2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)

Fast, Dynamic and Robust Byzantine Fault Tolerance Protocol for Consortium Blockchain

Anping Song
School of Computer Engineering and Science
Shanghai University
Shanghai, China
apsong@shu.edu.cn

Jing Wang, Wenjing Yu, Yi Dai, Hongtao Zhu
School of Computer Engineering and Science
Shanghai University
Shanghai, China

***Abstract*—Consortium blockchain is the preferred implementation form of blockchain for cooperation between enterprises. As an indispensable underlying technology of consortium blockchain, the Byzantine fault tolerant protocol guarantees that the replicas in network reach agreement even when there are arbitrary faults of a fraction of the replicas. Byzantine fault tolerant protocols that currently exist for consortium blockchain fail to meet the requirements of practical applications, such as satisfying low algorithm complexity, robustness and dynamic scalability at the same time. In this paper, a new Fast, Dynamic and Robust Byzantine Fault Tolerance (FDRBFT) protocol is proposed to address this problem. It applies random threshold signature consensus scheme, unique cryptographic algorithm and proactive recovery scheme to achieve fast agreement, dynamic scalability and robust system. FDRBFT protocol is implemented on Hyperledger Fabric and its performance is compared with existing widely accepted protocols. Experimental results show that FDRBFT achieves competitive throughput, dynamic scalability and better robustness.**

***Index Terms*—Consortium blockchain, Byzantine fault tolerance, Consensus, Threshold signature, Cryptographic algorithm.**

I. Introduction

Consortium blockchain is the most common type of blockchain in business applications nowadays [1]. Due to its restriction on participates who can join the network and contribute to the consensus of the system state, it provides protected privacy and fine control over replicas as well as their data.

The consensus problem is one of the most important problems in distributed systems. Consensus guarantees that the replicas in network reach agreement even when there are arbitrary faults of a fraction of the replicas. Considering the complexity of consortium blockchain applications in real-world environment, consensus in these applications must be able to tolerate Byzantine failures [2]. Under the assumption that there are no link failures [2], the kind of replicas' failure is either a Fail-stop failure or a Byzantine failure. Fail-stop failures occur when replicas fail by stopping. In the case of a Byzantine failure, the replicas fail arbitrarily.

Practical Byzantine Fault Tolerance (PBFT) [3] is the most widely used consensus protocol in consortium blockchain applications. And this protocol is a milestone in achieving Byzantine fault tolerance. PBFT can be used in practical applications owing to its ability to reduce algorithm complexity from exponential to polynomial. With synchronous setting, PBFT can reach agreement only if $f < n/3$ (f is the maximum of faulty replicas, n is the number of all replicas). However, there are still some problems about PBFT, such as large communication cost, poor resilience, weak scalability and time-consuming recovery when a replica is detected to be faulty [4].

In recent years, the problem of solving consensus in the presence of Byzantine failures is well-studied and a number of protocols for Byzantine fault tolerance (BFT) have been proposed to make up for the shortcomings of PBFT. Most of the researches about BFT protocols in early stages aim at reducing communication cost of replicas and improving throughput. However, Bach [5] points out that it is meaningless to measure a protocol only with its throughput. Subsequent works aim at eliminating the leader's effect, since many BFT protocols depend on a dedicated replica called leader to deliver transactions from client to other consensus replicas, and therefore the leader replica has a significant impact on throughput. Even if there exists several mechanisms to detect and recover from a faulty leader, the process is time-consuming and the leader could be smartly malicious [6]. For example, it can downgrade throughput to the detection threshold, without being detected. Some papers combine dynamic mechanism with PBFT so that a replica could join or exit the network when other replicas work normally. And many works propose recovery scheme to maintain long-term system security. However, none of existing protocols we are aware of solves the problems of PBFT simultaneously.

In this paper, a new protocol called FDRBFT is presented to achieve fast, dynamic and robust Byzantine fault tolerance. FDRBFT reaches fast agreement by deciding leader and consensus replicas randomly, using 2-phases communication between consensus replicas and collecting adequate valid threshold signatures. Compared with previous works, FDRBFT achieves dynamic scalability only by unique cryptographic algorithm and consensus. A new replica could join blockchain network by sending request to client. In addition, FDRBFT employs proactive recovery to maintain long-term system security. Since the latest version of Hyperledger Fabric project provides some Application Programming Interfaces (API) to customize consensus algorithms, FDRBFT is implemented based on this project. Compared with existing widely accepted protocols, we can conclude that FDRBFT achieves competitive throughput, dynamic scalability and better robustness.

The rest of the paper is organized as follows: we revisit the methods applied in BFT protocols and analyze their strengths and weaknesses in Sects.2. We introduce some background concepts in Sects.3. FDRBFT protocol is presented in Sects.4. We evaluate the performance of our protocol and compare it with other widely used protocols in Sects.5. In Sects.6, we make a conclusion and make suggestions for future research.

II. Related Work

According to our studies and observations, a practical BFT protocol for consortium blockchain should have three features: *high throughput*, *dynamic scalability* and *strong robustness*.

Agrawal [7] proposes there are two indicators to measure a consensus algorithm: (1) **round complexity**, the communication cost before the protocol terminates, and (2) **communication complexity**, the amount of information exchanged between replicas during the protocol.

Throughput of consortium blockchain applications is mainly limited by the communication price between replicas, calculation cost for encrypting and decoding message for reaching agreement [8].

A useful way to get high throughput is reducing consensus steps to reduce communication price. Fast Byzantine Fault Tolerance (FBFT) in [9] presents a 2-phases consensus, in the first phase, the leader proposes its value to all acceptors, then in the second phase, the acceptors accept this value and forward it to the learners. The typical PBFT protocol includes 3-phases: pre-prepare, prepare and commit [3]. Giving the fact that commit phase is not necessary, we use a 2-phases consensus in FDRBFT. Bosco [10] achieves one-step Byzantine consensus. However, strongly one-step could be achieved only when $7f<n$ (n is the number of replicas in the system and f is the maximum of Byzantine faulty replicas) and weakly one-step also need to satisfy $5f<n$. However, practical applications need higher fault-tolerate ratio. Another widely used method is random algorithm. Algorand [11] samples some replicas randomly to execute consensus process according to asset or computing power of replicas. The more asset or computing power owned by replica, the more possible it will be sampled.

PBFT relies on leader and view-change for recovering is time-consuming when leader is detected to be faulty. To eliminate the influence of leader, Pease [12] uses a random leader election sub-routine to solve Byzantine agreement with O(n^3) communication and O (1) round. Spin [13] monitors the execution time of leader, if the time overdue the threshold, the leader would be replaced. Redundant Byzantine Fault Tolerance (RBFT) [6] sets two instances with different leader and compare their throughput. Once the throughput of master instance is lower than the throughput of backup instance, the backup instance would replace the master instance. Borran [14] achieves a leader-free consensus. FDRBFT uses random leader election to avoid the adverse influence of leader, and reduces the communication price for recovery. Kursawe [15] proposes optimistic response mechanism to improve throughput. However, the performance of protocols with optimistic mechanisms are not always that good. In some specific cases, analyzed in [16], they could even be slower than PBFT. In FDRBFT, we use mature threshold signatures as our optimistic method. Once collecting enough signatures, the process moves forward.

PBFT is based on static Client/Server (C/S) structure so the number and identity of replicas in the network must be determined before startup and replicas could not be added or exited dynamically. So some protocols absorb the dynamic part of other consensus protocols and combine it with PBFT. Delegated Practical Byzantine Fault Tolerance (DPBFT) [4] combines the authorization mechanism of Delegated Proof of Stake (DPOS) [17] and PBFT, supplemented by the downgrade mechanism with score. And the consensus accounting representative can be dynamically updated. The protocol is slightly concentrated and contrary to the original intention of the blockchain. The protocol proposed in [18] combines heartbeat messages of Raft [19] and PBFT. It is usually used for private and licensed networks. In these networks, a replica could be a leader, a follower or a candidate. A leader is responsible for sending copies of history blocks to followers. It would regularly informing followers its survival by sending heartbeat messages. A new replica could join the system directly as a follower to get necessary blocks and public keys of other replicas. Once information is ready, it can participate in the next round of leader election. In Bitcoin, a replica in network could invite a new replica by sending a transaction to the new replica's address. Inspired by it, FDRBFT uses unique cryptographic algorithm to achieve a dynamic authentication mechanism. This mechanism could control the joining and exiting of replicas with consensus process, which makes the network scale easily.

Many BFT protocols could only tolerate one-third faulty replicas. Therefore we expect that faulty replicas could be repaired as soon as possible to avoid the situation where the proportion of faulty replicas exceeds one-third in a certain time window. FDRBFT uses proactive recovery [20], [21] to ensure that the system runs stably for a long time. In short, even if there is no reason to suspect that a replica is faulty, the replica would be restored periodically.

III. Background

A. Byzantine Agreement Problem

Solving the problem of consensus in the presence of Byzantine failures is known as the Byzantine Agreement problem [2], its most basic form is defined as follows: Let C be a consensus protocol among n replicas P = $\{P_1, P_2, ..., P_n\}$. B is a subset of P (B $\subset$ P) and every replica of B is Byzantine faulty. Each replica P_i starts with an input bit b_i, and P_i outputs a bit c_i at the end of the protocol. C is a Byzantine Agreement Protocol, if it satisfies the following conditions:

(1)Agreement: For any two non-faulty replicas $P_i \in P$ and $P_j \in P$, their output meets $c_i = c_j$.

(2)Validity: If $b_i = b$ for all non-faulty replicas $P_i \in P$, then $c_i = b$ for all non-faulty replicas P_i.

(3)Termination: Protocol C terminates with the probability of 1.

These conditions also define the $safety$ and $liveness$ conditions that a distributed consensus algorithm must satisfy. The agreement and validity conditions compose safety condition: safety will be violated if any two non-faulty replicas output different values. The termination condition specifies the liveness condition. If a system continues executing correctly until it terminates, all replicas must eventually obtain the same result.

A protocol is said to be f-fault tolerant if it operates correctly as long as no more than f replicas fail during execution. According to [12], there is a f-fault tolerant synchronous protocol to solve the Byzantine agreement problem only if $f < n/3$.

B. Cryptographic Primitives

In blockchain network, messages are exchanged between replicas based on digital signature scheme [22]. The digital signature scheme (G,S,V) consists of three fast algorithms:

(1)Key generator G**:** Given a security parameter k, $G(k, i)$ generates a pair of k-bit keys including a public key pk_i and a matching secret key sk_i for replica i.

(2)Signing S**:** Given a message m and a secret key sk_i of replica i, $S(m, sk_i)$ produces a signature σ_i.

(3)Verification V**:** V takes pk_i, a message m and a signature σ_i as inputs, then V outputs either $accept$ or $reject$. A signature is $valid$ only if the V outputs $accept$.

All signatures produced by S must be $valid$ and $unforgeable$. For digital signature scheme (G, S, V), it is hard to find signatures σ and σ', when $\sigma \neq \sigma'$, V outputs $accept$ for both of them with same inputs including public key pk and message m, i.e.,

$$\sigma \neq \sigma' \quad and \quad V(pk, m, \sigma) = V(pk, m, \sigma') = accept \tag{1}$$

The protocol proposed in this paper is based on threshold signature scheme, which is introduced by Desmedt [23], [24]. In an (n, k, t) threshold signature scheme, there are n replicas, up to t of which may be faulty, k is the threshold of valid signatures. The replicas who hold "share" could generate share signatures on individual messages by using S of digital signature scheme. At least k share signatures are both necessary and sufficient to construct a threshold signature. The only requirement of k is that $t < k \leq n-t$. The threshold signature scheme also consists of three algorithms:

(1)Signature verification SIV**:** SIV takes a message m, a signature σ and the public key pk as inputs. Then SIV determines whether the signature σ is valid.

(2)Share verification SHV**:** SHV takes a message m, a share signature on that message from a replica P_i, PK, VK, andVK_i as inputs. Then SHV determines whether the share signature is valid.

(3)Share combining SC**:** Given a message m, k valid share signatures on m, the public key pk and the verification key VK, SC would output either a valid threshold signature on m named as $\sigma(m)$ or not.

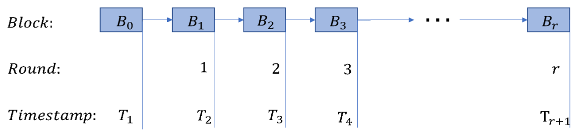


Fig. 1. Blockchain overview

The two basic security requirements of threshold signature scheme are robustness and non-forgeability. Robustness means it is computationally infeasible for an adversary to produce k valid share signatures such that the output of share combining algorithm is not a valid signature. Non-forgeability means it is computationally infeasible for the adversary to output a valid signature on a message. That signature was submitted as a signing request to less than $k - t$ non-faulty replicas.

C. Ideal Blockchain

Blockchain is a distrusted and shared ledger that multiple replicas record transactions in a verifiable and permanent way [1] based on State Machine Replication (SMR) and Cryptography.

At the beginning of a n-replicas blockchain network startup, there would be a genesis block B_0 to record the initial status of n replicas. The process of block generation could be broadly divided into 3 phases [1]: block packaging, consensus and termination. Let the process of r-th block generation be round r $(r \geq 1)$. In essence, the n replicas' initial status shows as Eq. (2).

$$S_1 = ((pk_1, B_0), ..., (pk_n, B_0)) \tag{2}$$

Consensus is the core of blockchain because it takes over 90 percent time of block generation process. If TX_r is the set of all transactions produced during round r, TX_r would be packaged into new block B_r. In round r, consensus in blockchain systems aims at collecting adequate replicas' valid signatures to prove a new block B_r is valid or invalid. Round r would cost $T_{r+1} - T_r$ time and the status of replicas in network would change as Eq. (3) shows.

$$r : S_r \xrightarrow{B_r} S_{r+1} \tag{3}$$

If B_r is valid, B_r along with its proof would consist $\overline{B_r}$, then $\overline{B_r}$ would be recorded on shared ledger. The valid blockchain is actually as follows:

$$Blockchain = \overline{B_0}, \overline{B_1}, ...\overline{B_r} \tag{4}$$

The intuitive overview of blochchain shows as Fig. 1

IV. FDRBFT Protocol

To deal with problems of BFT protocols, we propose a novel protocol which is called Fast, Dynamic and Robust Byzantine

Fault Tolerance (FDRBFT). FDRBFT combines random algorithm and threshold signature scheme for fast consensus. A unique cryptographic algorithm is applied in FDRBFT to achieve dynamic scalability and a proactive recovery scheme is adopted to restore replicas periodically.

We define the notion of (n, k, t)-random threshold signature consensus scheme: there are n replicas in network, up to t ($t < 3/n$) of which may be Byzantine faulty. k ($t < k \le n-t$) is the threshold of valid signatures. This scheme consists of six algorithms:

(1)Key generation G**:** Given a security parameter b, $G(b, i)$ generates a set of b-bit keys: a public key pk_i and a matching secret key sk_i, a global verification key VK and local verification keys VK_i for replica i.

(2)Random R**:** Given an initial status of network $S_r((pk_1, B_{r-1}), ..., (pk_n, B_{r-1}))$ and threshold k, $R(k)$ outputs the leader and consensus replicas for round r. R(k) generate a random binary array $randomArray \in \mathbb{R}^{1\times n}$ and a random number j ($j < k$) at first. $randomArray$ only includes "0" and "1", "1" in $randomArray$ means the corresponding replica own share. The number of "1" in $randomArray$ equals k and the number j is used to decide which replica is leader. For example, when $n = 4, replicas = (P_1, P_2, P_3, P_4), t = 1, k = 3$, $randomArray = (0, 1, 1, 1)$ and $j = 2$, then replicas P_2, P_3, P_4 own share, the replica P_3 which is corresponding to second "1" in $randomArray$ is leader.

(3)Share Signature generation SG**:** Given a message m, a secret key sk_i of replica P_i and its share ("0" or "1"), SG would produce a share signature of P_i named as $\sigma_i(m)$.

(4)Signature verification V**:** V takes a message m, a share signature $\sigma_i(m)$, VK and VK_i as inputs then outputs either valid or invalid.

(5)Share combining SC**:** Given a message m and k' valid share signature on m, the public keys $pk_i, ..., pk_{k'}$ and the global verification key VK. If $t < k' \le n-t$ and all keys are right, SC would output a valid threshold signature on m named as τ_m.

(6)Collect C**:** Given k'' valid threshold signature, if $t < k'' \le n-t$, C would output $accept$, else $reject$.

Replicas own share in round r are named as "Sreplicas", else "NSreplicas". NSreplicas would not take part in the consensus process. All threshold k, k', k'' are in same interval $(t, n-t]$. If a given k is close to $n-t$, then when k' or k'' exceeds t, all process would move forward rather than wait the maximum delay time.

FDRBFT is based on semi-synchronous setting. The transactions from client would not be recorded at once. In fact, they would be verified by the leader firstly. Then they would be saved in transaction pool. During the consensus period, leader would package some transactions from the pool into a new block with a specific packaging method. The Sreplicas in round r should be synchronous.

For a round r, the process of consensus includes *status, propose, prepare and collect*. During *status*, replicas in network would be divided into Sreplicas and NSreplicas. The leader would be selected from Sreplicas. During *propose* period, the leader would pack transactions into a new block B_r and sent proposal of B_r to other Sreplicas. Then a Sreplica except the leader would transfer the proposal to other Sreplicas. In *prepare* period, if a Sreplica receives k' valid share signatures within the maximum delay time MDT, it produces a valid threshold signature. As for *collect* period, if the collector receives k'' valid threshold signatures within the maximum delay time MDT', all replicas reach agreement and they would append B_r to their shared ledger. The pseudo code of consensus shows as Algorithm. 1

Algorithm 1 Random threshold signature consensus scheme (RTSC)

```
Input: Params: $r, S_r, SK, PK, VK, n, t, k, k', k''$,
       MDT,MDT';
       Fuctions: $R, SG, V, SC, C$
Output: string $accept/reject$
initial $r \ge 1$, $t = \lfloor n/3 \rfloor$,$k = n - t$, $k' = k'' = 0$;
status: leader,Sreplicas=$R(k)$;
propose: $transactions \xrightarrow[batch]{leader} B_r$;
$\sigma_{leader}(B_r)$=SG($B_r, SK_{leader}$, 1);
proposal=($B_r, \sigma_{leader}(B_r)$);
foreach $i$ ∈(Sreplicas-leader) do
    $i \xleftarrow{proposal} leader$;
for Sreplica $i$ has received proposal do
    $res = V(proposal, VK, VK_i)$;
    $time1 = getTime()$;
    if $res == valid$ then
        $k' += 1$;
        $shareSig_i = SG(proposal, SK_i, 1)$;
        for Sreplica $j \ne i$ and Sreplica$j \ne leader$ do
            $j \xleftarrow{shareSig_i} i$;
prepare:
foreach Sreplica $i$ do
    while $getTime() - time1 \le MDT$ do
        if $i$ receives a valid shareSig then
            $k' += 1$; if $k' \ge t + 1$ then
                $thresholdSig_i =$
                $SC(B_r, k'shareSig, PK, VK)$;
                break;
collect: $time2 = getTime()$;
while $getTime() - time2 \le MDT'$ do
    if $i$ receives a valid thresholdSig then
        $k'' += 1$;
        if $k'' \ge t + 1$ then
            break;
            return $accept$;
return $reject$;
```

In essence, agreement would be achieved by Sreplicas with 2-phases communication including *propose and prepare*, the communication price is reduced greatly.

An example is given in Fig. 2 with setting $n = 4, t = 1, k = 3$, when k' up to $t + 1 = 2$ and $k'' \in (1, 3]$, replicas could reach agreement by only 9 communication times.

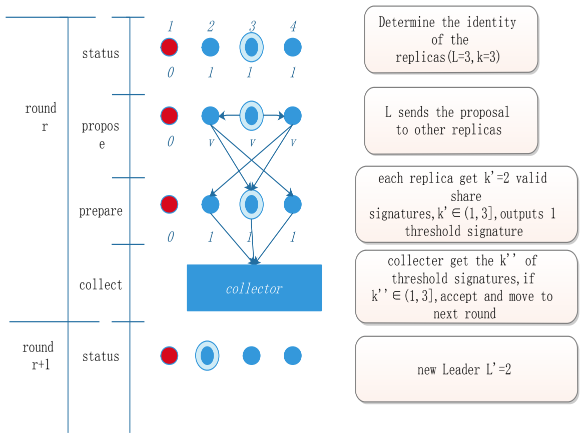

round r
status
propos e
prepare
collect
round r+1
status
1
2
3
4
0
1
1
1
0
v
v
v
0
1
1
1
collector
Determine the identity of the replicas(L=3, k=3)
L sends the proposal to other replicas
each replica get k'=2 valid share signatures, k' ∈ (1, 3], outputs 1 threshold signature
collecter get the k'' of threshold signatures, if k'' ∈ (1, 3], accept and move to next round
new Leader L'=2

Fig. 2. Example of a n=4 consensus. The red circle is a NSreplica.

Replicas in network verify messages from others based on the a unique cryptographic algorithm. Because of that, if a new replica could use this cryptographic algorithm, it would be recognized by other replicas. The unique cryptographic algorithm consists of two functions:

(1)Certificate generation CG**:** CG aims to generate a certificate $cert_i$ for a replica i , $cert_i$ is valid in the near future time T.

(2)Certificate verification CV**:** Given a certificate $cert_i$ of replica i, CV outputs whether $cert_i$ is $valid$.

In consortium blockchain, whether a new replica could join the network or a replica in network could exit need to be agreed by other replicas. Giving the fact that replicas could reach agreement by consensus, consensus described in Algorithm. 1 (RTSC) is used as the foundation for dynamic mechanism in FDRBFT. A replica could use a certificate generation algorithm CG to generate a certificate $cert$, the $cert$ is valid within time T. Then the replica could send a request to client with its certificate. The client would send the request as a transaction to leader. If the result of consensus is $accept$ within T, a replica could join or exit the network.

The concrete process is described in Algorithm. 2.

According to Algorithm 1, NSreplicas for round r would not participant in consensus, which creates favorable condition for proactive recovery because NSreplicas could be restored periodically without affecting the consensus process.

The recovery of NSreplicas could be divided into following steps:

Step1: NSreplicas ask the Sreplicas for checkpoint information;

Step2: NSreplicas check whether the blocks information of the checkpoint is consistent with their own blocks.

Step3: If not consistent, calculate height of blocks which are necessary to be restored.

Step4: NSreplicas ask the Sreplicas for missing blocks.

Step5: NSreplicas append the blocks to their own shared ledger.

Step6: NSreplicas update keys by using algorithm G firstly, then inform Sreplicas about the changes.

Algorithm 2 Dynamic mechanism(DM)

```
Input: $CV, CG, G, replicas, RTSC, b$
Output: $accept/reject$
$cert, T = CG(s); time = getTime(); pk_s, sk_s, vk_s, VK = G(b, s)$;
$request = (op, cert, s, PK_s, VK_s, VK)$;
$client \xleftarrow{request} s$;
$transation = request + \sigma_c client$
$leader \xleftarrow{transaction} client$;
if $CV(transaction) \neq valid$ then
    return $reject$;
else
    while $getTime() - time \leq T$ do
        $res = RTSC(r)$;
        if $res == accept\ and\ op == join$ then
            $s \xleftarrow{(S_r)} leader$;
            $replicas \leftarrow (VK_s, PK_s)$;
            $n = n + 1$;
            break;
            return $accept$;
        if $res == accept\ and\ op == exit$ then
            $s \xleftarrow{(S_r)} leader$;
            all $replicas$ delete $(VK_s, PK_s)$;
            $n = n - 1$;
            break;
            return $accept$;
        $r = r + 1$;
    return $reject$;
```

V. Performance Evaluation

A. Implementation

A blockchain network for evaluating FDRBFT is achieved as Fig. 3 shows. This blockchain network is implemented based on an open source project called Hyperledger Fabric. In this project, pluggable consensus framework for different protocols is provided. For convenience, Hyperledger Caliper program is applied as performance benchmark framework. With a simple smart contract, write TPS (W-TPS) and read TPS (R-TPS) could both be evaluated. W-TPS is more important because it reflects the ability of protocols to reach agreement and record transactions directly. Unless otherwise specified, the TPS that appears after in this article is W-TPS by default.

The configuration information for the replicas in our network is shown in Table. I.

B. Performance

In blockchain, TPS and latency are usually used to measure the performance of protocols. TPS and latency can be

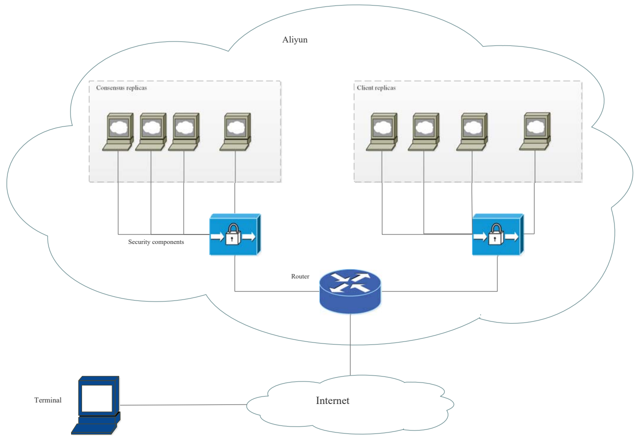


Fig. 3. The topology of blockchain evaluation network.

TABLE I
REPLICA CONFIGURATION

Items	Information
CPU/Memory	4v CPU/16G
Bandwith	8Mbps
Main frequency	2.5GHZ
OS	Ubuntu 16.04

calculated by Eq. (5) and Eq. (6) respectively. TPS evaluates the number of transactions could be successfully recorded by replicas per second, while latency, whose default unit is millisecond(ms), measures the time cost from proposal for block to replicas' record. In Eq. (6), $recordTime$ represents the time when replicas append a new block including transactions to their shared ledger and $requestTime$ represents the time when the leader generates the proposal for new block.

$$TPS = \frac{success\quad transactions}{totalTime} \tag{5}$$

$$Latency = recordTime - requestTime \tag{6}$$

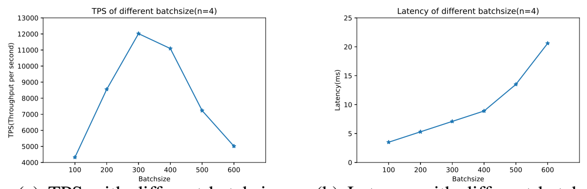
Generally speaking, there are two methods for block packaging. One is packaging by time, transactions would be packaged into a new block per $batchtime$. The other one is packaging by quantity of transactions, every $batchsize$ transactions would be packaged into a new block.

Packaging by $batchsize$ is more suitable for consortium blockchain because it achieve parallelization actually. The bandwidth that one round of communication between replicas would cost could be calculated by Eq. (7).

To find a suitable $batchsize$ and make the most use of the bandwidth. TPS and latency of FDRBFT are detected with different $batchsize$, the evaluation result is shown in Fig. 4.

$$bandCost = n * (n - 1) * blocksize \tag{7}$$

It is observed from Fig. 4(a) that TPS improves with the $batchsize$ increases when bandwidth is enough. However, when $bandCost$ is over than bandwidth, then TPS decreases

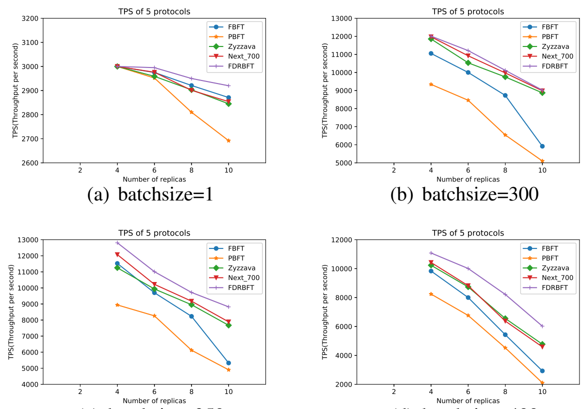


(a) TPS with different batchsize (b) Latency with different batchsize

Fig. 4. TPS and latency with different batchsize

because of network congestion. Fig. 4(b) shows the delay increases as the $batchsize$ increases because the block needs to be transmitted between the consensus replicas, the bigger the block is, the longer the processing time. To sum up, the optimal $batchsize$ is 300 to 400 because when $batchsize \in [300, 400]$, TPS is optimal and latency growth is within acceptable limits.

By using same smart contract in Caliper, we evaluate TPS of five protocols including FDRBFT, FBFT [9], PBFT [3], Next 700 [25], and Zyzzyva [26] with different number of replicas when $batchsize$ is 1, 300, 350, 400 respectively. For each case, we use one million transactions and count the time these transactions cost for consensus. TPS is calculated as Eq. (5). we test each case for 10 times and get its average TPS. The experimental results are shown in Fig. 5.



(a) batchsize=1 (b) batchsize=300

(c) batchsize=350 (d) batchsize=400

Fig. 5. TPS of 5 protocols with different $batchsize$ and n

As Fig.5 shows, when the number of replicas increases, the TPS would downgrade because of increasing communication price. The performance of PBFT is worst because of its $O(n^2)$ communication complexity and $O(3)$ round complexity.

FDRBFT performs best among the five protocols. Because of random algorithm, the round complexity of FDRBFT is $O(2)$. With 2-phases consensus, the communication complexity of FDRBFT is $O(k^2)$ ($k < n$). FBFT, Next 700 and Zyzzyva all achieves $O(n^2)$ communication complexity, but lower round complexity than PBFT.

With $batchsize = 300$ and $batchsize = 350$, TPS of FDRBFT is up to 9000 even when the number of replicas is 10, it could satisfy the requirement of most applications.

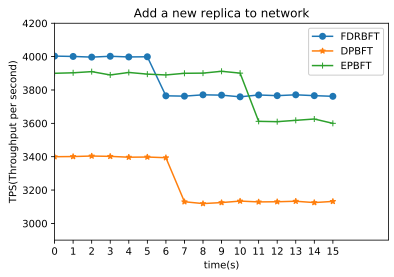


Fig. 6. Dynimic scalability with adding a new replica

C. *Dynamic scalability*

With $batchsize = 1$, $n = 4$, we test the dynamic scalability of DPBFT [4], EPBFT [18] and FDRBFT by sending a request of adding a new replica to the network at the same time (when $time = 5s$). According to Sect.5.2, TPS would decrease when the number of replicas in network increases. For this reason, throughput is monitored per second to see if the new replica succeed to join the network. Experimental result shows as Fig. 6.

In Fig. 6, TPS changes sharply after $time = 5s$ indicates that the replica has joined the network and could execute the consensus process normally like other replicas. Fig. 6 shows FDRBFT could finish the process within just 1 second, while DPBFT needs 2 seconds and EPBFT takes 6 seconds.

Actually, in a network using FDRBFT protocol, it would only take some milliseconds for a new replica to join the network if the transaction including the request could be packaged into new block at once. That is because our dynamic mechanism is based on consensus. When the agreement on transactions could be reached quickly, the network could scale in a high speed. As for DPBFT, it costs more time because of communication times used to select voters. In EPBFT, changing identity of a replica should wait many heartbeat messages and election messages.

D. *Robustness*

In Sects.5.2, we demonstrate that FDRBFT has competitive performance to other protocols in non-faulty case. In this section, DPBFT, EPBFT and FDRBFT are evaluated from two aspects in order to verify the security and robustness of them. On the one hand, we control the number of faulty replicas in network. On the other hand, we test the reaction of the network when leader is faulty.

Experiments are carried out under different settings:
(1)$n = 4, batchsize = 1, f = 1$, leader is not faulty;
(2)$n = 4, batchsize = 1, f = 1$, leader is faulty;
(3)$n = 4, batchsize = 1, f = 2$, leader is not faulty;
(4)$n = 4, batchsize = 1, f = 2$, leader is faulty.

Throughput would be detected to estimate whether the network works normally. Attacks would happen when $time = 10ms$.

Two possible attacks are defined according to our settings: (1)**attack-1**, when the leader is non-faulty and (2)**attack-2**, when the leader is faulty. The result is shown as Fig. 7.

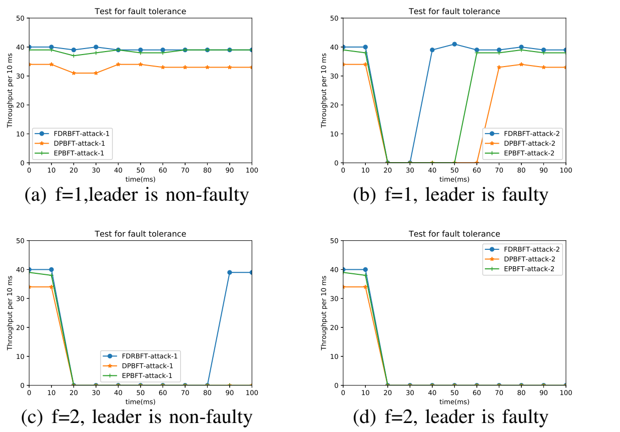


Fig. 7. Throughput of protocols under two possible attacks

As shown in Fig. 7(a), when attack-1 happens, the curve of throughput for DPBFT and EPBFT fluctuates, while the curve of throughput for FDRBFT is relatively smooth. The most likely reason is that the faulty replica happens to be a NSreplica. It can be seen from Fig. 7(b) that throughput drops to zero because other replicas can not receive messages from leader until the maximum delay time. It is obvious FDRBFT takes the least time to recover a faulty replica.

When attack-2 occurs, throughout drops to zero because only two non-faulty replicas could not reach agreement. However, in Fig. 7(c), the throughput of FDRBFT rises to a normal level. There might be a situation where one of faulty replica is a NSreplica in round r, at $time = 80ms$, the faulty NSreplica is restored and three non-faulty replica are selected as Sreplicas in next round $r + 1$.

According to the results in Fig. 7, FDRBFT shows better robustness in the face of faulty replicas.

VI. Conclusion and Future Work

In this paper, we propose a new protocol called FDRBFT. With a random threshold signature consensus scheme we achieve O(2) round and O(k^2) communication. The dynamic mechanism of FDRBFT is based on consensus and a unique cryptographic algorithm, which allows replicas join or exit the network quickly under the premise that other replicas do not stop working. We restore NSreplicas regularly with proactive recovery mechanism, which maintain the security of network.

Experimental results show that TPS of FDRBFT could be up to 9000 with $batchsize = 300$ and $batchsize = 350$ even when there are 10 replicas in network. The network scales quickly and faulty replicas could be repaired in limited time. Therefore, we can conclude FDRBFT achieves fast agreement, dynamic scalability and robustness at the same time.

Future work will continue to optimize algorithm details to further reduce comminication price and the consensus structure. Furthermore, future work would focus on cryptography to reduce computing price and achieve better security. Anyway, protocols should be used in a blockchain system which is in a practical field, and contributes to the application and popularization of blockchain.

References

[1] L. Zhu, H. Yu, SX. Zhan, WW. Qiu and QL. Li, " Research on High-Performance Consortium Blockchain Technology–A Case Study of Decentralized Securities Transaction System," Journal of Software, vol. 30, no. 6, pp. 1-18, 2019.

[2] L. Lamport, R. Shostak and M. Pease, "The Byzantine generals problem," ACM. T. PROGR. LANG. SYS. vol. 4, no. 3, pp. 382-401, 1982.

[3] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," In: OSDI, Vol. 99, pp. 173-186, 1999.

[4] X. F. LIU, "Research on blockchain performance improvement based on Byzantine Fault Tolerance consensus algorithm based on dynamic authorization," Zhejiang University, 2017.

[5] L. M. Bach, B. Mihaljevic and M. Zagar, "Comparative analysis of blockchain consensus algorithms," In: 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO) pp. 1545-1550. IEEE, May, 2018.

[6] P. L. Aublin, S. B. Mokhtar and V. Quéma, Rbft, "Redundant byzantine fault tolerance," In: 2013 IEEE 33rd International Conference on Distributed Computing Systems pp. 297-306. IEEE, July, 2013.

[7] S. Agrawal and K. Daudjee, "A Performance Comparison of Algorithms for Byzantine Agreement in Distributed Systems," In 2016 12th European Dependable Computing Conference (EDCC), pp. 249-260. September, 2016.

[8] F. Reid and M. Harrigan, "An analysis of anonymity in the bitcoin system," In: Security and privacy in social networks, Springer, New York, 2013, pp. 197-223.

[9] J. P. Martin and L. Alvisi, "Fast byzantine consensus," IEEE. T. DEPEND. SECURE., vol. 3, no. 3, pp. 202-215, 2006.

[10] Y. J. Song and R. V. Renesse, "Bosco: One-step byzantine asynchronous consensus," In: International Symposium on Distributed Computing, Springer, Berlin, Heidelberg. September, 2008, pp. 438-450.

[11] Y. Gilad, R. Hemo, S. Micali, G. Vlachos and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," In: Proceedings of the 26th Symposium on Operating Systems Principles, ACM, October, 2017, pp. 51-68.

[12] Pease, M., Shostak, R. and Lamport, L.: Reaching agreement in the presence of faults. Journal of the ACM (JACM), **27**(2), 228-234(1980).

[13] G. S. Veronese, M. Correia, A. N. Bessani and L. C. Lung, "Spin one's wheels Byzantine fault tolerance with a spinning primary," In: 2009 28th IEEE International Symposium on Reliable Distributed Systems pp. 135-144. IEEE, September, 2009.

[14] F. Borran and A. Schiper: Brief announcement, "a leader-free byzantine consensus algorithm," In: International Symposium on Distributed Computing, pp. 479-480. Springer, Berlin, Heidelberg, September, 2009.

[15] K. Kursawe, "Optimistic byzantine agreement," In: 21st IEEE Symposium on Reliable Distributed Systems, Proceedings , 2002, pp. 262-267.

[16] I. Abraham, G. Gueta, D. Malkhi, L. Alvisi, R. Kotla and J. P. Martin, "Revisiting fast practical byzantine fault tolerance," arXiv preprint arXiv:1712.01367. ,2017.

[17] D. Larimer, "Delegated proof-of-stake (dpos)," Bitshare whitepaper, 2014.

[18] J. Gan, Q. LI, Z. H. Chen and C. Zhang, "Improvement research of Blockchain Practical Byzantine Fault Tolerance Consensus Algorithm(EPBFT)," Journal of Computer Applications, 2019, pp. 1-10.

[19] D. Ongaro and J. Ousterhout, " In search of an understandable consensus algorithm," In: Annual Technical Conference, ATC, 2014, pp. 305-319.

[20] M. Castro and B. Liskov, "Practical Byzantine fault tolerance and proactive recovery," ACM Transactions on Computer Systems (TOCS), vol. 20, no. 4, pp. 398-461, 2002.

[21] P. Sousa, A. N. Bessani and M. Correia, "Highly Available Intrusion-Tolerant Services with Proactive-Reactive Recovery," IEEE Transactions on Parallel and Distributed Systems, vol. 21, no. 4, pp. 452-465, 2010.

[22] Y. Desmedt and Y. Frankel, "Shared generation of authenticators and signatures," In Annual International Cryptology Conference, pp. 457–469. Springer, Berlin, Heidelberg, August, 1991.

[23] Y. Desmedt and Y. Frankel, "Threshold cryptosystems," In Conference on the Theory and Application of Cryptology, pp. 307–315, Springer, New York, NY, August, 1989.

[24] Y. Desmedt, "Society and group oriented cryptography," A new concept. In Conference on the Theory and Application of Cryptographic Techniques, pp. 120–127. Springer, Berlin, Heidelberg, August, 1987.

[25] R. Guerraoui, N. Knežević, V. Quéma and M. Vukolić, "The next 700 BFT protocols," In: Proceedings of the 5th European conference on Computer systems, ACM, April , 2010, pp. 363-376.

[26] R. Kotla, L. Alvisi, M. Dahlin, A. Clement and E. Wong, "Zyzzyva: speculative byzantine fault tolerance," In: ACM SIGOPS Operating Systems Review Vol. 41, No. 6, pp. 45-58, ACM, October, 2007.