Tyr: Finding Consensus Failure Bugs in Blockchain System with Behaviour Divergent Model

Yuanliang Chen*, Fuchen Ma*, Yuanhang Zhou*, Yu Jiang*✉, Ting Chen†, and Jiaguang Sun*
*School of Software, Tsinghua University, KLISS, BNRist, Beijing, China
† University of Electronic Science and Technology of China, Chengdu, China

***Abstract*—Blockchain is a decentralized distributed system on which a large number of financial applications have been deployed. The consensus process in it plays an important role, which guarantees that legal transactions on the chain can be executed and recorded fairly and consistently. However, because of Consensus Failure Bugs (CFBs), many blockchain systems do not provide even this basic guarantee. The validity and consistency of blockchain systems rely on the soundness of complex consensus logic implementation. Any bugs which cause the blockchain consensus failure can be crucial.**

In this work, we introduce Tyr, an open-source tool for detecting CFBs in blockchain systems with a large number of abnormal divergent consensus behaviors. First, we design four oracle detectors to monitor the behaviors of nodes and analyze the violation of consensus properties. To trigger these oracles effectively, Tyr harnesses a behavior divergent model to constantly generate consensus messages and make nodes behave as differently as possible. We implemented and evaluated Tyr on six widely used commercial blockchain consensus systems, including IBM Fabric, WeBank FISCO-BCOS, ConsenSys Quorum, Facebook Diem, Go-Ethereum, and EOS. Compared with the state-of-the-art tools Peach, Fluffy, and Twins, Tyr covers 27.3%, 228.2%, and 297.1% more branches, respectively. Furthermore, Tyr has detected 20 serious previously unknown vulnerabilities, all of which have been repaired by the corresponding maintainers.

I. Introduction

At present, the blockchain has won numerous research recognition and public attention in the global innovation field [1]. As the backbone of the blockchain system, the consensus process coordinates all the nodes in decentralized network scenarios to verify and make agreements on transaction results. Various blockchain systems utilize different consensus processes. In practice, most consensus processes ensure that all honest nodes maintain a consistent view of the blockchain. Two fundamental properties should be guaranteed: **validity**, which is that all transactions published by one honest node will eventually be executed and recorded by all other honest nodes on their blockchain. **Consistency**, which is that given enough time, all ledgers in a decentralized environment will eventually arrive at the same content.

Unfortunately, due to the complexity of the node consensus process, it is hard to avoid bugs in the implementation of the consensus process. Since the consensus process plays a significant role in blockchain systems, any bugs may result in severe consequences, leading to the failure of consensus and endangering the security of the blockchain system. For example, three consensus vulnerabilities in Go-Ethereum [2]–[4] could cause a chain split, where vulnerable versions refuse to accept the canonical chain. Attackers may use them to control a part of the network and make double-spending attacks, violating the validity of the blockchain and causing severe economic losses. Another bug in Ethereum [5] makes nodes transition to inconsistent blockchain states and fail to reach consensus with other nodes. Consequently, Infura [6], the largest infrastructure service, went down; with it, some of the most popular Ethereum applications, such as MakerDAO [7] and Uniswap [8], went down. Such bugs that break key properties of blockchain systems, leading to consensus failures, are called Consensus Failure Bugs (CFBs).

To exercise blockchain systems' consensus process, more abnormal consensus behaviors than those occurring naturally during regular use need to be triggered. Testing tools such as Peach [9], Fluffy [10], Twins [11], etc., are routinely applied to the blockchain and have successfully detected some bugs. However, they lack both precise detectors and efficient testcase generation for CFBs. Most fuzzers like Peach identify bugs by monitoring whether the target program exits normally. They can not detect CFBs that violate consensus properties of blockchain systems, for example, by erroneously processing blocks without crashing the nodes. Differential testing tools, such as Fluffy, detect bugs by comparing Geth [12] and Parity's [13] results (Ethereum client implemented in Go and Rust, respectively). It is limited that the target under test needs different implementations to achieve the same consensus protocol. Most blockchain platforms such as Fabric [14], FISCO-BCOS [15] have only one implementation, making it hard to conduct differential testing. Furthermore, even when all clients return the same results, it is hard to ensure they correctly work because they might be affected by the same underlying bug. The consensus testing tool Twins detects byzantine behaviors in a mock environment, ignoring the runtime behaviors such as transaction execution, block verification, and message interaction, resulting in ineffective testing.

To effectively find CFBs in blockchain systems, there are two main challenges: (1) The first challenge is to come up with precise oracles that can identify whether a blockchain system behaves correctly. Blockchain consensus behaviors are complex and dynamic. Even though vulnerabilities are executed, it is hard to find them without precise detectors.

*Fuchen Ma have contributed equally to this work.
✉Yu Jiang and Ting Chen are the corresponding authors.

(2) The second challenge is that the CFBs tend to be hidden in the deep logic path, making it hard to trigger those bug oracles efficiently. The blockchain consensus process usually comes in multiple phases, with many nodes interacting and collaborating in different phases. Multiple nodes are required to execute a series of different behaviors to detect deep path logic vulnerabilities.

We propose Tyr, a testing tool for detecting CFBs in blockchain systems, to address these challenges. First, Tyr models four common bug oracles based on blockchain consensus properties: (1) Liveness – valid transactions should be executed eventually; (2) Safety – invalid transactions should not be committed; (3) Integrity – no hard fork happens; (4) Fairness – chances to be the leader should be fair. Then, four oracle detectors are proposed for monitoring and analyzing the nodes' consensus data in real time. Any nodes that violate the properties will be identified. To reach the deep consensus logic and trigger the oracle effectively, Tyr constructs a behavior divergent model, which contains consensus data and runtime data to describe the behavior divergence of nodes. Based on the model, Tyr continuously generates numerous abnormal divergent consensus messages to diverge the distributed nodes' behaviors as much as possible and performs effective testing for CFBs detection.

We implemented Tyr and evaluated its effectiveness on the consensus networks of 6 commercial blockchain systems: IBM Fabric, WeBank FISCO-BCOS, ConsenSys Quorum, Facebook Diem, Go-Ethereum, and EOS. Results show that Tyr averagely covers 27.3%, 228.2%, and 297.1% more branches than Peach, Fluffy, and Twins. In addition, Tyr has found 20 previously unknown vulnerabilities (5 in Fabric, 7 in FISCO-BCOS, 2 in Quorum, 3 in Go-Ethereum, 2 in EOS, and 1 in Diem). All of them were repaired by the corresponding maintainers, and 5 CVEs were assigned to US National Vulnerability Database. In summary, we make three key contributions:

- We define four bug oracles based on consensus properties and design four oracle detectors for efficiently detecting CFBs in blockchain consensus networks.
- We introduce the behavior divergent model to diverge the behaviors of nodes and trigger the bug oracles.
- We implement and evaluate Tyr on six widely used blockchain consensus systems. We will open-source Tyr[1] for practical usage. Compared with state-of-the-art tools, Tyr increases the branch coverage by 27.3%, 228.2%, and 297.1% on average. It detects 20 serious previously unknown bugs, which have been confirmed and repaired.

II. Background

A. Blockchain Consensus Process

Different from a centralized distributed system, transactions in a blockchain network need to be executed by all the replicas. The consensus protocol is proposed to ensure validity and consistency among all the replicas. Figure 1 illustrates the overall consensus process of a blockchain system. Once a user sends transactions to blockchain networks, the transactions are broadcasted between all nodes in the network and stored in their transaction pools. When a node becomes a leader/miner node according to the specific consensus protocol used by the network, it executes the transactions from the transaction pools and orders them in a block. Then the leader/miner seals the block with its signature and broadcasts it to the other nodes. When other nodes receive a block from the network, they become validators. They check the validity of the block and its transaction results. If the block is identified as legal, then it will be linked to their local blockchains. Finally, all the replicas update their global states on the ledger based on the transaction results. With the help of this consensus process, the ledger of all honest nodes will remain the same.

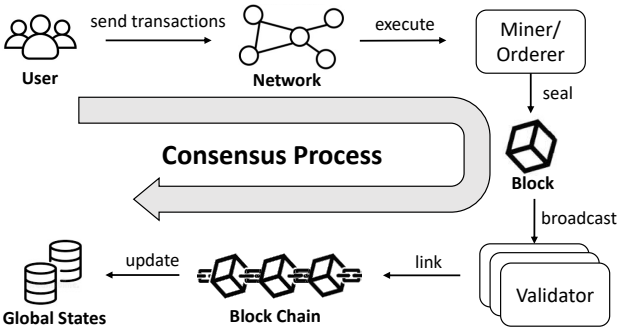

Fig. 1. The consensus process of blockchain system.

Consensus processes commonly used in blockchain systems can be divided into two types. The first type is Nakamoto consensus [16] – the longest chain mechanism. This includes protocols like PoW [17] (stands for Proof of Work) and PoS [18] (stands for Proof of Stake). They prevent malicious data manipulation by introducing a huge cost of doing evil. Each participating node competes fairly to become a miner by their computing power or stake resources proportion. The second type is the committee-based consensus mechanism. Typical protocols of this type contain PBFT [19] and Hot-Stuff [20]. Each participating node is elected leader via the 'viewchange' messages. Both consensus mechanisms satisfy the CAP theory [21], and they can tolerate node crash failures or even byzantine attacks. In addition to the effectiveness and consistency of the blockchain system, the correctness of transactions and the participants' fairness need to be guaranteed. As the basic service of financial applications, blockchain consensus protocols should be safe enough to prevent invalid transactions such as double-spending attacks [22], [23]. Fairness is essential because for blockchain systems with incentive mechanisms(e.g., Ethereum), being a leader means making a profit. Thus fairness is the concern for all participants. For those without incentive mechanisms, fair election mechanisms ensure decentralization and reduce the risks of centralized services, e.g., DDoS attacks. In summary, four key properties in blockchain systems should be guaranteed:

1) **Liveness.** The liveness property ensures that the consensus

[1] Tyr is available at: https://github.com/BlockFuzz/Tyr

process eventually produces a result even if there are malicious nodes. In a blockchain system, all valid transactions should be committed eventually.

2) **Safety.** The consensus protocol is safe if all honest nodes yield the same correct output. In the blockchain system, it makes sure all invalid transactions should never be committed.
3) **Integrity.** This property ensures that all honest nodes achieve the same block eventually in the blockchain system, and there is no node isolation in this system.
4) **Fairness.** In a blockchain consensus system, each node should have a fair probability of becoming the miner or being elected as the leader.

III. Motivation

A. CFBs in Blockchain Consensus Systems

Threat Model: Throughout this paper, we use the following threat model. First, we formally define a blockchain network as $\phi = \{h, k, T\}$. Specifically, h means the number of honest nodes which perform correctly in the network. k presents the malicious nodes, which are totally under the attacker's control. T presents the type of consensus protocols used in the blockchain network. We assume that each malicious node can connect to arbitrary nodes (including h honest nodes and $k-1$ malicious nodes). Attackers can send any number of message packets of any type to any connected nodes at any time. However, the proportion of attackers should satisfy the fault tolerance mechanisms corresponding to the different types of consensus protocols. If T is the Nakamoto consensus type, then $k/(k+h)$ should be smaller than 50%. If T is the committee-based consensus type. then $k/(k+h)$ should be smaller than $1/3$.

CFBs in blockchain consensus systems will lead to severe consequences. For example, the implementation of the PBFT protocol in FISCO-BCOS (version 3.0.0) has a liveness bug in the leader election process. The PBFT algorithm's view change mechanism is designed for leader election, which is rotated by each participant node. The 'viewchange' message proposes a request for a new leader in the next round. The number 'view' marks the current election round. And the value 'toView' tells nodes what round the next view should be. The code in figure 2 describes the process of view changing. In line 3, the node first iterates all the received 'viewChange' packets in its cache. Then, the node counts how many nodes have sent the 'viewChange' packets as shown in lines 6 - 15. The variable 'greaterViewWeight' stores the voting weights of all viewchange messages. In lines 16 - 19, if more than 'f+1' (represented as maxFaultyQuorum() + 1 in the code) votes are collected, and if the 'viewToReach' is bigger than the current one, then the node will catch up with the view.

However, this vulnerable code forgets to clear the weight in its cache after counting. If a malicious node constantly sends 'viewChange' packets with a large view, the honest nodes will store all the malicious packets in the cache first. After storing enough 'viewChange' packets from the malicious nodes in the cache, the 'greaterViewWeight' becomes larger than 'f+1', and the honest nodes set the large view from the malicious node as the target and start the view change process. Consequently, all nodes in the system begin the view change process uninterruptedly and stop processing other packets, such as transactions, blocks, etc. Attackers can use this bug to easily perform a DoS attack by preventing valid transactions from being processed. This previously unknown bug is detected by Tyr and has been assigned a CVE identifier: CVE-2022-26534 [24].

```
1  uint64_t greaterViewWeight = 0;
2  ViewType viewToReach = 0;
3  for (auto const& it : m_viewChangeCache) {
4     ...
5     // check the viewchange weight
6     auto viewChangeCache = it.second;
7     for (auto const& cache : viewChangeCache){
8        auto fromIdx = cache.first;
9        auto nodeInfo = m_config->
10          getConsensusNodeByIndex(fromIdx);
11       if (!nodeInfo){ continue;
12 // BUG: weight in cache should be clear to 0.
13       greaterViewWeight += nodeInfo->weight();
14    }
15 }
16 if (greaterViewWeight <
       (m_config->maxFaultyQuorum() +1)) return 0;
17 if (m_config->toView()>=viewToReach) return 0;
18 if (viewToReach > 0)
19 {  m_config->setToView(viewToReach - 1)};
```

Fig. 2. A CFB that breaks the liveness oracle in the implementation of PBFT protocol in FISCO-BCOS.

B. Challenges to Detect Such Bugs

Two main challenges need to be addressed to detect such CFBs: 1) These bugs are hard to trigger effectively. 2) It requires precise oracles for blockchain CFBs.

Tyr proposes a precise oracle detector and an efficient trigger mechanism for detecting such bugs. First, Tyr leverages a behavior divergent model to record the real-time behaviors of target nodes. Based on the model, Tyr effectively sends divergent consensus messages to various nodes and triggers this bug. In addition, Tyr designs an oracle based on the liveness property which identifies whether valid transactions are committed eventually.

Figure 3 shows how Tyr detects this bug. At first, Tyr randomly sends some target nodes (e.g., node A and node B) 'viewChange' packets with mutated 'toView' values (e.g., 7821, 7872). Then nodes A and B execute the above vulnerable code while others will not. By monitoring all nodes' behavior

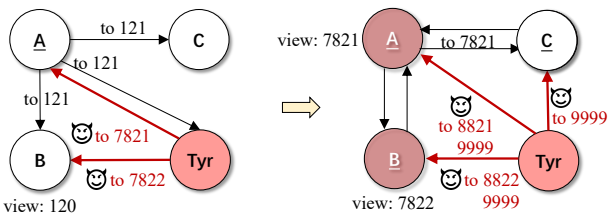

Fig. 3. The workflow of Tyr in detecting this bug. A, B, and C represent three honest nodes. Tyr constantly sends malicious packets to them.

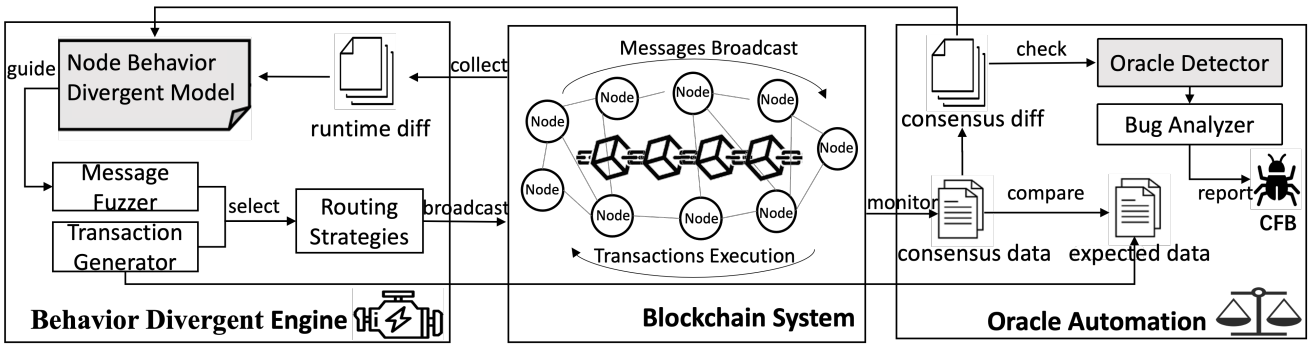

Fig. 4. An overview of Tyr. (1) Behavior divergent engine first constructs transactions and messages. (2) Then, the messages are sent to the target nodes, and the transactions are executed under the blockchain consensus network. (3) Oracle automation collects and analyzes the consensus data of distributed nodes in real time. (4) The difference between these consensus data is calculated. Then Tyr delivers it to the behavior divergent model for guiding the message fuzzer. (5) In the meanwhile, the execution runtime information of nodes will be collected and the difference between them will be calculated. (6) Oracle detector compares the consensus data with expected data and identifies if the nodes violate the oracle by checking the consensus difference. (7) Tyr proceeds to the next iteration (from step 1 to step 6) of the testing process until termination.

and calculating their difference in real time, Tyr senses the different behaviors and sends 'viewchange' packets again with new mutated values (e.g., 8822, 8821). As the figure shows, Tyr tries to persuade nodes A and B to change their views to 8821 and 8822. Due to this CFB, nodes A and B agree on the view change request and start changing their views. Tyr further senses these divergent behaviors and keeps resending 'viewChange' packets to more target nodes with large values (e.g., 9999, etc.). As a result, all the honest nodes trigger this bug, keep changing their views and stop other processing, including transaction handling. Furthermore, with the help of a well-design oracle, the liveness automation of Tyr found that transactions got stuck and reported this CFB.

IV. Tyr Design

Tyr is designed to find consensus failure bugs (CFBs) for most blockchain systems, from public blockchains, e.g., Ethereum [25] and EOS [26], to consortium blockchains, e.g., Fabric [14], Diem [27], FISCO-BCOS [15] and, Quorum [28]. Based on a node behavior divergent model(BDM), Tyr constantly generates consensus messages and transactions as test inputs to make nodes in the network behave as differently as possible. In the meanwhile, Tyr employs four oracle detectors to monitor and analyze the consensus data of target nodes in real time. If any nodes violate the oracle automation, Tyr records the contexts and reports the CFBs.

Figure 4 illustrates an overview of Tyr. There are two key components: Behavior Divergent Engine for constantly calculating the behavior difference of nodes and constructing plenty of messages and transactions based on it; Oracle Automation for analyzing nodes' consensus data and identifying consensus failure bugs in real-time. (1) Behavior divergent engine first constructs a set of messages by message fuzzer based on the node behavior divergent model. In the first testing iteration, this construction is random. Behavior Divergent Engine also generates a set of transactions as well as the expected data for validity check. (2) Then, the messages are broadcasted to the target nodes, and the transactions are executed under the consensus protocol of the blockchain system. (3) Oracle automation monitors and collects the consensus data of distributed nodes in real time. (4) The difference between these consensus data is calculated. Then it is delivered to the behavior divergent model for guiding the message fuzzer. (5) In the meanwhile, the execution runtime information of nodes will be collected by the behavior divergent model, and the difference between them will be recorded. (6) Finally, oracle automation compares the consensus data with the expected data and identifies if the nodes violate the oracle according to their consensus difference. The CFBs are reported once they are detected. (7) Tyr proceeds to the next iteration (from step 1 to step 6) of the testing process until termination.

A. Behavior Divergent Engine

Most consensus processes of blockchain systems in practice are aimed at guaranteeing eventual consistency. Transient inconsistencies in node behavior during the process are normal and allowed. However, a heuristic insight is that the ultimate consensus failure is the cumulative result of many transient inconsistencies in the consensus process. Hence, to help reach the deep consensus logic and trigger the consensus failure bugs efficiently, Tyr proposes a Behavior Divergent Engine to guide the message generation and make the nodes in the system behave as differently as possible.

Node Behavior Divergent Model. Figure 5 describes the behavior divergence model of nodes in blockchain systems. The node's behavior properties can be divided into two main types. Consensus data indicates the core consensus states of each node, including leader information, transaction data, block states, etc. Runtime data presents the key real-time states of each node as it provides services on the network. The coverage information $Cover_i$ indicates the code execution behavior of $node_i$. $Rmsg_i$ and $Smsg_i$ represent the received and sent message sets of $node_i$, respectively. Note that the transaction data Tx_i and block data B_i on the blockchain will become larger and larger over time. It is inefficient to calculate the difference between them directly. Hence, we only

Consensus Data

Num_{leader_i}: the number of times being leader/miner

TX_i : the transaction set in local pool

$TX_{i_{tyr}}$: the transaction constructed and marked by Tyr

B_i : the block chain data locally

$Height_{block}$: the height of the chain

$State_i$: the global state of ledger, is consistent with B_i

$State_{i_{tyr}}$: the state marked and monitored by Tyr

Runtime Data

$Cover_i$: the coverage information

$Rmsg_i$: the received message set

$Smsg_i$: the sent message set

Divergence

$$Divergence_{ij} = \text{diff}\left(Num_{leader_i}, Num_{leader_j}\right) + \text{diff}\left(Tx_{i_{tyr}}, Tx_{j_{tyr}}\right) + \text{diff}\left(State_{i_{tyr}}, State_{j_{tyr}}\right) + \text{diff}(Cover_i, Cover_j) + \text{diff}\left(Height_{block_i}, Height_{block_j}\right)$$

Fig. 5. Abstract description of the Node behavior Divergent Model. BDM mainly includes three parts: Consensus Data, Runtime Data, and their Divergence.

focus on $TX_{i_{tyr}}$ and $State_{i_{tyr}}$ to speed up the testing process. $TX_{i_{tyr}}$ means the transaction marked by Tyr, $State_{i_{tyr}}$ represents and the state monitored by Tyr. Different nodes can select and execute different transactions from the pool. $diff(Tx_{i_{tyr}}, Tx_{j_{tyr}})$ is the number of different transactions executed by $node_i$ and $node_j$. $diff(State_{i_{tyr}}, State_{j_{tyr}})$ is the number of different marked global states between $node_i$ and $node_j$. $diff(Num_{leader_i}, Num_{leader_j})$ is the difference in the number of elected leaders between $node_i$ and $node_j$. $diff(Height_{block_i}, Height_{block_j})$ is the difference in the height of local chains between $node_i$ and $node_j$. $diff(Txi_{tyr}, Txj_{tyr})$ is the number of different marked transactions between $node_i$ and $node_j$. The $Cover_i$ is the branch coverage of $node_i$, which is stored in a bitmap.The $diff(Cover_i, Cover_j)$ is the result of the XOR operation of the two bitmaps.

Behavior Guided Message Fuzzer. All the nodes' behaviors are driven by the messages in the blockchain system. To diverge the behaviors of nodes as much as possible, Tyr employs a behavior guided message fuzzer to constantly generate messages as test inputs to the target nodes in the blockchain system.

Algorithm 1 illustrates the process of the behavior guided message fuzzer. Before fuzzing, we set up a connection to each node in the blockchain network and monitor their consensus behaviors in real time. Tyr collects the messages in the network as the initial seeds and puts them into the $msgPool$, as shown in lines 1-5. In each fuzzing iteration, Tyr selects some target nodes from the network and mutates messages from the message pool. Then the mutated messages are sent to target nodes, and their behavior divergent models are updated and

Algorithm 1: Behavior Guided Message Fuzzing process.

```
   Input  : p2p : P2P Network under Fuzz
   Output: B_n: Consensus Failure Bugs
1  for node in p2p.getNodes() do
2      setupConnection(node);
3      monitorbehavior(node);
4      collectMessage(msgpool)
5  end
6  BDMs = {} ;
7  while true do
8      N' = randomSelect(neighbours) ;
9      for node in N' do
10         msg = msgPool.dequeue() ;
11         msg' = mutate(msg) ;
12         async:
13             feedback = p2p.send(msg');
14             BDM = updateBDM(feedback) ;
15             BDMs.append(BDM);
16         end async
17     end
18     diff = Divergence(BDMs, N');
19     newBugs = checkOracle(BDMs);
20     B_n.append(newBugs);
21     if (diff >0) or (newBugs != NULL) then
22         msgPool.updatePool(msg', msgPool);
23     end
24 end
```

analyzed in real time (Lines 11-15). The behavior divergence of nodes is calculated. In the meanwhile, four oracle detectors analyze their BDMs and identify whether they violate the liveness, safety, integrity, and fairness properties, as shown in lines 18-19. The detailed CFB detecting process will be introduced in Section IV-B. If the divergence exists or any new CFBs are found, then the messages will be regarded as interesting seeds and stored in the message pool to guide the subsequent fuzzing process, as presented in figure 6.

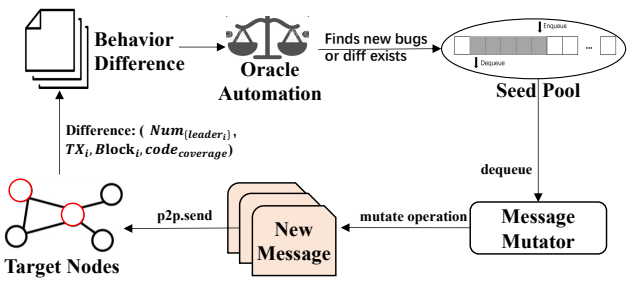

Fig. 6. Seed selection process of behavior guided fuzzing. If new mutated messages detect new oracle bugs or vary behaviors, then they will be stored in the seed pool.

The message packets in the blockchain are highly structured. As a mutation-based fuzzer, it is important to maintain

the structure of the message while mutating its content. To achieve this and generate high-quality inputs, Tyr mutates each message structurally. Since each blockchain implementation has its own message parser, Tyr first analyses the grammar structure of each message based on the inherent parsers. For example, Fabric uses protobuf [29] formation to serialize and deserialize the messages, so Tyr directly uses the *Marshal* and *Unmarshal* packages to parse messages. Then, based on the parsed content, Tyr generates new values of each field by mutating the old fields. Tyr uses different mutators for different types of fields. A numerical mutator will randomly convert the numeric type to another number. A string mutator will mutate it to a new string for the string type. For the struct type, Tyr mutates each field recursively. Finally, the well-mutated messages are generated and sent to the nodes in the blockchain network.

Take the bug in section III as an example. Once Tyr sends a 'viewChange' to the target node, then the behavior of this node will be different from other nodes. This difference will be captured and recorded by storing the 'viewChange' message in the message pool. Then in the following fuzzing iterations, Tyr has a higher probability of continuously sending 'viewChange' messages to the target node, which eventually triggers this liveness bug. With the help of the structural mutation process, the content (e.g., value $toView$) of 'viewChange' message is constantly mutated while its structure remains valid. In this way, the behavior divergent engine continuously generates high-quality messages to diverge the behaviors of distributed nodes and test the blockchain consensus network.

B. Oracle Automation

Oracle Definition: to help identify CFBs precisely, we model four oracles according to the four key properties of the blockchain consensus systems. Formally, we consider a blockchain system with m nodes taken from a finite set $\Pi = \{n_1, n_2, n_3, ...n_m\}$. The consensus data of each node $n_i \in \Pi$ can be represented as a triple $\langle Num_{leader_i}, TX_i, B_i \rangle$. More specifically, Num_{leader_i} means the number of times node n_i was elected as a leader/miner node. The finite set $TX_i = \{tx_{i1}, tx_{i2}, tx_{i3}, ..., tx_{in}\}$ means the transaction pool of node n_i. A finite set $B_i = \{b_{i1}, b_{i2}, b_{i3}, ..., b_{ip}\}$ represents the local blockchain of node n_i, and each block $b_{ij} \in B_i$ contains a set of confirmed transactions, and all transactions in B_i is part of TX_i. We also define the symbol $Height_{block_i}$ as the height of the chain. We use a finite set $TX_{invalid} = \{tx_1, tx_2, tx_3, ..., tx_q\}$ to represent all the invalid transactions constructed by Tyr's fuzz engine. The four oracles are formally defined as follows:

- **Liveness.** For each node $n_i \in \Pi$, $\forall tx \in TX_i$ and $tx \notin TX_{invalid}$, there always $\exists b_{ij} \in B_i$ that $tx \in b_{ij}$. The liveness oracle guarantees that all valid transactions must be executed, committed and stored in a specific block eventually.
- **Safety.** For each node $n_i \in \Pi$, $\forall tx \in TX_i$ and $tx \in TX_{invalid}$, $\forall b_{ij} \in B_i$ that $tx \notin b_{ij}$. The safety oracle guarantees that any invalid transactions are not allowed to be executed, committed, or stored in any blocks.
- **Integrity.** For each node $n_i \in \Pi$, $\forall n_i, n_j \in \Pi$, $\forall k : 1 \leq k \leq Min(Height_{block_i}, Height_{block_j})$, b_{ik} is equivalent to b_{jk}; and $abs(Height_{block_i}$ - $Height_{block_j})$ can not be too large. The integrity oracle ensures: (1) any block with the same block height should be equivalent to each other in all nodes; (2) block syncing mechanism should work normally. There is no node isolation in this network.
- **Fairness.** $\forall n_i, n_j \in \Pi$, $P \leq Num_{leader_i}$ / $Num_{leader_j} \leq P^{'}$, where P and $P^{'}$ are determined by specific consensus protocols. Fairness oracle means that all nodes should have a fair possibility to be elected as the leader node or miner node. In committee-based protocols, the possibility should be the same. In Nakamoto consensus protocols, the possibility should be consistent with the proportion of corresponding resources (e.g., computing power in pow).

Oracle Detector: Tyr utilizes four oracle detectors to analyze the consensus data of nodes in real time and check whether they violate the oracle definition. The process of the Liveness detector and Safety detector is illustrated in figure 7. Tyr first randomly selects some unlocked states from the global states and locks them, e.g., the balance of account A is x_a, the balance of account B is x_b, etc. Then the transaction constructor generates two sets of transactions according to the values of states. The first transactions set includes all valid transactions, e.g., transaction $A.send(B, x_a)$ means account A sends x_a to account B. The second transactions set contains all invalid transactions, e.g., transaction $C.send(D, y) y > x_c$ indicates that the balance of account D is not sufficient to pay account C. Specifically, transaction ① and transaction ② together are a double-spending attack (the balance of E x_e can only be spent once), which means that only one of them is a valid transaction and the other is an illegal transaction. In the meanwhile, the expected states are inferred and recorded by Tyr. For those valid transactions, Tyr executes them locally, and the execution results are the Liveness expected states. For the invalid double-spending transactions, Tyr executes all the combinations of them and outputs the Safety expected states. As shown in Figure 7, the expected states of F and G are either (F = $x_f + x_e$, G = 0) or (F = 0, G = $x_g + x_e$).

After the expected states are constructed, all the transactions are sent to the blockchain system and executed under the consensus process. Tyr monitors all the marked global states of target nodes and checks whether the marked states have changed or not. Finally, after a period of **decision time**, Liveness automation identifies whether all valid transitions are executed and committed successfully by checking if the corresponding states have changed to the expected states. e.g., if the balance of A has not changed to 0 or the balance of B has not changed to $x_b + x_a$, then the valid transaction $A.send(B, x_a)$ execution fails. A liveness CFB is detected. Safety Automation identifies whether all invalid transitions are not committed by checking if the marked states remain

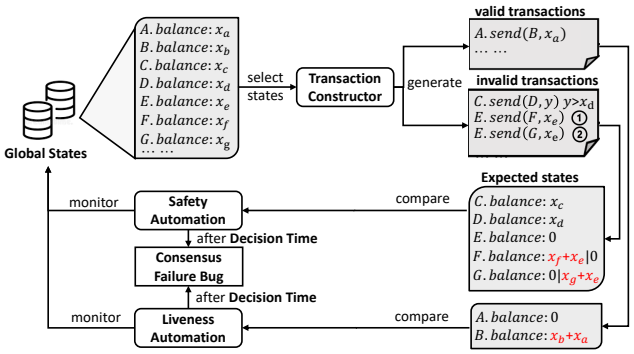

Fig. 7. The CFBs detection of Liveness automation and Safety automation. Liveness checks whether marked states changes as expected; Safety identifies whether marked state remains unchanged.

unchanged. e.g., if the balance of C and D have changed, then the invalid transaction $C.send(D, y) : y > x_c$ is executed erroneously, and a safety CFB is detected. For the double-spending transactions, if the balances of both F and G change, then a safety CFB is detected. If no CFB is detected, then remove the lock of the marked global states.

In a distributed environment, it's difficult to accurately identify whether all nodes are in finality state where all data are fixed and irreversible. Hence, the **decision time** mechanism is proposed in Tyr. It is critical to the precision of bug detection. Too short a decision time leads to many false positives because, in distributed scenarios, transaction execution has a certain delay, which depends on the network environment. However, too long a decision time affects the efficiency of the testing process since most of the global states remain locked. How to find a balanced decision time will be discussed in detail in Section V and Section VI-C.

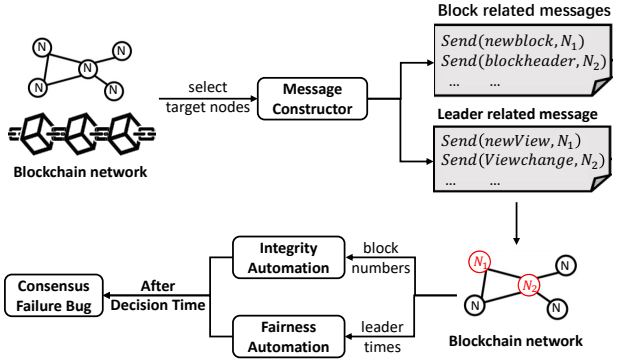

Fig. 8. The CFBs detection of Integrity automation and Fairness automation. Integrity checks all target nodes' block numbers; Fairness analyzes all target nodes' leader numbers.

Figure 8 represents the process of the Integrity detector and Fairness detector. Tyr first selects a part of the nodes in the blockchain network as target nodes. Then it generates a series of consensus messages based on the BDM and sends them to the target nodes. For example, message $send(newblock, N_1)$ means that Tyr sends a $newblock$ message to node N_1, and message $send(viewchange, N_2)$ indicates that Tyr sends a $viewchange$ message to N_2. Then, the number of times a node is elected as leader/miner and the height of the blockchain will be monitored and recorded in real time. The difference between nodes will be calculated. Integrity Automation checks the block data consistency and the blockchain's height of each node. If there is an inconsistency in the data of their local blocks with the same block number, a Integrity CFB is found. In addition, too large a difference in $Height_{block}$ indicates that the consensus network is separated. If the difference $(diff_{ij} = |Height_{block_i} - Height_{block_j}|/Height_{block_i})$ of $node_i$ and $node_j$ is greater than 10% over a **decision time**, then a Integrity CFB is detected. Fairness Automation compares the number of leaders/miners for each node. If the difference $(diff_{ij} = |Num_{leader_i} * P_j - Num_{leader_j} * P_i|/Num_{leader_i} * P_j)$ of $node_i$ and $node_j$ is greater than 10% over a **decision time**, then a Fairness CFB is detected. More specifically, P_i indicates the probability of $node_i$ being elected leader theoretically. In committee-based protocol, every node has the same P, which is $1/n$ when there are n nodes in the network. In the Nakamoto protocol, P of each node is the same as the proportion of resources they have, e.g., in pow, if the computing resource of $node_i$ accounts for 20% of the entire network, then $P_i = 0.2$.

V. Implementation

We implement Tyr on six commercial blockchain platforms, including four consortium blockchains, Fabric, FISCO-BCOS, Quorum, and Diem; and two public blockchains, Go-Ethereum and EOS; We chose them for two main reasons:

BlockChain Popularity: Hyperledger Fabric is one of the most popular enterprise-grade blockchains. It has been widely used in many industrial environments, such as A.P. Moller-Maersk, Allianz, Ant Group, Tencent, etc. FISCO-BCOS is another popular financial-grade consortium blockchain that has already been applied in many financial areas, e.g., loans. Quorum is a permissioned blockchain protocol forked from the well-known Ethereum blockchain protocol [30]. Diem is a blockchain system started by Facebook that drew regulatory blowback worldwide. Ethereum is one of the most widely used public blockchains in the world with the highest market cap $540.55B [31]. EOS is another fast, flexible, forward-driven public blockchain with higher scalability and throughput [26].

Blockchain Diversity: All six blockchain systems come from different organizations with various consensus protocols and languages. Fabric uses SmartBFT consensus protocol, developed by IBM in Go language. FISCO-BCOS uses PBFT consensus protocol, developed by WeBank in C++. Quorum uses QBFT consensus protocol, developed by ConsenSys in Go language. Diem uses DiemBFT consensus protocol, developed by Facebook in Rust language. Go-Ethereum uses POW consensus protocol, developed by Ethereum Org in Go language. EOS uses aBFT-DPOS consensus protocol, developed by block.one in C++ language. Implementation and evaluation of these blockchain systems can demonstrate that Tyr is a cross-platform and language-free testing framework with high scalability.

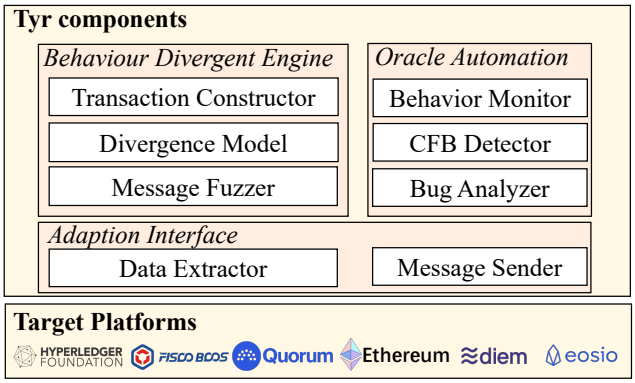

Fig. 9. Components of Tyr implementation are divided into three parts – Adaption Interface for uncoupling the target blockchain and Tyr; behavior Divergent Engine for calculating divergence and generating test inputs and Oracle Automation for detecting CFBs.

Figure 9 presents the components of Tyr, which can be divided into three main parts. The first part is the adaption part which is designed to standardize and encapsulate interfaces for testing. It is strongly associated with the target blockchain system. The second part is the behavior divergent engine which is implemented for generating high-quality transactions and messages to the target nodes based on their behavior divergence. The third part is the oracle automation for analyzing nodes' behaviors in real time and detecting CFBs. These two parts are independent and free from the target blockchain systems. The rest of the section describes notable implementation details.

Blockchain Adaption: The effort of adapting Tyr to other blockchain systems could be negligible. Modules in Tyr are well-encapsulated and loosely coupled. Hence, when adapting Tyr to a new blockchain, developers only need to implement two interfaces related to a specific chain. The first interface is 'BlockExtract()', which is responsible for extracting key consensus data from struct 'Block'. The second interface is 'p2p.send()', to send messages and transactions generated by Tyr to the target nodes in the blockchain system. Since each blockchain has its own 'DataExractor' and 'p2pSender' implementation, as inherent functions, Tyr reuses these components directly. The detailed adaption process is introduced in Appendix XI-A.

Coverage Instrumentation: Coverage Instrumentation: Language-specific instrumentation is required to collect runtime code coverage information of target nodes. For C/C++ programs, we use gcov [32]; for Rust programs, we use grcov [33]; for Go programs, we use gtest [34].

Transaction Construction: Before the testing process begins, Tyr will deploy some smart contracts in the blockchain system. These smart contracts contain a set of well-designed interfaces, including $query()$ and $transfer()$. Based on them, Tyr can easily search the global states and construct transactions to change them.

Initial Message Seeds: Initial seeds are critical to the performance of the message fuzzer. In our implementation, Tyr collects the messages in the network as the initial seeds. Once Tyr connects to the blockchain network, all messages from the normal nodes will be collected. Any messages which contribute to new code coverage or new behavior will be stored in the seed pool as the initial seeds.

Decision Time Setup: Decision time is critical to the precision of CFB detection. Too short a decision time may cause false positives, while too long affects the efficiency of the testing process. We did an empirical study on how Tyr performs on various decision time setups and found that the six-block confirmation time is a balanced value to help avoid most of the false positives while achieving efficient testing performance. In the field of blockchain, block confirmation time is a commonly used time unit [35], [36].

Bug Analyzer: Tyr collects all received messages of each node as runtime data, sorts them by the timestamp, and stores them in the behavior divergent model. When oracle detectors report a CFB in some target nodes, Tyr records the timestamp and marks it as bug time. When nodes join the consensus network and start receiving well-constructed messages from Tyr, Tyr records the timestamp and marks it as begin time. When a CFB occurs, Tyr replays these messages between begin time and bug time to reproduce the CFB and help analyze the root cause.

VI. Evaluation

To evaluate the effectiveness of Tyr, we compared it with three state-of-the-art tools: Peach [9], Twins [11], [37] and Fluffy [10] on six widely used blockchain networks. We ran each blockchain network with a 10-nodes setup. The entire network is isolated and set up locally. All nodes under test are honest nodes and are assumed to perform correctly. All the experiments are conducted several times, and the average values are used in this paper. The experiment environment is a 64-bit machine with 128 CPU cores (AMD EPYC 7742 64-Core Processor). The OS of this machine is Ubuntu 20.04.2 LTS, and the main memory is 512 GB. We design experiments to address the following research questions:

- **RQ1:** Is Tyr effective in finding CFBs of real-world blockchain systems?
- **RQ2:** Can Tyr cover more code of blockchain systems compared with state-of-the-art tools?
- **RQ3:** How does the decision time influence the efficiency and false positives of Tyr?
- **RQ4:** Does the behavior divergent model effectively improve testing performance?

A. CFBs in Blockchain Systems

We applied Tyr on all 6 target blockchain networks for CFBs detection. For comparison, we ran Peach on the same blockchain networks. Since Fluffy only supports EVM, we ran it on Ethereum and Quorum; Twins is implemented for DiemBFT, so we only ran it on Diem. Each experiment is conducted for 24 hours. In total, Tyr found 20 CFBs on six different target blockchain networks with 5 in Fabric, 7 in FISCO-BCOS, 2 in Quorum, 1 in Diem, 3 in Go-Ethereum,

TABLE I

CFBs WERE FOUND BY Tyr ON SIX BLOCKCHAIN SYSTEMS WITHIN 24 HOURS. Tyr FOUND FIVE BUGS IN FABRIC, SEVEN BUGS IN FISCO-BCOS, TWO BUGS IN QUORUM, ONE BUG IN DIEM, THREE BUGS IN GO-ETHEREUM, AND TWO BUGS IN EOS. EVEN ENHANCED BY THE PROPOSED FOUR ORACLE AUTOMATION, TWINS, FLUFFY, AND PEACH ONLY FOUND ZERO, ZERO, AND SIX BUGS, RESPECTIVELY.

#	Platform	Bug Type	Bug Description	Identifier
1	Fabric	Integrity	Missing Deletion of in-flight when syncing past the in-flight sequence.	CVE-2022-26297
2	Fabric	Safety	Asynchronous sync procedures cause some proposals to be double processed.	CVE-2022-26298
3	Fabric	Integrity	Repeat malicious consensus messages makes some honest nodes to be disconnected.	Bug#18167
4	Fabric	Fairness	Various viewchange message sequences make some nodes always skip leader.	Bug#17950
5	Fabric	Liveness	Random newView causes abnormal high-frequent viewchange and chaos in the network.	Bug#17875
6	FISCO-BCOS	Liveness	The nodes change view frequently and stop generating blocks.	CVE-2022-26534
7	FISCO-BCOS	Liveness	Transaction handling process is stuck after confusing nodes with different transaction headers.	Bug#2206
8	FISCO-BCOS	Liveness	Multi-thread bugs cause some transactions cannot to be executed anymore.	Bug#2204
9	FISCO-BCOS	Liveness	Some transactions cannot be processed correctly due to a deadlock.	Bug#2133
10	FISCO-BCOS	Liveness	Lack of the verification of the packet header and the view-change is continuously triggered.	Bug#2448
11	FISCO-BCOS	Safety	A malicious leader may fake a proposal's header and transactions cannot be processed .	Bug#2307
12	FISCO-BCOS	Fairness	A malicious node can always be the leader, thus stop producing new blocks..	CVE-2022-28937
13	Quorum	Liveness	Transactions get stuck in a pending state after receiving incorrect gas from a malicious node.	Bug#1371
14	Quorum	Integrity	Serial of malicious sync messages cause repeated "Full sync failed ", isolate normal node.	Bug#1107
15	Diem	Fairness	Malicious nodes affect the QC commit and the leader's reputation and cause unfair leader selection.	Bug#10362
16	Go-Ethereum	Integrity	Geth nodes no longer sync with Parity nodes after keep receiving malicious sync messages.	Bug#25243
17	Go-Ethereum	Integrity	Keep rejecting blocks and stopping the block syncing procedure, leading to node isolation.	Bug#24448
18	Go-Ethereum	Liveness	The client stopped transaction processing after receiving plenty of re-connection requests.	Bug#24832
19	EOS	Liveness	The producer node crashes when generating a test account through the txn_test_gen_plugin.	CVE-2022-26300
20	EOS	Integrity	Isolation occurs when multiple nodes produce blocks with the same index at the same time.	Bug#11063

and 2 in EOS. The detailed information on these previously unknown bugs is presented in Table I.

For the CFBs found by Tyr, all of them have been confirmed by the corresponding vendors, and 5 have been assigned as CVEs in U.S. National Vulnerability Database, the rest is in the CVE review process. 9 of the detected bugs (#5, #6, #7, #8, #9, #10, #13, #18, #19) violate the liveness properties of blockchain consensus protocol which miss or even stop executing valid transactions. Bugs #2 and #12 trigger the safety automation which results in executing some invalid transactions that may lead to potential economic losses. Six bugs #1, #3, #14, #16, #17, and #20 are detected by Integrity oracle, which leads to node isolation and network partition eventually. Bug #4, #12, and #15 make target nodes have more probability to be elected as leaders, violating the fairness of the leader election process.

False Negatives Evaluation: In our 24-hour experiments, Twins, Fluffy, and Peach did not find any bugs due to the lack of oracle detectors. To help analyze false negatives of Tyr, we have done further experiments and enhanced Peach with our 4 well-designed CFB detectors. Peach successfully detected 6 CFBs (#3, #5, #8, #9, #11, and #20), which demonstrates that our oracle is effective. However, the rest of the 14 bugs were not found by Peach because these CFBs are hidden in the deep path. To trigger them, many nodes with different interactions in multiple phases should be processed. With the help of the behavior divergent model, Tyr successfully detected all 20 logic vulnerabilities, proving the effectiveness of Tyr in detecting CFBs in real-world blockchain systems. Compared with other tools, Tyr found all the bugs that other tools found.

To better evaluate false negatives of Tyr, we also did an experiment by detecting known bugs. We first collected the latest 20 known CFBs from GitHub, and then used Tyr to detect them. The detailed bug information can be found in Section Appendix XI-B. Results show 90% of bugs can be detected by Tyr. Bug#11 and Bug#17 are not detected by Tyr because their root causes are data race, which is hard to reproduce within the 24-hour experiment.

1) Case Study: Now we use two cases to illustrate how the CFBs detected by Tyr affect the whole blockchain network. **The first case is the bug #1 listed in Table I**. This bug is an integrity CFB that causes some nodes to be isolated from others in the blockchain network. It is found in version 1.4 of Fabric and has been assigned with a CVE ID: CVE-2022-26297. The code snippet in figure 10 describes the detailed information of this integrity vulnerability.

Struct 'inFlight' stores all in-flight 'viewChange' messages which are waiting to be processed. If a node receives a 'viewChange' message with a larger 'viewId', then the function 'sync()' will be called to update the current 'view' from other nodes. However, if a malicious node syncs a smaller 'viewId' which is less than the current 'viewId', then this function ends immediately, as shown in line 10. As a result, the current node fails to sync the view and will call this function again. As long as the malicious node keeps synchronizing the smaller view, the current node will never be able to complete the synchronization and will gradually be isolated from the blockchain network. In this way, attackers can utilize this integrity bug to make a network split, causing potential economic losses. This CFB has already been fixed by adding the function 'maybePruneInFlight()' which is implemented to delete the in-flight proposal whose 'viewId' is smaller, as shown in lines 15-19.

In our experiments, this integrity CFB was only found by

```
1  func (c *Controller) sync() (viewNum uint64,
       seq uint64, decisions uint64) {
2     // Block any concurrent sync attempt.
3      c.grabSyncToken()
4      defer c.relinquishSyncToken()
5      syncResponse := c.Synchronizer.Sync()
6      decision := syncResponse.Latest
7  +     c.maybePruneInFlight(*md)
8     // check proposal and current view
9      if syncResponse.ViewId < c.currViewNumber {
10      return 0, 0, 0
11     }
12       ...
13      return view, md.LatestSequence + 1,
            md.DecisionsInView + 1
14 }
15 + func (c *Controller) maybePruneInFlight( ...
       ) {
16 +    inFlight := c.InFlight.InFlightProposal()
17 +      ...
18 +    c.InFlight.clear()
19 +}
```

Fig. 10. An integrity bug which can isolate target nodes in the fabric SmartBFT consensus process.

Tyr. To trigger it, a 'viewChange' message with a larger 'viewId' should be sent to the target node first. Then the function 'sync()' is called by the target node, and this unusual behavior needs to be captured in real-time. After observing this unique behavior, Tyr constantly syncs the 'viewChange' message with a smaller 'viewId'. Eventually, Tyr triggers this bug via the integrity oracle automation.

The second case is the bug # 2 listed in Table I. This is a safety CFB that causes some proposals to be processed more than once. It is found in version 1.4 of Fabric and has been assigned with a CVE ID: CVE-2022-26298. The code snippet in figure 11 describes the details of this vulnerability.

```
1  func (c *Controller) getCurrentViewNumber()
       uint64 {
2     ...
3  +  c.currViewLock.RLock()
4  +  defer c.currViewLock.RUnlock()
5     ...
6  }
7  func (c *Controller) sync() ( ... ) {
8      // we were syncing.
9      defer c.relinquishSyncToken()
10 +    c.syncLock.Lock()
11 +    defer c.syncLock.Unlock()
12     syncResponse := c.Synchronizer.Sync()
13    ...
14 }
15 func (med *MutuallyExclusiveDeliver) Deliver(
       ... ) {
16     ...
17 +    med.C.syncLock.Lock()
18 +    defer med.C.syncLock.Unlock()
19     ...
20 }
```

Fig. 11. A safety bug which can double processing block proposal in fabric SmartBFT consensus system.

Function 'getCurrentViewNumber()' is responsible for getting the current 'viewId'. Function 'sync()' is designed to synchronize 'viewId' from other nodes in the blockchain. The function 'deliver()' is implemented to deliver block proposals. If a node is unaware of the block proposal committed by the rest of the nodes, attempts a view change that fails, and triggers a view change timeout, then the function 'sync()' will be called. However, the sync procedure may then commit that proposal, and the proposal will be then attempted to be committed again once the view change resumes its operation, these two procedures are parallel. This CFB has already been fixed by the developer. Locks are added to ensure that the view change procedure hangs out until the sync procedure has ended, as shown in lines 17-18.

This safety vulnerability is hard to be detected because in most cases, the block delivery procedure and view change procedure are totally independent and can be executed concurrently. Due to network delay or malicious block messages, the target node is missing other nodes' block proposals and starts the delivery process. At the same time, an abnormal 'viewChange' message should be sent to the target node so that the sync procedure can be called. Tyr can easily trigger such concurrent conflict because the unique block delivery behavior and view sync behavior will be captured and analyzed in time. Based on the behavior divergence guided model, Tyr dynamically adjusts the message selection process and has a high probability to send both 'viewChange' and 'blockDeliver' messages simultaneously. Eventually, Tyr triggers this bug and detects it via the safety oracle automation.

Lessons from the cases. From the above two cases, we find that CFBs in blockchain tend to cause severe consequences. Either network split or proposal double processing may cause unrecoverable loss. From the first case, we can learn that the view sync procedure should take abnormal 'viewChange' messages from malicious nodes into account and implement corresponding defense mechanisms. The second case shows that a developer should always be aware of concurrent processes and validate them for conflict.

B. Effectiveness on Code Coverage

To evaluate the capacity of Tyr in code coverage of blockchain systems, we set up a 10-node network for each target blockchain and compared Tyr with other state-of-art tools in the same experimental setup. According to the empirical study of absfuzz [38], code, block, and branch coverage are highly correlated. Therefore, we collected the branch coverage for each tool in 24 hours as the evaluation metric. The statistics are shown in Table II. In conclusion, Tyr covers 43.6% and 228.2% more branches than Peach and Fluffy on Go-Ethereum and Quorum. Tyr covers 27.5% and 297.1% more branches than Peach and Twins on Diem. On Fabric, FISCO-BCOS, and EOS, Tyr covers 43.5%, 22.7%, and 13.5% more branches than Peach respectively.

Since Fluffy only supports Ethereum, we just compared it with Tyr on Go-Ethereum and Quorum. Tyr covers twice as many branches as Fluffy. This is because Fluffy is designed to test the transaction execution logic of EVM. Most of the consensus processes such as leader election, block verification,

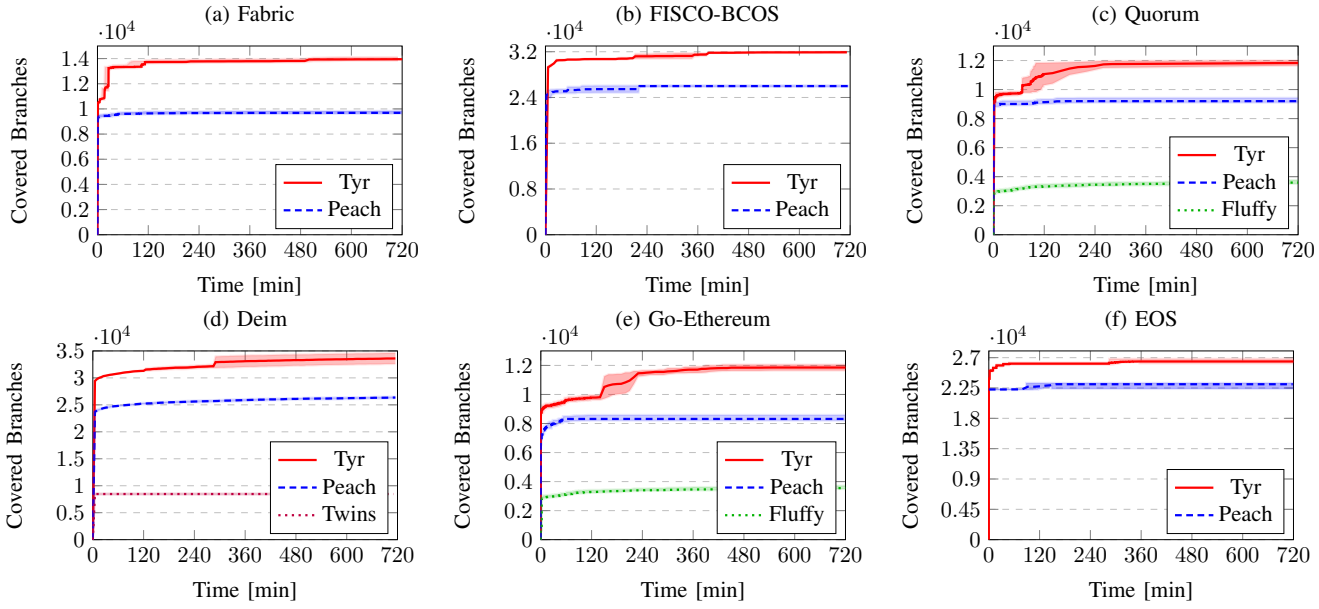

Fig. 12. Coverage trends evaluated for Tyr, Peach, Twins and Fluffy on Fabric, FISCO-BCOS, Quorum, Diem, Go-Ethereum and EOS in 12 hours. Fluffy only supports Quorum and Go-Ethereum. Twins only supports DiemBFT. Compared with these state-of-the-art tools, Tyr shows better branch coverage all the time on all the target blockchain networks.

TABLE II
BRANCH COVERAGE ON SIX BLOCKCHAIN NETWORKS IN 24 HOURS. FLUFFY ONLY SUPPORTS ETHEREUM AND QUORUM, TWINS ONLY SUPPORTS DIEM. Tyr COVERS 27.3%, 228.2%, AND 297.1% MORE BRANCHES COMPARED WITH PEACH, FLUFFY, AND TWINS.

	Peach	Fluffy	Twins	Tyr
Fabric	9731	-	-	13964
FISCO BCOS	25992	-	-	31902
Quorum	8766	3756	-	12662
Diem	26366	-	8463	33606
Go-Ethereum	8315	3712	-	11851
EOS	23305	-	-	26453

etc. cannot be tested by Fluffy. Twins only supports DiemBFT, so we ran it on Diem. Tyr covers almost three times as many branches as Twins. The reason is that Twins just tests BFT protocols in a mock environment, missing the actual transaction execution and block verification. Compared with Peach, Tyr covers over 27.3% more branches on all 6 blockchain networks because Tyr can analyze nodes' behaviors data in real time and send message inputs accordingly. While Peach only generates static messages based on predefined state models without utilizing runtime information of target nodes.

To observe the trends of coverage growth over time, we record the branch coverage every minute over 12 hours, as shown in figure 12. The shadow represents the range of values in multiple times of experiments. The results varied within +/- 5%. The line represents the average value. According to the figure, Tyr's branch coverage grows significantly in the first 4 hours on all six target blockchain networks. After around 12 hours, the coverage of Tyr gradually converges (only less than 1% coverage improvement is observed). But after 12 hours, Tyr can still generate some boundary values to trigger more CFBs. As for Peach and Fluffy, the coverage grows rapidly in the first 60-120 minutes. After that, the message inputs they generated can hardly cover more branches as it does at the beginning of the fuzzing process. As a unit test generator, the coverage of Twins does not change over time because Twins can not collect the feedback information of target nodes and generate new test inputs accordingly.

Compared to other tools Peach, Fluffy and Twins, we can see from figures (a) - (f) that Tyr always achieves more branch coverage and its coverage grows much faster. The main reason is that Tyr can collect their behavior information in real time. Benefiting from this, Tyr utilizes the behavior guided fuzzing algorithm to constantly select and mutate a significant number of high-quality message seeds. In this way, the well-generated message inputs can reach deeper code logic, achieving better fuzzing performance.

C. Efficiency and False Positives Analysis

To evaluate the influence of the decision time on fuzz efficiency and false positives, we also conduct the experiment that runs Tyr on different decision time values, from 1 bct (block confirmation time) to 15 bct. First, we recorded the number of transactions generated by Tyr per second and calculated the average TPS (transaction per second). Then, we collected all the consensus failure bugs reported by Tyr in 24 hours on all six target blockchain systems. Then we manually analyzed the false positives of those bugs.

As shown in table III, as the decision time increases, the transaction generation rate of Tyr declines slightly, from 377.8 TPS to 274.6 TPS. However, the number of false positives reported by Tyr decreases rapidly, from 67 to 0. These

TABLE III
THE AVERAGE TPS, FALSE POSITIVES AND TRUE POSITIVES OF Tyr ON VARIOUS DECISION TIME SETUPS.

Decision Time	Average TPS	False Positives	True Positives
1 bct	377.8	67	20
2 bct	362.5	16	20
4 bct	345.2	7	20
6 bct	324.7	0	20
8 bct	301.1	0	20
10 bct	274.6	0	20

statistics demonstrate that although a long decision time affects the efficiency of transaction generation, it effectively reduces false positives, which adequately answers **RQ3**. According to the experimental results, a six-block confirmation time is sufficient to remove all the false positives for all 6 target blockchain systems. The main reason is that after a six-block confirmation time, nearly all blocks are in an irreversible state, where all data are fixed and can be calculated. The number of real CFBs has not changed over different decision time values, which demonstrates that the decision time has no effect on the true positives.

D. Effectiveness of Behavior Guided Fuzzing

To evaluate the effectiveness of the behavior divergent model, we also conducted the experiment that compares Tyr with Tyr^-, the version of Tyr which disables the behavior divergent model and generates messages randomly. We collected the branch coverage as well as the number of bugs in 24 hours on all six target blockchain networks.

TABLE IV
COMPARISON OF Tyr^- AND Tyr ON 6 TARGET BLOCKCHAIN NETWORKS IN 24 HOURS. Tyr WITH BEHAVIOR DIVERGENT MODEL DETECTS 12 MORE BUGS AND COVERS 18.5% MORE BRANCHES.

	Number of Bugs		Branch Coverage	
	Tyr^-	Tyr	Tyr^-	Tyr
Fabric	2	5	10725	13964
FISCO BCOS	4	8	29426	31902
Quorum	1	2	9228	12662
Diem	0	1	27523	33606
Go-Ehteruem	0	2	8632	11851
EOS	1	2	24513	26453
Total	8	20	110047	130438
Improvement	-	+150%	-	+18.5%

As shown in table IV, with the help of the behavior divergent model, Tyr can detect all 20 bugs in 24 hours, while Tyr^- only detects 8 of them. Besides that, compared with Tyr^-, Tyr always achieves more branch coverage in all 6 target blockchain systems. In total, Tyr covers 20,391 more branches, achieving an improvement of 18.5% branch coverage. The main reason is that most CFBs are hidden in the deep code path. To cover them, many nodes with different behaviors should be executed first. Benefiting from the behavior divergent model, Tyr successfully explores more deep logic paths in all target blockchain systems. Thus, we can conclude that the behavior divergent testing strategy achieves better performance on both code coverage and bug detection. It significantly improves the testing performance, which adequately answers **RQ4**.

VII. DISCUSSION

In this section, we will discuss some advantages and limitations of Tyr and our future work.

Generability of Tyr. Currently, Tyr has supported 4 CFB oracles based on blockchain consensus properties, including Liveness, Safety, Integrity, and Fairness. Tyr has already been adapted to six widely used blockchain systems and has found 20 previously unknown bugs with 5 CVEs assigned. However, in practice, there are still some other types of bugs, such as memory-related vulnerabilities, privacy issues, etc. hidden in the implementation of the blockchain systems. It will also pose a threat to the ecology of blockchain systems.

Tyr is designed to be generalizable to other scenarios. Tyr can find other types of bugs if equipped with corresponding oracles. For example, if enhanced with ASAN (Address Sanitizer [39]), Tyr can find memory-related bugs. Take CVE-2021-35041 for example, it's a memory unfree bug and can be easily detected by Tyr. If extended with privacy oracle, by trying to access private data and checking its visibility, Tyr can also find privacy issues in blockchain networks. Take the code snippet in figure 17 as an example. This is a privacy vulnerability in Quorum. When an unauthorized node makes a call to a private contract, the return value should be an empty string. But in practice, the function returns an error code via JSON RPC rather than an empty string, which leaks information about the contract's behavior. By keeping calling private contracts and checking whether the result of the unauthorized call is an empty string, Tyr can easily detect this privacy vulnerability.

However, the privacy perspective in blockchain varies for personal and organizational data. Although privacy rules are applicable to personal data, more stringent privacy rules apply to sensitive and organizational data. The flexibility of blockchain privacy makes it hard to design a general privacy oracle for various blockchain systems. More works need to be explored to address this challenge. Privacy and other types of bugs deserve our attention in the future.

Finer-grained Runtime Information. At present, Tyr utilizes a behavior divergent model to make distributed nodes behave as differently as possible. Through analyzing nodes' leader election times, transaction execution, and code coverage in real-time, Tyr triggers the oracle automation more efficiently and covers 20,391 more branches, and detects 12 more CFBs compared with Tyr^-. However, some finer-grained runtime information of target nodes can also be collected as better guidance for the testing process.

For example, the timestamp of messages can be collected and analyzed in each target node. The timestamp determines the exact moment in which the messages have been sent and received by the blockchain network. As a temporal parameter, each node's sending and receiving message sequence can be constructed from it. Based on that, Tyr may construct

more efficient message sequences as inputs and may achieve better testing performance. However, finer-grained information analysis brings higher overhead. How to find a balance point needs to be explored in the future.

VIII. Related Work

Blockchain Consensus Network Testing. Fluffy [10] and Twins [37] are representative tools for blockchain consensus network testing and are also the work most relevant to this paper. The core idea of Fluffy [10] is differential testing. It generates multi-transactions and uses different Ethereum virtual machines as cross-referencing factors to observe abnormal behaviors. Although the execution of the transaction sequence involves consensus logic, what Fluffy can expose is only the vulnerabilities in the virtual machine implementation. Twins [37] is an automated unit test generator of Byzantine attacks. It replicates node information, systematically generates three kinds of Byzantine attack scenarios, and executes them in a controlled manner. However, Twins' attack scenarios are limited and miss the runtime information in the consensus process. There are also some tools designed for consensus protocols in general distributed systems, such as DEMi [40]. Its core idea is to minimize faulty executions in Raft [41] and Spark [42] by filtering out key events. But it does not focus on detecting vulnerabilities in the consensus network.

Blockchain Smart Contract Testing. Some work also focuses on detecting smart contracts vulnerabilities that may also influence the security of blockchain systems [43]. For example, Oyente [44] is a symbolic execution tool for bug detection in solidity contracts by exploring a contract CFG. However, Oyente does not support detection in cross-contract scenarios. To handle this problem, Pluto [45] supports inter-contract call modeling. In this way, Pluto found some previously-unknown inter-contract vulnerabilities. SCStudio [46] integrates several commonly-used detection tools such as Securify [47] and Pied-Piper [48] to provide more accurate analysis results.

Logic Bug Detection. Logic bugs usually refer to semantic vulnerabilities in the implementation of the code, and the detection target is often the improper behavior or execution results of the program rather than crashes. Elle [49] is a novel checker which infers a dependency graph among client-observed transactions. By identifying cycles in the graph, Elle can expose a particular set of transactions which implies an anomaly has occurred that violates the consistency properties. Symbolic QED [50] aims to find critical design flaws in integrated circuits. Its core idea is combining redundant execution and control flow checking with bounded model checking-based formal analysis. For file systems, Hydra [51] designs an in-house crash consistency checker, the SibylFS oracle, and the existing file system-specific assertions to find three common types of semantic bugs. Modulo [52] checks consistency for distributed databases by generating test inputs and injecting them into tested systems. Differential fuzzing is also an effective way to find logic bugs in different fields. DPIFuzz [53] generates and mutates QUIC streams to compare the server-side interpretations of different QUIC implementations.

Network Protocol Fuzzing. Fuzzing is an effective method to detect bugs in network protocol implementations. For traditional network protocols, fuzzers are generally divided into two types: generation-based and mutation-based. The former reads the user-provided data models to obtain the format specification of each element, then utilizes it to complete test seeds. Representative tools are Peach [9], Sulley [54] and their extensions [55], [56]. However, these fuzzers rely on manual efforts in the acquisition of the protocol model, which leads to low testing effectiveness. The latter uses valid inputs and modifies them via mutation operations. Due to the lack of targeted oracle definitions and strategy guidance, most of them are inefficient in triggering deep logic vulnerabilities.

Main Difference. Different from the traditional fuzzers which focus on coverage-guided and state-guided algorithms, to the best of our knowledge, Tyr is the first behavior-guided fuzzer for diverging blockchain's behavior and detecting CFB in blockchain systems. Tyr designs 4 CFB oracle detectors and collects nodes' behaviors information in real time for testing guidance. With well-defined oracles and efficient behavior divergent models, Tyr can effectively trigger and monitor the abnormal behavior of nodes in the blockchain network in addition to crashes. According to the behavior information feedback, Tyr will preferentially select the message that triggers the abnormal behavior of the node for more mutation operations, thus enabling Tyr to reach deeper consensus code paths more easily. Furthermore, based on the scalability of the testing framework, Tyr can be quickly adapted to different blockchain systems.

IX. Conclusion

In this paper, we propose Tyr, an automatic testing tool for detecting CFBs in blockchain systems based on the behavior divergent model. Tyr first designs 4 CFB oracle detectors for monitoring the behaviors of nodes and analyzing the violation of consensus properties. Tyr employs a behavior divergent model to collect real-time behaviors states and divergent distributed nodes' behaviors in the system as much as possible. We implement and evaluate Tyr on 6 commercial blockchain systems: Fabric, FISCO-BCOS, Quorum, Diem, Go-Ethereum, and EOS. The results show that Tyr covers 27.3%, 228.2%, and 297.1% more branches on average compared with state-of-the-art tools. Tyr successfully detected 20 previously unknown CFBs with 5 CVE IDs assigned. Our future work will consider enhancing Tyr with finer feedback guidance in finding more CFBs.

X. Acknowledgements

This research is sponsored in part by the National Key Research and Development Project (No. 2022YFB3104000), NSFC Program (No. 62022046, 92167101, U1911401), and Webank Scholar Project (20212001829).

REFERENCES

[1] Y. Lu, "The blockchain: State-of-the-art and research challenges," *Journal of Industrial Information Integration*, vol. 15, pp. 80–90, 2019.

[2] CVE-2021-39137, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-39137, 2021.

[3] CVE-2020-26265, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-26265, 2020.

[4] CVE-2020-26241, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-26241, 2021.

[5] C. Harper, "Ethereum's 'unannounced hard fork' was trying to prevent the very disruption it caused," https://finance.yahoo.com/news/ethereum-unannounced-hard-fork-trying-230144206.html, 2022.

[6] INFURA, "The infura platform," https://infura.io/product/overview, 2022.

[7] makerdao, "The maker protocol: Makerdao's multi-collateral dai (mcd) system," https://makerdao.com/en/whitepaper/, 2022.

[8] e. a. Hayden Adams, "Uniswap v3 core," https://uniswap.org/whitepaper-v3.pdf, 2022.

[9] M. Eddington, "protocol-fuzzer-ce," https://gitlab.com/gitlab-org/security-products/protocol-fuzzer-ce, 2021, accessed at April 6, 2022.

[10] Y. Yang, T. Kim, and B.-G. Chun, "Finding consensus bugs in ethereum via multi-transaction differential fuzzing," in *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*, 2021, pp. 349–365.

[11] S. Bano, A. Sonnino, A. Chursin, D. Perelman, Z. Li, A. Ching, and D. Malkhi, "Twins: Bft systems made robust," in *25th International Conference on Principles of Distributed Systems (OPODIS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[12] Go-Ethereum, "Official go implementation of the ethereum protocol," https://geth.ethereum.org/, 2022.

[13] openethereum, "Parity ethereum. the fastest and most advanced ethereum client," https://github.com/openethereum/parity-ethereum, 2022.

[14] Hyperledger, "Hyperledger fabric," https://www.hyperledger.org/use/fabric, 2021, accessed at April 6, 2022.

[15] FISCO, "Fisco bcos," https://github.com/FISCO-BCOS/FISCO-BCOS, 2021, accessed at April 6, 2022.

[16] L. Ren, "Analysis of nakamoto consensus," *Cryptology ePrint Archive*, 2019.

[17] J. FRANKENFIELD, "Proof of work," https://www.investopedia.com/terms/p/proof-work.asp, 2021, accessed at April 6, 2022.

[18] ——, "Proof of stake," https://www.investopedia.com/terms/p/proof-stake-pos.asp, 2021, accessed at April 6, 2022.

[19] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, no. 1999, 1999, pp. 173–186.

[20] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus in the lens of blockchain," *arXiv preprint arXiv:1803.05069*, 2018.

[21] E. A. Brewer, "Towards robust distributed systems," in *PODC*, vol. 7, no. 10.1145. Portland, OR, 2000, pp. 343 477–343 502.

[22] S. Zhang and J.-H. Lee, "Double-spending with a sybil attack in the bitcoin decentralized network," *IEEE transactions on Industrial Informatics*, vol. 15, no. 10, pp. 5715–5722, 2019.

[23] A. Begum, A. Tareq, M. Sultana, M. Sohel, T. Rahman, and A. Sarwar, "Blockchain attacks analysis and a model to solve double spending attack," *International Journal of Machine Learning and Computing*, vol. 10, no. 2, pp. 352–357, 2020.

[24] CVE-2022-26534, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-26534, 2022.

[25] Ethereum, "Welcome to ethereum," https://ethereum.org/en/, 2021, accessed at April 6, 2022.

[26] B. Xu, D. Luthra, Z. Cole, and N. Blakely, "Eos: An architectural, performance, and economic analysis," *Retrieved June*, vol. 11, p. 2019, 2018.

[27] Diem, "Welcome to the diem project," https://www.diem.com/en-us/, 2021, accessed at April 6, 2022.

[28] J. M. Chase, "Quorum white paper," *Accessed: Jan*, vol. 17, p. 2019, 2016.

[29] Google, "Protocol buffer structure encoding," https://developers.google.com/protocol-buffers/docs/encoding, 2021, accessed at April 6, 2022.

[30] J. Polge, J. Robert, and Y. Le Traon, "Permissioned blockchain frameworks in the industry: A comparison," *Ict Express*, vol. 7, no. 2, pp. 229–233, 2021.

[31] CoinMarketCap, "Coinmarketcap," https://coinmarketcap.com, 2022, accessed at April 6, 2022.

[32] G. documentation, "A test coverage program," https://gcc.gnu.org/onlinedocs/gcc/Gcov.html, 2022.

[33] ——, "Codecov and grcov," https://about.codecov.io/tool/grcov/, 2022.

[34] ——, "Googletest coverage," https://github.com/google/googletest, 2022.

[35] J. Stauffer, "What is a block confirmation on ethereum," https://jaredstauffer.medium.com/what-is-a-block-confirmation-on-ethereum-e27d29ca8c01, 2022.

[36] N. REIFF, "How does a block chain prevent double-spending of bitcoins," https://www.investopedia.com/ask/answers/061915/how-does-block-chain-prevent-doublespending-bitcoins.asp, 2022.

[37] S. Bano, A. Sonnino, A. Chursin, D. Perelman, and D. Malkhi, "Twins: White-glove approach for bft testing," *arXiv preprint arXiv:2004.10617*, 2020.

[38] C. Salls, A. Machiry, A. Doupe, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, "Exploring abstraction functions in fuzzing," in *2020 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2020, pp. 1–9.

[39] C. . documentation, "Address sanitizer," https://clang.llvm.org/docs/AddressSanitizer.html, 2021.

[40] C. Scott, V. Brajkovic, G. Necula, A. Krishnamurthy, and S. Shenker, "Minimizing faulty executions of distributed systems," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, 2016, pp. 291–309.

[41] D. Ongaro and J. Ousterhout, "The raft consensus algorithm," 2015.

[42] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10)*, 2010.

[43] M. Ren, Z. Yin, F. Ma, Z. Xu, Y. Jiang, C. Sun, H. Li, and Y. Cai, "Empirical evaluation of smart contract testing: what is the best choice?" in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021, pp. 566–579.

[44] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 254–269.

[45] F. Ma, Z. Xu, M. Ren, Z. Yin, Y. Chen, L. Qiao, B. Gu, H. Li, Y. Jiang, and J. Sun, "Pluto: Exposing vulnerabilities in inter-contract scenarios," *IEEE Transactions on Software Engineering*, 2021.

[46] M. Ren, F. Ma, Z. Yin, Y. Fu, H. Li, W. Chang, and Y. Jiang, "Making smart contract development more secure and easier," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 1360–1370.

[47] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Buenzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 67–82.

[48] F. Ma, M. Ren, L. Ouyang, Y. Chen, J. Zhu, T. Chen, Y. Zheng, X. Dai, Y. Jiang, and J. Sun, "Pied-piper: Revealing the backdoor threats in ethereum erc token contracts," *ACM Transactions on Software Engineering and Methodology*, 2022.

[49] K. Kingsbury and P. Alvaro, "Elle: Inferring isolation anomalies from experimental observations," *arXiv preprint arXiv:2003.10554*, 2020.

[50] E. Singh, D. Lin, C. Barrett, and S. Mitra, "Logic bug detection and localization using symbolic quick error detection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[51] S. Kim, M. Xu, S. Kashyap, J. Yoon, W. Xu, and T. Kim, "Finding semantic bugs in file systems with an extensible fuzzing framework," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 147–161.

[52] B. H. Kim, T. Kim, and D. Lie, "Modulo: Finding convergence failure bugs in distributed systems with divergence resync models."

[53] G. S. Reen and C. Rossow, "Dpifuzz: A differential fuzzing framework to detect dpi elusion strategies for quic," in *Annual Computer Security Applications Conference*, 2020, pp. 332–344.

[54] P. Amini and A. Portnoy, "Sulley," https://github.com/OpenRCE/sulley, 2012.

[55] Z. Luo, F. Zuo, Y. Shen, X. Jiao, W. Chang, and Y. Jiang, "Ics protocol fuzzing: coverage guided packet crack and generation," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[56] F. Zuo, Z. Luo, J. Yu, Z. Liu, and Y. Jiang, "Pavfuzz: State-sensitive fuzz testing of protocols in autonomous vehicles."
[57] A. P. Joshi, M. Han, and Y. Wang, "A survey on security and privacy issues of blockchain technology," *Mathematical foundations of computing*, vol. 1, no. 2, p. 121, 2018.

XI. Appendix

A. An Example of Tyr Adaption

When adapting Tyr to a new blockchain system, two key interfaces are needed to implement. The first interface is 'BlockExtract()' which is responsible for extracting key consensus data from struct 'Block'. The second interface is 'p2p.send()', to send messages and transactions generated by Tyr to the target nodes in blockchain system. Here we take Tyr adaption in Go-Ethereum for example.

Figure 13 illustrates how Tyr interacts with Go-Ethereum network through these two interfaces. Tyr first extracts the $Height_{num}$, TX and $State$ from blocks through BlockExtract(). Then Tyr calculates the divergences based on the extracted consensus data and converts them to the BDMs. Then, Tyr utilizes Oracle Detector to identify whether they violate the oracle definition and reports CFBs if violations are found. Based on the BDMs, Tyr mutates consensus messages by Algorithm 1. Finally, Tyr gossips the mutated messages to the Ethereum network via p2p.send().

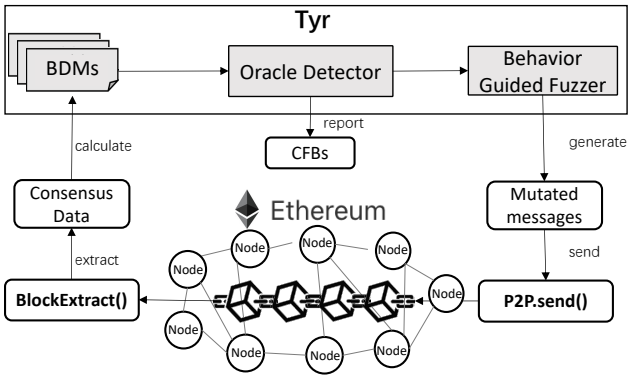

Fig. 13. Workflow of Tyr implementation in Go-Ethereum. Tyr first analyses consensus data through the interface 'BlockExtract' and calculates behavior divergences. Based on them, Tyr mutates messages and sends them to the network through the interface 'P2P.send()'.

As the code in Figure 14 shows, the 'BlockExtract' interface will check whether the target chain ($provide\_url + chain$) exists. Then it extracts block data in the target chain from the $start\_block$ to the end_block. To speed up this process, we use a multi-threaded parallel calculation with the number of threads determined by the value of $max_workers$. The statement $job.run()$ will call the concrete block parser provided by Go-Ethereum official implementation. Finally, the interface 'BlockExtract' outputs the consensus data which contains two main parts – the block data part and the transaction data part. The block data includes block number, miner identification, hash value, timestamps, etc. The transaction data includes all transaction execution values. All these data are sent to Tyr for calculating node behavior divergence and updating BDMs.

```
def extract_blocks(start_block, end_block,
    batch_size, provider_uri, max_workers,
    blocks_output,  transactions_output,
    chain='ethereum'):
"""Exports blocks and transactions."""
   provider_uri =
       check_classic_provider_uri(chain,
       provider_uri)
   if blocks_output is None and
       transactions_output is None:
      raise ValueError('Either --blocks-output
          or --transactions-output options must
          be provided')
   job = ExportBlocksJob(
      start_block=start_block,
      end_block=end_block,
      batch_size=batch_size,
      batch_web3_provider=ThreadLocalProxy(
      lambda:
          get_provider_from_uri(provider_uri,
          batch=True)),
      max_workers=max_workers,
      item_exporter=
      blocks_item_exporter(blocks_output,
          txs_output),
      export_blocks=blocks_output is not None,
      export_transactions=txs_output is not None)
   job.run()
```

Fig. 14. An example of the BlockExtract interface for Go-Ethereum. Tyr uses this interface to extract consensus data from blocks.

As the code in Figure 15 shows, the 'P2P.send()' interface will first check whether the target nodes $targets$ are in the connected neighborhood nodes. Then it constructs Ethereum-style messages based on msg_code and $data$ and encodes them with concrete message formatter provided by Go-Ethereum official implementation. Finally, interface 'P2P.send()' sends all mutated messages to target nodes in the blockchain network.

B. Bug Reproduce Evaluation

To better understand the false negatives of Tyr, we also did an experiment by detecting known bugs. We first collected the latest 20 known CFBs from GitHub (4 bugs from fabric, 3 bugs from FISCO-BCOS, 2 bugs from Quorum, 3 bugs from Diem, 5 bugs from Go-ethereum and 3 bugs from EOS). Then we ran Tyr in each blockchain platform for 24 hours to try to detect them. The detailed bug information can be listed in Table V. Results show 90% bugs (18 bugs) can be detected by Tyr. Only Bug#11 and Bug#17 are not detected by Tyr because their root causes are data race, which is hard to reproduce within the 24-hour experiment.

C. Finding Bugs in Other Scenarios

In addition to consensus failure bugs, memory-related bugs in blockchain implementations are also important. Thanks to the generability of Tyr, Tyr can find memory-related bugs if enhanced with ASAN (Address Sanitizer [39]). Take CVE-2021-35041 for example, it's a memory unfree bug. The code

```
def P2P_send(self, targets, msg_code, data):
"""sends messages."""
   if len(self.nodes_connected) < len(targets):
      self.debug_print("Peers limit reached.")

   for target in targets:
      if not self.check_ip_to_connect(target):
         self.debug_print("connect_to: Cannot
             connect!!")
         targets.remove(target)

   try:
      data = convert_ethereum_msg(msg_code,data)
      self.sock.sendall(targets,
          data.encode_ethereum())

   except Exception as e:
      self.main_node.debug_print(
      "NodeConnection.send: Unexpected
          ercontent/ror:"
      + str(sys.exc_info()[0]))
      self.main_node.debug_print("Exception: " +
          str(e))
      self.terminate_flag.set()
```

Fig. 15. An example of the P2P.send() interface for Go-Ethereum. Tyr uses this interface to send mutated messages to the blockchain network.

TABLE V
THE LATEST 20 KNOWN CFBS FROM GITHUB. WE RUN Tyr FOR 24 HOURS IN EACH BLOCKCHAIN PLATFORMS AND RECORD WHETHER Tyr CAN DETECT THEM.

#	Platform	Link
1	Fabric	https://jira.hyperledger.org/projects/FAB/issues/FAB-18535
2	Fabric	https://jira.hyperledger.org/projects/FAB/issues/FAB-18526
3	Fabric	https://jira.hyperledger.org/projects/FAB/issues/FAB-14470
4	Fabric	https://jira.hyperledger.org/projects/FAB/issues/FAB-18239
5	FISCO-BCOS	https://github.com/FISCO-BCOS/FISCO-BCOS/issues/2254
6	FISCO-BCOS	https://github.com/FISCO-BCOS/FISCO-BCOS/issues/2206
7	FISCO-BCOS	https://github.com/FISCO-BCOS/FISCO-BCOS/issues/2101
8	Quorum	https://github.com/ConsenSys/quorum/issues/1081
9	Quorum	https://github.com/ConsenSys/quorum/issues/1379
10	Diem	https://github.com/diem/diem/issues/8704
11	Diem	https://github.com/diem/diem/issues/8423
12	Diem	https://github.com/diem/diem/issues/7643
13	Go-Ethereum	https://github.com/ethereum/go-ethereum/issues/26022
14	Go-Ethereum	https://github.com/ethereum/go-ethereum/issues/26020
15	Go-Ethereum	https://github.com/ethereum/go-ethereum/issues/25953
16	Go-Ethereum	https://github.com/ethereum/go-ethereum/issues/25787
17	Go-Ethereum	https://github.com/ethereum/go-ethereum/issues/25870
18	EOS	https://github.com/EOSIO/eos/issues/7600 Safety
19	EOS	https://github.com/EOSIO/eos/issues/3835 Safety
20	EOS	https://github.com/EOSIO/eos/issues/10104 Safety

snippet in Figure 16 describes the detailed information of the vulnerability.

The function 'P2PMessageRC2::decode' is used to decode the received packets. The code at line 2 reads the first 4 bytes of data as the length of the received packet. If the current size of the packet is less than the target length, the node believes that the current packet has not been completely received and returns a signal: 'PACKET_INCOMPLETE' whose value is 0. This signal will be handled by the code at line 12, which will read more data from the session. The function 'doRead()' will allocate memory with size m_length for the incoming packet. By constantly sending packets with large m_length

```
ssize_t P2PMessageRC2::decode(...){ ...
   m_length =
       ntohl(*((uint32_t*)&buffer[offset]));
   if (size < m_length) {
// the value of PACKET_INCOMPLETE is 0
    return dev::network::PACKET_INCOMPLETE;
   }
   ...
}
// code for handling the decoding result
ssize_t result =
    message->decode(s->m_data.data(),
    s->m_data.size());
...
else if (result == 0) {
// m_length size of memory is allocated
   s->doRead();
   break;
}
```

Fig. 16. Code snippet that constantly allocate new memory. An attacker can sustain sending maliciously constructed packets to consume all the memory of the honest node's host and break it down.

size, the node will consume the memory sustainably until out of memory. With the help of ASAN, this memory bug can be detected by Tyr.

Furthermore, privacy issue is important in blockchain systems. The goal of privacy protection is to make it extremely difficult for users to access other users' private data [57]. Currently, the techniques for blockchain privacy protection can be divided into two main types: access control and cryptographic methods. Both of them can prevent invalid private data access. If extended with privacy oracle, for example, by constantly trying to query private data and checking its visibility, Tyr can also find privacy issues in blockchain networks. Take the code snippet in figure 17 as an example. This is a privacy vulnerability in Quorum. When an unauthorized node makes a call to a private contract, the return value should be an empty string. But in practice, the function returns an error code via JSON RPC rather than an empty string, which leaks information about the contract's behavior. By keeping calling private contracts and checking whether the result of the unauthorized call is an empty string, Tyr can easily detect this privacy vulnerability.

```
func (s *PublicBlockChainAPI) doCall( ... )
    (string, *big.Int, error) {
   ...
   gp := new(core.GasPool).AddGas(common.MaxBig)
   res, gas, err := core.ApplyMessage(vmenv,
       msg, gp)
   if err := vmError(); err != nil {
      // BUG: the first parameter should be
          common.ToHex(nil), instead of "0x"
      return "0x", common.Big0, err
   }
   return common.ToHex(res), gas, err
}
```

Fig. 17. A privacy vulnerability occurs in Quorum.