International Journal of Scientific Advances
ISSN: 2708-7972
Volume: 4 | Issue: 3 | May – Jun 2023 Available Online: www.ijscia.com
DOI: 10.51542/ijscia.v4i3.1

Dynamic Protocol Blockchain for Practical Byzantine Fault Tolerance Consensus

Ali Asad Sarfraz[1], Shiren Ye[1]*, Tianshu CheneyPen Zhao[1], Muhammad Usama Raza[2], and Tianshu Chen[1]

[1]Aliyun School of Big Data, Changzhou University, Changzhou 213164, China

[2]School of Materials Science and Engineering, Changzhou University, Changzhou 213164, China

E-mail: Aazi7777@gmail.com; yes@cczu.edu.cn; zhaopeng@cczu.edu.cn; usama132741.raza@gmail.com; tensoulchen@gmail.com

*Corresponding author details: Shiren Ye; yes@cczu.edu.cn

ABSTRACT
This work details the byzantine fault-tolerant protocols that dynamically allow replicas to join and exit. Byzantine fault-tolerant (BFT) protocols and the blockchain now play an essential role in achieving consensus. There are numerous drawbacks to PBFT, despite its multiple positives. The first thing to note is that it runs in an environment completely isolated from the rest of the world. The entire system must be shut down before any nodes can be added or removed. Second, it ensures liveness and safety if no more than (n-1)/3 out of total n replicas, PBFT takes no action to cope with ineffective or malicious counterparts. This is bad for the system and will lead to its eventual failure. These flaws have far-reaching consequences in real life. The Randomization PBFT is an alternative way of dealing with these issues. In recent decades, as computer technology has advanced, so has our reliance on the products, services, and capabilities that computers provide.

Keywords: DRBFT; PBFT; protocol; Blockchain; Byzantine fault tolerance; Dynamic PBTF

INTRODUCTION
Economic interests are becoming more prominent, which increases our vulnerability to system failures. Even if these systems are being attacked maliciously or if there are technical flaws, we would expect them to function correctly [1][2]. Researchers focus a lot of their efforts on replication to build reliable and protected computer systems. Replication research has focused mainly on strategies that only accept harmless mistakes. Crashing is considered the worst conceivable behavior for replications and nodes by these techniques, which assume that some steps are skipped entirely. Because malicious assaults, poor software, and human mistakes can occur in the real world, it is impossible to rely on the earlier premise. These flawed reasons may lead to clones behaving erratically [3].

Numerous hours have been devoted to research that relies on Byzantine mistakes. Tolerance for Byzantine faults has been explained in several articles. The primary PBTF Phases are shown in Figure 1.

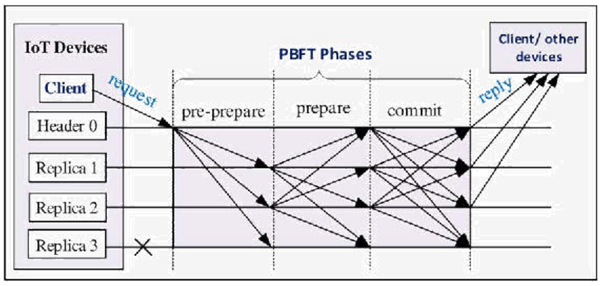

FIGURE 1: PBFT phases.

Replicating state machines using PBFT is a method. If at least $(\frac{n-1}{3})$ the total number of replicas has been mistaken, it has liveliness and safety qualities. If the proportion of malfunctioning nodes is lower than the security inception, then control, software problems, or operator blunders cannot cause a system crash [4]. Even though PBFT has numerous advantages, there are significant disadvantages to using it. Starting with a wholly confined arrangement, any nodes that seek to join or depart the network must pause the system as a whole. If most b $(\frac{n-1}{3})$ of n total replicas are defective, PBFT guarantees *liveness and safety*; nevertheless, it does not take any precautions to deal with malicious representations, which are detrimental to the system and will eventually cause it to crash. PBFT, on the other hand, does not specify a standard for determining whether or not a replica is active.

Many people in the system prefer to rely on others and avoid their responsibilities to save money, which is terrible for the system's security. It's hardly surprising that these flaws are unpleasant in practice. The Dynamic PBFT is an alternative way of dealing with these issues [5].

Many of its advantages may be traced back to its predecessor, which is why it is called randomization PBFT. Our protocol has the same amount of liveliness and security as PBFT. Like random PBFT, it is a protocol based on week synchrony conventions, which is a necessary attribute to have to make it work over the Internet. The moniker "Dynamic PBFT" recommends replicas, and nodes can join or leave the consensus network without downtime.

Our protocol has the same amount of liveliness and security as PBFT. Like random PBFT, it is a protocol based on week synchrony conventions, which is a necessary attribute to have to make it work over the Internet. The moniker "Dynamic PBFT" recommends replicas, and nodes can join or leave the consensus network without downtime. Because the network does not need to be taken offline, this is an option. Aside from that, it makes the system more resilient by allowing the removal of malicious nodes and nodes that have been down for an extended period [6]. A brand-new concept, the Participation Degree, is also part of our procedure. A node's level of activity can be gauged using this metric. Increasing the cost of avoiding consensus can successfully increase the system's security. The cumulative cost of participating in the consensus process is one way to do this. Adding them to a blocklist raises the malicious price nodes must pay for their actions, making being a malicious node more expensive.

THE BASIC IDEA OF PBFT

Tolerating Byzantine faults in a distributed network is the primary objective of this research, which focuses primarily on the construction of workable simulated state machines founded on PBFT. The fundamental concept of PBFT must be presented in this part to comprehend Dynamic PBFT better. The Practical Byzantine Fault Tree (PBFT) was the first clarification of Byzantine faults in a synchronous situation like the Internet that was both practical and effective.

Even though there are at most $(\frac{n-1}{3})$ flawed replicas out of a total of n replicas that are fixed and well-defined, PBFT can assure that the input requirements projected by a client will be processed in a similar order by at least all direct replicas and that these replicas will reoccurrence the correct and consistent consequences to the client. PBFT uses two methods to serialize requests to realize the qualities listed above. These methods are as follows: *Primary-Backup* and *Quorum* Replication can be found here.

Distributed system architectures typically use this approach. Views are the primary idea of PBFT, and they refer to the numerous configurations into which the copies can be put. For each perspective, a primary and backup copy is selected for each perspective. Replicas are assigned sequential numbers using a numeric value in the range [0, n-1], known as a node ID. Views are also numbered sequentially [7].

A. The Quorum's Replica

The quorum mechanism is generally used in dispersed systems to ensure data termination and consistency. Mathematics' fundamental concept may be traced to this simple idea: pigeonholes. Most nodes must be present in a distributed system before a transaction can proceed, known as a "quorum." One of the most important aspects of having a quorum is that it must be easily accessible.

- (Intersection) Every pair of quorums has at least one standard and correct replica. Quorums are always available that are free of false representations.

- (Availability) For example, ensure the distributed system's message has been appropriately saved if the message is sent to all members of the quorum and all members respond by acknowledging the message. As long as all of the quorum members react to the replica's communication, it is considered a success. In a distributed classification with n replicas, PBFT undertakes that n equals *3f + 1*, where *fi* is the maximum number of models with errors. PBFT uses this assumption [8].

There must be at least *2f + 1* clone for a quorum to be established. According to another idea known as weak certificates, if at least *f + 1* copies of the identical message are stored, there must be further than one benign replica also accumulating the news.

B. Normal Case Operation

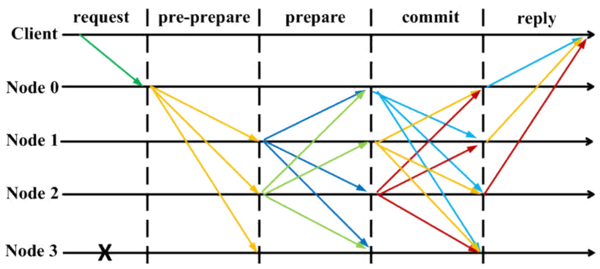

FIGURE 2: Normal Case Operation.

Figure 2 shows a typical PBFT scenario in which Node 0 represents the primary Node free of errors, and Node 3 illustrates an error node. The client will broadcast requests to all replicas with a timestamp to start. Once the recommendations have been received, the main will continue the procedure following a three-primary phase protocol. Pre-prepare, prepare, and commit are the steps in this process; client requests can be atomically broadcast to replicas. Replicas are responsible for completing all requests in the order specified by the primary and returning the results to clients [9]. *F+1* answers from distinct duplicates with the same effect and time stamp are waited for by the client. An invalid certificate is generated, ensuring that at least one accurate replica has responded to a result. As a result, the consumer has a strong basis for believing that the results are reliable and credible.

C. Checkpoints

The pre-preparation phase is the first step of the three-phase process. A pre-prepared message is sent to all replicas, and the primary advises the order in which requests should be processed. After receiving the pre-prepared message, each replication immediately confirms the proposal's authenticity by adding communication to its log and sending a prepared communication for others to show that it has received and recognized the proposal and is ready for deployment [10]. The replica will enter the preparation phase when other clones have delivered prepared messages. This case is published with a commit message and moved on to the commit phase once it has gathered a sufficient number of scheduled messages identical to the pre-prepared note. It accumulates *2f + 1* commit messages to ensure that enough copies have recorded the primary's proposal and that it can be implemented reliably.

Every legitimate three-phase communication provided to a node must be documented separately. PBFT creates a technique for removing the three-phase notifications of completed requests from the log. A checkpoint is a state that is established when all of the Krequests are executed, and replicas will record it as these requests are completed [11].

Creating a checkpoint causes the replica to broadcast a message to the entire cluster, which logs all subsequent checkpoint messages. A checkpoint can be proven correct if *2(f + 1)* identical checkpoint from distinct object replicas is collected. In the case of stable checkpoints, replicas can clean their logs of any three-phase messages with a sequence number lower than or equal to the checkpoint's correctness proof [12].

D. View change

The three-phase protocol indicates that the PBFT relies on a primary node regarded as benign to multicast a pre-prepared message and begin a round. However, the main might also serve as a valuable target for an attack. PBFT devised a method known as view-change to ensure liveness in the event of a damaged primary. As soon as the primary becomes corrupted or stops working, the backups will inform the next accessible Node of their desire to switch the primary [13][14]. An aberrant main can be discovered and changed by a quorum of *2(f+1*) copies, at which point the subsequent primary takes control.

SYSTEM MODEL

This Section will review the system entities' different functions and give an overview of the Dynamic PBFT.

A. Aspects of the System

Our algorithm assumes that the nodes are connected via a network in an asynchronous distributed system. Depending on the type of Node, the system can be divided into two groups: CA Replica and Node.

Two replica nodes receive client requests, perform consensus procedures like the three-phase protocol, and deliver the correct results back to the clients [15]. Assumptions are made to assure that the system will work properly, as shown below. Bound $f = (\frac{n-1}{3})$ is a constraint on the number of defective duplicates, which assume to be true. Our protocol uses Node *CA* to investigate and differentiate between all options, making it far easier to satisfy this assumption than the PBFT, which inflexibly makes this assumption. It is possible to distinguish between primary and backup replicas based on the purposes for which they are used. The primary Node is responsible for sorting and communicating with the backup nodes during each iteration of the consensus process [16]. For the PBFT to work, it relies on a closed network of well-defined nodes. Assuming that all replicas in the system have already preserved public keys, this is how it's possible to achieve.

No new replicas can enter or exit the network unless the PBFT system is stopped and the configuration is amended. Practically, no one can tolerate this strategy in practice. According to Assumption 2, a single node keeps track of all the replicas' data to enable the consensus system's dynamic property. Configuring a node in the system to provide security services is standard practice. For instance, IBM created "Membersrvc" specifically for usage in Hyper Ledger Fabric. (Assumption 2) Let's assume that the Dynamic PBFT system uses a security service provider called Node *CA*. The Node *CA* provides three goals similarly but not identical to the conventional *CA*. A company's position in the industry and the authenticity of the identity information supplied to Node *CA* determine whether or not a company is allowed to register and become a part of the system.

The system's Node *CA* is responsible for granting and revoking certificates for replicas [17]. The Node Certificate Authority maintains a list of node IDs, IP addresses, public keys, and the condition of each Node. Replicas have total faith in the identification information contained in Node *CA*. Replicas should not commence the Network Dynamic Consensus protocol based solely on the information provided by Node *CA* to prevent an undue reliance on Node *CA*. Several steps involve leaving a consensus network, such as notifying the other replicas of your request and filing an application with Node *CA* to revoke your certificate [18].

Once Nodes receives an exit request from Node j and a CRL from Node *CA*, it will only begin the *EXIT* protocol. Node *j* certificate has been revoked, and that information will be included in the CRL.

B. System Flow Chart

Byzantine flaws are not an issue with our protocol because it uses the three-phase approach PBFT uses to resolve distributed consensus issues on Byzantine fault-infested networks. On the other side, PBFT has specific fatal weaknesses, such as the inability to handle dynamically joining or leaving nodes and the lack of any mechanisms to punish rogue nodes, whether they are primary or backups. PBFT only uses the view-change protocol to hide the harmful primary and imposes no penalties on it in the process.

The sender's signature must be appended to every message delivered through the NDC protocol. The digest of a message is called *mas D (m),* and a message signed by replicas *I* called *mas D (m).* A message digest is marked instead of the complete statement and then added to the original text [19]. Standard procedure has been in place for a long time.

To avoid confusion, the remainder of this Section shall refer to *m*||*m I* simply as m||m *I.* The NDC considers active participation, active exit, passive exit, and passive exit of evil primary and backup. Create different sub-protocols for each use case for *JOIN REQ, EXIT, PCLEAR REQ, and CLEAR REQ*. The three-phase protocol is suspended on all nodes during these sub-protocols to make necessary adjustments to the NDC system settings [20]. Our protocol will continue to function normally if we follow Assumption 3. For example (Assumption 3), Node *CA* will immediately broadcast a *JOIN* message to all of the other nodes in the system as soon as the candidate has successfully registered.

DYNAMIC PBFT

PBFT is the initial practical Byzantine fault-tolerance protocol that can achieve dynamic properties. Applicable Byzantine Fault-Tolerance Protocol is known as PBFT.

A. Candidate Pool

Because dynamic PBFT allows nodes to enter and exit the graph at any time, the number of nodes in the system is never constant. The formula p = *v* cannot be used to select the primary, as it does not consider other factors [21].

Table 1 contains the node information list each replica must update and maintain, and the N-node *CA* following our protocol. Each representation is identified by its unique IP address, public key, and present state. A node has three possible states: benign, absent, and malevolent. If a node continues to contribute to the establishment of the consensus and is eligible to be declared a suitable node, it is considered benign. A node's status should default be set to Benign when it is formed. Node *CA* and other replicas will be absent if a node actively leaves the network [22].

A node's state will be modified if other nodes in the network judge it as doing evil or not functioning correctly. A node like a Node 1 is designated as the primary Node during the system's initialization process. Switching from v to v + 1 and selecting a new primary is done when nodes have to run the NDC protocol. Consider, for example, that the primary Node in Table 1 is Node 1 and that the current view is View NDC-enables replicas will transition to View 2 and designate the primary Node as t he second-to-last Node, Node 2. In the first place, there are tables.

TABLE 1: Node Information List.

Node ID	IP	PK	State
1	192.168.10.1	1PK	Benign
2	192.168.10.3	2PK	Absent
3	192.168.10.5	3PK	Evil
4	192.168.10.7	4PK	Benign

Nodes 2 and 3 are not considered safe, bringing us to number 4. Replicas will go back to the beginning of the list if the current primary is positioned further down the list than usual. An unusual situation has arisen here. In this scenario, the recent primary is Node 4, and the list suggests that Node 5 is safe; however, after some time has elapsed, replicas learn that Node 5 is malicious. If Node 6 is deemed benign, they will run the NDC protocol and make it the principal Node.

B. Active Participation.

The first step is to have the new Node registered with the Node CA, which is the first step. Node CA checks its data to remove nodes with negative business ratings or criminal history. Once the Node's identifying information is confirmed authentic, it will be added to the end of the list and given an appropriate identifier, such as j. Communication is known as JOINREQ, IP, and P K j will then be sent to all replicas to seek their participation in the network. After receiving a join request, nodes in the network, such as Nodes, will verify the signature [23]. Finally, they will match the join request's data to the data saved at Node CA. "Node *I* will do the JOIN protocol, as shown in Figure 3, if the validations are successful.

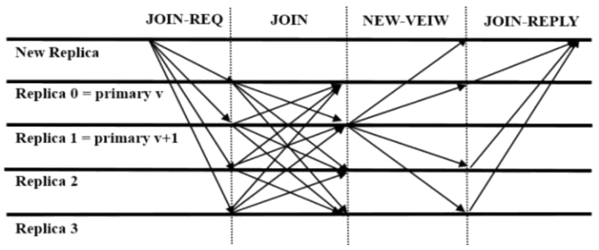

FIGURE 3: JOIN Req Protocol.

Node Protocol for Joining commences the process of forming the *JOIN* consensus when the three-phase protocol message is no longer sent and received. When a new message is received, it sends a "*JOIN*" message to all of the other nodes in the network that includes "his" (the sequence number of Node *I* records' most recent stable checkpoint) and a "Cis" (a series of " *2(f + 1)* checkpoint messages that prove the accuracy of s). P m contains a good pre-prepared communication and *2(f + 1)* matching statements for each contact that was prepared at Node *I* with a categorization number greater than h, where j is the node id of the sender of the join request [24].

A message will be sent to all the replicas, including the new node *j,* if the new primary collects *2(f + 1)* valid *JOIN* messages. *V + 1, V, O, j p* will be the message format for this one. Use the essence of joining communications and the senders' ID to simplify the communication complexity. O is a set of pre-prepared messages calculated in two different steps. Vis is a set of valid *JOIN* messages from *2(f + 1)* replicas. After determining the most recent stable checkpoint in V, the primary p selects the sequence number max-s with the highest value based on a V-prepared message [25].

The primary generates a new pre-prepared message for each sequence number that falls within the *view v+ 1* minimum and maximum time limit.

As soon as a new primary sends out a new-view message, Node does an audit to ensure that the signature is correct and valid. It completes a calculation similar to the primary to evaluate this. Every Node in the system has agreed to Node *j* involvement up to this point. They add Node *j* identity to the collection of node details that they already have in their possession [26]. By way of example, have a look at Node *I* communication with Node j: "*JOIN-REPLY, v + 1, J P O, I* i." Here P, O has a series of communications tied to client requests so that Node *j* can handle these requests in their new view the same way as the other messages.

Once Node *j* reaches a certain number of legitimate join-reply messages, it will start participating in the consensus. A node's exit can be classified as either active or passive, depending on its occurrence. A dynamic entry is when a node intentionally disconnects from the network for its benefit. The process by which malicious nodes are removed from the web by other nodes is referred to as "passive exit." Nodes can actively depart the network using a protocol we'll go through in the following few paragraphs.

If Node *j* decides to leave the network, it must first apply to Node CA to have its certificate and any other necessary information revoked. Afterward, it sends out an exit request communication to all other replicas in the form of EXIT REQ (j), IP (j), and P K (j). Another requirement is to contribute to the three-phase protocol until it receives a sufficient amount of departure reply messages from the rest of the network. To ensure that all of the system's nodes are aware of the revocation status of the Node *j* certificate, the Node CA adds the certificate to its Certificate Revocation List (CRL). The aspect of change phases is depicted in the figure below.

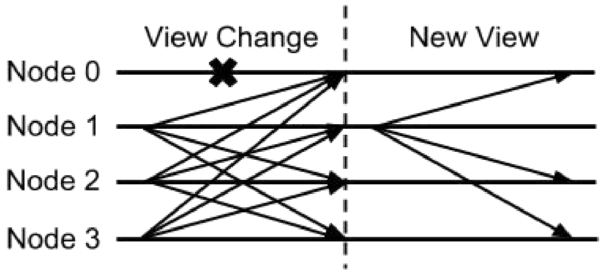

FIGURE 4 : View Change And New View.

The state of these removed nodes is changed to Preoccupied by Node *CA* upon the completion of multicasting CRL. After collecting all the exit requests, Node keeps a local log of them. When Node *I* receive a CRL, it only selects certificates that have not yet expired from the CRL's list of credentials [27]. After that, it scans its logs for similar exit requests and adds the node IDs of those certificates to a set E. ' Node *I* will initiate the EXIT operation if E is not empty, as depicted in Figure 5.

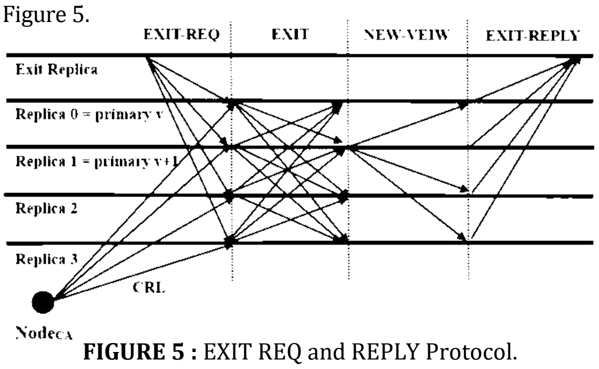

FIGURE 5 : EXIT REQ and REPLY Protocol.

To get things started, Node *I* multicasts an EXIT message to all of its replicas. This message appears as follows: It's time to get out of here. All images except the exit nodes will get a multicast message titled "*N EW V EIW*" when the new primary p has amassed two-and-a-half valid *EXIT* messages. Upon getting a *NEW - VIEW* communication from the new primary, Node, *I* first verify the signature's validity before evaluating whether or not *V, O,* and *E* are correct.

The network nodes have reached a consensus about removing nodes like Node *j.* As a result, they update their node information list to show that Node j is no longer present. Afterward, all nodes communicate with Node j by sending an exit-reply message, such as *'EXIT REPLY, v + 1, j, I I'.* At least one valid *EXIT-REPLY* message from numerous copies of Node *j* indicates that it has successfully exited the system and is now free to withdraw from the consensus.

C. Passive Exit: primary source

Dynamic PBFT can tolerate a certain amount of wrong copies, but any false documents should be avoided at all costs. Our protocol removes faulty nodes from the system and does not allow them to rejoin in the future, unlike PBFT, which does not take any action to deal with them. As a result, there will be no more network outages in the future. Making it more difficult and expensive can lessen clones' likelihood of bad conduct.

When the current primary fails to function or is malignant, replicas begin a *PCLEAR* procedure to remove it from the system. Restoring the regular operation of the network is the result of this. The protocol is shown graphically in Figure 6.

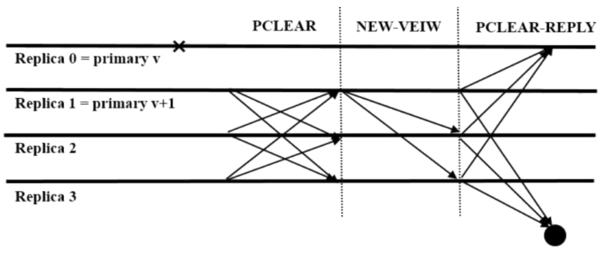

FIGURE 6 : PCLEAR REQ Protocol.

An initial message from Node *I* to other replicas reads: "*P CLEAR v + 1, h, C, P,* and *I,"* where *I* is the malicious main's node ID. All other replicas receive a message called a *NEW VIEW, v + 1, V, O, z p* when the new primary collects *2(f + 1)* valid PCLEAR messages; however, the primary Node was previously used for nefarious reasons and is omitted. New - VIEW message from the central and Node *I* checks to see whether the signature is accurate and whether V, O, and z *I* is suitable and valid is shown in Figure 6.

To date, every Node in the system has agreed on how to get rid of the wrong main that has outlived its usefulness. In a list of node information, modify the data of Node z into Evil's current status. Followed by PCLEAR-REPLY messages from all nodes like P CLEAR-REPLY messages sent to the Node z and Node CA. It will become evil if the Node CA receives more than one legitimate PCLEAR-REPLY message from other nodes. To prevent these rogue nodes from joining the system in the future, the Node CA will add them to its block list.

In Passive Exit, Letter E, the Evil Backup Replicas can quickly identify the properties of a main. There is no objective benchmark to employ.

A mechanism has been devised to address this problem as part of our protocol. As a first step, let's clarify the following terms:

D. The Participation Level (PD)

It's a metric for gauging the activity level among the nodes participating in the consensus process. PD is close to an integer between 0 and 3. Every Node in the local network maintains a list of the other nodes to record their actions. The PD of a newly-created node is set to three. By joining the consensus once, one copy will gain one point and lose one point in PD (up to an overall maximum of three). A backup node is considered faulty if its PD falls below zero. Contribute meaningfully to the process of obtaining an agreement. During a round of the three-phase protocol, if a commit communication from Node *j* is reliable with the majority *2(f+1),* Node *j* is reflected to have successfully entered consensus. Figure 7 shows that because no replicas received the commit communication from Node 3 in this round, they lowered the PD of Node 3 by one in their local list. This is because any of the copies has not received the communication from Node 3.

To keep track of the number of replicas that join consensus, Node *I* begin counting when a certificate is received on a message with a sequence number of l-t. Although network latency is taken into account, this still occurs. Here's a digit or two. CLEAR protocol is initiated when Node *I* learn that a sufficient number of nodes have had their PDs reduced to zero, and the IDs of these nodes are stored in the set Z.

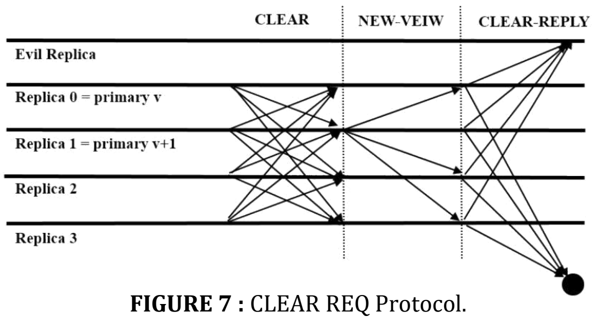

FIGURE 7 : CLEAR REQ Protocol.

CLEAR, v + 1, h, C, P, Z, and *I,* where *Z* is a list of node IDs whose PD will drop until it reaches 0, will be multicast to other replicas by Node I. A multicast message with the following contents is broadcast to all other representations, save for the malicious nodes, when the new primary collects *2(f + 1)* valid Req CLEAR communications [24[[25][26][27].

Thus far, all nodes in the system have achieved a consensus for removing wicked nodes. In Z Evil's state was changed in their node information list by altering the information of the nodes. Nodes then transmit a CLEAR-REPLY message, such as *"CLEAR" REP LY, "v+1," "Z," "I"* to wicked nodes, and "Node CA" back. Any nodes receiving valid CLEAR-REPLY messages from the Node CA server will have their state changed to Evil. These malicious nodes have been added to the Node CA's block list, which means they will no longer be allowed to join the system.

A. Reply from the Customer

The client must contact the Node CA and obtain information about the replicas when connecting to the network for the first time or reconnecting after a significant service disruption. Multicast queries to these copies and waiting for their responses will be possible. Node information has been updated, and the network's current number of active nodes is known after running NDC protocols [26][27][28].

However, because the client did not participate in the consensus, it has no idea how many nodes are currently operational. Consequently, it is impossible to identify the minimal number of reliable responses required to assure that the outcome is accurate. There are two options.

The client consults Network Security (CNS) after receiving a validated reply from a Dynamic PBFT network to determine the number of node conditions is Benign. Method 3 (CNS). When the customer obtains a third-party service, N CA /3 + 1 reliable *REPLY* messages must be received before calculating valid results.

Node j first registers at Node's CA's, then is sent into the network, so Node's CA Node. In the *Req CLEAR* and *Req PCLEAR* protocols, the wicked nodes are first sent out of the network and then notified to s Node's CA. An exit node instantly ends the three-phase protocol and begins working on the *EXIT* consensus, resulting in Node *CA = N* once it receives the CRL in the EXIT protocol. It is clear that Node *CA+1*, and N/N/3+1, which can guarantee the authenticity of the response, are the case.

New nodes are registered at Node *CA*; however, the join-REQ message is not delivered to all replicas for an extended period. Investigate this situation. In this case, N may be decreased to N. Node *CA* is equivalent to /3 + 1N+ 1 if Node *CA* exceeds 3N. As a result, the customer will never be able to gather enough consistent responses. The customer must check in with Node CA as soon as they obtain a response while using this method. A load-balanced cluster configuration of the Node CA may be implemented to handle many concurrent users.

Once a network replica has authenticated the client's request, it will select reproductions randomly from the list of local nodes in the client's network. Then it asks these copies about the active system's node number N c. The client trusts these replicas and waits for N *c /3 + 1* legitimate response before determining the right results if most of these copies match the beginning percentage p and send the similar N.

To carry out the quantitative analysis, we will presume that n equals 10 and f equals 3. Table 2 below shows the results. However, the likelihood did not rise linearly when the number of trials increased, as seen in Table 2. Furthermore, as p rises, Probe's value will probably decrease [28][29][30]. After selecting five random replicas to query, the client should wait until at least three replications return consistent results.

TABLE 2: Prob With Dissimilar K and P.

K / P / Prob	>1/3	=>2/3	>2/3
1	0.8	0.8	0.8
2	0.49	0.49	0.49
3	0.85	0.83	0.32
4	0.77	0.77	0.77
5	1	0.54	0.81
6	1	0.63	o.73
7	1	1	0.4
8	1	1	1

CORRECTNESS

Safety and liveliness are two areas in which our protocol has been examined in this work. Safety is defined in this study as all benign replicas will constantly on the sequence numbers of requests committed locally.

In the three-phase technique, the Dynamic PBFT is just as safe as the standard PBFT [31][32][33]. NDC protocol also guarantees that benign replicas agree with the sequence of local requests in different views. Node C protocol, the following subjects will be discussed in the next series for simplicity: only when Node *I* have put in the message does prepare *(m, v, and n)* become true. Unless and until this is the case, the statement is untrue.

2(f + 1) prepare messages from multiple nodes written in its local log for the min view with the sequence number n and a pre-prepared message for the min view. There is *f + 1* benign replicas in a set, therefore, committed *(m, v, n)* is only actual if that's the case.

For Node, *I* to have received *2f+ 1* commits from distinct nodes that match the pre-prepared message form, prepared *(m,n,v, i)* must also be true. The presence of a replica set R 1 containing at least *f + 1* benign counterpart that considers prepared *(m,v,n, i*) to be true implies that a request commits locally at a benign node with the sequence number in view v if. Each valid NEW - VIEW message in the NDC protocol contains *JOIN* messages from *2(f + 1)* replicas inside a replica set R2 (*EXIT, PCLEAR, CLEAR*) Because there are $f = (\frac{n-1}{3})$ replicas in the network, and at least most of the replicas are faulty, both R 1 and R2 have at least one healthy model. Node r *JOIN* signals may convey the new primary data that was improperly prepared in a prior view. Messages with the same sequence number in a previous view will not be committed if this strategy is used.

CONCLUSION AND FUTURE WORK

A "permission" network, such as a confederation of cryptocurrencies, can benefit from the scalability of Dynamic PBFT, a Byzantine fault-tolerant consensus protocol. High safety and liveliness are provided by Dynamic's PBFT, which is based on PBFT. It overcomes such lethal challenges as being entirely closed and taking no protections against faulty clones in the PBFT environment. Our protocol permits nodes to join or exit the consensus network, which prevents the entire system from restarting. It also includes a method for describing "wicked" backups. The blocklist of malicious primary and backup nodes is also built into the protocols used to gain consensus on these nodes' presence in the system and remove them. Our protocol raises the cost of being malicious. The protocol developed has improved dynamic characteristics and resilience over PBFT because it is descended from the former.

When combined with the NDC protocol or the three-phase protocol, this protocol can ensure the safety property. Customers who submit queries can rest assured that they will receive proper responses because of this protocol, which guarantees the system's liveness. Replicas are forced to change their perspective if they cannot fulfill a task. In addition, at least 2(f+1) benign duplicates should be in the same view for at least 2(f+1) benign copies. This technique has three steps to meet these requirements, which are as follows.

An NDC protocol (e.g., the *JOIN* protocol) should not be started early to keep the timer from running out prematurely. Hence a replica should wait for *2f + 1 JOIN* requests from other counterparts before initiating a timer. Allows the protocol to be launched at a more appropriate time. Until the timer expires, the replica will send another *JOIN* request to view *v + 2* if it has not received a valid *NEW - VIEW* message. At this point, a 2-minute timer is started.

After receiving more than one valid join message for an additional view that exceeds its current stance, the benign replica should immediately send a join message as other counterparts.

Finally, malicious copies cannot assault the system by continually initiating the NDC protocol. We cannot interfere with other replicas until one of our faulty ones becomes the primary because we need at least *f+1* valid *JOIN* messages to create an NDS protocol. Despite this, it is still possible to find and delete the faulty primary with the *PCLEAR* technique. Dynamic PBFT can guarantee liveness in a whole network by taking these precautions.

REFERENCES

[1] Zhan, Y., Wang, B., Lu, R., & Yu, Y. (2021). DRBFT: Delegated randomization Byzantine fault tolerance consensus protocol for blockchains. *Information Sciences*, *559*, 8-21.

[2] Chen, Y., Li, M., Zhu, X., Fang, K., Ren, Q., Guo, T., ... & Deng, Y. (2022). An improved algorithm for practical byzantine fault tolerance to large-scale consortium chain. *Information Processing & Management*, *59*(2), 102884.

[3] Jeon, S., Doh, I., & Chae, K. (2018, January). RMBC: Randomized mesh blockchain using a DBFT consensus algorithm. In *2018 International Conference on Information Networking (ICOIN)* (pp. 712-717). IEEE.

[4] Barger, A., Manevich, Y., Meir, H., & Tock, Y. (2021, May). A byzantine fault-tolerant consensus library for hyperledger fabric. In *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)* (pp. 1-9). IEEE.

[5] Wang, Y. (2021, October). The adversary capabilities in practical byzantine fault tolerance. *International Workshop on Security and Trust Management* (pp. 20-39). Springer, Cham.

[6] Tholoniat, P., & Gramoli, V. (2019). Formal verification of blockchain byzantine fault tolerance. *arXiv preprint arXiv:1909.07453*.

[7] Thai, Q. T., Yim, J. C., Yoo, T. W., Yoo, H. K., Kwak, J. Y., & Kim, S. M. (2019). Hierarchical Byzantine fault-tolerance protocol for permissioned blockchain systems. *The Journal of Supercomputing*, *75*(11), 7337-7365.

[8] Crain, T., Gramoli, V., Larrea, M., & Raynal, M. (2018, November). Dbft: Efficient leaderless byzantine consensus and its application to blockchains. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)* (pp. 1-8). IEEE.

[9] Ahmed, N. O., & Bhargava, B. (2020). Bio-inspired formal model for space/time virtual machine randomization and diversification. *IEEE Transactions on Cloud Computing*.

[10] Li, B., Weichbrodt, N., Behl, J., Aublin, P. L., Distler, T., & Kapitza, R. (2018, June). Troxy: Transparent access to byzantine fault-tolerant systems. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (pp. 59-70). IEEE.

[11] Zhang, Y., & Tseng, L. (2021). Byzantine Concensus: Theory and Applications in a Dynamic System.

[12] Loveless, A., Dreslinski, R., Kasikci, B., & Phan, L. T. X. (2021, May). IGOR: Accelerating byzantine fault tolerance for real-time systems with eager execution. In *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)* (pp. 360-373). IEEE.

[13] Ahmed, N. O., & Bhargava, B. (2018). From byzantine fault-tolerance to fault-avoidance: An architectural transformation to attack and failure resiliency. *IEEE Transactions on Cloud Computing*, *8*(3), 847-860.

[14] Yu, S., Zhu, J., & Yang, J. (2021, December). Efficient two-dimensional self-stabilizing byzantine clock synchronization in walden. In *2021 IEEE 27th International Conference on Parallel and Distributed Systems (ICPADS)* (pp. 723-730). IEEE.

[15] Tanwar, S. (2022). Distributed Consensus for Permissioned Blockchain. In *Blockchain Technology* (pp. 211-249). Springer, Singapore.

[16] Yuan, Y., Li, F., Yu, D., Yu, J., Wu, Y., Lv, W., & Cheng, X. (2019, July). Fast fault-tolerant sampling via random walk in dynamic networks. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)* (pp. 536-544). IEEE.

[17] Cohen, R., Haitner, I., Makriyannis, N., Orland, M., & Samorodnitsky, A. (2022). On the round complexity of randomized Byzantine agreement. *Journal of Cryptology*, *35*(2), 1-51.

[18] Kou, C., & Lundström, O. (2020). Self-Stabilizing Byzantine Fault-Tolerant State Machine Replication-Rust Implementation, Experimental Evaluation and Applications in Trucks.

[19] Li, T., Tseng, L., Higuchi, T., Ucar, S., & Altintas, O. (2021, November). Poster: Fault-tolerant Consensus for Connected Vehicles: A Case Study. In *2021 IEEE Vehicular Networking Conference (VNC)* (pp. 133-134). IEEE.

[20] Li, P., Peng, J., Yang, L., Zheng, Q., & Pan, G. (2018, December). Crux—A New Fast, Flexible and Decentralized Consensus Algorithm with High Fault Tolerance Rate. In *International Conference on Smart Blockchain* (pp. 66-76). Springer, Cham.

[21] Georgiou, C., Marcoullis, I., Raynal, M., & Schiller, E. M. (2021, May). Loosely-self-stabilizing Byzantine-tolerant binary consensus for signature-free message-passing systems. In *International Conference on Networked Systems* (pp. 36-53). Springer, Cham.

[22] Madsen, M. F., Gaub, M., Kirkbro, M. E., & Debois, S. (2019, September). Transforming byzantine faults using a trusted execution environment. In *2019 15th European Dependable Computing Conference (EDCC)* (pp. 63-70). IEEE.

[23] Baird, L., & Luykx, A. (2020). The Hashgraph protocol: Efficient asynchronous BFT for high-throughput distributed ledgers. *2020 International Conference on Omni-layer Intelligent Systems (COINS)* (pp. 1-7). IEEE.

[24] Abraham, I., Nayak, K., Ren, L., & Xiang, Z. (2021, July). Good-case latency of byzantine broadcast: A complete categorization. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing* (pp. 331-341).

[25] Coelho, I. M., Coelho, V. N., Araujo, R. P., Yong Qiang, W., & Rhodes, B. D. (2020). Challenges of PBFT-inspired consensus for blockchain and enhancements over neo dBFT. *Future Internet*, *12*(8), 129.

[26] Tseng, L. (2019, September). Eventual Consensus: Applications to Storage and Blockchain. In *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)* (pp. 840-846). IEEE.

[27] Crain, T., Natoli, C., & Gramoli, V. (2021, May). Red belly: a secure, fair and scalable open blockchain. In *2021 IEEE Symposium on Security and Privacy (SP)* (pp. 466-483). IEEE.

[28] Bonomi, S., Decouchant, J., Farina, G., Rahli, V., & Tixeuil, S. (2021, July). Practical Byzantine Reliable Broadcast on Partially Connected Networks. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)* (pp. 506-516). IEEE.

[29] Rullo, A., Serra, E., & Lobo, J. (2019). Redundancy as a measure of fault-tolerance for the Internet of Things: A review. *Policy-Based Autonomic Data Governance*, 202-226.

[30] Hou, R., Jahja, I., Luu, L., Saxena, P., & Yu, H. (2018, April). Randomized view reconciliation in permissionless distributed systems. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications* (pp. 2528-2536). IEEE.

[31] Hajiaghayi, M. T., Kowalski, D. R., & Olkowski, J. (2022, June). It improved communication complexity of fault-tolerant consensus. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing* (pp. 488-501). Luo, J., Kang, M., Bisse, E., Veldink, M., Okunev, D., Kolb, S., ... & Canedo, A. (2020). A Quad-Redundant PLC Architecture for Cyber-Resilient Industrial Control Systems. *IEEE Embedded Systems Letters*, *13*(4), 218-221.

[32] Gueta, G. G., Abraham, I., Grossman, S., Malkhi, D., Pinkas, B., Reiter, M., ... & Tomescu, A. (2019, June). Sbft: a scalable and decentralized trust infrastructure. In *2019 49th Annual IEEE/IFIP international conference on dependable systems and networks (DSN)* (pp. 568-580). IEEE.

[33] Hazari, S. S., & Mahmoud, Q. H. (2019). Comparative evaluation of consensus mechanisms in cryptocurrencies. *Internet Technology Letters*, *2*(3), e100.