngan, P. Druschel, and D. S. Wallach, "Eclipse attacks on overlay networks: Threats and defenses," in *Proc. IEEE INFOCOM 25th IEEE Int. Conf. Comput. Commun.*, Apr. 2006, pp. 1–12.

[43] X. Sui, S. Duan, and H. Zhang, "Marlin: Two-phase BFT with linearity," in *Proc. 52nd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Baltimore, MD, USA, Jun. 2022, pp. 54–66.

[44] M. Tran, I. Choi, G. J. Moon, A. V. Vu, and M. S. Kang, "A stealthier partitioning attack against Bitcoin peer-to-peer network," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 894–909.

[45] P.-A. Vervier, O. Thonnard, and M. Dacier, "Mind your blocks: On the stealthiness of malicious BGP hijacks," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2015, pp. 1–15.

[46] Y. Wang and R. Wattenhofer, "Asynchronous Byzantine agreement in incomplete networks," in *Proc. 2nd ACM Conf. Adv. Financial Technol.*, vol. 99, Oct. 2020, pp. 178–188.

[47] Q. Xing, B. Wang, and X. Wang, "POSTER: BGPCoin: A trustworthy blockchain-based resource management solution for BGP security," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 2591–2593.

[48] C. Xu, C. Zhang, and J. Xu, "VChain: Enabling verifiable Boolean range queries over blockchain databases," in *Proc. Int. Conf. Manag. Data (SIGMOD)*, 2019, pp. 141–158.

[49] L. Yang, S. J. Park, M. Alizadeh, S. Kannan, and D. Tse, "DispersedLedger: High-throughput Byzantine consensus on variable bandwidth networks," in *Proc. 19th USENIX Symp. Netw. Syst. Design Implement.*, 2022, pp. 493–512.

[50] Z. Yang, Y. Cui, B. Li, Y. Liu, and Y. Xu, "Software-defined wide area network (SD-WAN): Architecture, advances and opportunities," in *Proc. 28th Int. Conf. Comput. Commun. Netw. (ICCCN)*, 2019, pp. 1–9.

[51] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "HotStuff: BFT consensus with linearity and responsiveness," in *Proc. ACM Symp. Princ. Distrib. Comput.*, Jul. 2019, pp. 347–356.

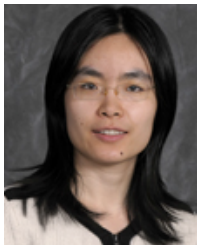
[52] Y. Zhang, S. T. V. Setty, Q. Chen, L. Zhou, and L. Alvisi, "Byzantine ordered consensus without Byzantine oligarchy," in *Proc. USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2020, pp. 633–649.

Sitong Ling received the B.E. degree from the Huazhong University of Science and Technology, Wuhan, China, in 2018, and the M.S. degree from Tsinghua SIGS, Shenzhen, China, in 2021. He is currently pursuing the Ph.D. degree with Tsinghua University. His research area includes Byzantine fault tolerance and blockchain.

Zhuotao Liu received the Ph.D. degree from the University of Illinois at Urbana-Champaign, USA. He is currently an Associate Professor with Tsinghua University. Before joining Tsinghua, he was a Technical Lead with Google, managing one of world's largest software-defined datacenter networks. His research interests include data/AI security and privacy, blockchain and applied cryptography, and secure internet architecture.

Qi Li (Senior Member, IEEE) received the Ph.D. degree from Tsinghua University. He is currently an Associate Professor with the Institute for Network Sciences and Cyberspace, Tsinghua University. His research interests include internet and cloud security, mobile security, and machine learning security. He is an Editorial Board Member of IEEE TRANSACTIONS ON DEPENDABLE AND SECURITY COMPUTING, ACM TOPS, and ACM DTRAP.

Xinle Du received the B.E. degree from the Department of Computer Science and Technology, Xidian University, Xi'an, China, in 2018, and the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2023. He has been the Chief Engineer of the Computer Network and Protocol Laboratory, Huawei, since 2023. His research interests include networking and LLM systems.



Jing Chen received the bachelor's and master's degrees in computer science from Tsinghua University and the Ph.D. degree in computer science from MIT. She is a Professor with the Computer Science Department, Tsinghua University. She was a Chief Scientist and the Head of theory research with Algorand Inc.; and a Faculty Member of the Computer Science Department, Stony Brook University. She was a Post-Doctoral Researcher with the Institute for Advanced Study, Princeton. Her main research interest is theory of computing, in particular algorithmic game theory, mechanism design, and problems related to blockchains. She received the NSF CAREER Award for her work on mechanism design.

Ke Xu (Fellow, IEEE) received the Ph.D. degree from Tsinghua University, Beijing, China. He is currently a Full Professor with the Department of Computer Science and Technology, Tsinghua University. He has published more than 200 technical articles in the research areas of next-generation internet, blockchain systems, and network security. He has won the IWQoS 2024 Best Paper Award and the Distinguished Paper Award at USENIX Security 2023 and 2024.