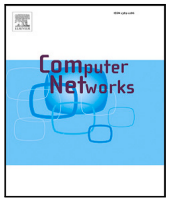
Computer Networks 249 (2024) 110404

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



BFTDiagnosis: An automated security testing framework with malicious behavior injection for BFT protocols

Jitao Wang, Bo Zhang, Kai Wang, Yuzhou Wang, Weili Han [*,1]

Laboratory for Data Security and Governance, Fudan University, China

ARTICLE INFO

Dataset link: https://github.com/Wangert/BFTDiagnosis

Keywords:
Consistency security
BFT protocols
Automated testing
Malicious behavior
Blockchain

ABSTRACT

In peer-to-peer computing networks, Byzantine Fault Tolerance (BFT) protocols are a popular solution to ensure the consistency security in the presence of some malicious nodes, thus are widely employed in blockchain systems. However, BFT protocols still face various security threats in practice. Currently, testing the security of BFT protocols often requires manual operations. I.e., it lacks comprehensive automation testing techniques. This makes it difficult to cover various scenarios of malicious node behaviors. Therefore, it is an urgent requirement to design an automated testing framework that can comprehensively and efficiently evaluate the security of BFT protocols.

This paper proposes BFTDiagnosis, an automated security testing framework with malicious behavior injection for BFT protocols. The framework can automatically configure and initiate protocol testing tasks, and construct consensus nodes to execute different BFT protocols. During testing, consensus nodes can inject malicious behaviors based on pattern matching strategies and malicious behavior triggering mechanisms to simulate various malicious scenarios. Through an Analyzer, BFTDiagnosis can collect protocol runtime data from consensus nodes and calculate four security quantification indicators to evaluate protocol performance.

We conducted a load consumption evaluation of BFTDiagnosis. The results indicate its acceptable performance. Additionally, we tested PBFT, Basic-HotStuff, and Chained-HotStuff protocols using the BFTDiagnosis framework, validating the effectiveness of the framework and the rationality of security quantification indicators. Finally, we compared it with nine related technologies and found that BFTDiagnosis is good at testing scenario comprehensiveness and fault localization capability, and can intuitively evaluate the security of BFT protocols through four quantification indicators. The above results show that BFTDiagnosis is an effective and comprehensive security testing framework for BFT protocols.

1. Introduction

Byzantine Fault Tolerant (BFT) [1] protocols ensure that honest nodes reach a consistent state in a distributed system, as long as the number of nodes showing malicious (i.e., Byzantine) behaviors is less than a fault tolerance threshold f. At present, BFT protocols are widely used in blockchain systems, e.g., FISCO BCOS [2], Cosmos and AntChain [3], to achieve the consistency of ledger data. Among many BFT protocols, PBFT [4] is the first practical BFT protocol, which divides nodes into the Primary and Backup (called Primary-Backup structure). Emerging BFT protocols, e.g., SBFT [5], Tendermint [6], the HotStuff series [7], DiemBFT [8], AptosBFT [9] and LWSBFT [10], often use the Primary-Backup structure. And these protocols are collectively referred to as Primary-Backup BFT (PB-BFT) protocols. Although researchers theoretically evaluate the performance and security of these protocols, practical applications of these protocols often expose new issues with performance and security due to differences in protocol implementation. Therefore, performance and security testing of BFT protocols is crucial in practical applications.

In recent years, researchers proposed a doze of testing methods for blockchain systems or consensus protocols. On the one hand, some testing frameworks for blockchain systems, e.g., Caliper [11] and HyperBench [12], can effectively evaluate the performance of blockchain systems, but cannot evaluate the security of blockchain systems. In addition, researchers utilized BFT-SMaRt [13] to evaluate the performance of BFT protocols in the development of new protocols. Compared with the testing frameworks of blockchain systems, BFT-SMaRt has the ability to intuitively reflect the performance of BFT protocols. However, BFT-SMaRt cannot evaluate the security of BFT protocols.

* Corresponding author.
E-mail address: wlhan@fudan.edu.cn (W. Han).
[1] Member, IEEE.

https://doi.org/10.1016/j.comnet.2024.110404
Received 25 October 2023; Received in revised form 11 March 2024; Accepted 6 April 2024
Available online 9 May 2024

The above frameworks can evaluate the performance of BFT protocols in practical applications, rather than evaluate the security of BFT protocols. On the other hand, Twins [14] is an automated unit test generator to detect consistency conflicts in DiemBFT. Twins simulate attack scenarios by creating a malicious node with the same identity as an honest node and having that malicious node launch malicious behaviors. Nevertheless, testers need to generate attack scenarios in advance, which results in a significant amount of time being spent on scenario generation when conducting large-scale security testing, making Twins inflexible. In addition, malicious nodes cannot synchronize with the execution state of honest nodes so that testers cannot simulate timing attack scenarios. Further, Twins can only detect the conflicts of DiemBFT protocol, but cannot detect other security problems, e.g., malicious attacks cause abnormal latency. And Twins also cannot accurately locate the source of conflicts. As a result, the practical and flexible security testing methods for BFT protocols are absent so far.

Designing a security testing method for BFT protocols poses significant challenges, mainly due to two critical issues. Firstly, the method should be able to flexibly simulate various malicious scenarios. During the execution, BFT protocols may encounter various malicious attacks. And these malicious attacks are closely related to behaviors of consensus nodes. Therefore, nodes launching malicious behaviors autonomously is the primary difficulty in flexibly simulating malicious scenarios. In addition, some malicious attacks rely on the timing of attacks. Although the timing of malicious attacks can be predetermined, it is meaningless because the state changes of the protocol execution process before testing are unpredictable. Therefore, choosing a reasonable time for launching malicious behaviors is the second difficulty. Secondly, the method should be able to quantify the security of BFT protocols. In current, researchers theoretically verified the security (i.e., Consistency) of BFT protocols under the condition of satisfying a security threshold f. However, the security of BFT protocols in practical applications is determined not only by consistency but also by the extent to which attacks affect performance, which is affected by various factors such as hardware devices, network environment, and implementation methods. So it is difficult to analyze the security of the BFT protocol theoretically in such a complex scenario. In contrast, quantitative methods can intuitively reflect the changes in numerical values. Therefore, we need to quantify the security of BFT protocols.

Compared to prior works [14–19], our designed testing framework can encompass a broader range of malicious scenarios, thereby comprehensively and efficiently evaluating the security of BFT protocols. The framework exhibits greater applicability and flexibility, and objectively reflects the robustness of protocols under different malicious scenarios through four quantitative security metrics. While meeting the testing demands of large-scale systems, our testing framework possesses robust fault localization capabilities, allowing for intuitive identification of fault-prone phases within the protocol, thus enhancing protocol reliability and stability.

Contributions. In this paper, we propose `BFTDiagnosis`,[2] a automated security testing framework for BFT protocols. `BFTDiagnosis` can support BFT protocols that meet the Primary-Backup structure to evaluate their security. To simulate various malicious scenarios flexibly, we design a node mode with behavior-triggering to control the various behaviors of consensus nodes, enabling the automatic execution for behaviors of consensus nodes. Then, we design a malicious message forgery method to construct malicious interactive messages of the consensus node, and combined with the node mode to achieve the purpose of triggering malicious behaviors. Further, we present a protocol phase division rule to divide the protocol phase reasonably, so that `BFTDiagnosis` enable control the trigger timing of malicious behaviors. On the other hand, we define four indicators to quantify the security of BFT protocols, including mean latency, abnormal latency rate, mean latency variation rate, mean throughput, mean throughput variation rate and consistency disruption degree. In addition, we design and develop three core components to establish the `BFTDiagnosis` framework: the Controller, which controls the process of security testing; the Protocol Actuator, which is the running container of BFT protocols; the Analyzer, which collects and analyzes the protocol running data.

Finally, we integrated representative PBFT, Basic-HotStuff, and Chained-HotStuff protocols into `BFTDiagnosis`. Through latency and throughput comparisons, we demonstrated that `BFTDiagnosis` had little impact on the performance of BFT protocols. Further, we use `BFTDiagnosis` to test the security of the above three protocols in nine malicious behavior scenarios, including FeignDeath (Leader/Replica), Duplicate (Leader/Replica), Delay (Leader/Replica), Ambiguous (Leader/Replica), and Conspire (Replica). We obtained a large number of testing results, which show that `BFTDiagnosis`'s ability to intuitively reflect the security indicators of three protocols in various malicious scenarios. Therefore, `BFTDiagnosis` can serve as a valuable security evaluation reference for other BFT protocols and their variants. With the maturity of the BFT protocol testing framework, BFT protocols can be better applied to fields such as multi-level network construction [20] and industrial production [21,22].

Organization. The rest of this paper is organized as follows: Section 2 presents the preliminary. Section 3 describes the specific design and implementation principle of `BFTDiagnosis`. Section 4 presents the experiment of `BFTDiagnosis`. Section 5 discusses the achievements of our framework. Section 6 discusses the related work. Section 7 concludes the paper and presents the future work.

2. Preliminary

In this section, we describe the relevant knowledge points involved in this paper.

2.1. Primary-backup BFT protocols

The Primary-Backup BFT protocol, which originated from the development of PBFT (Practical Byzantine Fault Tolerance), has become a popular choice for implementing Byzantine fault tolerance in distributed systems. This type of protocol typically has three main features:

(a) Primary-Backup structure. The protocol divides nodes into Primary and Backup roles, with the Primary node responsible for organizing and initiating a new consensus request or block to the Backup nodes, and the Backup nodes responsible for verifying the consensus request or block.
(b) Phased sequential execution. The Primary-Backup BFT protocol is characterized by its strong sequential execution, whereby each individual or batch consensus request is required to undergo multiple verification steps that depend on each other in a specific order.
(c) View-change. View-change is a crucial mechanism in Primary-Backup BFT protocols that enables a system to recover from failures. In Primary-Backup BFT protocols, each node maintains its own view of the current leader and a log of all the messages exchanged between nodes. When a view-change occurs, nodes work together to elect a new leader and transition to the next view. This collaborative process ensures the safety and liveness of the system despite failures that may occur.

[2] This framework is open source by accessing "https://github.com/Wangert/BFTDiagnosis".

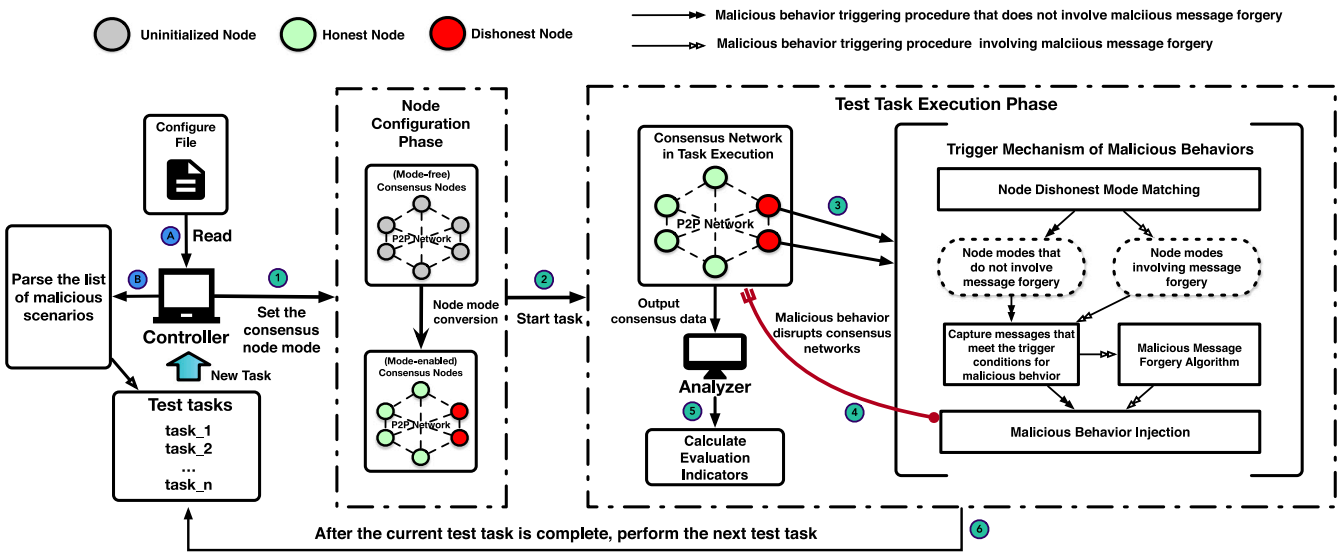


Fig. 1. Overview of BFTDiagnosis.

2.2. Malicious behaviors of primary-backup BFT protocols

In the PB-BFT protocol, nodes generally perform three types of behaviors: sending messages, receiving messages and locally processing. Malicious behaviors refer to intentional and targeted actions by malicious nodes to disrupt the normal operation of the system. We collect the literature [14,23–26] related to BFT protocol security issues and then classify that malicious behavior by nodes typically includes the following:

(a) From the perspective of message transmission, nodes can perform three types of malicious behavior: Delayed transmission, Duplicate transmission, and Feign death. Delayed transmission refers to a node intentionally waiting for a period of time before sending a message when the conditions for sending a message and triggering malicious behavior are met. Duplicate transmission refers to a node sends a message that has already been sent before when the conditions for sending a message and triggering malicious behavior are met. Feign death refers to a node fails to send a message when the conditions for sending the message and triggering malicious behavior have been met but then sending the message when the conditions for triggering malicious behavior are no longer present.

(b) From the perspective of combining local processing and message transmission, malicious nodes can perform the malicious behavior of sending ambiguous messages. This means that when the conditions for sending a message and triggering malicious behavior are met, the node locally constructs messages that are contradictory to each other and distributes them to other nodes.

3. Design of BFTDiagnosis

In this section, we describe the design of BFTDiagnosis, a security testing framework for Primary-Backup BFT protocols.

3.1. Overview

3.1.1. Framework components

BFTDiagnosis is composed of three key components, namely the Controller, the Protocol Actuator and the Analyzer.

- **Controller**. The Controller efficiently manages the operations of the Protocol Actuator and Analyzer during security testing. Upon initialization, it generates testing tasks from the configuration file. During the execution of security testing tasks, the Controller ensures proper configuration of the Protocol Actuator and Analyzer before initiating the testing. Moreover, the Controller automatically switches between security testing tasks.
- **Protocol Actuator**. The Protocol Actuator is a container designed to execute BFT protocols. Initially, it lacks any specific protocol logic, and therefore, the user must implement a series of specific interfaces to furnish the required protocol logic. Once the necessary protocol execution logic is incorporated, the Protocol Actuator transforms into a runnable-consensus-node whose behavior may differ based on the security testing tasks. Moreover, the consensus node persistently feeds running data to the Analyzer during the testing process.
- **Analyzer**. The Analyzer is responsible for collecting and analyzing protocol running data. The Analyzer has the ability to make evaluation decisions based on the protocol running data, and it calculates various security indicators of the protocol.

3.1.2. Workflow

BFTDiagnosis is capable of adapting to different BFT protocols. Prior to testing a BFT protocol with BFTDiagnosis, the core logic of the BFT protocol needs to be integrated into BFTDiagnosis through the interface exposed by the protocol executor. This integration allows for the construction of consensus nodes with the ability to run the protocol. Subsequently, a certain number of consensus nodes, controllers, and analyzers are launched according to the requirements. Finally, the protocol testing begins.

The overall execution process of BFTDiagnosis is illustrated in Fig. 1, before BFTDiagnosis executes the protocol security testing task, it needs to establish a test task list. The controller goes through steps A and B, reads the protocol test configuration file written by the user, and parses the list of malicious scenarios to be simulated according to the configuration file. Finally, it establishes and outputs the task list to be executed. Once the test task list is ready, BFTDiagnosis completes a test task execution through steps ①-⑥:

① The Controller retrieves a new test task from the test task list and sets the mode of the consensus nodes according to the scenario requirements of the test task. When the consensus nodes in the consensus network receive instructions from the Controller to set the mode, each consensus node transitions from a no-mode state to a mode state, and the mode of each consensus node depends on the scenario requirements of the test task;

② After the consensus nodes are configured, BFTDiagnosis begins to execute the test task. During the task start period, the consensus

data generated by the consensus network nodes is continuously sent to the Analyzer;

③ Nodes in Dishonest mode in the consensus network will trigger the malicious behavior mechanism in the protocol executor. This mechanism matches the Dishonest mode of the current consensus node. If the current Dishonest mode does not require message forgery, the mechanism will directly capture the corresponding message and perform the injection process of malicious behavior; if the current Dishonest mode requires message forgery, the mechanism will first capture the corresponding message, then forge the message according to the malicious message forging algorithm, and finally perform the injection process of malicious behavior;

④ Nodes in Dishonest mode will initiate corresponding malicious behaviors to disrupt the operation of the consensus network due to the malicious behavior mechanism;

⑤ When a sufficient amount of consensus data has been received, the analyzer will use the consensus data collected from each consensus node to calculate security evaluation indicators, thereby completing the current round of testing tasks;

⑥ After the current round of testing tasks is completed, a new round of testing tasks is performed according to the task list to be executed, until all testing tasks are completed.

3.2. Node mode with behavior-triggering

The execution process of the BFT protocol is closely related to the behavior of consensus nodes, so the malicious behavior of consensus nodes may lead to protocol abnormalities. To simulate malicious scenarios arising from different malicious behaviors of consensus nodes, it is common to impersonate honest nodes and launch malicious attacks externally to disrupt normal protocol execution. However, this method requires real-time tracking of running nodes to capture the timing of the attack and may not accurately create specific attack scenarios. Therefore, the primary solution to the aforementioned issue is to internally control the interaction behavior of consensus nodes during protocol execution. To achieve this, we designed a set of node modes for consensus nodes.

Uninitialized Mode. The node in the Uninitialized mode represents its initial state, in which it is incapable of exhibiting any honest or dishonest behavior. In other words, all other node modes stem from the Uninitialized mode.

Honest Mode. The Honest mode of a consensus node enables it to initiate honest behavior while refraining from any dishonest conduct that could disrupt the protocol's regular operation. Such a node is expected to maintain the correct interaction behavior consistently, thereby ensuring proper protocol execution across the entire network.

Dishonest Mode. The consensus node in Dishonest mode can trigger honest behavior, but under certain conditions, it may engage in various types of dishonest behavior to disrupt the normal operation of the protocol. However, the specific types of dishonest behavior exhibited by consensus nodes are diverse and can result in various malicious scenarios. Therefore, it is crucial to precisely define the types of dishonest behavior displayed by consensus nodes. To address this, we have further classified the malicious behaviors of consensus nodes as follows:

- *FeignDeath(Leader/Replica)*. When a consensus node acts as a Leader or Replica, it engages in malicious behavior by deliberately ignoring messages from other consensus nodes under trigger conditions for feigning inactivity in the consensus network.
- *Duplicate(Leader/Replica)*. When a consensus node acts as a Leader or Replica, it engages in malicious behavior by deliberately increasing message redundancy in the consensus network by duplicately sending the same message to other consensus nodes, under the trigger condition of satisfying for message repetition.
- *Delay(Leader/Replica)*. When a consensus node acts as a Leader or Replica, it engages in malicious behavior by deliberately delaying the transmission of messages under the condition of satisfying the trigger condition for message delay. This aims to simulate the malicious behavior of delayed message propagation in the consensus network.
- *Ambiguous(Leader/Replica)*. When a consensus node acts as a Leader or Replica, it engages in malicious behavior by differently processing the same message from another consensus node to generate distinct response messages under the trigger condition for ambiguous messages. This results in different response messages being sent to different consensus nodes, causing the message to be ambiguous in the network.
- *Conspire(Replica)*. When a consensus node acts as a Replica, it engages in malicious behavior by colluding with other replicas in Conspire mode to process the same message in the same malicious way under the trigger condition of conspire. This results in the formation of an identical malicious response message, which is then sent to other honest consensus nodes, thereby achieving the malicious behavior of Replica collusion.

In the scenario simulation, each consensus node has one of the aforementioned node modes at any given time, and we are able to alter the behavior of consensus nodes by changing their node modes. This enables us to internally control the interaction behavior of consensus nodes, instead of injecting externally.

3.3. Malicious messages forgery for protocols

Among the above-mentioned node modes, some of the malicious behaviors (e.g., Ambiguous and Conspire) in Dishonest mode need to be triggered by malicious tampering with protocol messages. To enable consensus nodes to independently forge malicious messages and trigger malicious behaviors, `BFTDiagnosis` employs a message forgery method based on field chain, which is embedded in the node mode (The specific logic can be seen in Algorithm 1). It is well-known that generating a fake message for protocols with known message structures is relatively easy. However, forging a message for protocols with unknown message structures is difficult, and therefore obtaining the message structure of protocols becomes the first step.

To parse the message structure of protocols, users need to implement the *send_protocol_phase_messages* interface of Protocol Actuator when customizing a protocol. As shown in Fig. 2 Steps①-③, the Controller triggers a consensus node to execute the *send_protocol_phase_messages* method, and then the node sends messages of each phase in its protocol as a JSON serialization string to the Controller. The Controller parses fields of the JSON serialization string to obtain the tree structure of message fields. For example, as shown in Fig. 2, Field_2 has two sub-fields s_Field_1 and s_Field_2, and s_Field_2 also has two sub-fields ss_Field_1 and ss_Field_2, so Field_2 and s_Field_2 are structure types. However, fields that do not have sub-fields are primitive types (Null, Bool, Number, String, Array). At this time, the Controller has obtained the message structure of each phase in this protocol and enables to forge messages for this protocol, but this is not the final goal, we hope that consensus nodes can construct malicious messages autonomously under certain conditions.

To enable consensus nodes to construct malicious messages independently based on specific message fields accurately, `BFTDiagnosis` embeds the field chain into the dishonest mode. As shown in Fig. 2 Step④, since non-leaf fields are generated from leaf fields, only leaf fields of the message structure tree can be used as forged fields. Controller stores the field path of each leaf field locally to form a field chain list. When a consensus node obtains a field chain, it can accurately match and modify a field of the message. As shown in Fig. 2 Steps⑤-⑧, the Controller adds a field chain parameter to the Dishonest mode to set the mode of consensus nodes. When the triggering conditions are met, this consensus node maliciously modifies the message field according to this field chain to construct a malicious message.

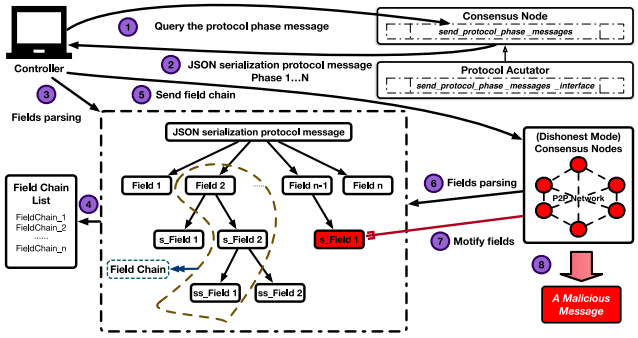


Fig. 2. The process of forging a malicious message.

Algorithm 1 forging a malicious message.

Input: $node, FC, msg\_bytes, phase$
Output: msg^{err}
//modify messages field
Function modify_msg (msg_field_map, FC, error_content)
Match $msg\_field\_map[FC[0]]$ **do**
Case Null **or** Bool **or** Number **or** String **or** Array:
return $msg\_field\_map[FC[0]] = error\_content$
Case Object(sub_map):
return $FC.remove(0)$
new_msg_field_map=modify_msg(sub_map,FC)
End Match
if $FC.length == 0$ **then**
return $new\_msg\_field\_map$
end if
// generate a malicious message
Function generate_err_message(node,FC,msg_bytes,phase)
phase_msg_type = node.phase_map[phase]
msg_struct_map = json_parse(msg_bytes)
new_msg_struct_map=modify_msg(msg_struct_map,FC)
err_msg = map_to_json_string(new_msg_struct_map)
return $malicious\_msg$

3.4. Phase division rule

Different triggering times of malicious behavior have varying impacts on BFT protocols. During the execution process of BFT protocols, the same node exhibiting malicious behavior in different phases may result in completely different outcomes. In other words, a particular malicious behavior may have a negative impact on a specific execution, and the degree of impact on such executions may differ. Hence, the time of triggering malicious behavior is crucial for simulating malicious scenarios in BFT protocols.

Unfortunately, it is infeasible to control the trigger time of malicious behaviors. Due to the impact of network environment, hardware devices, or implementation logic of the protocol on the actual execution process, we cannot predict the running condition of the protocol, making it difficult to accurately set the trigger time point of malicious behavior in advance to simulate specific malicious scenarios. To overcome this challenge, we regard the protocol execution phase as the triggering point for the malicious behavior of consensus nodes. Structurally, the execution process of BFT protocols proceeds link by link without considering parallel processing. Importantly, in a normal network environment, the processing time of each link should be short, which makes it reasonable and feasible to use each execution phase as a triggering point for malicious behaviors.

Typically, BFT protocols have their own phase design logic, but some phases are still complex and indistinguishable. To divide the execution process of PB-BFT protocols more fine-grained to support more comprehensive security tests, BFTDiagnosis presents a protocol phase division rule:

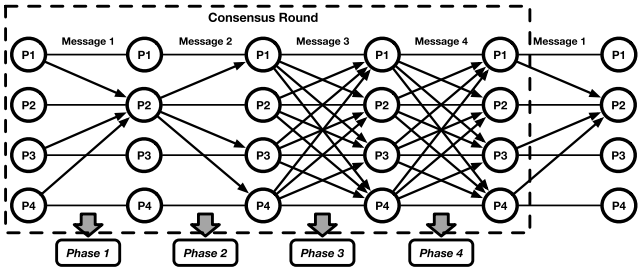


Fig. 3. An example of protocol phase division rule.

During the process of executing a BFT protocol, BFTDiagnosis assumes that each consensus round is triggered by an initial message called M_{init}, where M_{init} is used as a marker to separate different consensus rounds. Within the same consensus round, different types of messages sent through unicast or broadcast are considered as distinct behaviors by consensus nodes, and are used to divide the protocol into different phases.

The hypothetical example depicted in Fig. 3 is a BFT protocol. Nodes send various messages at different times, and following the protocol phase division rule of BFTDiagnosis, Message 1 acts as a separator for consensus rounds. Within each consensus round, Messages 1, 2, 3, and 4 correspond to Phases 1, 2, 3, and 4, respectively.

According to the protocol phase division rule of BFTDiagnosis, different BFT protocols can be easily and clearly divided into phases, and these phases are taken as the trigger time points for malicious behaviors.

3.5. Security testing execution

BFTDiagnosis divides the security testing execution process into malicious behavior injection and security evaluation. On the one hand, we use methods and rules proposed above to inject malicious behaviors of consensus nodes into protocols, so as to simulate various malicious scenarios. On the other hand, we define four security indicators to evaluate the security of BFT protocols in various malicious scenarios.

Malicious behavior injection. Injection processes of nine malicious behaviors are shown in Fig. 4. During security testing, the Controller of BFTDiagnosis can dynamically inject corresponding malicious behaviors into workable consensus nodes by setting a node mode. Generally, for BFT protocols, the trigger type and trigger time of malicious behaviors will lead to different abnormal effects. (The specific logic can be seen in Algorithm 2) In the scenario where Replica initiates malicious behaviors, the number of malicious replicas is also a crucial factor. Therefore, in order to comprehensively test the protocol, BFTDiagnosis introduces four general parameters in the node model, namely, role parameter R (Leader/Replica), trigger phase parameter P (based on phase division rules), malicious replicas parameter N_r and stable time parameter ΔT. The stable time parameter is to avoid the impact of the instability in the early stage of protocol operation on testing results. Therefore, malicious behaviors are injected after the protocol reaches a stable state. In addition to the four general parameters, different malicious behaviors also have special parameters. The injection process of different malicious behaviors will be described in detail as follows (The specific logic can be seen in Algorithm 3):

- $FeignDeath(\Delta T, R, P, N_r)$. When the protocol is in phase P after ΔT time, the consensus node of $FeignDeath$ mode will trigger the malicious behavior of $Feigndeath$, that is, ignore the received protocol message at this time and stop the broadcast of the message in the next phase. When $R = Leader$, the N_r parameter is ignored. Only the consensus node in $FeignDeath$ mode acting as Leader will trigger feign death. When $R = Replica$, N_r replicas are in $FeignDeath$ mode. These replicas trigger feign death when they meet the conditions.

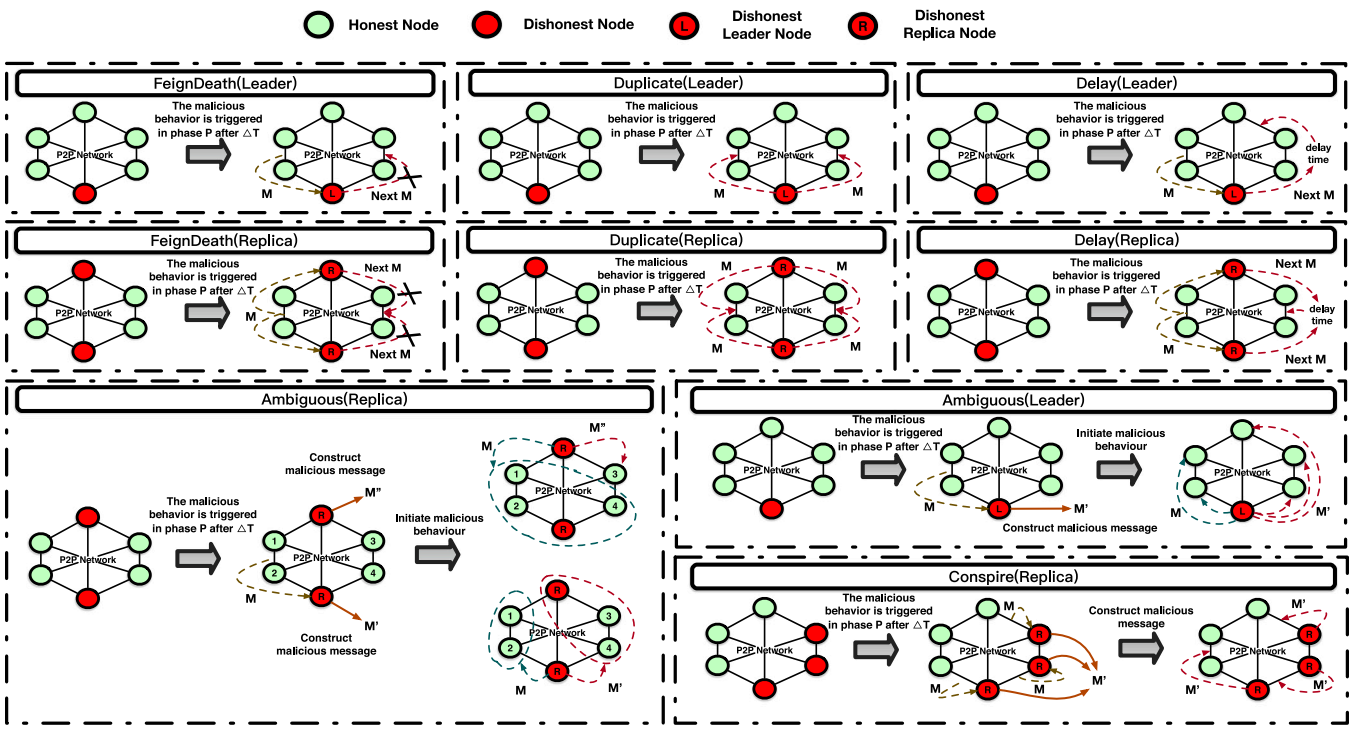


Fig. 4. Injection processes for 9 malicious behaviors, including FeignDeath (Leader/Replica), Duplicate (Leader/Replica), Delay (Leader/Replica), Ambiguous (Leader/Replica), and Conspire (Replica).

- $Duplicate(\Delta T, R, P, N_r)$. When the protocol runs statically for ΔT time and is in stage P, the consensus node in the *Duplicate* mode will trigger the malicious behavior of repeated message sending. That is, once the protocol message M is received, the consensus node will repeatedly send the next stage message M_{next} to the consensus network. When $R = Leader$, the N_r parameter is ignored. Only when the *Duplicate* consensus node serves as the Leader, the message duplication behavior is triggered. When $R = Replica$, there are N_r replicas.
- $Delay(\Delta T, R, P, T_{delay}, N_r)$. When the protocol runs statically for ΔT time and the protocol is in phase P, the consensus node in *Delay* mode will trigger the malicious behavior of delaying message sending. That is, once receiving the protocol message M, the consensus node will temporarily detain the response message M_{next} locally and send M_{next} to the consensus network after T_{delay} time. When $R = Leader$, the N_r parameter is ignored. The message *Delay* behavior is triggered only when the consensus node in delay mode acts as the Leader. When $R = Replica$, N_r replicas trigger message delay.
- $Ambiguous(\Delta T, R, P, FC, S_a, N_r)$. When the protocol is stable for ΔT time and the protocol is in phase P, the *Ambiguous* mode consensus node triggers the ambiguous message's malicious behavior. Ambiguous malicious behavior injection has its own parameters, FC is the forgery parameter of ambiguous message, that is, the field chain of protocol message, consensus node will forge the message of the front wheel according to FC parameters; S_a is the ambiguous message number set (Assuming $S_a = \{M_0, \ldots, M_{N_r}\}$), where M_0 is the ambiguous message number of the Leader node, and the rest $M_1 \ldots M_{N_r}$ corresponds to N_r Replica. When $R = Leader$, the N_r parameter is ignored, and when the consensus node in *Ambiguous* mode acts as the Leader, it generates a malicious message M' of the correct message M according to the FC parameter, sending M' randomly to M_0 consensus nodes and M to the remaining consensus nodes. When $R = Replica$, N_r replicas generate malicious messages based on the correct message M and send malicious messages to different nodes based on the S_a set. As shown in Fig. 4, one Replica generates malicious messages M' and sends M' to three consensus nodes. Another Replica generates a malicious message $M\varepsilon$ and sends $M\varepsilon$ to a consensus node.
- $Conspire(\Delta T, P, FC, REQ, N_r)$. The *Conspire* Replica in mode triggers malicious behavior that nodes conspire to forge messages if the protocol is in phase P after stable operation for ΔT time. Has its own parameters when Ambiguous malicious behavior is injected, where FC is a forged parameter for ambiguous messages; REQ is the consensus request identifier. N_r malicious replicas forge the REQ request message based on FC to generate malicious messages, and then send the malicious messages to the consensus network. As shown in Fig. 4, three malicious replicas conspired to forge M' and sent it to the consensus network.

During the actual testing, all variable parameters (including the phase, number of malicious nodes, malicious fields, and node roles) of different malicious behavior scenarios will be iterated for `BFTDiagnosis` to comprehensively test the security of protocols.

Security evaluation. Although the malicious behavior injection can flexibly simulate the malicious scenario of the primary and secondary BFT protocols, the result of the protocol security is still obscure. To evaluate the protocol security more intuitively, we designed a security evaluation strategy based on the protocol running data. Due to the differences in the running data of different protocols, in order to simplify the complex diversity of the running data of different protocols and avoid the difficulty of calculating security indicators, we set two simple protocol running output data in the Protocol Actuator:

(a) $C^{start} = \{Hash(REQ), TS^{start}\}$. Consists of a round of consensus request content Hash value and consensus start time stamp.
(b) $C^{end} = \{Hash(REQ), TS^{end}\}$. Consists of a round of consensus request content Hash value and consensus end time stamp.

In a consensus round, the consensus node sends C^{start} and C^{end} data to the analyzer at the beginning and end, respectively. Therefore, the analyzer continues to receive running data from different consensus nodes during protocol testing execution. Despite the simple

Algorithm 2 The Controller manages test tasks.

Input:
N: Number of consensus nodes
ΔT: stable runtime parameter of the protocol
T_{delay}: Delay time under 'delay' malicious behavior
tasks: list of test tasks
Ctrl: Controller
fp: Configuration file path
$nodes^{\text{all}}$: Collection of all consensus nodes

// initialization phase
$(N, \Delta T, T_{\text{delay}}, \text{tasks}) \leftarrow$Ctrl.read_config_file(fp)
while *true* **do**
  $N' \leftarrow$ gossip protocol listens for started consensus nodes
  if $N = N'$ **and** N' consensus nodes are running properly **then**
    $C_{\text{phase}} \leftarrow$Ctrl.query_"protocol_phase_count(any_node)"
    Ctrl.distribute_keypair($nodes^{\text{all}}$)
    Ctrl.set_consensus_mode("Uninitialized", $nodes^{\text{all}}$)
    Break
  end if
end while

// Set testing tasks
for each task in tasks **do**
  if task.role = "Leader" **then**
    for each P in $1, 2, \ldots, C_{\text{phase}}$ **do**
      if task.type = "FeignDeath" **then**
        $consensus\_mode = FeignDeath(Leader, P)$
      else if task.type = "Duplicate" **then**
        $consensus\_mode = Duplicate(Leader, P)$
      else if task.type = "Delay" **then**
        $consensus\_mode = Delay(Leader, P, T\_delay)$
      else if task.type = "Ambiguous" **then**
        $consensus\_mode$=Ambiguous(Leader,P,FC,S_a)
      end if
      ($nodes^{\text{target}}$, $nodes^{\text{other}}$)=select_node($nodes^{\text{all}}$,1)
      Ctrl.set_consensus_mode(consensus_mode,$nodes^{\text{target}}$)
      Ctrl.set_consensus_mode("Honest",$nodes^{\text{other}}$)
      wait for instructions from Analyzer to execute the next round of test task
    end for
  else if task.role = "Replica" **then**
    for each $N_r in 1, 2, \ldots, N$ **do**
      for each P in $1, 2, \ldots, C_{\text{phase}}$ **do**
        if task.type = "FeignDeath" **then**
          consensus_mode = $FeignDeath(Replica, P, N_r)$
        else if task.type = "Duplicate" **then**
          $consensus\_mode = Duplicate(Replica, P, N_r)$
        else if task.type = "Delay" **then**
          consensus_mode = $Delay(Replica, P, T_{\text{delay}}, N_r)$
        else if task.type = "Ambiguous" **then**
          consensus_mode=$Ambiguous(Replica, P, FC, S_a, N_r)$
        else if task.type = "Conspire" **then**
          consensus_mode=$Conspire(Replica, P, FC, REQ, N_r)$
        end if
        ($nodes^{\text{target}}$, $nodes^{\text{other}}$)=select_node($nodes^{\text{all}}$,1)
        Ctrl.set_consensus_mode(consensus_mode,$nodes^{\text{target}}$)
        Ctrl.set_consensus_mode("Honest",$nodes^{\text{other}}$)
        wait for instructions from Analyzer to execute the next round of test task
      end for
    end for
  end if
end for

Algorithm 3 malicious behaviors triggering.

Input: P, node, msg
Output:
Messages are maliciously processed according to the consensus node mode

// Consensus node malicious behavior trigger function
Function malicious_triggering(P, node, msg)
msg_{next} = build_next_msg(msg)
if IsDishonest(node.mode) **and** node.mode.phase == P **then**
  Switch (node.mode)
  Case FeignDeath:
    ignore "msg"
  Case Duplicate:
    multiple broadcast msg_{next} to other consensus node
  Case Delay:
    sleep(node.mode,T_{delay})
    broadcast msg_{next} to other consensus node
  Case Ambiguous:
    fc = node.mode.fc
    $msg^{\text{malicious}}_{\text{next}}$=generate_error_message(fc,msg_{next})
    // The consensus node to which "$msg^{\text{malicious}}_{\text{next}}$" is sent is selected
    C_{amb} = node.ambiguous_count
    (nodes1, nodes2)=select_nodes($nodes^{\text{all}}$, C_{amb})
    broadcast msg_{next} to nodes1
    broadcast $msg^{\text{malicious}}_{\text{next}}$ to nodes2
  Case Conspire:
    R_{mode}=node.mode.request
    R_{msg}=msg.request
  if IsReplicaNode(node) **and** $R_{\text{mode}} == R_{\text{msg}}$ **then**
    fc = node.mode.fc
    $msg^{\text{malicious}}_{\text{next}}$=generate_error_message(fc, msg_{next})
    Broadcast $msg^{\text{malicious}}_{\text{next}}$ to other consensus node
  end if
  End Switch
end if

structure of the two output data, the analyzer can use these data to calculate a variety of security metrics. In a round of testing, assuming that there are N consensus nodes in the network, the analyzer will store the running data of these N consensus nodes $S_{RD} = \{D_0, D_1, \ldots, D_N\}$, where $D_i = \{CS^{start}, CS^{end}\}$, CS^{start} and CS^{end} are C^{start} and C^{end} sets of different consensus requests, respectively. $REQ_k = \{(C^{start}_{k1}, C^{end}_{k1}), (C^{start}_{k2}, C^{end}_{k2}), \ldots, (C^{start}_{kN}, C^{end}_{kN})\}$, where C^{start}_{kj} or C^{end}_{kj} of any consensus node may be empty due to malicious scenarios. In the testing process, the run time data of malicious nodes will have a huge impact on the testing results. Therefore, to ensure the rationality of the results, we further eliminate the run time data of malicious nodes to obtain REQ^m_k, which does not contain malicious node data. In addition, to indicate whether a consensus request completes the final output on a consensus node, we define γ_{kj} value:

$$\gamma_{kj} = \begin{cases} 1, & C^{start}_k \in D_j \wedge C^{start}_k \in D_j \\ 0, & C^{start}_k \notin D_j \vee C^{start}_k \notin D_j \end{cases}$$

Based on the situation of each consensus node, whether each consensus request meets the final consistency can be obtained, which is expressed by the value of Γ_k:

$$\Gamma_k = \bigwedge_{j=1}^{j=N-M} \gamma_{kj}$$

Firstly, we defined and calculated two auxiliary indicators, which provide support for calculating the security indicators. (assuming the number of malicious nodes is $M < N$ and there are Q consensus requests)

- Mean latency (L_{mean}). A consensus request has run time data for $N - M$ honest nodes, based on which the consensus request latency for each is calculated, and then the average latency for

all consensus requests that meet the conformance condition is calculated.

$$L_{mean} = \frac{\sum_{k=1}^{k=Q} \sum_{j=1}^{j=N-M} \Gamma_k (TS_{kj}^{start} - TS_{kj}^{end})}{Q(N-M)}$$

- Mean Throughput (T_{mean}).The number of consensus requests that meet the consistency conditions within a period of time is counted as the number of requests that complete the consensus ΔT_{exec}.

$$T_{mean} = \frac{\sum_{k=1}^{k=Q} \Gamma_k}{\Delta T_{exec}}$$

According to the protocol running data obtained above, and for different malicious scenarios, we define and calculate the following four security indicators. (assuming the number of malicious nodes is $M < N$ and there are Q consensus requests)

- Abnormal latency rate ($AR_{latency}$). In the testing process, each consensus request is taken as an independent sample to calculate the latency of each consensus request. When the latency exceeds the threshold χ, we mark it as an anomaly, where χ is the standard deviation of the consensus request latency in the protocol without injected malicious behavior. Calculate the proportion of the number of abnormal requests to the total number of requests based on the number of abnormal latency.

$$\chi = \sqrt{\frac{\sum_{k=1}^{Q} \sum_{j=1}^{N} ((TS_{kj}^{start} - TS_{kj}^{end}) - L_{mean}^{normal})^2}{QN}}$$

$$\alpha_k = \begin{cases} 1, & (TS_k^{end} - TS_k^{start}) > \chi \\ 0, & (TS_k^{end} - TS_k^{start}) \leq \chi \end{cases}$$

$$AR_{latency} = \frac{\sum_{k=1}^{k=Q} \alpha_k}{Q}$$

- Mean latency variation rate ($VR_{latency}$). The average delay in the normal and malicious scenarios is calculated respectively, and the difference ratio of the average delay in the malicious scenario compared with that in the normal scenario is calculated.

$$VR_{latency} = \frac{L_{mean}^{malicious}}{L_{mean}^{normal}}$$

- Throughput variation rate ($VR_{throughput}$). The throughput in the normal and malicious scenarios is calculated respectively, and the throughput difference ratio between the malicious scenario and the normal scenario is calculated.

$$VR_{throughput} = \frac{T_{mean}^{malicious}}{T_{mean}^{normal}}$$

- Consistency disruption degree ($DD_{consistency}$). Reflected in the impact of malicious behavior on consensus request consistency in the process of protocol execution, the number of Q consensus requests that do not meet the consistency conditions accounted for the proportion of the total number.

$$DD_{consistency} = \frac{\sum_{k=1}^{k=Q} \Gamma_k}{Q}$$

In general, malicious behavior injection and security evaluation strategies can be used to conduct comprehensive security tests on different primary and secondary BFT protocols, and obtain multidimensional security indicator values to evaluate the protocol execution process in malicious scenarios.

3.6. Architecture of BFTDiagnosis

Based on the above mechanism, we design and develop the Controller, the Protocol Actuator and the Analyzer to build the framework of `BFTDiagnosis`. Three components cooperate with each other to perform the security testing of BFT protocols. And their design structure is shown in Fig. 5.

The Controller is composed of *Security Test Management* module and *Instruction Sending* module to manages the security testing process of BFT protocols. Before testing, the Controller reads the configuration file through the *Test Parameter Parsing* submodule to set up the testing environment, including the IP addresses and ports of the Protocol Actuator and the Analyzer, the list of malicious behavior testing contents, the attack intensity and the number of consensus requests. Then, the *Key Distribution* and *Consensus Node Mode Setting* submodules are used to allocate keys and set the node mode for the Protocol Actuator (i.e., consensus node), completing the testing initialization process. Next, the Controller uses the *Test Task Start* submodule to start the protocol operation of the Protocol Actuator and continuously sends consensus requests to the consensus network through the *Consensus Request Generation* submodule. All of the aforementioned instructions are received, processed, and sent to the Protocol Actuator and the Analyzer by the *Instruction Sending* module.

The protocol integration process is entirely dependent on the Protocol Actuator, which consists of the *Network* module, *Digital Signature* module, *Protocol* module, *Mode Management* module, and *Signal Processing* module. The *Protocol* module serves as the core of the Protocol Actuator, while the other modules operate around it. During the operation of the Protocol Actuator, the *Protocol* module uses the *Network* module and *Digital Signature* module to achieve message transmission and signature verification. The *Mode Management* module dynamically manages the modes of the Protocol Actuator based on the security testing content, and triggers various behaviors according to modes of the Protocol Actuator and the *Protocol* module. The *Signal Processing* module is used to receive external and internal signals, such as mode updates, protocol start, protocol stop, and behavior triggering. In the Protocol Actuator, all modules except for the *Protocol* module can run independently. However, the *Protocol* module requires users to implement module interfaces based on the logic of the BFT protocol being tested. The *Protocol* module has four key interfaces that need to be implemented:

- *Protocol Initialization Interface.* This interface requires the implementation of initialization work before protocol execution, such as key distribution and protocol parameter configuration.
- *Protocol Phases Interface.* This interface is responsible for implementing the protocol phase message structure. `BFTDiagnosis` will obtain the phase information of the protocol being tested through this interface before testing.
- *Protocol Message Handler Interface.* This interface is responsible for implementing the core logic of the BFT protocol being tested, including the processing methods for different messages during the protocol execution process.
- *Timeout Handler Interface.* This interface is responsible for implementing timeout processing logic during the protocol execution process, such as the view-change process after a timeout.

The Analyzer is composed of *Task Receiving* module, *Data Acquisition* module, *Data Storage* module and *Security Indicator Calculation* module to evaluates the security of BFT protocols. After the Analyzer is started, it receives instructions of testing tasks from the Controller through the *Task Receiving* module. During the testing process, the *Data Acquisition* module continuously receives the protocol running data from the Protocol Actuator and transmits it to the *Data Storage* module to store the protocol running data of each consensus node. In addition, the *Security Indicator Calculation* module can obtain the protocol running data of the consensus nodes from the Data Storage module and calculate two auxiliary indicators and four security indicators.

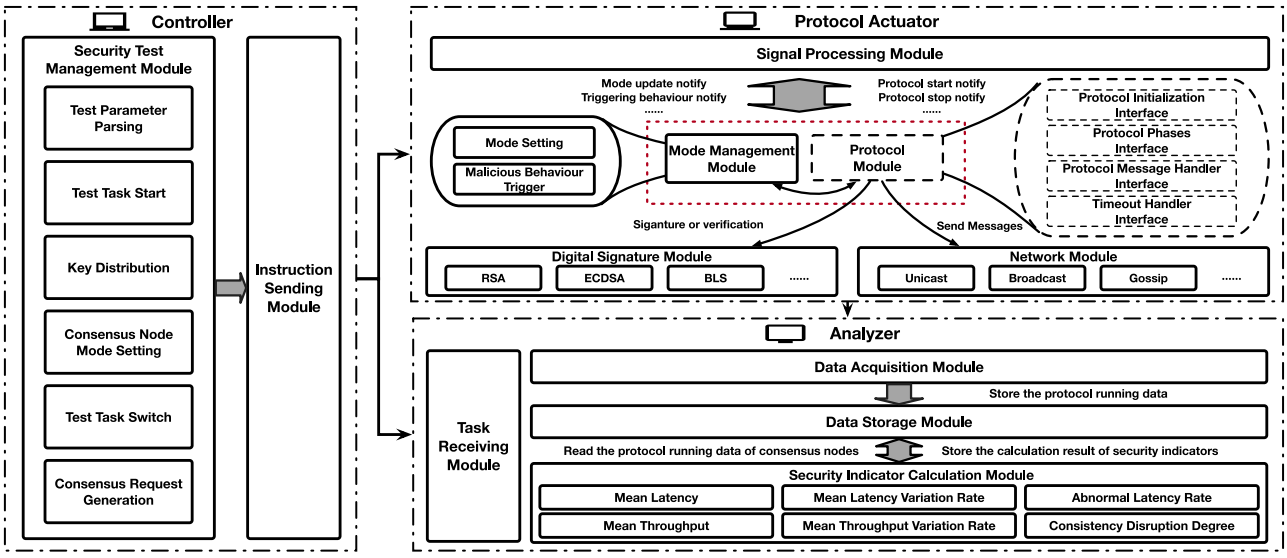


Fig. 5. The architecture of BFTDiagnosis, including the structure design of the Controller, the Protocol Actuator and Analyzer.

4. Experimental evaluation

To validate the effectiveness of BFTDiagnosis in conducting security tests on BFT protocols, we utilized the framework to test and analyze three representative BFT protocols. Additionally, we have open-sourced the BFTDiagnosis framework along with the three protocols (PBFT, Basic-HotStuff, and Chain-HotStuff), which can be accessed at https://github.com/Wangert/BFTDiagnosis.git.

4.1. Experimental setup

Hardware and software. We run consensus nodes on four CentOS Linux release 7.6.1810 (Core) servers with 128G of memory, 16 cores and 3.5T SSD storage, and we run the Controller and the Analyzer on two CentOS Linux release 7.6.1810 (Core) desktops with 16G of memory and 1T SSD storage. And, the message communication delay between the four servers running the consensus node ranges from 3 to 4 ms.

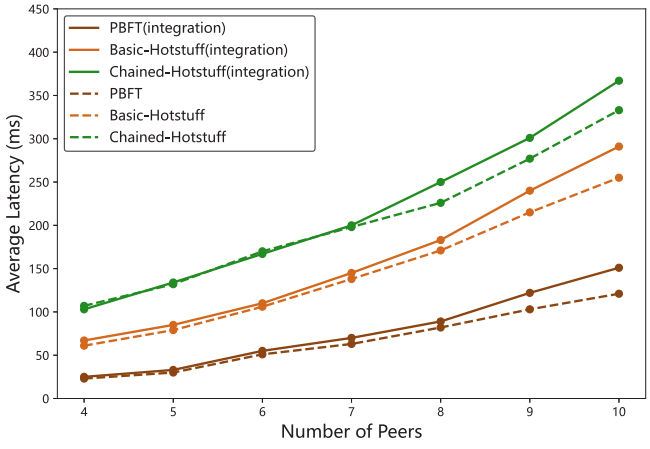
Protocol implementation. Among the numerous BFT protocols, the HotStuff series optimized from PBFT, which has linear communication complexity, is currently a hot spot in the study of consensus protocols, and the design ideas of DiemBFT and AptosBFT is derived from its structure. So we implemented three types of BFT protocols in Rust language and integrated them into the Protocol Actuator, including PBFT, Basic HotStuff, and Chained-HotStuff.

By utilizing protocol phase partitioning rules, we can divide PBFT, Basic-HotStuff, and Chained-HotStuff into distinct phases.

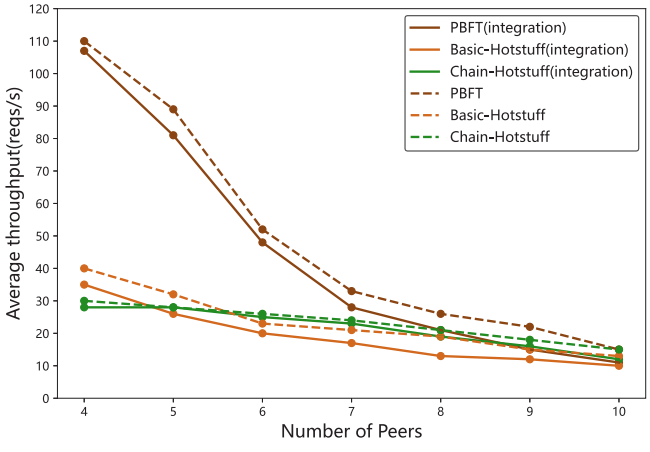
- PBFT (three phases): PrePrepare (1 phase), Prepare (2 phase), and Commit (3 phase).
- Basic-HotStuff (six phases): Prepare (1 phase), PrepareVote (2 phase), PreCommit (3 phase), PreCommitVote (4 phase), Commit (5 phase), and CommitVote (6 phase).
- Chained-HotStuff (two phases): Generic (1 phase) and GenericVote (2 phase).

4.2. Evaluation of framework overhead

We evaluate the load on the BFTDiagnosis framework from two aspects: (1) the resource consumption of consensus nodes; (2) the impact of the BFTDiagnosis framework on the latency and throughput of the consensus protocol.



(a) Comparison of latency



(b) Comparison of throughput

Fig. 6. Comparison of performance between the original protocols and the protocol implemented using BFTDiagnosis. (PBFT, Basic-HotStuff and Chained-HotStuff).

First, we run multiple consensus nodes built on BFTDiagnosis on a single server to measure the resource consumption of the consensus nodes. The results indicate that each consensus node built based on

Table 1
The testing results of Delay, Duplicate and FeignDeath Malicious behaviors on the PBFT, Basic-HotStuff and Chained-HotStuff (Leader-fault).

	PBFT (Phase Number)			Basic-Hotstuff (Phase Number)						Chained-Hotstuff (Phase Number)	
	1	2	3	1	2	3	4	5	6	1	2
Delay	**VRL(59.3)** **VRT(55.6)** **AR (62.8)** DDC(0.0)	VRL(11.1) VRT(20.9) AR (13.0) DDC(0.0)	VRL(29.4) **VRT(53.8)** AR (22.6) DDC(1.4)	VRL(14.5) VRT(21.9) AR (17.9) DDC(0.0)	VRL(1.7) VRT(13.2) AR (1.6) DDC(0.0)	VRL(18.1) VRT(20.5) AR (23.7) DDC(0.0)	VRL(4.8) VRT(9.6) AR (6.8) DDC(0.0)	VRL(18.1) VRT(20.5) AR (23.7) DDC(0.0)	VRL(4.8) VRT(9.6) AR (6.0) DDC(0.0)	**VRL(32.9)** **VRT(41.6)** **AR (84.7)** DDC(1.0)	VRL(4.3) VRT(19.8) AR (9.5) DDC(0.0)
Duplicate	VRL(7.7) VRT(19.0) AR (6.0) DDC(0.0)	VRL(7.7) VRT(18.5) AR (9.2) DDC(0.0)	VRL(7.7) VRT(19.6) AR (6.2) DDC(0.0)	VRL(1.7) VRT(12.3) AR (3.9) DDC(0.0)	VRL(1.7) VRT(9.6) AR (1.8) DDC(0.0)	VRL(1.7) VRT(9.6) AR (1.1) DDC(0.0)	VRL(3.3) VRT(8.3) AR (2.2) DDC(0.0)	VRL(3.3) VRT(12.3) AR (3.8) DDC(0.0)	VRL(1.7) VRT(10.3) AR (2.3) DDC(1.5)	VRL(6.0) VRT(21.9) AR (10.4) DDC(0.0)	VRL(4.8) VRT(20.7) AR (8.3) DDC(0.0)
FeignDeath	**VRL(79.4)** **VRT(89.6)** AR (9.4) DDC(0.4)	VRL(7.1) VRT(13.2) AR (7.5) DDC(0.0)	VRL(10.3) VRT(13.0) AR (5.5) DDC(0.0)	**VRL(81.5)** **VRT(85.0)** AR (21.1) DDC(4.5)	VRL(3.6) VRT(1.0) AR (4.5) DDC(0.0)	**VRL(77.1)** **VRT(85.4)** AR (22.2) DDC(4.8)	VRL(3.6) VRT(1.1) AR (4.6) DDC(0.0)	**VRL(82.3)** **VRT(85.9)** AR (16.5) DDC(0.0)	VRL(5.4) VRT(0.2) AR (7.1) DDC(0.7)	**VRL(70.8)** **VRT(92.1)** **AR (35.1)** DDC(4.2)	VRL(3.5) VRT(18.4) AR (8.6) DDC(0.4)

VRL: Mean latency variation rate; **VRT**: Mean throughput variation rate; **AR**: Abnormal latency rate; **DDC**: Consistency disruption degree.

BFTDiagnosis occupies approximately 17% of the CPU and 10% of the memory.

Secondly, we recognize that the performance of BFT protocols may be affected by the utilization of BFTDiagnosis, which could indirectly impact the results of security testing. Thus, we conducted tests on the latency and throughput of PBFT, Basic-HotStuff, and Chained-HotStuff protocols without integration into the BFTDiagnosis framework, and compared these results to those of the same protocols integrated into the BFTDiagnosis framework.

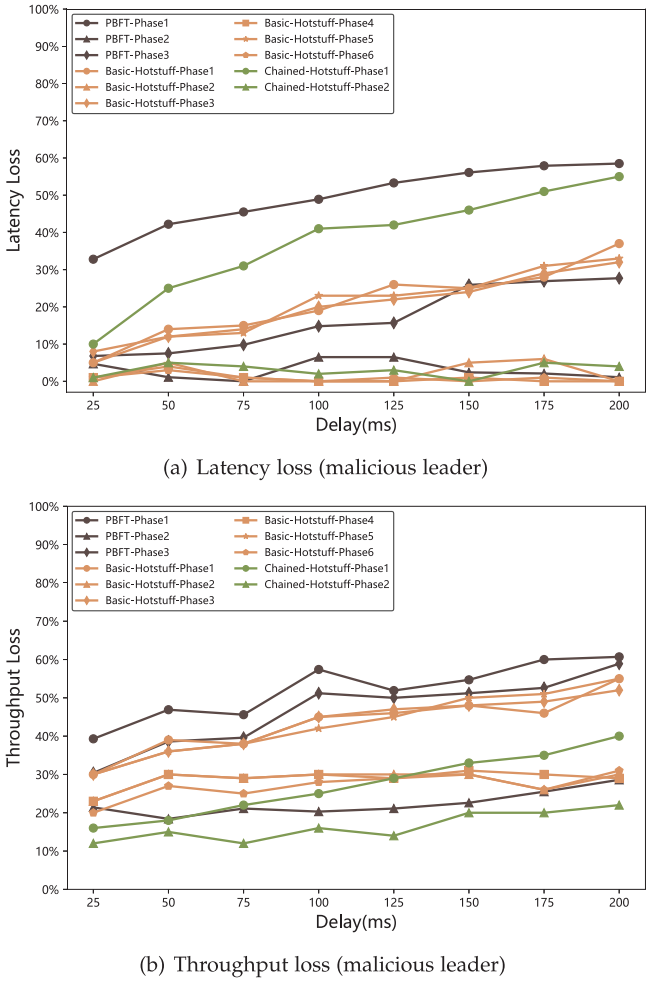
Due to the limited number of servers in our experimental environment, if the number of consensus nodes exceeds four, we evenly distribute them among the available servers. From a resource utilization perspective, this allocation scheme may affect the authenticity of the protocol performance. However, we can still compare the performance of the protocol under the same allocation scheme to effectively evaluate the degree of influence that BFTDiagnosis has on the protocol.

Fig. 6 shows that the latency and throughput of PBFT, PBFT(integration), Basic-HotStuff, Basic-HotStuff(integration), Chained-HotStuff and Chained-HotStuff(integration) protocols under 4 to 10 consensus nodes. The figure illustrates that the three BFT protocols integrated with the BFTDiagnosis framework exhibit consistent latency and throughput trends with those without integration. When the number of consensus nodes is less than 7, the latency of the three protocols differs from their corresponding integrated protocols by no more than 5 ms. However, when the number of consensus nodes is greater than or equal to 7, the difference in latency increases because BFTDiagnosis requires additional CPU resources. In our experimental environment, more than 4 consensus nodes need to share the same server, which means that CPU resources cannot provide services for each consensus node normally. In addition, the throughput difference between the three protocols and their corresponding integrated protocols remains between 5 and 10. Considering the network latency in our experimental environment, the performance overhead of BFTDiagnosis on the BFT protocols is acceptable.

4.3. Security testing results

To ensure that consensus nodes have sufficient CPU resources, we execute security tests on PBFT, Basic-HotStuff, and Chained-HotStuff with four consensus nodes. Furthermore, we run the four consensus nodes on separate servers (128G of memory, 16 cores and 3.5T SSD storage) to avoid contention for system resources among the nodes. Below, we analyze the testing results of BFTDiagnosis from two perspectives: leader-fault and replica-fault.

Malicious scenarios of leader-fault. We set the attack intensity of malicious behavior to trigger every 10 ms to simulate the scenario of Leader-fault for the three protocols. Table 1 shows the results of the four security indicators when the Leader initiates the FeignDeath



(a) Latency loss (malicious leader)

(b) Throughput loss (malicious leader)

Fig. 7. Latency and throughput losses when there is a malicious leader. (PBFT, Basic-HotStuff and Chained-HotStuff)

malicious behavior, Duplicate, and Delay. When Leader initiates the FeignDeath behavior in PBFT, Basic-HotStuff, and Chained-HotStuff during odd phases, it leads to a significant increase in VRL and VRT indicators, and slightly higher AR indicators than normal levels. This is because the leader's inactivity during odd-numbered phases leads to view-change and re-consensus on requests. However, this malicious behavior has a small impact on the DDC indicator. When the Leader executes a Delay (50 ms) malicious behavior, it has a similar impact

Table 2
The testing results of Ambiguous malicious behavior on the PBFT (Leader-fault).

PBFT							
	LeaderAmbiguous (number of nodes)				LeaderAmbiguous (number of nodes)		
	1	2	3		1	2	3
(1,view)	VRL(5.7) VRT(10.7) AR (5.4) DDC(0.4)	**VRL(77.3) VRT(90.3)** AR (9.0) DDC(0.4)	**VRL(85.0) VRT(89.7)** AR (5.4) DDC(0.0)	(2,signature)	VRL(17.5) VRT(15.0) AR (7.7) DDC(1.7)	VRL(17.5) VRT(16.1) AR (8.1) DDC(0.4)	VRL(19.5) VRT(20.4) AR (11.2) DDC(0.0)
(1,m_hash)	VRL(9.4) VRT(13.9) AR (8.3) DDC(4.3)	**VRL(77.4) VRT(91.0)** AR (5.6) DDC(0.5)	**VRL(85.9) VRT(90.9)** AR (14.1) DDC(0.0)	(3,view)	VRL(13.2) VRT(24.0) AR (16.6) DDC(0.0)	VRL(13.2) VRT(23.8) AR (15.8) DDC(0.5)	VRL(17.5) VRT(32.0) AR (19.2) DDC(0.0)
(1,signature)	VRL(8.3) VRT(15.0) AR (8.3) DDC(0.0)	**VRL(77.2) VRT(97.8)** AR (11.4) DDC(0.0)	**VRL(87.8) VRT(90.4)** AR (13.6) DDC(0.0)	(3,m_hash)	VRL(18.8) VRT(23.7) AR (16.1) DDC(0.5)	VRL(16.1) VRT(22.5) AR (12.0) DDC(0.0)	VRL(23.5) VRT(28.0) AR (20.7) DDC(0.0)
(2,view)	VRL(15.4) VRT(15.5) AR (11.4) DDC(0.9)	VRL(15.4) VRT(14.7) AR (11.0) DDC(0.9)	VRL(19.5) VRT(13.5) AR (10.2) DDC(0.4)	(3,signature)	VRL(18.8) VRT(23.5) AR (17.9) DDC(0.5)	VRL(18.8) VRT(24.0) AR (13.2) DDC(0.0)	VRL(18.8) VRT(23.6) AR (17.8) DDC(1.0)
(2,m_hash)	VRL(17.5) VRT(19.7) AR (9.2) DDC(0.5)	VRL(17.5) VRT(18.5) AR (11.4) DDC(1.4)	VRL(17.5) VRT(18.9) AR (9.6) DDC(0.0)				

VRL: Mean latency variation rate; **VRT**: Mean throughput variation rate; **AR**: Abnormal latency rate; **DDC**: Consistency disruption degree; **(1, A)**: indicates that field A of phase 1 is forged.

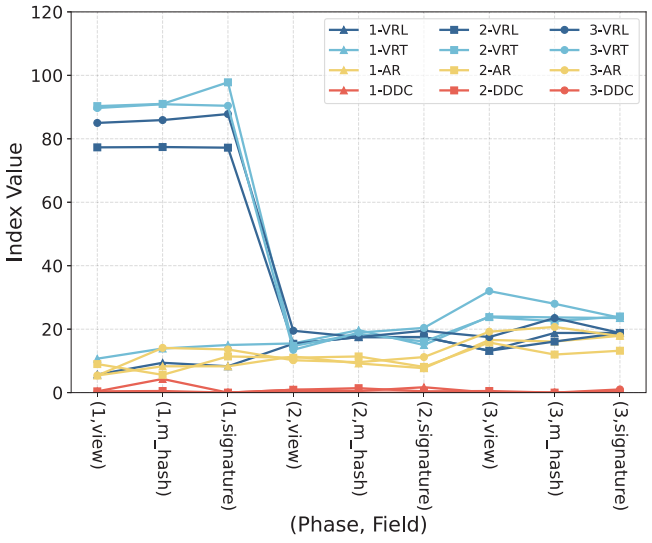


Fig. 8. The values of security indexes for the PBFT protocol under the scenario of "Ambiguous" malicious behavior triggered by leader.

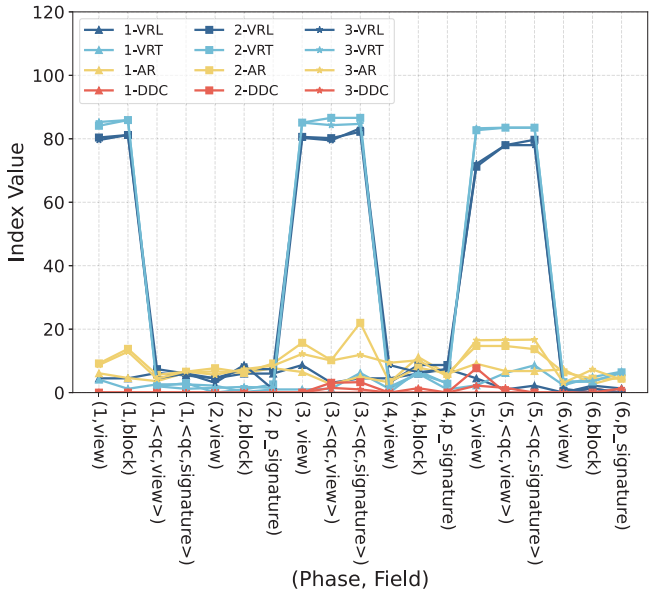


Fig. 9. The values of security indexes for the Basic-HotStuff protocol under the scenario of "Ambiguous" malicious behavior triggered by leader.

on the odd phases of the three protocols as the FeignDeath malicious behavior, but the impact is smaller since a 50 ms delay may not trigger a view change. However, in PBFT, the Leader's delay malicious behavior has a more significant impact in the first phase. According to our analysis, the Leader's delay in the first phase greatly affects the message processing time of other Replicas, while in the third phase, it only affects its own Commit time. In addition, the Duplicate malicious behavior of Leader has little impact on the security indicators of each protocol.

To demonstrate more clearly the impact of leader latency malicious behavior on the three protocols, we conducted tests under different latency conditions. As depicted in Fig. 7, as the delay increases, the latency and throughput loss of odd phases of the three protocols caused by Delay malicious behavior also increase significantly. This effect is particularly pronounced in the first phase of PBFT and Chained-HotStuff, highlighting the importance of the leader's message processing time for these protocols.

Table 2 shows the experimental results of Ambiguous (Leader) malicious behavior on PBFT. When the Leader sends a fake message to one Replica, any forgery of any field in any phase will not have a significant impact on its four indicators. When the Leader sends fake messages to two Replicas, forgery of any field in phase 1 will result in a significant increase in VRL and VRT indicators, but will not have a significant impact in phases 2 and 3. When the Leader sends fake messages to three Replicas, forgery of any field in phase 1 will result in a significant increase in VRL and VRT indicators, but will not have a significant impact in phases 2 and 3. These contrasts are clearly displayed in Fig. 8. These malicious scenarios will not undermine the consistency of PBFT (i.e., the DDC indicator is not affected).

Table 3 shows the impact of Leader Ambiguous malicious behavior on Basic HotStuff, which, like other leader malicious behaviors, only affects the odd phases. When forging the fields as *view* and *block*, the VRL, VRT, and AR indicators are relatively high. When forging the fields as $\langle qc, view\rangle$ and $\langle qc, signature\rangle$, only the VRL, VRT, and AR of phases 3 and 5 are relatively high, while the VRL, VRT, and AR of phase 1 are relatively low. This situation is demonstrated in Fig. 9.

Table 4 shows that injecting Leader Ambiguous malicious behavior into phase 1 of Chained-HotStuff can have a huge impact on VRT metrics. In Fig. 10, it is evident that in phase 1, both VRT and VRL are notably higher than in phase 2.

In summary, the malicious behaviors of a leader has a significant impact on the VRL, VRT, and AR security indicators of the odd phases of the three protocols, but has almost no effect on DDC. Moreover, in the case of Ambiguous malicious behavior, maliciously forging a few fields in some phases may not pose a threat to the security of three protocols.

Table 3
The testing results of Ambiguous malicious behavior on the Basic-HotStuff (Leader-fault).

Basic-Hotstuff							
	LeaderAmbiguous (number of nodes)				LeaderAmbiguous (number of nodes)		
	1	2	3		1	2	3
(1,view)	VRL(4.5) VRT(4.1) AR (6.1) DDC(0.0)	**VRL(80.4) VRT(84.1)** AR (9.2) DDC(0.0)	**VRL(79.6) VRT(85.3)** AR (8.7) DDC(0.0)	(4,view)	VRL(4.5) VRT(1.2) AR (3.2) DDC(0.0)	VRL(3.1) VRT(0.0) AR (3.4) DDC(0.0)	VRL(8.7) VRT(1.8) AR (9.4) DDC(0.0)
(1,block)	VRL(4.5) VRT(1.2) AR (4.6) DDC(0.0)	**VRL(81.2) VRT(85.9)** AR (13.8) DDC(0.0)	**VRL(81.2) VRT(85.9)** AR (12.9) DDC(0.0)	(4,block)	VRL(6.0) VRT(6.0) AR (11.2) DDC(0.0)	VRL(8.7) VRT(6.8) AR (7.9) DDC(0.0)	VRL(6.4) VRT(6.0) AR (10.2) DDC(1.4)
(1,<qc,view>)	VRL(6.2) VRT(2.6) AR (3.6) DDC(0.0)	VRL(7.4) VRT(2.0) AR (5.6) DDC(0.0)	VRL(4.5) VRT(2.0) AR (4.4) DDC(0.0)	(4,p_signature)	VRL(7.4) VRT(1.0) AR (5.9) DDC(0.0)	VRL(8.7) VRT(2.8) AR (6.0) DDC(0.0)	VRL(7.4) VRT(3.0) AR (5.1) DDC(0.0)
(1,<qc,signature>)	VRL(5.4) VRT(2.6) AR (6.3) DDC(0.0)	VRL(6.0) VRT(3.0) AR (6.6) DDC(0.0)	VRL(6.0) VRT(1.2) AR (6.7) DDC(0.0)	(5,view)	VRL(4.5) VRT(2.3) AR (9.1) DDC(0.0)	**VRL(71.2) VRT(82.7)** AR (14.7) DDC(7.7)	**VRL(72.1) VRT(83.3)** AR (16.5) DDC(2.2)
(2,view)	VRL(4.5) VRT(2.2) AR (6.7) DDC(0.0)	VRL(4.5) VRT(0.0) AR (7.7) DDC(0.0)	VRL(3.1) VRT(1.5) AR (5.6) DDC(0.0)	(5,<qc,view>)	VRL(1.0) VRT(6.2) AR (6.8) DDC(0.0)	**VRL(78.0) VRT(83.5)** AR (14.7) DDC(0.0)	**VRL(78.0) VRT(83.5)** AR (16.6) DDC(1.5)
(2,block)	VRL(6.0) VRT(0.2) AR (6.2) DDC(0.0)	VRL(7.4) VRT(1.0) AR (6.3) DDC(0.0)	VRL(8.7) VRT(1.8) AR (7.4) DDC(0.0)	(5,<qc,signature>)	VRL(2.2) VRT(8.6) AR (6.8) DDC(0.0)	**VRL(79.7) VRT(83.5)** AR (13.7) DDC(0.0)	**VRL(78.0) VRT(83.5)** AR (16.7) DDC(0.0)
(2,p_signature)	VRL(6.0) VRT(1.0) AR (7.7) DDC(0.0)	VRL(7.4) VRT(2.6) AR (9.2) DDC(0.0)	VRL(1.0) VRT(1.5) AR (8.5) DDC(0.0)	(6,view)	VRL(0.0) VRT(2.3) AR (7.3) DDC(0.0)	VRL(0.5) VRT(3.3) AR (6.4) DDC(0.0)	VRL(1.2) VRT(3.5) AR (3.1) DDC(0.0)
(3,view)	VRL(8.7) VRT(1.0) AR (6.5) DDC(0.0)	**VRL(80.6) VRT(85.1)** AR (15.7) DDC(0.0)	**VRL(80.3) VRT(85.1)** AR (12.2) DDC(0.0)	(6,block)	VRL(2.2) VRT(4.9) AR (2.7) DDC(1.3)	VRL(1.3) VRT(3.5) AR (4.4) DDC(0.0)	VRL(0.0) VRT(3.6) AR (7.3) DDC(0.0)
(3,<qc,view>)	VRL(3.1) VRT(1.2) AR (2.6) DDC(0.0)	**VRL(80.2) VRT(86.6)** AR (10.2) DDC(3.1)	**VRL(79.5) VRT(84.3)** AR (10.1) DDC(1.5)	(6,p_signature)	VRL(1.3) VRT(6.6) AR (5.1) DDC(0.0)	VRL(0.0) VRT(6.4) AR (4.5) DDC(0.0)	VRL(0.0) VRT(6.2) AR (4.1) DDC(1.3)
(3,<qc,signature>)	VRL(4.5) VRT(6.2) AR (5.0) DDC(0.0)	**VRL(82.2) VRT(86.6)** AR (22.0) DDC(3.3)	**VRL(83.3) VRT(84.7)** AR (11.9) DDC(1.0)				

VRL: Mean latency variation rate; **VRT**: Mean throughput variation rate; **AR**: Abnormal latency rate; **DDC**: Consistency disruption degree; **(1, A)**: indicates that field A of phase 1 is forged.

Table 4
The testing results of Ambiguous malicious behavior on the Chained-HotStuff (Leader-fault).

Chained-Hotstuff							
	LeaderAmbiguous (number of replicas)				LeaderAmbiguous (number of nodes)		
	1	2	3		1	2	3
(1,view)	VRL(22.1) **VRT(89.4)** AR (3.4) DDC(4.5)	VRL(4.5) **VRT(89.8)** AR (10.8) DDC(18.2)	VRL(18.2) **VRT(90.4)** AR (10.2) DDC(5.0)	(2,view)	VRL(2.8) VRT(11.9) AR (6.3) DDC(0.0)	VRL(2.1) VRT(15.9) AR (7.7) DDC(0.0)	VRL(4.1) VRT(14.9) AR (9.6) DDC(0.6)
(1,block)	VRL(0.0) **VRT(88.9)** AR (3.3) DDC(3.6)	**VRL(32.2) VRT(89.9)** AR (12.1) DDC(4.3)	**VRL(45.5) VRT(90.1)** AR (12.5) DDC(5.5)	(2,block)	VRL(7.9) VRT(10.8) AR (14.6) DDC(0.0)	VRL(7.3) VRT(16.3) AR (11.8) DDC(0.0)	VRL(4.4) VRT(16.3) AR (8.6) DDC(0.0)
(1,<qc,view>)	VRL(28.2) **VRT(89.5)** AR (5.0) DDC(0.0)	**VRL(68.9) VRT(90.1)** AR (15.0) DDC(4.5)	**VRL(55.1) VRT(89.4)** AR (15.8) DDC(0.0)	(2,p_signature)	VRL(2.1) VRT(12.3) AR (5.1) DDC(0.6)	VRL(3.4) VRT(11.8) AR (7.3) DDC(0.0)	VRL(12.5) VRT(18.2) AR (10.0) DDC(0.0)
(1,<qc,signature>)	**VRL(38.1) VRT(87.7)** AR (11.1) DDC(9.1)	**VRL(68.3) VRT(88.2)** AR (20.0) DDC(15.4)	**VRL(53.8) VRT(89.5)** AR (20.4) DDC(16.0)				

VRL: Mean latency variation rate; **VRT**: Mean throughput variation rate; **AR**: Abnormal latency rate; **DDC**: Consistency disruption degree; **(1, A)**: indicates that field A of phase 1 is forged.

Table 5
The testing results of the impact of Delay, Duplicate and FeignDeath malicious behaviors on the PBFT (Replica-fault).

	PBFT								
	1 malicious replica (Phase number)			2 malicious replica (Phase number)			3 malicious replica (Phase number)		
	1	2	3	1	2	3	1	2	3
Delay (50ms)	VRL(0.0) VRT(23.4) AR (4.6) DDC(0.0)	VRL(0.0) VRT(21.7) AR (2.4) DDC(1.2)	VRL(4.0) VRT(25.8) AR (7.3) DDC(0.0)	VRL(4.0) VRT(24.3) AR (4.3) DDC(0.0)	**VRL(48.9) VRT(53.7) AR (44.8)** DDC(0.0)	**VRL(54.7) VRT(53.7) AR (37.7)** DDC(0.0)	VRL(4.7) VRT(27.4) AR (4.5) DDC(1.3)	**VRL(46.7) VRT(52.2) AR (43.0)** DDC(0.0)	**VRL(55.0) VRT(54.6) AR (48.4)** DDC(0.0)
Duplicate	VRL(4.2) VRT(28.3) AR (6.8) DDC(1.3)	VRL(4.2) VRT(22.4) AR (5.1) DDC(0.0)	VRL(4.2) VRT(21.4) AR (6.2) DDC(0.0)	VRL(4.2) VRT(24.3) AR (6.1) DDC(3.7)	VRL(4.2) VRT(26.4) AR (6.3) DDC(0.0)	VRL(6.0) VRT(27.1) AR (9.5) DDC(0.0)	VRL(11.9) VRT(31.4) AR (13.5) DDC(0.0)	VRL(8.0) VRT(25.5) AR (12.6) DDC(1.2)	VRL(11.5) VRT(29.7) AR (14.5) DDC(0.0)
FeignDeath	VRL(0.0) VRT(2.3) AR (2.3) DDC(0.4)	VRL(12.2) VRT(22.4) AR (6.1) DDC(1.4)	VRL(0.0) VRT(11.2) AR (2.7) DDC(2.7)	VRL(7.1) VRT(12.1) AR (9.6) DDC(0.8)	**C**	**C**	VRL(0.0) VRT(10.3) AR (6.3) DDC(0.4)	**C**	**C**

VRL: Mean latency variation rate; **VRT**: Mean throughput variation rate; **AR**: Abnormal latency rate; **DDC**: Consistency disruption degree; **C**: indicates that the system crashes.

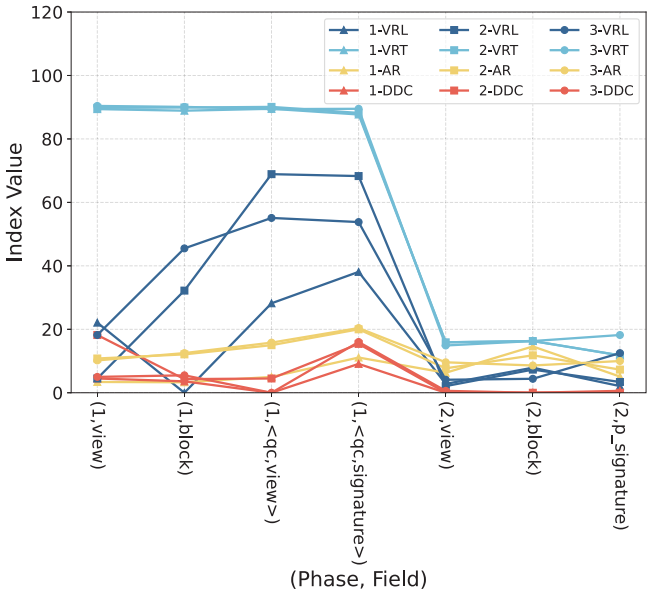


Fig. 10. The values of security indexes for the Chained-HotStuff protocol under the scenario of "Ambiguous" malicious behavior triggered by leader.

Malicious scenarios of replica-fault. We set the attack intensity of malicious behavior to trigger every 10 ms to simulate the scenario of Replica-fault for three protocols.

Table 5 shows that the experimental results of the malicious behaviors of Delay (Replica), Duplicate (Replica), and FeignDeath (Replica) on PBFT. For the Delay malicious behavior, injecting the behavior in phases 2 and 3 of PBFT will increase the VRL, VRT, and AR indicators when the number of malicious Replicas in the PBFT protocol is greater than or equal to 2; while it remains normal in all three phases when the number of malicious Replicas in the PBFT protocol is 1. The Duplicate malicious behavior has no effect on PBFT in any scenario. For the FeignDeath malicious behavior, injecting the behavior in phases 2 and 3 of PBFT will cause system crashes when the number of malicious Replicas in the PBFT protocol is greater than or equal to 2; while it remains normal in all three phases when the number of malicious Replicas in the PBFT protocol is 1. In Fig. 11(a), it is evident that under duplicate behavior, these values remain stable. However, for Delay and FeignDeath behaviors, the VRL, VRT, and AR values show a notable increase in phases 2 and 3 after malicious replicas reach 2 or more. This indicates that these malicious scenarios will affect the normal execution of PBFT.

Table 6 shows that the experimental results of the malicious behaviors of Delay (Replica), Duplicate (Replica), and FeignDeath (Replica) on Basic-HotStuff. For the Delay malicious behavior, injecting the behavior in phases 2, 4, and 6 of Basic-HotStuff will increase the VRL, VRT, and AR indicators when the number of malicious Replicas in the Basic-HotStuff protocol is greater than or equal to 2; while it remains normal in all six phases when the number of malicious Replicas in the Basic-HotStuff protocol is 1. The Duplicate malicious behavior has no effect on Basic-HotStuff in any scenario. For the FeignDeath malicious behavior, injecting the behavior in phases 2, 4, and 6 of the Basic-HotStuff protocol will cause system crashes when the number of malicious Replicas in the Basic-HotStuff protocol is greater than or equal to 2; while it remains normal in all six phases when the number of malicious Replicas in Basic-HotStuff is 1. In Fig. 11(b), it is evident that under the duplicate behavior, these values remain stable. However, under the Delay and FeignDeath behaviors, there is a significant impact on phases 2, 4, and 6 when the number of malicious replicas is greater than or equal to 2.

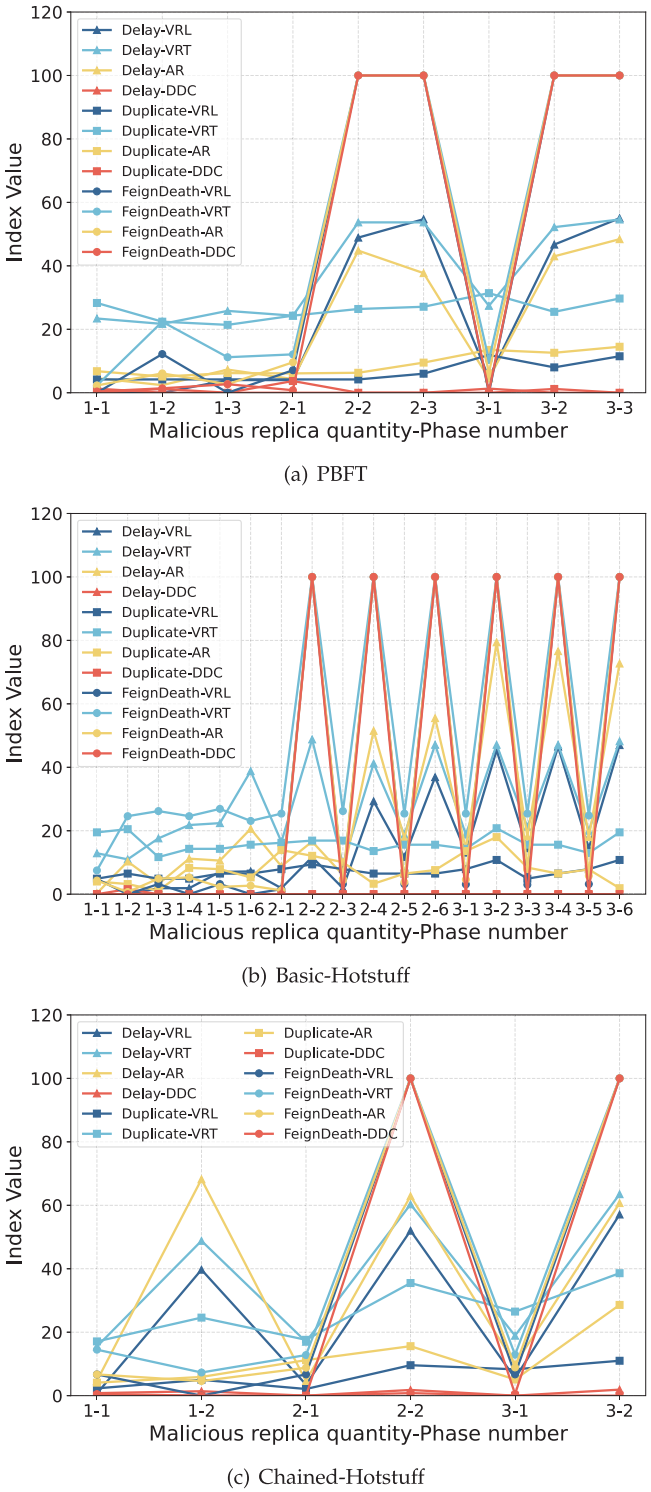


Fig. 11. The values of security indexes for three protocols under the scenarios of "Delay", "Duplicate", and "FeignDeath" by replica.

Table 7 shows the experimental results of malicious behaviors Delay (Replica), Duplicate (Replica), and FeignDeath (Replica) on Chained-HotStuff. Regarding Delay behavior, when the number of malicious Replicas in the Chained-HotStuff protocol is greater than or equal to 2, injecting malicious behavior in its phase 2 will increase the values of VRL, VRT, and AR indicators. Interestingly, when the number of malicious Replicas in the Basic-HotStuff protocol is 1, it can be observed that injecting malicious behavior in its phase 2 also increases

Table 6
The testing results of the impact of Delay, Duplicate and FeignDeath malicious behaviors on the Basic-HotStuff (Replica-fault).

	Basic-Hotstuff					
	1 malicious replica (Phase number)		2 malicious replica (Phase number)		3 malicious replica (Phase number)	
	1	2	1	2	1	2
Delay (50ms)	VRL(0.0) VRT(12.9) AR (0.3) DDC(0.0)	VRL(0.0) VRT(11.0) AR (10.3) DDC(0.0)	VRL(1.9) VRT(16.5) AR (8.8) DDC(0.0)	VRL(11.7) **VRT(48.8)** AR (16.6) DDC(0.0)	VRL(13.1) VRT(18.8) AR (14.4) DDC(0.0)	**VRL(45.4) VRT(47.1) AR (79.4)** DDC(0.0)
Duplicate	VRL(4.9) VRT(19.5) AR (4.0) DDC(0.0)	VRL(6.5) VRT(20.5) AR (3.2) DDC(1.6)	VRL(7.9) VRT(16.2) AR (13.9) DDC(0.0)	VRL(9.4) VRT(16.9) AR (12.1) DDC(0.0)	VRL(7.9) VRT(14.3) AR (13.6) DDC(0.0)	VRL(10.8) VRT(20.8) AR (18.0) DDC(0.0)
FeignDeath	VRL(4.8) VRT(7.5) AR (4.3) DDC(0.0)	VRL(0.0) VRT(24.6) AR (0.7) DDC(0.0)	VRL(1.6) VRT(25.4) AR (1.2) DDC(0.0)	**C**	VRL(3.1) VRT(25.4) AR (4.6) DDC(0.0)	**C**
	3	4	3	4	3	4
Delay (50ms)	VRL(1.9) VRT(17.6) AR (3.2) DDC(0.0)	VRL(1.9) VRT(21.8) AR (11.2) DDC(0.0)	VRL(1.9) VRT(9.4) AR (6.1) DDC(0.0)	**VRL(29.3) VRT(41.2) AR (51.5)** DDC(0.0)	VRL(14.5) VRT(21.2) AR (18.2) DDC(0.0)	**VRL(46.5) VRT(47.1) AR (76.6)** DDC(0.0)
Duplicate	VRL(4.9) VRT(11.7) AR (1.1) DDC(0.0)	VRL(4.9) VRT(14.3) AR (8.3) DDC(0.0)	VRL(7.9) VRT(16.9) AR (10.1) DDC(0.0)	VRL(6.5) VRT(13.6) AR (3.3) DDC(0.0)	VRL(4.9) VRT(15.6) AR (8.4) DDC(0.0)	VRL(6.5) VRT(15.6) AR (6.5) DDC(0.0)
FeignDeath	VRL(3.2) VRT(26.2) AR (4.9) DDC(0.0)	VRL(0.0) VRT(24.6) AR (5.3) DDC(0.0)	VRL(3.2) VRT(26.2) AR (4.4) DDC(0.0)	**C**	VRL(3.0) VRT(25.4) AR (7.9) DDC(0.0)	**C**
	5	6	5	6	5	6
Delay (50ms)	VRL(7.0) VRT(22.4) AR (10.6) DDC(0.0)	VRL(7.2) **VRT(38.8)** AR (20.6) DDC(0.0)	VRL(11.7) VRT(18.8) AR (17.7) DDC(0.0)	**VRL(36.9) VRT(47.1) AR (55.5)** DDC(0.0)	VRL(14.5) VRT(20.0) AR (18.0) DDC(0.0)	**VRL(47.0) VRT(48.2) AR (72.7)** DDC(0.0)
Duplicate	VRL(6.5) VRT(14.3) AR (7.9) DDC(0.0)	VRL(6.5) VRT(15.6) AR (5.3) DDC(0.0)	VRL(6.5) VRT(15.6) AR (6.5) DDC(0.0)	VRL(6.5) VRT(15.6) AR (7.6) DDC(0.0)	VRL(7.9) VRT(13.0) AR (7.8) DDC(0.0)	VRL(10.8) VRT(19.5) AR (1.9) DDC(0.0)
FeignDeath	VRL(3.2) VRT(26.9) AR (2.3) DDC(0.0)	VRL(0.0) VRT(23.1) AR (2.7) DDC(0.0)	VRL(3.2) VRT(25.4) AR (3.8) DDC(0.0)	**C**	VRL(3.2) VRT(24.8) AR (5.8) DDC(0.0)	**C**

VRL: Mean latency variation rate; **VRT**: Mean throughput variation rate; **AR**: Abnormal latency rate; **DDC**: Consistency disruption degree; **C**: indicates that the system crashes.

the values of VRL, VRT, and AR indicators in Fig. 11(c). Through our in-depth analysis, due to the high-frequency View-Change process of Chained-HotStuff, the probability of malicious Replica being selected as the Leader is high. When this Replica is selected as the Leader of the next round, it may still be performing Delay behavior, thus blocking the execution of the next round of consensus. Additionally, other scenarios of Delay behavior remain normal in Chained-HotStuff. Regarding Duplicate behavior, any malicious scenario has no effect on Chained-HotStuff. For FeignDeath behavior, when the number of malicious Replicas in the Chained-HotStuff protocol is greater than or equal to 2, injecting malicious behavior in its phase 2 will result in system crash. However, when the number of malicious Replicas in the Chained-HotStuff protocol is 1, it remains normal in its phase 1. In these malicious scenarios, the DDC indicator of Chained-HotStuff is not affected abnormally.

Table 8 shows the experimental results of Conspire (Replica) malicious behavior on PBFT. When there is only 1 malicious replica in the PBFT protocol, it does not have a significant impact on its 4 indicators. When there are 2 malicious replicas in the PBFT protocol, conspiring to forge any field of messages in phases 2 and 3 will cause the system to crash, while having no significant impact in phase 1. When there are 3 malicious replicas in the PBFT protocol, the DDC indicator will increase significantly in phases 2 and 3, but it will not cause the system to crash because the 3 malicious replicas reached consensus and dominated the consensus result of the protocol. The aforementioned phenomenon is more intuitively demonstrated in Fig. 12.

Table 9 shows the experimental results of Conspire (Replica) malicious behavior on Basic-HotStuff. When there is 1 malicious replica in the Basic-HotStuff protocol, conspiring to forge messages on the view, block, $\langle qc, view\rangle$, and $\langle qc, signature\rangle$ fields in phases 1, 3, and 5 will not have a significant impact, while there will be a small increase in VRL, VRT, and AR indicators in phases 2, 4, and 6. When there are 2 malicious replicas in the Basic-HotStuff protocol, it will cause the system to crash in phases 2 and 3, but it will not have a significant impact in phase 1. When there are 3 malicious replicas in the Basic-HotStuff protocol, it will cause the system to crash in phases 2 and

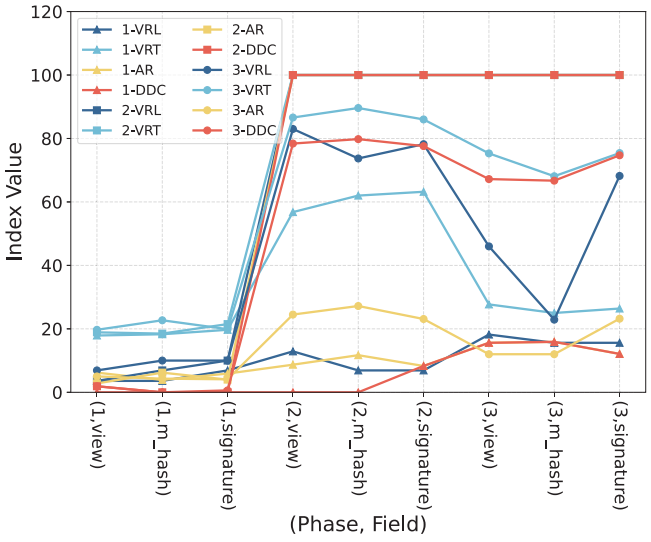


Fig. 12. The values of security indexes for the PBFT protocol under the scenarios of "Conspire" by replica.

3, but it will not have a significant impact in phase 1. Compared to PBFT, these malicious scenarios will not disrupt the consistency of Basic-HotStuff (i.e., DDC indicators are not affected). The significant impact of phase 2 and 3 when malicious replicas are greater than or equal to 2 is clearly illustrated in Fig. 13(a).

Table 10 shows the experimental results of Conspire (Replica) malicious behavior on Chained-HotStuff protocol. When there is 1 malicious Replica in the Chained-HotStuff protocol, colluding forgery of the view, block, $\langle qc, view\rangle$, and $\langle qc, signature\rangle$ fields in phase 1 will not have a significant impact on it, while there will be a slight increase in VRL, VRT, and AR indicators in phase 2. When there are 2 malicious Replicas in the Chained-HotStuff protocol, the system will crash in phase 2,

Table 7
The testing results of the impact of Delay, Duplicate and FeignDeath malicious behaviors on the Chained-HotStuff (Replica-fault).

	Chained-Hotstuff					
	1 malicious replica (Phase number)		2 malicious replica (Phase number)		3 malicious replica (Phase number)	
	1	2	1	2	1	2
Delay (50ms)	VRL(0.7) VRT(15.2) AR (4.7) DDC(0.8)	**VRL(39.7) VRT(48.8) AR (68.2)** DDC(1.4)	VRL(3.5) VRT(17.0) AR (4.6) DDC(0.0)	**VRL(52.7) VRT(60.3) AR (62.9)** DDC(1.8)	VRL(5.5) VRT(18.9) AR (9.9) DDC(0.0)	**VRL(57.1) VRT(63.5) AR (60.7)** DDC(1.9)
Duplicate	VRL(2.3) VRT(17.1) AR (4.1) DDC(0.0)	VRL(5.0) VRT(24.6) AR (5.9) DDC(0.0)	VRL(2.1) VRT(17.6) AR (11.2) DDC(0.0)	VRL(9.6) VRT(35.5) AR (15.6) DDC(0.8)	VRL(8.3) VRT(26.5) AR (5.1) DDC(0.0)	VRL(11.0) **VRT(38.6) AR (28.6)** DDC(0.0)
FeignDeath	VRL(6.7) VRT(14.5) AR (6.7) DDC(0.5)	VRL(0.0) VRT(7.3) AR (4.6) DDC(0.0)	VRL(6.7) VRT(12.8) AR (8.8) DDC(0.0)	**C**	VRL(6.7) VRT(12.9) AR (8.9) DDC(0.5)	**C**

VRL: Mean latency variation rate; **VRT**: Mean throughput variation rate; **AR**: Abnormal latency rate; **DDC**: Consistency disruption degree; **C**: indicates that the system crashes.

Table 8
The testing results of Conspire malicious behavior on the PBFT (Replica-fault).

	PBFT						
	ReplicaConspire (number of replicas)				ReplicaConspire (number of replicas)		
	1	2	3		1	2	3
(1,view)	VRL(3.6) VRT(17.9) AR(6.2) DDC(0.0)	VRL(3.6) VRT(18.9) AR(2.8) DDC(1.9)	VRL(6.9) VRT(19.7) AR(5.0) DDC(1.9)	**(2,signature)**	VRL(6.9) **VRT(63.2)** AR(8.3) DDC(4.6)	**C**	**VRL(78.2) VRT(86.2) AR(23.1) DDC(77.6)**
(1,m hash)	VRL(3.6) VRT(18.3) AR(3.9) DDC(0.0)	VRL(6.9) VRT(18.5) AR(6.2) DDC(0.0)	VRL(10.0) VRT(22.7) AR(4.3) DDC(0.0)	**(3,view)**	VRL(18.2) VRT(27.7) AR(15.6) DDC(2.4)	**C**	**VRL(46.0) VRT(75.3)** AR(12.0) **DDC(67.2)**
(1,signature)	VRL(6.9) VRT(19.7) AR(5.9) DDC(0.0)	VRL(10.0) VRT(21.5) AR(4.0) DDC(0.0)	VRL(10.0) VRT(19.9) AR(4.1) DDC(0.6)	**(3,m hash)**	VRL(15.6) VRT(25.0) AR(15.9) DDC(3.8)	**C**	VRL(22.9) **VRT(68.1)** AR(12.0) **DDC(66.7)**
(2,view)	VRL(12.9) VRT(56.8) AR(8.7) DDC(0.0)	**C**	**VRL(83.0) VRT(86.6) AR(24.5) DDC(78.4)**	**(3,signature)**	VRL(15.6) VRT(26.4) AR(12.1) DDC(3.9)	**C**	**VRL(68.2) VRT(75.4) AR(23.2) DDC(74.7)**
(2,m hash)	VRL(6.9) **VRT(62.0)** AR(11.7) DDC(0.0)	**C**	**VRL(73.7) VRT(89.6) AR(27.2) DDC(79.8)**				

VRL: Mean latency variation rate; **VRT**: Mean throughput variation rate; **AR**: Abnormal latency rate; **DDC**: Consistency disruption degree; **C**: indicates that the system crashes; **(1, A)**: indicates that field A of phase 1 is forged.

Table 9
The testing results of Conspire malicious behavior on the Basic-HotStuff (Replica-fault).

	Basic-Hotstuff						
	ReplicaConspire (number of malicious replicas)				ReplicaConspire (number of malicious replicas)		
	1	2	3		1	2	3
(1,view)	VRL(1.6) VRT(16.2) AR (6.0) DDC(0.0)	VRL(1.6) VRT(17.2) AR (7.0) DDC(0.0)	VRL(2.0) VRT(18.3) AR (5.8) DDC(0.0)	**(4,view)**	VRL(6.2) VRT(21.5) AR (13.8) DDC(0.0)	C	C
(1,block)	VRL(1.6) VRT(18.3) AR (7.1) DDC(0.0)	VRL(3.2) VRT(20.4) AR (5.0) DDC(0.0)	VRL(3.2) VRT(17.4) AR (8.2) DDC(0.0)	**(4,block)**	VRL(7.7) VRT(23.6) AR (17.1) DDC(0.0)	C	C
(1,<qc,view>)	VRL(0.0) VRT(15.1) AR (3.4) DDC(0.0)	VRL(1.6) VRT(17.2) AR (4.8) DDC(0.0)	VRL(6.2) VRT(22.5) AR (6.6) DDC(0.0)	**(4,p_signature)**	VRL(9.1) VRT(23.6) AR (17.3) DDC(0.0)	C	C
(1,<qc,signature>)	VRL(1.6) VRT(17.2) AR (7.3) DDC(0.0)	VRL(1.6) VRT(17.0) AR (3.5) DDC(0.0)	VRL(3.2) VRT(17.3) AR (8.7) DDC(0.0)	**(5,view)**	VRL(7.7) VRT(23.6) AR (5.6) DDC(0.0)	VRL(4.8) VRT(21.6) AR (6.7) DDC(0.1)	VRL(7.7) VRT(26.8) AR (9.3) DDC(2.2)
(2,view)	VRL(3.2) VRT(18.3) AR (5.0) DDC(0.0)	C	C	**(5,<qc,view>)**	VRL(6.2) VRT(23.6) AR (7.6) DDC(0.0)	VRL(6.2) VRT(23.6) AR (12.5) DDC(0.0)	VRL(7.7) VRT(25.7) AR (10.0) DDC(1.5)
(2,block)	VRL(11.8) VRT(23.6) AR (17.1) DDC(0.0)	C	C	**(5,<qc,signature>)**	VRL(6.2) VRT(22.5) AR (12.8) DDC(0.0)	VRL(6.2) VRT(21.5) AR (11.3) DDC(0.0)	VRL(7.7) VRT(22.5) AR (10.6) DDC(0.0)
(2,p_signature)	VRL(10.4) VRT(26.8) AR (14.1) DDC(0.0)	C	C	**(6,view)**	VRL(6.2) VRT(22.5) AR (8.5) DDC(0.0)	C	C
(3,view)	VRL(6.2) VRT(22.3) AR (8.2) DDC(0.0)	VRL(6.2) VRT(21.5) AR (10.0) DDC(0.0)	VRL(9.1) VRT(21.5) AR (9.7) DDC(0.0)	**(6,block)**	VRL(9.1) VRT(26.5) AR (10.8) DDC(1.3)	C	C
(3,<qc,view>)	VRL(6.2) VRT(22.3) AR (14.6) DDC(0.0)	VRL(6.2) VRT(23.6) AR (13.0) DDC(3.1)	VRL(7.7) VRT(23.6) AR (10.8) DDC(1.5)	**(6,p_signature)**	VRL(7.7) VRT(21.5) AR (10.8) DDC(0.0)	C	C
(3,<qc,signature>)	VRL(7.7) VRT(23.6) AR (11.8) DDC(0.0)	VRL(7.7) VRT(22.3) AR (10.3) DDC(3.3)	VRL(9.1) VRT(23.6) AR (12.1) DDC(1.0)				

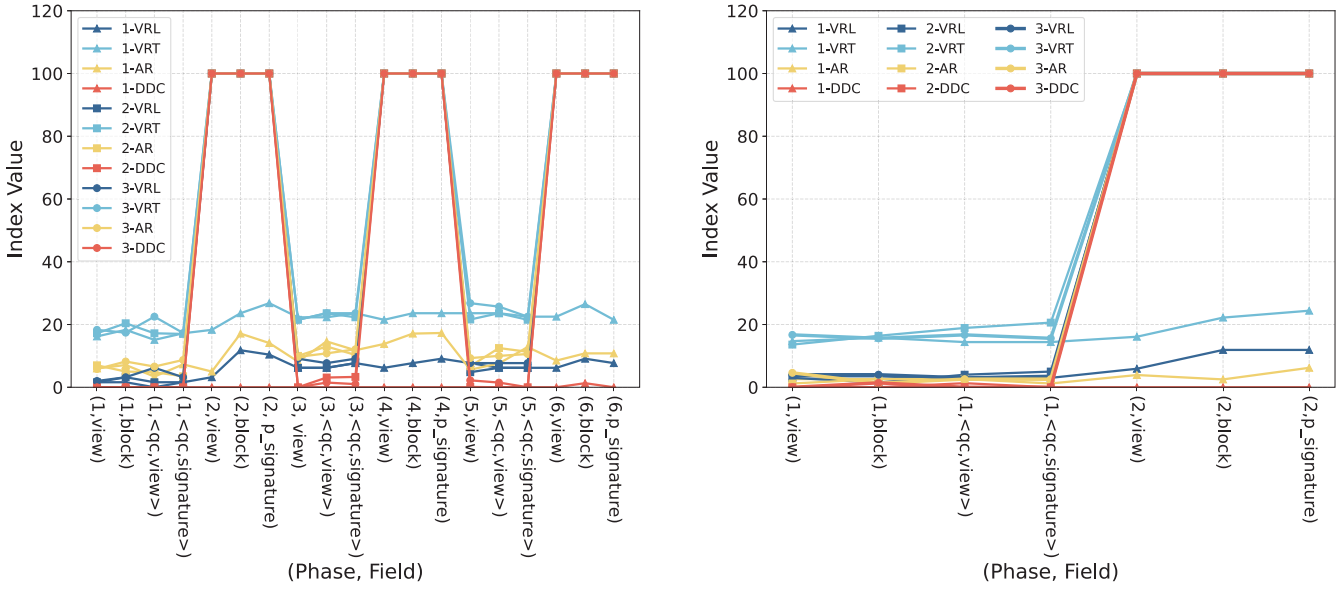
VRL: Mean latency variation rate; **VRT**: Mean throughput variation rate; **AR**: Abnormal latency rate; **DDC**: Consistency disruption degree; **C**: indicates that the system crashes; **(1, A)**: indicates that field A of phase 1 is forged.

while there will be no significant impact in phase 1. When there are 3 malicious Replicas in the Chained-HotStuff protocol, the system will also crash in phase 2, while there will be no significant impact in phase 1. After the malicious replicas reach two or more, the crashes in the second phase are clearly evident in Fig. 13(b). Similar to Basic-HotStuff, these malicious scenarios will not undermine the consistency of Chained-HotStuff (i.e., DDC indicator is not affected).

Table 10
The testing results of Conspire malicious behavior on the Chained-HotStuff (Replica-fault).

	Chained-Hotstuff						
	ReplicaConspire (number of replicas)	ReplicaConspire (number of replicas)	ReplicaConspire (number of replicas)		ReplicaConspire (number of replicas)	ReplicaConspire (number of replicas)	ReplicaConspire (number of replicas)
	1	2	3		1	2	3
(1,view)	VRL(3.5) VRT(14.7) AR (2.6) DDC(0.0)	VRL(3.0) VRT(13.6) AR (1.2) DDC(0.0)	VRL(4.0) VRT(16.7) AR (4.6) DDC(0.0)	**(2,view)**	VRL(5.9) VRT(16.1) AR (3.9) DDC(0.0)	**C**	**C**
(1,block)	VRL(3.5) VRT(15.8) AR (2.6) DDC(0.0)	VRL(1.5) VRT(16.4) AR (2.6) DDC(0.0)	VRL(4.0) VRT(15.6) AR (1.3) DDC(1.4)	**(2,block)**	VRL(11.9) VRT(22.2) AR (2.5) DDC(0.0)	**C**	**C**
(1,<qc,view>)	VRL(2.5) VRT(14.4) AR (2.6) DDC(1.3)	VRL(4.0) VRT(18.9) AR (2.7) DDC(0.0)	VRL(3.0) VRT(16.7) AR (2.6) DDC(0.0)	**(2,p_signature)**	VRL(11.9) VRT(24.4) AR (6.2) DDC(0.0)	**C**	**C**
(1,<qc,signature>)	VRL(3.0) VRT(14.4) AR (1.2) DDC(0.0)	VRL(5.0) VRT(20.6) AR (2.4) DDC(0.0)	VRL(3.5) VRT(15.6) AR (2.6) DDC(0.0)				

VRL: Mean latency variation rate; **VRT**: Mean throughput variation rate; **AR**: Abnormal latency rate; **DDC**: Consistency disruption degree; **C**: indicates that the system crashes; **(1, A)**: indicates that field A of phase 1 is forged.



(a) Basic-Hotstuff (b) Chained-Hotstuff

Fig. 13. The values of security indexes for three protocols under the scenarios of "Delay", "Duplicate", and "FeignDeath" by replica.

Our experimental results indicate that the BFTDiagnosis framework exhibits significant advantages and potential in conducting security testing of BFT protocols. Through testing various protocols such as PBFT, Basic-HotStuff, and Chained-HotStuff, we found that this framework can effectively simulate various malicious behavior scenarios and comprehensively evaluate the performance of protocols using four quantified security indicators.

4.4. Analysis of security quantification indicators

The Analyzer in `BFTDiagnosis` is capable of collecting consensus data from each consensus node. With this data, it can calculate the average latency of the consensus process in the absence of malicious behavior, as well as the latency for each consensus request to complete the process in the presence of malicious behavior. When consensus nodes in the network exhibit malicious behavior, honest nodes may fail to process messages at normal speeds, leading to delays or message loss during the consensus process. These delays or losses can impact the efficiency of the entire consensus network and prolong the time to achieve consensus. Therefore, the abnormal latency rate effectively reflects the proportion of consensus processes experiencing abnormal delays in malicious scenarios, thereby indicating the impact of malicious behavior on protocol latency. A higher abnormal latency rate may suggest a greater influence of corresponding malicious behavior on the entire consensus network.

Furthermore, the average latency variation rate measures the degree of change in average latency during the consensus process in malicious scenarios compared to normal conditions, thus evaluating the impact of malicious behavior on consensus process latency. A higher average latency variation rate typically indicates greater latency fluctuations during the consensus process in malicious scenarios, which may adversely affect the performance and stability of the entire consensus network.

On the other hand, the throughput in the consensus process under normal conditions refers to the average number of messages or transactions that consensus nodes can process in a healthy state. When malicious behavior occurs, honest consensus nodes may fail to process messages at normal speeds, resulting in reduced throughput. Therefore, the throughput variation rate measures the degree of change in throughput during the consensus process in malicious scenarios, thereby evaluating the impact of malicious behavior on consensus process throughput. A higher throughput variation rate usually indicates greater fluctuations in throughput during the consensus process in malicious scenarios.

The degree of consistency disruption reflects the protocol's resistance to malicious behavior by consensus nodes. A lower degree of consistency disruption implies that the system can maintain a higher level of consistency when facing abnormal situations, indicating greater robustness. Quantitative analysis of inconsistencies generated during the consensus process enables a more accurate assessment of the performance and stability of the consensus protocol.

Table 11
Multidimensional comparison of BFTDiagnosis with 9 related techniques.

	Technology method	Performance testing				Malicious scenario simulation					Fault localization	Security quantification	Testing subjects
		CPU	RAM	Latency	Throughput	Delay	Duplicate	FeignDeath	Ambiguous	Conspire			
Twins [14]	Simulation	✗	✗	✗	✗	✔	✔	✔	✔	✗	✗	✗	Diem
Fluffy [15]	Fuzzing	✗	✗	✗	✗	✔	✔	✔	✔	✗	✗	✗	Ethereum
LOKI [16]	Fuzzing	✗	✗	✗	✗	✔	✔	✔	✔	✗	✗	✗	Most Blockchain
Hyperledger Caliper [11]	Adapter	✔	✔	✔	✔	✗	✗	✗	✗	✗	✗	✗	Hyperledger Ethereum
HyperBench [12]	Adapter	✔	✔	✔	✔	✗	✗	✗	✗	✗	✗	✗	Hyperledger Ethereum
BFT-SMaRt [13]	Simulation	✔	✔	✔	✔	✗	✗	✗	✗	✗	✗	✗	BFT
Delphi-BFT [19]	Simulation	✔	✔	✔	✔	✗	✗	✗	✗	✗	✗	✗	BFT
BFT-Bench [18]	Simulation	✔	✔	✔	✔	✔	✔	✔	✗	✗	✗	✗	BFT
Tool [17]	Simulation	✔	✔	✔	✔	✔	✔	✔	✗	✗	✗	✗	BFT
BFTDiagnosis	Adapter Simulation	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	BFT

Simulation: Provides the basic components of the scenario and detects them through simulation;
Fuzzing: Security detection based on fuzzy testing;
Adapter: Common interfaces are exposed to the outside world and detected by means of adaptation.

4.5. Comparison of existing relevant technologies

To emphasize the advantages of BFTDiagnosis, we collected and analyzed nine other relevant technologies and tools, comparing them with BFTDiagnosis.

As shown in the Table 11, Twins, Fluffy, and LOKI are security testing methods for blockchain systems, supporting malicious behavior scenarios such as Delay, Duplicate, Feign Death, and Ambiguous. Among them, Twins employs simulation, while Fluffy and LOKI rely on fuzz testing methods. Overall, LOKI outperforms Twins and Fluffy, as it can test the security of most blockchain systems, whereas Twins and Fluffy are limited to Diem and Ethereum, respectively. Hyperledger Caliper and HyperBench are primarily used for performance testing of Hyperledger blockchain systems and Ethereum, adapting to test metrics such as CPU, RAM, latency, and throughput. BFT-SMaRt and Delphi-BFT employ simulation methods for testing BFT protocols. BFT-Bench and Tool use simulation to test BFT protocols and simulate malicious scenarios, supporting behaviors like Delay, Duplicate, and Feign Death.

Our BFTDiagnosis framework can perform performance testing and simulate malicious scenarios for BFT protocols using adaptation or simulation methods, supporting behaviors such as Delay, Duplicate, Feign Death, Ambiguous, and Conspire. Additionally, BFTDiagnosis quantifies the security of BFT protocols, reflecting protocol execution through four security metrics and pinpointing fault locations based on the triggering phase of malicious behavior. Compared to the other nine technologies and tools, BFTDiagnosis offers a more comprehensive testing scenario for BFT protocols, making it more practical. However, its scope of applicability is relatively weaker compared to LOKI.

5. Discussion

In this section, we discuss BFTDiagnosis in terms of its universality, practicality, and scalability, and summarize its limitations.

5.1. The universality of BFTDiagnosis

To adapt to various BFT protocols, we design and develop the Protocol Actuator in BFTDiagnosis. For users, the Protocol Actuator is a highly flexible component that allows them to implement various BFT protocols with a Primary-Backup structure based on the inherent network communication module and key management module in the Protocol Actuator. Furthermore, the Protocol Actuator does not restrict the core logic of BFT protocols, and users only need to supplement the core logic of protocols based on the external interfaces of the Protocol Actuator. Therefore, BFTDiagnosis is universal for BFT protocols.

5.2. The practicality of BFTDiagnosis

BFTDiagnosis achieves the synchronization of BFT protocol development and security testing processes. During the actual protocol development, different coding approaches can lead to various running problems. BFTDiagnosis provides an intuitive presentation of the security indicator values, which enables the determination of the source of problems based on these indicators. Interestingly, during the implementation of PBFT, Basic-HotStuff, and Chained-HotStuff protocols, we unexpectedly discovered issues that needed attention in the Chained-HotStuff protocol's code implementation. Specifically, the Chained-HotStuff's Generic message contains nested logic of QC, and as the consensus round increases, the size of the Generic message also increases. This leads to memory overflow when the consensus round reaches a certain quantity. Therefore, it is necessary to clear the old QC in the Generic message or generate a fixed-size verification tag for the old QC. Moreover, although BFTDiagnosis is a security testing framework for BFT protocol, it also supports performance testing for BFT protocol. Therefore, BFTDiagnosis is very practical in the actual application process of BFT protocol development.

5.3. The complexity of expanding malicious scenarios

Although BTFDiagnosis currently supports simulating various malicious scenarios with 9 types of malicious behaviors, it still needs the ability to extend to new malicious scenarios. With the method proposed in this paper, we can easily add new node modes to support more node behaviors, and add more control parameters in the node modes to simulate more malicious scenarios, thereby testing potential unknown attack methods. Therefore, BTFDiagnosis has a strong extendibility to malicious scenarios.

5.4. Limitations

Despite the superior capabilities of BFTDiagnosis compared to existing technologies, there are still some inevitable limitations. Firstly, for emerging or insufficiently understood malicious behavior patterns, BFTDiagnosis may struggle to accurately identify and simulate them, potentially affecting the comprehensiveness and accuracy of test results. Secondly, BFTDiagnosis may not be suitable for certain types of BFT protocols, especially those with non-standard or highly customized designs, which may be incompatible with the testing framework of BFTDiagnosis. Lastly, while BFTDiagnosis

provides comprehensive security assessments at the protocol level, it does not directly apply to security testing of blockchain systems, which is a limitation that requires special attention.

6. Related work

6.1. BFT consensus protocols

Due to the continuous advancement of blockchain technology in recent years, some consensus protocols based on BFT are also constantly proposed and improved [27–29]. PBFT [4] is the first practical BFT consensus protocol. However, when the number of nodes participating in the consensus is too large, its communication complexity also increases exponentially, so it is generally used in consensus networks with few nodes. To reduce the communication complexity of PBFT, some novel improved PBFT protocols are proposed, such as SBFT [5] and BFT-SMART [13]. However, PBFT leadership switching still has significant performance problems. Tendermint [6] uses Gossip broadcast to simplify the PBFT broadcast process, while Casper [30] combines the idea of probabilistic consensus protocol PoS with the idea of BFT on the basis of Tendermint, which enhances the security of the protocol and greatly improves the leader mechanism. HotStuff [7,31,32] protocol presents a three-stage QC proof, which has greater throughput compared to PBFT and greatly reduces the complexity of leader handover. In addition, many asynchronous BFT protocols have been proposed, such as DLS [33], HoneyBadger [34], BEAT [35], and Dumbo [36–38]. However, asynchronous BFT protocols are still not mature enough to be suitable for practice.

6.2. Blockchain testing

In recent years, researchers proposed a doze of performance testing frameworks or tools for blockchain systems, such as Caliper [11], Hyperbench [12], EthereumTester [39], and others. Some recent research also has testing frameworks or tools for blockchain. Blockbench [40], the first evaluation framework for analyzing private chains, is a way to fairly compare different blockchain platforms, allowing us to gain a deeper understanding of different system design choices.

Several studies have proposed performance indicators for blockchain systems, such as overall performance and detailed performance indicators, to provide users with an accurate understanding of the performance of different stages of the blockchain. One study [41] designed a log-based performance monitoring framework with low overhead and good scalability. Additionally, another study [42] developed a method to evaluate the performance of consensus algorithms on private blockchain platforms, specifically Ethereum and Hyperledger. Through quantitative analysis of latency and throughput, the study obtained performance evaluation results for consensus algorithms with different transaction numbers. These experimental results provide quantitative data support for further research on consensus algorithms.

6.3. Security analysis of BFT protocols

Although the BFT protocol is theoretically proved to be secure when $f < N/3$[43] is satisfied, it is mentioned in some studies [44–46] whether the BFT protocol can tolerate more Byzantine nodes. Recently, Multi-Threshold BFT [24] proposed four fault thresholds for synchronous, asynchronous, and partially synchronous BFT protocols. For the case that the security threshold of BFT protocol is exceeded, BFT Forensics [23] conducted a research on whether BFT protocol supports Forensics, that is, in the case that the number of Byzantine nodes is greater than or equal to $N/3$, the number of malicious copies that each BFT protocols can identify is analyzed. In addition, Marco Platania et al. [47] classify BFT protocols into server-side or client-side, and theoretically analyze their security from the perspectives of performance attacks and correctness attacks, providing guidance for builders of BFT application systems in selecting appropriate BFT protocols. Twins [14] is an automated unit test generator to detect consistency conflicts in DiemBFT. Twins simulate attack scenarios by creating a malicious node with the same identity as an honest node and having that malicious node launch malicious behaviors. Newer BFT protocols like sync-Hotstuff also fail to resist force-locking Attack [48]. Therefore, BFT protocol may still have some subtle security threats, which will lead to security vulnerabilities in the process of protocol design or protocol code implementation.

7. Conclusion and future works

In this work, we propose an automated security testing framework, referred to as `BFTDiagnosis`, for Byzantine Fault Tolerant (BFT) protocols. The framework consists of three components: the Controller, the Protocol Actuator, and the Analyzer. `BFTDiagnosis` automates the management and configuration of testing tasks through the Controller. The protocol Actuator serves as a container for running BFT protocols, driving the consensus nodes to initiate malicious behaviors into the network using malicious node mode matching strategies and triggering mechanisms to simulate various malicious scenarios. The Analyzer collects node data during the testing process and outputs four quantified security indicators. Experimental analysis shows excellent performance of `BFTDiagnosis`. Leveraging this framework, we conduct security testing on PBFT, Basic-HotStuff, and Chained-HotStuff, validating the effectiveness and rationality of the four security quantification indicators proposed. Compared to existing technologies, `BFTDiagnosis` offers broader coverage in testing scenarios.

In the future, we will extend and optimize `BFTDiagnosis`: (1) We will extend it to support for asynchronous BFT protocols, which can effectively explore the security issues of asynchronous BFT protocols and provide a powerful auxiliary tool for the practical application of asynchronous BFT protocols; (2) We will optimize the execution performance of the `BFTDiagnosis`.

CRediT authorship contribution statement

Jitao Wang: Data curation, Methodology, Project administration, Resources, Software, Supervision, Validation, Writing – original draft, Writing – review & editing. **Bo Zhang:** Software, Writing – original draft, Writing – review & editing. **Kai Wang:** Conceptualization, Writing – original draft, Writing – review & editing. **Yuzhou Wang:** Software, Writing – original draft, Writing – review & editing. **Weili Han:** Conceptualization, Funding acquisition, Methodology, Project administration, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

https://github.com/Wangert/BFTDiagnosis.

Acknowledgments

This paper is supported by National Key Technologies R&D Program of China (2022YFC3301300), NSFC (No. 62172100, 92370120). Thank Mr. Changhao Wu for his language supports.

Appendix. Abbreviation index

Symbol	Explanation
R	Consensus node role parameters: Leader or Replica
P	Protocol stage parameters triggered by malicious behavior
N	Number of consensus nodes
N_r	Number of malicious replicas
ΔT	Stable runtime parameter of the protocol, to prevent abnormal data caused by instability in the early stages of protocol execution
T_{delay}	Delay time parameter under 'delay' malicious behavior
FC	Message spoofing parameter, indicating the hierarchical index of message structure fields
S_a	Ambiguous message quantity set under 'ambiguous' malicious behavior
$M^{'}M^{''}$	Forged malicious messages
REQ	Consensus request identifier, used to differentiate between different consensus requests
C^{start}	The relevant data for consensus initiation output to the analyzer, including the request identifier and the start timestamp
C^{end}	The relevant data for consensus initiation output to the analyzer, including the request identifier and the end timestamp
C_{kj}^{start}	The consensus initiation data sent by consensus node with ID j to analyzer with ID k
C_{kj}^{end}	The consensus termination data sent by consensus node with ID j to analyzer with ID k
TS^{start}	Timestamp of when the consensus request begins consensus
TS^{end}	Timestamp of when the consensus request completes consensus
CS^{start}	Set of timestamps of when consensus requests begin consensus
CS^{end}	Set of timestamps of when consensus requests complete consensus
Q	Number of consensus requests
L_{mean}	Average delay for consensus request completion
L_{mean}^{normal}	Average delay for consensus request completion under normal conditions
$L_{mean}^{malicious}$	Average delay for consensus request completion under malicious scenarios
T_{mean}	Average throughput of the protocol
$T_{mean}^{malicious}$	Average throughput of the protocol under malicious scenarios
ΔT_{exec}	Execution time of the protocol
$AR_{latency}$	Abnormal delay rate, indicating the proportion of consensus requests experiencing abnormal delay over a period of time compared to all consensus requests
$VR_{latency}$	Average delay change rate, indicating the change in average delay relative to the situation without malicious behavior occurring in consensus nodes
$VR_{throughput}$	Throughput change rate, indicating the change in throughput relative to the situation without malicious behavior occurring in consensus nodes
$DD_{consistency}$	Degree of consistency violation, indicating the proportion of consensus requests that fail to achieve consensus, relative to all consensus requests. In the absence of malicious behavior, this should be close to 0 (considering other factors such as network environment leading to partial data loss)

References

[1] L. Lamport, R.E. Shostak, M.C. Pease, The Byzantine generals problem, ACM Trans. Program. Lang. Syst. 4 (3) (1982) 382–401, http://dx.doi.org/10.1145/3335772.3335936.

[2] FISCO, FISCO BCOS: An enterprise-level financial blockchain underlying technology platform, 2017, https://github.com/FISCO-BCOS.

[3] Ant Group, AntChain, 2016, https://antchain.antgroup.com/.

[4] M. Castro, B. Liskov, Practical Byzantine fault tolerance, in: Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation, OSDI, New Orleans, Louisiana, USA, February 22-25, 1999, USENIX Association, 1999, pp. 173–186, https://www.usenix.org/conference/osdi-99/practical-byzantine-fault-tolerance.

[5] G. Golan-Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M.K. Reiter, D. Seredinschi, O. Tamir, A. Tomescu, SBFT: A scalable decentralized trust infrastructure for blockchains, 2018, pp. 568–580, http://dx.doi.org/10.1109/DSN.2019.00063, CoRR abs/1804.01626. http://arxiv.org/abs/1804.01626.

[6] E. Buchman, J. Kwon, Z. Milosevic, The latest gossip on BFT consensus, 2018, CoRR abs/1807.04938, https://api.semanticscholar.org/CorpusID:49744920.

[7] M. Yin, D. Malkhi, M.K. Reiter, G. Golan-Gueta, I. Abraham, HotStuff: BFT consensus with linearity and responsiveness, in: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, on, Canada, July 29 - August 2, 2019, ACM, 2019, pp. 347–356, http://dx.doi.org/10.1145/3293611.3331591.

[8] The Diem Team, DiemBFT protocol, 2021, https://developers.diem.com/papers/diem-consensus-state-machine-replication-in-the-diem-blockchain/2021-08-17.pdf.

[9] Aptos-Labs, AptosBFT protocol, 2022, https://github.com/aptos-labs/aptos-core.

[10] K. Huang, R. Hou, Y. Zeng, LWSBFT: Leaderless weakly synchronous BFT protocol, Comput. Networks 219 (2022) 109419, http://dx.doi.org/10.1016/J.COMNET.2022.109419.

[11] Hyperledger, Hyperledger caliper: A blockchain benchmark tool, 2018, https://github.com/hyperledger/caliper.

[12] Hyperchain, HyperBench: A distributed stress testing tool on blockchain platforms, 2020, https://github.com/meshplus/HyperBench.

[13] A.N. Bessani, J. Sousa, E.A.P. Alchieri, State machine replication for the masses with BFT-SMART, in: Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, June 23-26, 2014, IEEE Computer Society, 2014, pp. 355–362, http://dx.doi.org/10.1109/DSN.2014.43.

[14] S. Bano, A. Sonnino, A. Chursin, D. Perelman, Z. Li, A. Ching, D. Malkhi, Twins: BFT systems made robust, in: Proceedings of the 25th International Conference on Principles of Distributed Systems, OPODIS 2021, December 13-15, 2021, Strasbourg, France, in: LIPIcs, vol. 217, 2021, pp. 7:1–7:29, https://api.semanticscholar.org/CorpusID:237235563.

[15] Y. Yang, T. Kim, B. Chun, Finding consensus bugs in ethereum via multi-transaction differential fuzzing, in: A.D. Brown, J.R. Lorch (Eds.), 15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021, July 14-16, 2021, USENIX Association, 2021, pp. 349–365, https://www.usenix.org/conference/osdi21/presentation/yang.

[16] F. Ma, Y. Chen, M. Ren, Y. Zhou, Y. Jiang, T. Chen, H. Li, J. Sun, LOKI: State-aware fuzzing framework for the implementation of blockchain consensus protocols, in: NDSS, 2023, http://dx.doi.org/10.14722/ndss.2023.24078.

[17] P.-L. Wang, T.-W. Chao, C.-C. Wu, H.-C. Hsiao, Tool: An efficient and flexible simulator for Byzantine fault-tolerant protocols, in: 2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN, IEEE, 2022, pp. 287–294, http://dx.doi.org/10.1109/DSN53405.2022.00038.

[18] D. Gupta, L. Perronne, S. Bouchenak, BFT-bench: A framework to evaluate BFT protocols, in: A. Avritzer, A. Iosup, X. Zhu, S. Becker (Eds.), Proceedings of the 7th ACM/SPEC International Conference on Performance Engineering, ICPE 2016, Delft, the Netherlands, March 12-16, 2016, ACM, 2016, pp. 109–112, http://dx.doi.org/10.1145/2851553.2858667.

[19] C. Berger, S.B. Toumia, H.P. Reiser, Simulating BFT protocol implementations at scale, 2022, http://dx.doi.org/10.48550/ARXIV.2208.14745, CoRR abs/2208.14745.

[20] S. Al Janabi, M.A. Salman, M. Mohammad, Multi-level network construction based on intelligent big data analysis, in: Big Data and Smart Digital Environment, Springer, 2019, pp. 102–118.

[21] M.A. Salman, M.A. Mahdi, S. Al-Janabi, A GMEE-WFED system: Optimizing wind turbine distribution for enhanced renewable energy generation in the future, Int. J. Comput. Intell. Syst. 17 (1) (2024) 5, http://dx.doi.org/10.1007/S44196-023-00391-7.

[22] G.S. Mohammed, S. Al-Janabi, An innovative synthesis of optmization techniques (FDIRE-GSK) for generation electrical renewable energy from natural resources, Results Eng. 16 (2022) 100637, http://dx.doi.org/10.1016/j.rineng.2022.100637, [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2590123022003073.

[23] P. Sheng, G. Wang, K. Nayak, S. Kannan, P. Viswanath, BFT protocol forensics, in: Proceedings of the CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021, ACM, 2021, pp. 1722–1743, http://dx.doi.org/10.1145/3460120.3484566.

[24] A. Momose, L. Ren, Multi-threshold Byzantine fault tolerance, in: Proceedings of the CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021, ACM, 2021, pp. 1686–1699, http://dx.doi.org/10.1145/3460120.3484554.

[25] M.N. Halgamuge, Estimation of the success probability of a malicious attacker on blockchain-based edge network, Comput. Networks 219 (2022) 109402, http://dx.doi.org/10.1016/J.COMNET.2022.109402.

[26] R. Saltini, BigFooT: A robust optimal-latency BFT blockchain consensus protocol with dynamic validator membership, Comput. Networks 204 (2022) 108632, http://dx.doi.org/10.1016/J.COMNET.2021.108632.

[27] Y. Amir, B.A. Coan, J. Kirsch, J. Lane, Prime: Byzantine replication under attack, IEEE Trans. Dependable Secur. Comput. 8 (4) (2011) 564–577, http://dx.doi.org/10.1109/TDSC.2010.70.

[28] A. Clement, M. Kapritsos, S. Lee, Y. Wang, L. Alvisi, M. Dahlin, T. Riché, Upright cluster services, in: J.N. Matthews, T.E. Anderson (Eds.), Proceedings of the 22nd ACM Symposium on Operating Systems Principles 2009, SOSP 2009, Big Sky, Montana, USA, October 11-14, 2009, ACM, 2009, pp. 277–290, http://dx.doi.org/10.1145/1629575.1629602.

[29] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, E.L. Wong, Zyzzyva: Speculative Byzantine fault tolerance, ACM Trans. Comput. Syst. 27 (4) (2009) 7:1–7:39, http://dx.doi.org/10.1145/1658357.1658358.

[30] V. Buterin, V. Griffith, Casper the friendly finality gadget, 2017, CoRR abs/1710.09437. https://api.semanticscholar.org/CorpusID:11301538.

[31] I. Abraham, D. Malkhi, K. Nayak, L. Ren, M. Yin, Sync HotStuff: Simple and practical synchronous state machine replication, in: Proceedings of the 2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020, IEEE, 2020, pp. 106–118, http://dx.doi.org/10.1109/SP40000.2020.00044, https://api.semanticscholar.org/GorpusID:150386693.

[32] M.M. Jalalzai, J. Niu, C. Feng, Fast-HotStuff: A fast and resilient HotStuff protocol, 2020, pp. 1–17, http://dx.doi.org/10.1109/TDSC.2023.3308848, CoRR abs/2010.11454. https://api.semanticscholar.org/CorpusID:225040881.

[33] C. Dwork, N.A. Lynch, L.J. Stockmeyer, Consensus in the presence of partial synchrony, J. ACM 35 (2) (1988) 288–323, http://dx.doi.org/10.1145/42282.42283.

[34] A. Miller, Y. Xia, K. Croman, E. Shi, D. Song, The honey badger of BFT protocols, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016, ACM, 2016, pp. 31–42, http://dx.doi.org/10.1145/2976749.2978399.

[35] S. Duan, M.K. Reiter, H. Zhang, BEAT: asynchronous BFT made practical, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, on, Canada, October 15-19, 2018, ACM, 2018, pp. 2028–2041, http://dx.doi.org/10.1145/3243734.3243812.

[36] B. Guo, Z. Lu, Q. Tang, J. Xu, Z. Zhang, Dumbo: Faster asynchronous BFT protocols, in: Proceedings of the CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020, ACM, 2020, pp. 803–818, http://dx.doi.org/10.1145/3372297.3417262.

[37] Y. Lu, Z. Lu, Q. Tang, G. Wang, Dumbo-MVBA: Optimal multi-valued validated asynchronous Byzantine agreement, revisited, in: Proceedings of the PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020, ACM, 2020, pp. 129–138, http://dx.doi.org/10.1145/3548606.3559379.

[38] B. Guo, Y. Lu, Z. Lu, Q. Tang, J. Xu, Z. Zhang, Speeding dumbo: Pushing asynchronous BFT closer to practice, IACR Cryptol. ePrint Arch. 2022 (2022) 27, http://dx.doi.org/10.1145/3548606.3559379, https://eprint.iacr.org/2022/027.

[39] Ethereum Team, Ethereum tester: A ethereum testing tool, 2018, https://github.com/ethereum/eth-tester.

[40] T.T.A. Dinh, J. Wang, G. Chen, R. Liu, B.C. Ooi, K. Tan, BLOCKBENCH: A framework for analyzing private blockchains, in: S. Salihoglu, W. Zhou, R. Chirkova, J. Yang, D. Suciu (Eds.), Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017, ACM, 2017, pp. 1085–1100, http://dx.doi.org/10.1145/3035918.3064033.

[41] P. Zheng, Z. Zheng, X. Luo, X. Chen, X. Liu, A detailed and real-time performance monitoring framework for blockchain systems, in: Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2018, Gothenburg, Sweden, May 27 - June 03, 2018, ACM, 2018, pp. 134–143, http://dx.doi.org/10.1145/3183519.3183546.

[42] Y. Hao, Y. Li, X. Dong, L. Fang, P. Chen, Performance analysis of consensus algorithm in private blockchain, in: Proceedings of the 2018 IEEE Intelligent Vehicles Symposium, IV 2018, Changshu, Suzhou, China, June 26-30, 2018, IEEE, 2018, pp. 280–285, http://dx.doi.org/10.1109/IVS.2018.8500557.

[43] C. Dwork, N.A. Lynch, L.J. Stockmeyer, Consensus in the presence of partial synchrony, J. ACM 35 (2) (1988) 288–323, http://dx.doi.org/10.1145/42282.42283.

[44] J. Li, D. Mazières, Beyond one-third faulty replicas in Byzantine fault tolerant systems, in: Proceedings of the 4th Symposium on Networked Systems Design and Implementation (NSDI 2007), April 11-13, 2007, Cambridge, Massachusetts, USA, Proceedings, USENIX, 2007, http://dx.doi.org/10.5555/1973430.1973440, http://api.semanticsholar.org/CorpusID:13199075.

[45] D. Kane, A. Fackler, A. Gagol, D. Straszak, Highway: Efficient consensus with flexible finality, 2021, http://dx.doi.org/10.48550/arXiv.2101.02159, CoRR abs/2101.02159.

[46] Z. Xiang, D. Malkhi, K. Nayak, L. Ren, Strengthened fault tolerance in Byzantine fault tolerant replication, in: Proceedings of the 41st IEEE International Conference on Distributed Computing Systems, ICDCS 2021, Washington DC, USA, July 7-10, 2021, IEEE, 2021, pp. 205–215, http://dx.doi.org/10.1109/ICDCS51616.2021.00028.

[47] M. Platania, D. Obenshain, T. Tantillo, Y. Amir, N. Suri, On choosing server- or client-side solutions for BFT, ACM Comput. Surv. 48 (4) (2016) 61:1–61:30, http://dx.doi.org/10.1145/2886780.

[48] A. Momose, Force-locking attack on sync hotstuff, IACR Cryptol. ePrint Arch. 2019 (2019) 1484, https://api.semanticscholar.org/CorpusID:209528893.

Jitao Wang is a Ph.D student at Fudan University. He received his master's degree from Nanjing University of Posts and Telecommunications in 2021. He is currently a member of the Laboratory of Data Analytics and Security. His research interests mainly include blockchain consensus protocol, consensus protocol testing and blockchain data analytics.

Bo Zhang is a graduate student at Fudan University. He received his B.S. degree from Shandong University in 2020. He is currently a member of the Laboratory of Data Analytics and Security. His research interest mainly includes blockchain consensus protocol and blockchain data analytics.

Kai Wang is a Ph.D student at Fudan University. He received his B.S. degree from Fudan University in 2020. He is currently a member of the Laboratory of Data Analytics and Security. His research interest mainly includes blockchain data analytics and system security.

Yuzhou Wang is a graduate student at Fudan University. He received his B.S. degree from Southeast University in 2022. He is currently a member of the Laboratory of Data Analytics and Security. His research interest mainly includes blockchain consensus protocols and Web3 security.

Weili Han is a full Professor at Software School, Fudan University. He received his Ph.D. at Zhejiang University in 2003. Then, he joined the faculty of Software School at Fudan University. From 2008 to 2009, he visited Purdue University as a visiting professor funded by China Scholarship Council and Purdue University. His research interests are mainly in the fields of Data Systems Security, Access Control, and Password Security. He is now the distinguished member of CCF and the members of the IEEE, ACM, SIGSAC. He serves in several leading conferences and journals as PC members, reviewers, and an associate editor.