A Framework for Evaluating BFT

James R. Clavin[1,*], Yue Huang[2,*] ,
Xin Wang[1], Pradeep M. Prakash[1], Sisi Duan[3,⋆], Jianwu Wang[1], Sean Peisert[4]
[1]University of Maryland, Baltimore County. {jclavinm,xinwang11,pmargas1,jianwu}@umbc.edu
[2]Shandong Institute of Blockchain. huangyue@sdibc.cn [3]Tsinghua University. duansisi@mail.tsinghua.edu.cn
[4]Lawrence Berkeley National Laboratory. sppeisert@lbl.gov

***Abstract*—We present a framework for evaluating the performance of Byzantine fault-tolerant (BFT) protocols theoretically. Our motivation is to identify protocols suitable for a particular power grid application. In this application, replicas are located in a LAN network where latency is the priority. To fully understand the performance of BFT, we provide a generic approach that quantifies the performance of BFT protocols based on the number of cryptographic operations under five different scenarios (in the presence of failures and without failures).**

We present the performance of three representative BFT protocols: PBFT, Prime, and SBFT. To validate our framework, we also evaluate the protocols experimentally in the CloudLab testbed. Our experimental results match the findings predicted by the framework. Although a variety of factors may affect the performance of the protocols, our framework can be used as a valuable reference to understand the performance of BFT.

***Index Terms*—BFT, cryptography, power grid**

I. Introduction

Fault-tolerant state machine replication is a technique that masks failures in computing systems. Fault-tolerant protocols typically focus on masking either crash failures or Byzantine failures. These protocols leverage replicas — typically $2f + 1$ or $3f + 1$ — to overcome f failures.

Applications critical enough to require fault-tolerant protocols have varying requirements in terms of latency, throughput, number of replicas required, processing capacity, the types of failures tolerated (crash vs. Byzantine), and so on. As a result, different fault-tolerant protocols are appropriate for different situations. Determining which protocol to leverage can be challenging since although tens or hundreds of fault-tolerant protocols have been developed and published, there are no straightforward or generic means that we are aware of for evaluation and comparison of these protocols to determine their appropriateness for given situations. This paper describes a theoretical approach to accomplish an evaluation of Byzantine fault-tolerant (BFT) protocols given a set of constraints and validates the approach experimentally.

A. Motivating Use Case

Electrical transmission lines that carry a load of 345kV and higher are critical components in the power grid. U.S. power grids have been generally reliable for almost a century. However, aging infrastructure is being modernized, including elements that are connected to computer networks. While technical advancements in the power grids have improved the efficiency, reliability, and capacity of the systems, but it has also introduced challenges, including components like protective relays that can be attacked over computer networks. The dependence of the public on critical infrastructure providing secure and reliable electricity requires new means to ensure secure transmission of electricity through power grids.

Electrical power grids have three main functions at the systems level: generation, transmission, and distribution. Electricity is generated at power stations, transported through high-voltage transmission lines, stepped down in substations, and distributed through low-voltage lines to residence and business customers. At each level, various grid-attached components exist and can all be the target of manipulations resulting in grid instability, theft of intellectual property, and denial of service. These attacks can manifest themselves externally over a network, through internal components in a supply chain attack, or via insider attack.

The transmission grid is the portion of the grid that transports high-voltage electricity from large-scale generation facilities to cities, where voltage is stepped down to where it can be ingested in buildings. Given the vital role the transmission grid serves as the arteries for powering large portions of the grid affecting many people and facilities, it is, therefore, the most important to protect.

In electrical engineering, circuit breakers are leveraged when faults are detected in order to protect the circuit wiring and any devices connected to the circuit. In the power grid, protective relays are used to trip circuit breakers when faults, including over-current, over-voltage, and other potentially damaging conditions might occur. In the transmission grid, these relays protect equipment both up and down stream from where they are installed, including the generation facilities and the distribution systems that the transmission systems supply.

Like many control systems, modern protective relays are now computer-controlled and networked. As such, an attacker could potentially control the relays to cause an incorrect action based on false sensor readings from the electrical lines they affect. In order to protect these vital pieces of equipment so that the correct action is taken based on the current grid state, Byzantine fault tolerance (BFT) is being evaluated.

∗ Co-first author. Yue Huang performed the work while she was at UMBC.
⋆ Sisi Duan performed this work while she was at UMBC and an LBNL Affiliate

B. Needs of BFT in the Grid

In our study on using BFT in transmission grid relays, extremely low latency is the focus, both without failures and under Byzantine failures. In particular, a response that is too slow might fail to protect grid-attached equipment in time to prevent damage. A number of prior efforts have studied the application of BFT in the Supervisory Control and Data Acquisition (SCADA) system [2, 3, 19], a control system architecture used in the power grid, among other places. However, our focus is on the use of BFT for controlling modern protective relays, not SCADA in general. Such a use case favors low latency over high throughput and scalability.

In this paper, we evaluate a variety of BFT protocols and propose a suitable implementation for the power grid relay use case via the protocols' performance parameters, including types of failures tolerated, resilience, latency, message complexity, hardware requirements, and cryptographic modules.

C. Framework for Evaluating BFT

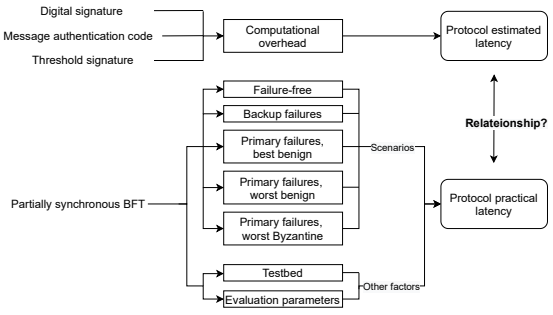


Figure 1: The framework for evaluating BFT protocols.

The paper aims to accomplish "apples-to-apples" evaluations of BFT protocols both theoretically and experimentally, and with a focus on latency. As shown in Fig. 1, we propose a framework to quantify the performance of BFT by analyzing the accurate number of cryptographic operations in different scenarios. Previous studies have shown that the number of cryptographic operations is directly related to performance, especially latency, in local area networks (LAN) [12, 15]. Furthermore, most studies only discuss the authenticator complexity in general, rather than the concrete number of operations. In contrast, we provide a customized framework that considers multiple scenarios. This way, we can determine if protocols meet the requirements of the target application under all conditions. Thus, our framework provides a generic way to evaluate BFT theoretically and comprehensively.

We also evaluate the performance of BFT protocols experimentally to validate the framework's predictions. In particular, we evaluate the latency of PBFT [6], SBFT [13], and Prime [1] in the same CloudLab distributed environment [11]. We compare the performance of our experimental results to the results of our framework. The evaluation shows that our framework provides a reasonable estimation of protocol performance.

II. Motivation

A. Overview of BFT

The goal of BFT is for the replicas to operate correctly to reach a consensus on the order of client requests. BFT can be categorized into synchronous BFT (where there exists a known upper bound on message transmission and processing delay), partially synchronous BFT (where there exists an unknown upper bound), and asynchronous BFT (where there does not exist such a bound). Starting with PBFT [6], a a partially synchronous BFT, an impressive number of efficient BFT protocols have been proposed (e.g., [9, 12, 17]). Numerous efforts have been made to improve the performance of partially synchronous BFT [7,8,14,18]. Meanwhile, there are efficient protocols for completely asynchronous environments [10,16].

Our work focuses on partially synchronous BFT, although our approach is generic to all categories. We chose partially synchronous BFT for two reasons. First, partially synchronous BFT is more practical than synchronous BFT. Compared to asynchronous BFT, partially synchronous BFT achieves lower latency which fits the needs of our case study. Second, by modeling different components (normal operation, view changes) of protocols in the same category, we obtain a fair and comprehensive comparison. We consider protocols that require $n \geq 3f+1$ replicas, where f the number of Byzantine replicas and n is the total number of replicas in the system. Without loss of generality, we consider $n = 3f + 1$. In BFT, a replica *delivers client requests*, each submitted by a client.

B. Needs and Challenges

BFT protocols have been studied extensively. To analyze system's performance, most prior studies of BFT have focused on criteria such as message complexity, communication complexity, etc. Some studies present the number of cryptographic operations of the bottleneck server to understand the performance better. However, a wide variety range of different approaches have been used in these prior studies since many factors can affect the performance, i.e., programming languages used, different system parameters, communication libraries used.

Furthermore, the design of most protocols focuses on the performance during the normal operation, i.e., when there are no failures or only backup failures. The protocols are evaluated in a failure-free case partially because it is extremely challenging, if not impossible, to *simulate* Byzantine behaviors. While it is reasonable to consider that failures are rare, there is still a lack of *full picture* of how a protocol performs, especially under failures and attacks. It leaves a gap for understanding how useful a BFT approach will be since we cannot know how it will perform under attack.

In our work, we seek to provide a more comprehensive framework that provides such a full picture of how a protocol performs. For instance, some protocols may have the same message complexity but significantly different numbers of concrete cryptographic operations. Also, some protocols may sacrifice performance under failures to enjoy the improved

performance in failure-free cases. While such an observation is *natural* since there are always trade-offs in the protocol design, our framework *quantifies* the trade-offs. Thus, our framework provides a detailed and "apples-to-apples" comparison of the BFT protocols in terms of the performance of each step and performance under primary/backup failures.

III. The Framework for Evaluating BFT Protocols

This section describes our framework for evaluating BFT protocols. We first discuss five scenarios in the framework. Then we discuss TheoLat, the key number in our framework to evaluate the protocols theoretically.

A. Evaluation Scenarios

Partially synchronous protocols are usually leader-based and usually include two major components: *normal operation* and *view change*. During normal operation, leader *proposes* the sequence of client requests and replicas agree on the order of requests. When the leader is suspected to be faulty, a new one needs to be elected via *view change*.

We consider five scenarios of a partially synchronous BFT protocol to determine performance, including performance without failures and under failures.

- **Failure-free**: The normal operation when there are no failures.
- **Backup failures**: The normal operation when there are only backup failures.
- **Primary failures**: When the primary fails, we consider three different sub-scenarios:
  - **Best benign scenario**: The primary crashes and the following designated primary is correct, i.e., replicas resume to normal operation upon one view-change.
  - **Worst benign scenario**: f primaries crash, i.e., replicas run f rounds of view-changes before resuming to normal operation.
  - **Worst Byzantine scenario**: f primaries are Byzantine, i.e., replicas resume to normal operation upon f rounds of view-changes.

B. TheoLat

TheoLat is an approach to evaluate the performance of the five scenarios given a BFT protocol. In particular, we calculate TheoLat that can be used to quantify the performance of a protocol. TheoLat consists of two components: $\mathsf{TheoLat_{NO}}$ and $\mathsf{TheoLat_{VC}}$. $\mathsf{TheoLat_{NO}}$ evaluates the normal operation, and $\mathsf{TheoLat_{VC}}$ evaluates the view-changes. We use several system parameters such as the type of cryptographic operation, batch size, size of the request. Based on the system parameters, we calculate $\mathsf{TheoLat_{NO}}$ and $\mathsf{TheoLat_{VC}}$ of the bottleneck replicas. The notations are shown in Table I.

As shown in Fig. 2, TheoLat consists of $\mathsf{TheoLat_{NO}}$ and $\mathsf{TheoLat_{VC}}$. For the normal operation, we count the number of cryptographic operations per step per phase using the number of nodes and batch size as the input. Then we calculate the result via the algorithm as $\mathsf{TheoLat_{NO}}$ similarly according to different scenarios.

Table I: Notations

Notation	Meaning
Gen	Generation of a cryptographic authenticator
Vrf	Verification of a cryptographic authenticator
S_σ	Unit latency of σ operation
σ_d	An operation of digital signature
$\sigma_{t,s}$	A partial threshold signature signing operation
$\sigma_{t,c}$	A partial threshold signature combining operation
$\sigma_{t,v}$	A verification operation of threshold signature
N	The number of Gen or Vrf operations
b	Batch size
ρ	Number of requests carried in the view change messages

Algorithm 1: $\mathsf{TheoLat_{NO}}$
- Based on unit latency (S_σ) and the number of phases i, calculate the number of N_{Gen} and N_{Vrf} respectively.
- The theoretical latency $\mathsf{TheoLat_{NO}}$ of one BFT in NO can be computed as $\mathsf{TheoLat_{NO}} = \sum_{j=0}^{i}(S_{\sigma_{\mathsf{Gen}}} \times N_{\mathsf{Gen}}) + \sum_{j=0}^{i}(S_{\sigma_{\mathsf{Vrf}}} \times N_{\mathsf{Vrf}})$, where i is the total number of steps.

Algorithm 2: $\mathsf{TheoLat_{VC}}$
- Based on the unit latency (S_σ), the number of phases i, number of requests in each view change ρ, and f, calculate the number of N_{Gen} and N_{Vrf} respectively.
- The theoretical latency $\mathsf{TheoLat_{VC}}$ of one BFT in VC can be computed as follows, where i is the number of steps.

– In the best benign scenario
$\mathsf{TheoLat_{VC}} = \sum_{j=0}^{i}(S_{\sigma_{\mathsf{Gen}}} \times N_{\mathsf{Gen}}) + \sum_{j=0}^{i}(S_{\sigma_{\mathsf{Vrf}}} \times N_{\mathsf{Vrf}})$
– In primary failures - worst benign scenario
$\mathsf{TheoLat_{VC}} = f\sum_{j=0}^{i}(S_{\sigma_{\mathsf{Gen}}} \times N_{\mathsf{Gen}}) + f\sum_{j=0}^{i}(S_{\sigma_{\mathsf{Vrf}}} \times N_{\mathsf{Vrf}}) + \rho$
– In the primary failures - worst Byzantine scenario
$\mathsf{TheoLat_{VC}} = f\sum_{j=0}^{i}(S_{\sigma_{\mathsf{Gen}}} \times N_{\mathsf{Gen}}) + f\sum_{j=0}^{i}(S_{\sigma_{\mathsf{Vrf}}} \times N_{\mathsf{Vrf}}) + f * \rho$

Figure 2: The algorithm to calculate TheoLat.

Note that a BFT protocol usually has a checkpoint sub-protocol which is used for garbage collection. Checkpoint protocol may affect the performance of the system as discussed in prior works [5]. The reason why we do not include the evaluation of the checkpoint sub-protocol here is two-fold. First, the checkpoint sub-protocol is relatively independent of the normal operation and view-changes, i.e., choosing a different checkpoint sub-protocol typically does not affect the correctness of the BFT protocol. Second, most BFT protocols use the same checkpoint sub-protocol. Although the frequency of garbage collection affects the performance of the BFT protocols, the impact of any BFT protocols is similar.

IV. Evaluating BFT Under the Framework

While our approach is generic that can be applied to any BFT protocol, in this paper, we focus on three protocols: PBFT, Prime, and SBFT. We chose these three protocols for two reasons. First, we consider the three protocols *representative* of our purpose. PBFT is a classic BFT protocol, and a large number of partially synchronous BFT studies have compared the performance of their protocols with PBFT [4, 7,8,13,20]. Prime improves the performance of PBFT under attack, and it has been extensively applied to study in SCADA environments [2, 3], which makes it a perfect choice for our study. SBFT simplifies the workflow of PBFT, so it is worth evaluating the performance trade-offs between different designs. The second reason for selecting these three is that all are widely studied, so there are mature, open-source

implementations. It is important because, in our work, we aim to *validate* our framework via experimental evaluation. Note that Prime and SBFT both consider benign failures in addition to crash failures, e.g., slow replicas. To conduct a fair comparison between the protocols, we consider that all replicas crash when we consider benign failures. Furthermore, proactive recovery is another option for BFT protocols [4, 6]. Both PBFT and Prime support proactive recovery and the approach can be integrated with other BFT protocols as well. For simplicity, we do not consider it during our evaluation.

A. PBFT

Workflow. PBFT has three phases: PRE-PREPARE, PREPARE, and COMMIT. In PRE-PREPARE phase, the primary broadcasts a *pre-prepare* message to all replicas. In PREPARE phase, upon receiving a valid *pre-prepare* message, each replica broadcasts a *prepare* message to all replicas. In COMMIT phase, upon receiving $2f$ *prepare* messages (or $2f + 1$ messages if we consider the *pre-prepare* message from the primary is also a *prepare* message), each replica broadcasts a *commit* message to all replicas and receives $2f + 1$ *commit* messages. Then a replica delivers the client request and replies to the client.

PBFT's view-change has two phases, VIEW-CHANGE and NEW-VIEW. If a replica starts the view change process, it increments its view number and sends a *view-change* message to all replicas. The *view-change* message consists of the last stable checkpoint, the requests that have been prepared since the last stable checkpoint, the largest sequence number, and the view number. Upon receiving $2f$ *view-change* messages, the designated primary, in NEW-VIEW phase, sends a *new-view* message to all replicas that consists of the last stable checkpoint, a sequence of requests that have been prepared or committed in the previous view, the view number, and $2f$ *view-change* messages. Replicas then move to the new view if the *new-view* message is well-formatted.

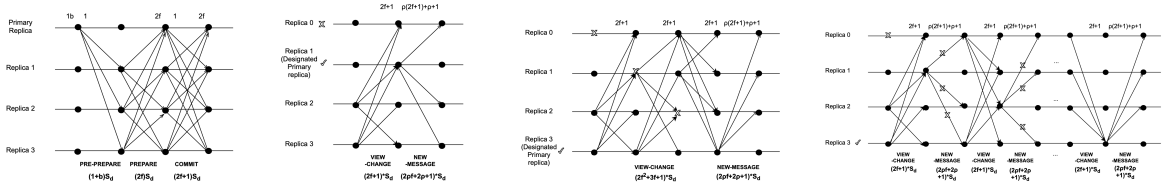
TheoLat *in the five scenarios.* The bottleneck server of PBFT is the primary. Latency of the primary is calculated as follows.

- *Failure-free (Fig. 3a):* $\mathsf{TheoLat}_{\mathsf{NO}}$ is $(4f + 2 + b) \times S_d$ for each round. In PRE-PREPARE phase, the primary executes b $N_{\mathsf{Vrf}} = \sigma_d$ operations for the client requests, and executes one $N_{\mathsf{Gen}} = \sigma_d$ operation when generating the *prepare* message. In PREPARE phase, the primary verifies $2f$ *pre-prepare* message(s) and generates one authenticator for the *commit* message. In COMMIT phase, primary replica executes one $N_{\mathsf{Gen}} = \sigma_d$, and $2f$ $N_{\mathsf{Vrf}} = \sigma_d$ operations.
- *Backup failures (Fig. 3a):* The normal operation of PBFT also deals with the backup failure scenario. Therefore, $\mathsf{TheoLat}_{\mathsf{NO}}$ of backup failure is as same as that in the failure-free scenario.
- *Primary failures best benign scenario (Fig. 3b):* $\mathsf{TheoLat}_{\mathsf{VC}} = (2\times\rho\times f+2f+2\rho+2)\times S_d$. The new primary verifies $2f+1$ $N_{\mathsf{Vrf}} = \sigma_d$ *view-change* message(s). In addition, it includes ρ *pre-prepare* messages, where ρ is the number of requests that have been processed since last stable checkpoint. For each request, in NEW-VIEW phase, the primary also verifies whether there exists a prepare certificate (with $(2f + 1) \times \rho$ $N_{\mathsf{Vrf}} = \sigma_d$ digital signatures). In other words, in NEW-VIEW phase, $1+\rho$ $N_{\mathsf{Gen}} = \sigma_d$ operations and $(2f+1) \times \rho$ $N_{\mathsf{Vrf}} = \sigma_d$ operations are executed.
- *Primary failures worst benign scenario (Fig. 3c):* $\mathsf{TheoLat}_{\mathsf{VC}} = (2f^2 + 3f + 2\rho f + 2\rho + 2) \times S_d$. In the worst case, f VIEW-CHANGE phases are triggered. In each VIEW-CHANGE, every replica verifies $2f + 1$ $N_{\mathsf{Vrf}} = \sigma_d$ *view-change* message(s). After f VIEW-CHANGES, the correct primary executes $(2f + 1) \times \rho$ $N_{\mathsf{Vrf}} = \sigma_d$ and $1 + \rho$ $N_{\mathsf{Gen}} = \sigma_d$ operations.
- *Primary failures worst Byzantine scenario (Fig. 3d):* $\mathsf{TheoLat}_{\mathsf{VC}} = (2f^2+2\rho\times f^2+4f+4\rho\times f+2\rho+2)\times S_d$. In the worst case, f VIEW-CHANGE phases are triggered where each primary generates an incorrect *new-view* message. In each round, every replica verifies $2f + 1$ *view-change* message(s), i.e., $2f+1$ $N_{\mathsf{Vrf}} = \sigma_d$ operations are executed. In each NEW-VIEW, the faulty replica first executes $(2f+1)\times\rho$ $N_{\mathsf{Vrf}} = \sigma_d$ to verify *pre-prepare* messages in *view-change* messages, and then executes $1 + \rho$ $N_{\mathsf{Gen}} = \sigma_d$ when broadcasting a faulty *new-view* message. The number of operations for all f view changes is therefore $2f+2\rho\times f+2\rho+2$. In the worst case, after f view changes, the primary replica is correct. The primary executes $2f + 1$ $N_{\mathsf{Vrf}} = \sigma_d$ operations for the *view-change* messages and executes $(2f + 1) \times \rho$ $N_{\mathsf{Vrf}} = \sigma_d$ and $1 + \rho$ $N_{\mathsf{Gen}} = \sigma_d$ operations.

B. Prime

Workflow. Prime is a leader-based BFT. It has two sub-protocols, PREORDERING and GLOBAL ORDERING. The PREORDERING sub-protocol has three phases, PO-REQUEST, PO-ACT, and PO-SUMMARY. In each round, one replica broadcasts a *po-request* message consisting of a batch of client requests and a pair of server number and sequence number. Upon receiving a *po-request* message, each replica broadcasts a *po-ack* message. After receiving $2f$ matching *po-ack* messages, the correct replica creates a preorder certificate ($2f+1$ signatures). Each correct replica then broadcasts *po-summary* messages. After that, replicas move to the global ordering phase, which is analogous to PBFT normal operation.

The view change protocol has three phases: STATE DISSEMINATION, PROOF GENERATION, and REPLAY. It covers both the best and worst scenarios of primary failure. In STATE DISSEMINATION, a replica broadcasts a *report* message and a *pc-set* message consisting of the prepare certificates. Upon collecting $2f + 1$ complete states, replicas broadcast *vc-list* messages. In PROOF GENERATION phase, when a replica receives a *vc-list* message, it broadcasts a *vc-partial-msg* message. Upon collecting $2f+1$ *vc-partial-msg* messages, the replicas sign the message using threshold signature and each replica broadcasts a *vc-proof* message. Finally, in REPLAY phase, the leader receives the *vc-proof* message, and then broadcasts a *reply* message. Replicas then move to NEW-VIEW phase and resume normal operation.



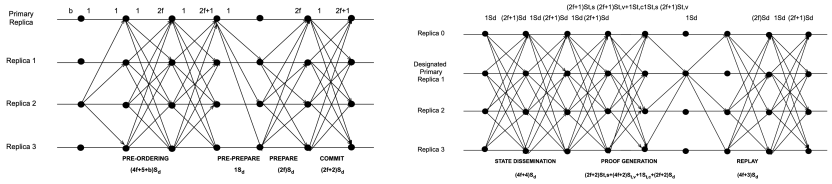
(a) Failure-free and backup failures scenario (f=1).

(b) Primary failures best benign scenario (f=1).

(c) Primary failures worst benign scenario (f=1).

(d) Primary failures worst Byzantine scenario (f=1).

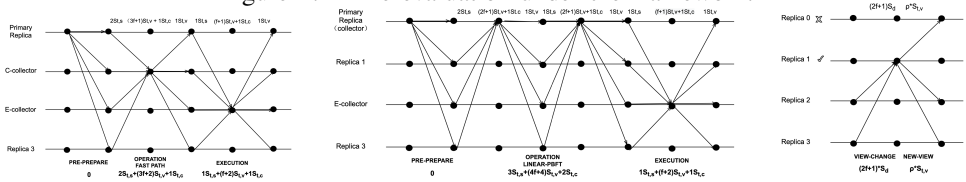
Figure 3: PBFT evaluation under the framework.



(a) Failure-free and backup failures scenario (f=1).

(b) Primary failures in best and worst scenario (f=1).

Figure 4: Prime evaluation under the framework.



(a) Failure-free scenario (f=1).

(b) Backup failures scenario (f=1).

(c) Primary failures best benign scenario (f=1).

Figure 5: SBFT evaluation under the framework.

TheoLat *in the five scenarios.* The bottleneck server of Prime is the primary. Latency of the primary is calculated as follows.

- *Failure-free (Fig. 4a):* $\mathsf{TheoLat}_{\mathsf{NO}} = (8f+8+b) \times S_d$. The failure-free case for Prime is similar to two PBFT's normal operation, as shown in the figure.
- *Backup failures (Fig. 4a):* The scenario is the same as that in failure-free scenario, i.e., $\mathsf{TheoLat}_{\mathsf{NO}} = (8f+8+b) \times S_d$.
- *Primary failure best and worst scenarios (Fig. 4b):* Prime's view change protocol ensures that a *correct* replica is elected after the view change. Therefore, the number of cryptographic operations is the same under primary failures. In the best and worst scenarios, $\mathsf{TheoLat}_{\mathsf{VC}} = (10f + 9) \times S_d + (4f + 2) \times S_{t,v} + 1 \times S_{t,c} + (2f + 2) \times S_{t,s}$. In STATE DISSEMINATION phase, each replica executes one $N_{\mathsf{Gen}} = \sigma_d$ operation for the *report* message and one $N_{\mathsf{Gen}} = \sigma_d$ operation for *pc-set* message. Each replica verifies $2f+1$ σ_d operations for the *report* messages and $2f + 1$ σ_d operations for the *pc-set* messages. In PROOF GENERATION phase, the designated primary executes one $N_{\mathsf{Gen}} = \sigma_d$ and $2f + 1$ $N_{\mathsf{Vrf}} = \sigma_d$ operations for the *vc-list* messages. After that, a replica executes $2f + 1$ $N_{\mathsf{Gen}} = \sigma_{t,s}$, $2f + 1$ $N_{\mathsf{Vrf}} = \sigma_{t,v}$, and one $N_{\mathsf{Vrf}} = \sigma_{t,c}$ operation for *vc-partial-sig* messages. Finally, the primary executes one $N_{\mathsf{Gen}} = \sigma_{t,s}$ and $2f + 1$ $N_{\mathsf{Vrf}} = \sigma_{t,v}$ operations for *vc-proof* messages. In REPLAY phase, the primary executes a $N_{\mathsf{Gen}} = \sigma_d$ operation and a $N_{\mathsf{Vrf}} = \sigma_d$ operation for *replay* message. In REPLAY-PREPARE and REPLAY-COMMIT, the number of cryptographic operations are the same as PREPARE and COMMIT in Prime's normal operation, i.e., $(4f + 3) \times S_d$.

C. SBFT

Workflow. SBFT is designed to enhance the scalability of conventional BFT protocols. SBFT is structured by primary and replicas (including $1 + c$ C-collector and $1 + c$ E-collector) and three threshold signatures schemes, σ with threshold $3f + c + 1$, τ with threshold $2f + c + 1$, π with threshold $f+1$. The normal operation of SBFT has two phases, OPERATION, and EXECUTION. OPERATION phase consists of a FAST PATH protocol in the failure-free scenario and a fall back LINEAR-PBFT protocol under failures. In this work, we consider $c = 0$.

In FAST PATH protocol, upon receiving requests from clients, the primary broadcasts a *pre-prepare* message. Upon accepting *pre-prepare* messages, each replica signs, and sends a *sign-share* message to C-collector. The C-collector verifies σ signature shares, combines them and sends them back to all replicas. In EXECUTION phase, replicas sign and send a *sign-state* message to E-collector. Upon collecting τ messages, The E-collector verifies and combines them into one signature and broadcast *full-commit-proof* messages. Each replica verifies and accepts the message.

LINEAR-PBFT is a fall-back protocol in normal operation. The primary also acts as C-collector and E-collector in this protocol. The protocol has four phases: SIGN-SHARE, PREPARE, COMMIT-PROOF, and PBFT COMMIT-PROOF. In SIGN-SHARE phase, replicas use τ as authenticator for *commit* message, and send to the primary. Upon collecting σ or

τ signatures hares, the primary combines them and broadcasts a *prepare* message. In PREPARE phase, replicas send *commit* messages to the primary. In PBFT COMMIT-PROOF phase, the primary combines $2f + 1$ shares, and broadcasts *full-commit-proof-slow* message. After verifying the message, replicas enter EXECUTION phase.

The view change protocol of SBFT is similar to that in PBFT.

TheoLat *in the five scenarios.* SBFT assumes point-to-point authenticated channels (via TLS). In our framework, we ignore the authenticators of the underlying channels and consider cryptographic operations at the application level. We consider the extreme case where a replica is a primary, E-collector, and also C-collector. Indeed, this might be the default setting in some setups. The latency of the bottleneck server is calculated as follows.

- *Failure-free (Fig. 5a):* $\mathsf{TheoLat_{NO}} = 3 \times S_{t,s} + 2 \times S_{t,c} + (4f+4) \times S_{t,v}$. SBFT executes FAST PATH and EXECUTION in failure-free scenario. In the FAST PATH phase, the primary executes a $N_{\mathsf{Gen}} = \sigma_{t,s}$ operation and the C-collector verifies $3f + 1$ $N_{\mathsf{Vrf}} = \sigma_{t,v}$ shares and combines them (executing a $N_{\mathsf{Vrf}} = \sigma_{t,c}$ operation). In EXECUTION phase, each replica verifies one $N_{\mathsf{Vrf}} = \sigma_{t,v}$ signature and generates a $N_{\mathsf{Gen}} = \sigma_{t,s}$ threshold signature. The E-collector verifies $f + 1$ $N_{\mathsf{Vrf}} = \sigma_{t,v}$ and combines the threshold signatures (one $N_{\mathsf{Vrf}} = \sigma_{t,c}$ operation). Each replica then verifies the message, executing one $N_{\mathsf{Vrf}} = \sigma_{t,v}$ operation.
- *Backup failures (Fig. 5b):* $\mathsf{TheoLat_{NO}} = 4 \times S_{t,s} + 3 \times S_{t,c} + (5f + 6) \times S_{t,v}$. SBFT consists of a LINEAR-PBFT phase and an EXECUTION phase under backup failure(s). The major difference from the failure-free case is that each replica creates two threshold signature shares in each round.
- *Primary failures best benign scenario:* $\mathsf{TheoLat_{VC}} = (2f + 1) \times S_d + \rho \times S_{t,v}$. This scenario is analogous to PBFT's view change. The main difference occurs in NEW-VIEW phase: the designated primary verifies all historical *commit-proof* messages sent in the *view-change* messages via threshold signatures (i.e. ρ $N_{\mathsf{Vrf}} = \sigma_{t,v}$).
- *Primary failures worst benign scenario:* $\mathsf{TheoLat_{VC}} = (2f^2 + 3f + 1) \times Sd + \rho \times S_{t,v}$. According to the illustration above, there are f rounds in worst case and the $f + 1$-th designated primary is correct. $\mathsf{TheoLat_{VC}}$ per round is $(2f + 1) \times S_d$ in terms of benign failures. Therefore, the $\mathsf{TheoLat_{VC}}$ in total is $(2f + 1) \times S_d \times (f + 1) + \rho \times S_{t,v}$.
- *Primary failures worst Byzantine scenario:* $\mathsf{TheoLat_{VC}} = (2f^2 + 3f + 1) \times Sd + (\rho \times f + \rho) \times S_{t,v}$. There are f rounds in worst case. $\mathsf{TheoLat_{VC}}$ per round is $(2f + 1) \times S_d + \rho \times S_{t,v}$ in terms of Byzantine failures. Therefore, the $\mathsf{TheoLat_{VC}}$ in this scenario is $((2f + 1) \times S_d + \rho \times S_{t,v}) \times (f + 1)$.

V. EXPERIMENTAL EVALUATION

A. Overview

We evaluate the BFT protocols experimentally. Our goal is to validate our theoretical results in the framework.

We present the performance of three representative BFT protocols: PBFT, Prime, and SBFT. Their corresponding open-source codebase libraries are Sawtooth-PBFT from Hyperledger[1], Prime from Johns Hopkins University[2], and Concord-BFT from VMware[3]. We consider all five scenarios as discussed in our framework. We focus mainly on the performance of latency as it is the motivation of our work. Our evaluation results show that our theoretical framework is useful in evaluating BFT protocols, especially for the latency breakdown of different phases.

B. Evaluation Setup

We use CloudLab [11] platform for our evaluation as CloudLab provides an environment similar to that in the control systems of the power grid. We use 6 Intel(R) Xeon(R) CPU E5-2640 v4 with 2.40GHz instances running in Utah cluster. By default, it provides 2×10 Gbps network interfaces, and each node has 64-bit ARM processors. What's more, we also considered experiments with different testbeds, which also include a local Dell/EMC docker cluster. For Dell/EMC docker cluster, all Docker deployed on a Dell/EMC local machine with Intel(R) Core™ i7-8700 CPU with 3.20GHz $\times$ 12 processors and 15.5GiB memory.

C. Latency Breakdown in Normal Operation

We first assess the latency of PBFT, Prime, and SBFT in the failure-free scenario, with four replicas. Table II shows the latency per request and its breakdown in each phase. For each experiment, we run it several times and report the average values.

Protocol/Phase	PRE-PREPARE	PREPARE	COMMIT	REPLY	Total
PBFT	8.32	3.21	3.85	2.38	17.76
Protocol/Phase	**PREORDERING**	**PRE-PREPARE**	**PREPARE**	**COMMIT**	**Total**
Prime	13.49	4.78	4.73	5.12	28.12
Protocol/Phase	**PRE-PREPARE**	**OPERATION**	**EXECUTION**		**Total**
SBFT	0.3	1.67	0.74		2.72

Table II: Latency breakdown of BFT protocol under failure-free scenario (unit: ms).

For PBFT, the latency of PRE-PREPARE, PREPARE, COMMIT, and REPLY phases account for 46.85%, 18.07%, 21.68%, and 13.4% of the overall latency, respectively. Our results show that the PRE-PREPARE phase incurs the highest latency. This result matches the fraction of cryptographic operations under our framework (i.e. $2:0:1:1$ by calculating $\mathsf{TheoLat_{NO}}$ letting $f = 0$ and $b = 1$).

For Prime, the latency of PREORDERING, PRE-PREPARE, PREPARE and COMMIT phases accounts for 47.97%, 17%, 16.82%, and 18.21% of the latency, respectively. According to the results of our framework, the fraction of cryptographic operations of four phases is $6:1:0:2$, which matches our experimental evaluation results.

The results for SBFT are similar. The latency of PRE-PREPARE, OPERATION, and EXECUTION accounts for 11.03%, 61.4% and 27.2% of the latency, respectively. Since

[1]Sawtooth-pbft: https://github.com/hyperledger/sawtooth-pbft
[2]Prime: http://www.dsn.jhu.edu/prime/
[3]Concord-bft: https://github.com/vmware/concord-bft

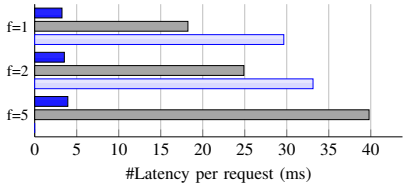


Figure 6: Latency of BFT protocols when f changes (Gray: PBFT, light blue: Prime, dark blue: SBFT).

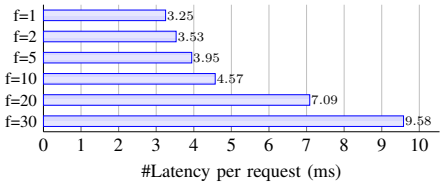


Figure 7: Scale-up latency of SBFT.

the SBFT uses threshold signatures, we evaluate the unit latency $S_{t,s}$, $S_{t,c}$ and $S_{t,v}$ and put them in the results of our framework. According to our framework, the fraction of different phases in SBFT is $0 : 11 : 9$, which approximately matches the experimental results.

D. Latency Breakdown under Backup Failures

We assess the performance under backup failures. We evaluate the latency and report the breakdown for $f = 1$, $f = 2$, and $f = 5$. For all experiments, we simply stop f backup replicas. The results are summarized in Figure 6.

For all three BFT protocols, the overall latency increases as f grows. Besides, Prime has the highest latency among the three protocols. The results match those in our framework. Compared to the other two protocols, the latency of SBFT does not increase significantly as f grows. Since the latency of threshold signatures is related to f (instead, the latency of digital signature is static), we add a scale-up evaluation for SBFT, where we evaluate latency using up to $f = 30$, as shown in Figure 7. The evaluation shows that when f is large, the latency of SBFT increases significantly.

We show the fraction of each phase for all three protocols ($f = 1$) in Figure 8a and compare the theoretical results with the experimental results. The results show that the fractions of each phase roughly match that of the experimental results. Furthermore, we also evaluate the latency for S_d, $S_{t,s}$, $S_{t,c}$, and $S_{t,v}$, put them in our framework, and *evaluate* the latency based on TheoLat. As shown in Figure 8b, the latency obtained from our framework is consistently lower than that in the experimental results. It is expected since the latency of a protocol also involves significantly more other operations and network communication. The overall trend of the latency, however, matches that of the experimental results.

Besides the above evaluation of latency under backup failures, we also report the latency breakdown for the protocols in Figure 9. As illustrated in Figure 9a, Figure 9b and 9c, each phase in BFT protocols achieves a higher latency when f is larger and its latency is expanded in every phases, as expected in our framework.

E. Latency under Primary Failure - Best Scenario

We then evaluate the overall latency under primary failures and assess the best scenario. In this scenario, the primary replica crashes only once while the next primary is correct so that all replicas resume normal operations upon only one round of view change. Based on this, we set up the BFT evaluation by starting a $f = 1$ network, applying a normal operation workload, and killing one replica in the middle of each experiment. During the evaluation, we record the latency of view change protocol and report latency breakdown of the protocols as shown in Table III.

According to the $\mathsf{TheoLat}_{VC}$ values, the fractions of different steps during view changes for SBFT, PBFT and Prime are $3S_d : \rho S_{t,v}$, $3S_d : (4\rho+1)S_d$ and $8S_d : 4S_{t,s} + S_{t,c} + 6S_{t,v} + 4S_d : 7S_d$, respectively. In our experiments, ρ is around 10. The results match the experimental results except for PBFT. It is mainly because, in the implementation, verification of the messages. are executed in the VIEW-CHANGE phase.

F. Latency Breakdown under Primary Failures - Worst Scenarios

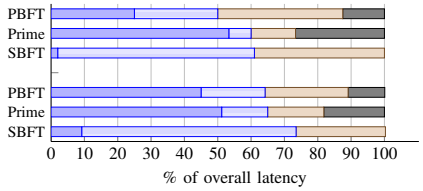
For the worst scenarios under primary failures, we first consider the crash failures where f primary replicas all crash. Then all replicas resume normal operations after view changes. We start a $f = 5$ network, apply a normal operation workload, and kill all f primary replicas (identify primary replicas by the replica IDs) in the middle of the experiments. Similar to the best scenario under primary failure, we assess the latency breakdown of the protocols, as shown in Table IV.

According to $\mathsf{TheoLat}_{VC}$, the fraction of view change steps for SBFT and PBFT are both $11S_d : \rho S_{t,v}$ and $11S_d : (12\rho + 1)S_d$. These results do not match exactly the experimental results. This is mainly because verification of committed requests in the *view-change* messages are implemented in the view change step, incurring a higher fraction of latency. The view change latency for SBFT is shorter than PBFT, which is expected in our framework. This is mainly because each primary verifies fewer authenticators due to the use of threshold signatures.

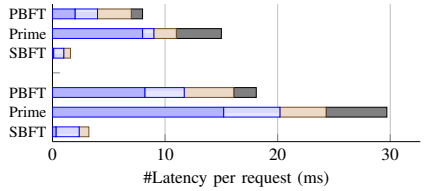
We also assess the performance of the worst-case scenarios under Byzantine failures. When we simulate the behavior of Byzantine replicas, we set up all f Byzantine primaries by letting them broadcast inconsistent messages to other replicas. During the evaluation, we find that the performance of Byzantine failures scenario is similar to that in the benign failures scenario. Therefore, the results we present in Table IV represent both the worst benign scenario and the worst Byzantine scenario of BFT consensus.

VI. Conclusion and Lessons Learned

We present a theoretical framework for evaluating BFT protocols. We provide a generic approach that calculates the concrete number of cryptographic operations in five different scenarios to understand the performance of BFT protocols. We

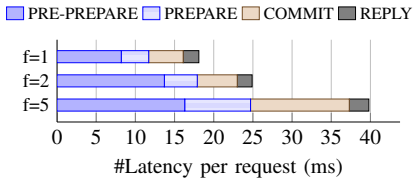


(a) Fractions of different phases over the overall latency.

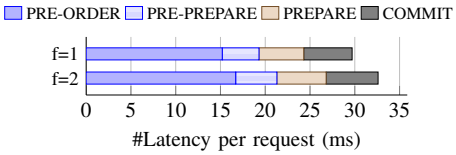


(b) Latency evaluated from the framework by inserting experimental results for cryptograhic operations.

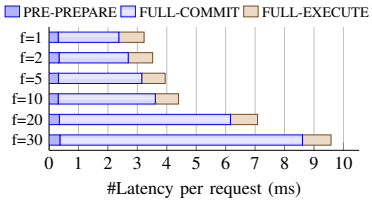
Figure 8: Theoretical and experimental practical results of BFT protocols in the backup failure scenario where $f = 1$ (the upper three are theoretical results, and others are experimental results).



(a) Latency breakdown of PBFT.



(b) Latency breakdown of Prime.



(c) Latency breakdown of SBFT.

Figure 9: Latency breakdown of PBFT, Prime, and SBFT when maximum number of faulty replicas f changes.

Protocol/Phase	VIEW-CHANGE	NEW-VIEW	Total
SBFT	5	18	23
Protocol/Phase	**VIEW-CHANGE**	**NEW-VIEW**	**Total**
PBFT	35.9	66.8	102.7
Protocol/Phase	**STATE-DISSEMINATION**	**PROOF-GENERATION**	**REPLAY**	**Total**
Prime	4.1	7.1	3.7	14.9

Table III: Latency breakdown of BFT protocol under primary failure best scenario (unit: ms).

Protocol/Phase	VIEW-CHANGE	NEW-VIEW	Total
SBFT	404	15	419
Protocol/Phase	**VIEW-CHANGE**	**NEW-MESSAGES**	**Total**
PBFT	739	94.2	833.2

Table IV: Latency breakdown of BFT protocol under primary crash failure worst scenario (unit: ms).

also evaluate the performance of selected protocols experimentally in the same LAN environment. Our experimental results validate most of our theoretical results.

Evaluating BFT protocols and understanding how each protocol behaves experimentally, especially under failures, has always been a challenge in the literature. In this work, we follow prior works that calculate the number of cryptographic operations. In practice, the performance of the protocols is related to many factors such as programming languages used and the underlying communication approach. Our experience with evaluating the protocols experimentally has shown that our framework is useful in estimating the fractions of latency in each phase of the protocols and the performance improvement/degradation of different protocols. Therefore, for the scenarios that are difficult to evaluate experimentally, the theoretical framework serves as a reference for assessment.

VII. Acknowledgements

This research was supported in part by the Director, Cybersecurity, Energy Security, and Emergency Response, Cybersecurity for Energy Delivery Systems program, of the U.S. Department of Energy, under the Grid Modernization Initiative under contract DE-AC02-05CH11231. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsors of this work.

References

[1] Y. Amir, B. Coan, J. Kirsch, and J. Lane. Prime: Byzantine replication under attack. *TDSC*, 8(4):564–577, 2011.

[2] A. Babay et al. Deploying intrusion-tolerant scada for the power grid. In *DSN*, pages 328–335. IEEE, 2019.

[3] A. Babay, T. Tantillo, T. Aron, M. Platania, and Y. Amir. Network-attack-resilient intrusion-tolerant scada for the power grid. In *DSN*, pages 255–266. IEEE, 2018.

[4] A. Bessani, J. Sousa, and E. E. Alchieri. State machine replication for the masses with bft-smart. In *DSN*, pages 355–362. IEEE, 2014.

[5] A. N. Bessani, M. Santos, J. Felix, N. F. Neves, and M. Correia. On the efficiency of durable state machine replication. In *ATC*, 2013.

[6] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *TOCS*, 20(4):398–461, 2002.

[7] S. Duan, K. Levitt, H. Meling, S. Peisert, and H. Zhang. ByzID: Byzantine fault tolerance from intrusion detection. In *SRDS*, 2014.

[8] S. Duan, H. Meling, S. Peisert, and H. Zhang. BChain: Byzantine replication with high throughput and embedded reconfiguration. In *OPODIS*, pages 91–106, 2014.

[9] S. Duan, S. Peisert, and K. N. Levitt. hBFT: Speculative byzantine fault tolerance with minimum cost. *TDSC*, 12(1):58–70, 2015.

[10] S. Duan, M. K. Reiter, and H. Zhang. BEAT: Asynchronous bft made practical. In *CCS*, pages 2028–2041, 2018.

[11] D. Duplyakin et al. The design and operation of CloudLab. In *ATC*, 2019.

[12] R. Guerraoui, N. Knežević, V. Quéma, and M. Vukolić. The next 700 bft protocols. *TOCS*, 32(4):12:1–12:45, 2015.

[13] G. G. Gueta et al. SBFT: a scalable decentralized trust infrastructure for blockchains. In *DSN*, 2019.

[14] R. Kapitza et al. CheapBFT: Resource-efficient Byzantine fault tolerance. In *EuroSys*, pages 295–308, 2012.

[15] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: speculative byzantine fault tolerance. In *SOSP*, pages 45–58, 2007.

[16] C. Liu, S. Duan, and H. Zhang. EPIC: efficient asynchronous BFT with adaptive security. In *DSN*, pages 437–451. IEEE, 2020.

[17] S. Liu, P. Viotti, C. Cachin, V. Quéma, and M. Vukolić. XFT: Practical fault tolerance beyond crashes. In *OSDI*, pages 485–500, 2016.

[18] P. J. Marandi, M. Primi, and F. Pedone. High performance state-machine replication. In *DSN*, pages 454–465. IEEE, 2011.

[19] A. Nogueira, M. Garcia, A. Bessani, and N. Neves. On the challenges of building a bft scada. In *DSN*, pages 163–170. IEEE, 2018.

[20] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In *PODC*, 2019.