The Hermes BFT for Blockchains

Mohammad M. Jalalzai, Chen Feng, *Member, IEEE*, Costas Busch,
Golden G. Richard III, *Member, IEEE*, Jianyu Niu

Abstract—The performance of partially synchronous BFT-based consensus protocols is highly dependent on the primary node. All participant nodes in the network are blocked until they receive a proposal from the primary node to begin the consensus process. Therefore, an honest but slack node (with limited bandwidth) can adversely affect the performance when selected as primary. *Hermes* decreases protocol dependency on the primary node and minimizes transmission delay induced by the slack primary while keeping low message complexity and latency with high scalability. Hermes achieves these performance improvements by relaxing strong BFT agreement (safety) guarantees only for a specific type of Byzantine faults (also called equivocated faults). Interestingly, we show that in Hermes equivocating by a Byzantine primary is *expensive* and *ineffective*. Therefore, the safety of Hermes is comparable to the general BFT consensus. We deployed and tested Hermes on 190 Amazon $EC2$ instances. In these tests, Hermes's performance was comparable to the state-of-the-art BFT protocol for blockchains (when the network size is large) in the absence of slack nodes. Whereas, in the presence of slack nodes Hermes outperforms the state-of-the-art BFT protocol significantly in terms of throughput and latency.

Index Terms—Blockchains, Byzantine Fault Tolerance, Consensus, Performance, Scalability, Security, Throughput.

1 Introduction

Scaling of a consensus protocol to support a large number of participants in the network is a desired feature. It not only allows more participants to join the network but also improves decentralization [1]. Additionally, reliability of a consensus protocol also depends on the actual number of faults it can tolerate [2], [3]. In a BFT-based (Byzantine Fault Tolerant) protocol if the total number of participating nodes is n then the number of faults that can be tolerated is bounded by $f < n/3$ [4]. Therefore, to improve the fault tolerance a scalable protocol has to be designed so that it can operate well with larger values of n. However, the increase in number of nodes generally results in higher message complexity which adversely affects the BFT performance [5], [6], [7]. Recent works have tried to address the scalability issues in the BFT protocols [8], [9], [10], [11], [12].

In BFT-based protocols the primary node acts as a serializer for requests. The consensus epoch begins as the primary proposes request batch/block to nodes in the network. The epoch ends after nodes agree on a block and add it to their chain. In practice, usually the size of the block proposed by the primary may range from several Kilobytes to several Megabytes. Therefore, by increasing the number of participants/nodes in the network the primary node with limited bandwidth has to spend most of its time broadcasting the block to a large number of nodes. Moreover, consensus nodes are blocked until they receive the proposal from the primary. Inversely, a primary node will be blocked until it receives at least $n - f$ responses from other nodes before proposing the next block. By designing a protocol that shortens this blocking time, BFT-based consensus protocol performance can be improved.

The Hermes protocol design has two new elements. The first design element includes proposing a block to a subset of nodes of size c and having a round of message exchange among this subset to make sure no request (block) equivocation has taken place by the primary. This design element helps us to achieve two goals, namely: (1) Mitigating the impact of slack primary nodes; and (2) Efficient Optimistic Responsiveness. The second design element involves removing a round (phase) of message exchange among all nodes (equivalent to the first phase of PBFT) from happy case execution (normal mode) and adding a recovery step to the view change mode. This design element helps Hermes to achieve design goal (3) Latency. In addition, we make use of a design element in [13] in order to achieve design goal (4) Scalability. This element involves node communication through a subset of nodes of size c also called the *impetus committee*.

Although the use of a committee to improve BFT performance has been done previously [13], [14], [15], [16], we use the impetus committee for completely different design goals (1 through 3). Moreover, a novel combination of the above three design elements to improve the overall BFT performance is something that to our best knowledge has not been done before.

Below we present key design goals of Hermes BFT that have pushed the performance to a next level especially for a blockchain setup.

Mitigating the impact of slack primary nodes. Slack nodes are honest nodes that have lower upload bandwidth compared to other nodes in the network. Lower upload bandwidth increases transmission delay (which is the time taken to put packets on the wire/link). Therefore, a slack primary can significantly diminish consensus protocol per-

• *M.M. Jalalzai, C. Feng, and J. Niu are with the Blockchain@UBC and the School of Engineering, The University of British Columbia (Okanagan Campus), Kelowna, BC V1V 1V7, Canada. C. Busch is with the School of Computer and Cyber Sciences, Augusta University. G. G Richard III is with the Computer Science and Engineering Division, Louisiana State University. E-mail: {m.jalalzai, chen.feng, jianyu.niu}@ubc.ca, kbusch@augusta.edu, golden@cct.lsu.edu.*

formance. On the contrary, prompt nodes have higher upload bandwidth and have lower transmission delay. In practice, slackness can be a temporary or permanent condition. In the case of slackness as a temporary condition, a node might become slack due to underlying issues in the network or a bug. Slackness might also be a permanent condition as a node simply does not have enough bandwidth. Addressing slackness in both cases is important. First, the temporary slackness is a real-time problem and thus difficult to address instantly due to the dynamic nature of the Internet. Secondly, if slackness is permanent then it may not be a good idea to remove slack nodes from the network as it will make the network more centralized. Indeed, nodes from the regions where the Internet cost is high (or nodes that cannot afford high bandwidth) should be able to be selected as the primary. In blockchain networks, primary nodes might be rotated after proposing a certain number of blocks (say, $10K$) so that all the nodes get the opportunity to propose blocks. In Section 6, we will discuss how view change is optimized in Hermes, making it easier to rotate primaries (view change). Therefore Hermes can be useful in both of the above mentioned cases. In normal BFT based protocols [17], [18], [19] the primary has to broadcast a block to all nodes in the network of size n. Whereas in Hermes the primary broadcasts a block only to a small subset of nodes of size c (also called impetus nodes). In Hermes the growth of c is sub-linear to n, therefore an increase in n will not have a significant impact on c, and hence on performance of Hermes. Therefore, Hermes mitigates the negative performance impact of the slack primary. This leads to another interesting property that we call Efficient Optimistic Responsiveness.

Efficient Optimistic Responsiveness. Responsiveness is an important property of BFT state machine replication (SMR) protocols [20], [21]. In BFT SMR protocols with responsiveness the primary drives the protocol towards consensus in a time that depends on actual message delay, regardless of any upper bound on message delivery [17], [20]. A protocol is optimistically responsive if it achieves responsiveness when additional constrains are met. HotStuff [17] is optimistically responsive in which the primary has to wait for $n-f$ responses to send next proposal. In Hermes during period of synchrony, a correct primary will only wait for $\lfloor c/2 \rfloor + 1$ responses (lower than $n-f$ responses) to propose the next block that will make progress with high probability. After a round of agreement these c nodes (also called impetus committee) if agreed, will forward the proposed block to all nodes in the network. In case of Hermes, conditions for optimistic responsiveness include receipt of $\lfloor c/2 \rfloor + 1$ responses by the primary. For different values of c chosen in our experiments in Section 7 the probability of having at least one prompt node among a subset of size c is approximately $1 - 10^{-9}$ (when the number of prompt nodes are at least $f + 1$). This means that with high probability there will be at least one prompt node in the impetus committee that can forward the block to the regular nodes efficiently. Therefore we call Hermes efficient optimistic responsive as messages in Hermes propagate to nodes with the wire speed comparable to the wire speed of prompt nodes.

Shortening Latency (in the first phase). BFT-based protocols [5], [11], [22] generally operate in two phases (excluding the $Pre-prepare$ phase). In the first phase each node receives $2f + 1$ responses before moving to the second phase. The first phase has two objectives. First, it guarantees that the request is unique. Hermes achieves this objective by making it difficult and expensive to perform equivocation through message exchange among a small subset of nodes of size c in the $Pre - Proposal$ phase shown in Figure 1. Second, it guarantees that if a request is committed in the second phase by at least one node just before a view change, all other correct nodes will eventually commit this request. This means for Hermes the second objective of the first phase is only necessary when there is a view change. To save $n \times n$ broadcast and processing latency (while achieving second objective), we remove this phase from the happy case mode and instead add a phase with $c \times c$ broadcast and an additional recovery phase to the view change subprotocol in Hermes.

Scalability. Hermes is using the impetus committee to maintain performance in the presence of a slack primary and achieve efficient optimistic responsiveness. Therefore, the same impetus committee can be leveraged to improve message complexity as done in Proteus [13]. Thus, we extend a single round of broadcast message where each node receives $n - f$ responses before committing a block into $Proposal$, $Confirm$ and $Approval$ phases (as show in Figure 1). This reduces the message complexity from $O(n^2)$ to $O(cn)$.

The trade-off for improvements in Hermes is a relaxed safety guarantee (R-safety) in the presence of equivocated faults in which a Byzantine primary proposes multiple blocks for the same height just before a view change. Informally, R-safety can be defined as a block will never be revoked if it is committed by at least $2f + 1$ nodes, out of which at least $f + 1$ are correct (honest) nodes. Whereas in Strong safety (S-safety) a block will never be revoked if it is committed by at least one correct node. Strong safety will not hold in the presence of equivocated faults. Moreover we also show that the equivocation faults will not have any affect on clients. That means if a client receives commit approval for a transaction from the network, then that transaction will never be revoked. Therefore, the main purpose of equivocation, which is double spending attacks, is not possible. The only side affect of the equivocation faults is the network recovery cost from failure. We also show that due to the design of Hermes, equivocating faults are expensive for a Byzantine primary and its acquaintances in the impetus committee (the primary as well as the Byzantine nodes in the impetus committee that have signed for two different blocks at the same height will lose their stake in the network and can be blacklisted). Therefore, a Byzantine primary finds equivocation expensive and has no incentive to perform it.

Paper Outline. The paper is organized as follows. In Section 2 we give our system model and definitions. In Section 3 we present the overview of the Hermes protocol. Section 4 provides a detailed operation of Hermes protocol. Proof of correctness for Hermes appears in Section 5 and in Section 6 we describe Efficient Optimistic Responsiveness as protocol optimization. Section 7 contains the experimental analysis and in Section 8 we present related work. We conclude our work in Section 9.

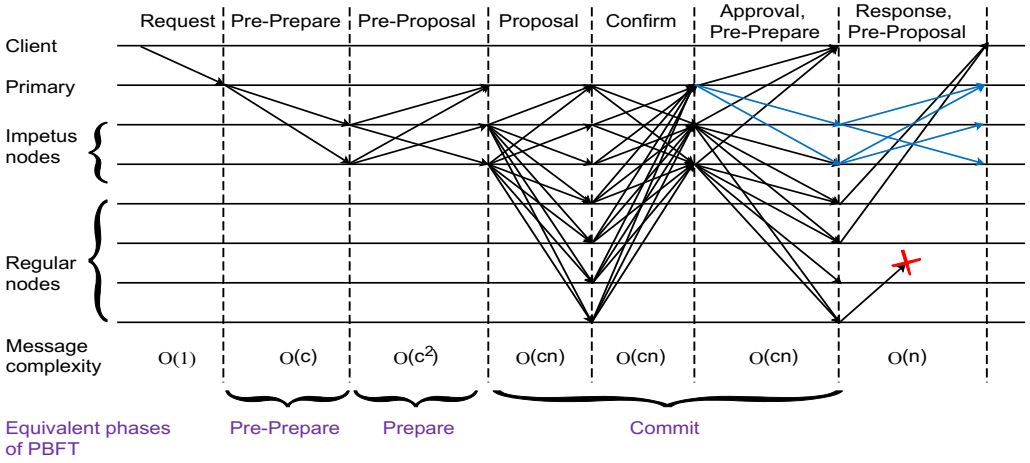


Fig. 1: Message pattern in each phase of normal mode in Hermes BFT

2 Definitions and Model

Hermes operates under the Byzantine fault model. Byzantine faults include but are not limited to hardware failures, software bugs, and other malicious behavior. Our protocol can tolerate up to f Byzantine nodes where the total number of nodes in the network is n such that $n = 3f+1$. The nodes that follow the protocol are referred to as correct nodes. In this model there can be up to f number of slack nodes ($n_s \leq f$, where n_s is total number of slack nodes).

Hermes is a permissioned blockchain. In a permissioned blockchain, nodes in the network are known to each other and have access to each other's public keys. In permissioned blockchains, nodes can join the network through an access control list. In this model nodes are not able to break encryption, signatures and collision-resistant hashes. We assume that all messages exchanged among nodes are signed. A message m' signed by the node i is denoted by $\langle m' \rangle_i$, a message m' with aggregated signature from impetus committee quorum (of size $\lfloor c/2 \rfloor + 1$) is denoted by $\langle m' \rangle_{\sigma_r}$, a message m' with aggregated signature from regular node quorum (of size $2f+1$) is denoted by $\langle m' \rangle_{\sigma}$, and a message m' with aggregated signature from view change quorum (of size $f+1$) is denoted by $\langle m' \rangle_{\sigma_v}$.

As a state machine replication service Hermes needs to satisfy following properties.

Definition 1 (Relaxed Safety). *A protocol is R-safe against all Byzantine faults if the following statement holds: in the presence of f Byzantine nodes, if $2f+1$ nodes or $f+1$ correct (honest) nodes commit a block at the sequence (blockchain height) s, then no other block will ever be committed at the sequence s.*

Definition 2 (Strong Safety). *A protocol is S-safe if the following statement holds: in the presence of f Byzantine nodes, if a single correct node commits a block at the sequence (blockchain height) s, then no other block will ever be committed at the sequence s.*

Definition 3 (Liveness). *A protocol maintains liveness if it guarantees progress in the presence of at most f Byzantine nodes.*

It should be noted that liveness in Hermes protocol is probabilistic. As discussed in Section 3.2, the largest probability of total failure (for liveness) is $3.8 \cdot 10^{-22}$ considering different practical sizes of c and n that we have used in Section 7.

To circumvent the FLP impossibility result [4], Hermes assumes partial synchrony [23] model. In this model, there is a fixed but unknown upper bound on message delay. An epoch identifies a period of time during which a block is generated. Each epoch can be associated with a specific sequence of blocks or height of blockchain. Whenever a node expects a message from the primary, it starts its timer and stops it when a decision is reached on the message. In case a node does not reach any decision after some period (timeout period), it times out and multicasts its timeout complaint to the root committee. This will either result in a view change or the node will download the missing message/block (if it did not receive the expected message). The node then doubles its timeout value to make sure that primaries get enough time to send their messages. This doubling of timeout value is called exponential backoff in the literature [5].

For ease of understanding first we describe and prove correctness of the basic Hermes protocol in which the primary proposes the next block once the current proposed block is committed. Then in Section 6 we discuss how Hermes can be optimized so that the primary can propose next block once it collects at least $\lfloor c/2 \rfloor + 1$ responses from the impetus committee (Efficient Optimistic Responsiveness).

3 Protocol Overview

In Hermes, the primary proposes a block to the impetus committee of size c. The impetus committee is running an agreement phase based algorithm (Algorithm 1). If $\lfloor c/2 \rfloor + 1$ impetus committee nodes agree on the block proposed by the primary and $2f+1$ regular nodes agree on the proposed block (which they received from the impetus committee), then the block is committed locally by a node and will be added to the blockchain. During normal execution of Hermes we use the same name for a message and its respective phase.

Normal execution of the protocol is shown in Figure 1 and can be summarized as follows:

1) The primary proposes the block (which contains transactions sent by clients) to the impetus commit-

tee ($Pre\text{-}Prepare$ message).
2) Each impetus committee member verifies the block and makes sure that there is only one block proposed for the expected height (by collecting signatures from at least $\lfloor c/2 \rfloor + 1$ impetus committee members through $Pre\text{-}Proposal$ messages).
3) The block is then proposed using the $Proposal$ message which includes primary signature along with $\lfloor c/2 \rfloor + 1$ aggregated signatures of impetus committee members from $Pre\text{-}Proposal$ messages.
4) Upon receipt of a block, regular nodes verify the aggregated signature and the transactions within the block.
5) If the block is found to be valid, each regular node responds with the signed $Confirm$ message.
6) Upon receipt of $2f + 1$ $Confirm$ messages from regular nodes, each impetus committee member as well as the primary commits the block. Each member of the impetus committee broadcasts approval of the majority nodes in the form of $2f + 1$ aggregated signatures from regular nodes in $Approval$ message.
7) Upon receipt of approval, each regular node commits the block, which is then added to the local history.
8) After committing locally, each node sends a $Reply$ message to the client. The client considers its transaction to be committed upon receipt of $2f + 1$ $Reply$ messages (at least $f + 1$ similar $Reply$ messages are from correct nodes).
9) Soon after committing a block the primary begins the next epoch (messages are shown in blue) and proposes a new block to the impetus committee using $Pre\text{-}Prepare$ message. The impetus committee members begin the $Pre\text{-}Proposal$ phase and the protocol progresses through each of its phases as described above.

3.1 Accountability

To discourage a malicious primary as well as impetus committee nodes from equivocation, Hermes nodes will have to provide a fixed amount of stake while joining the network as done in Proof-of-Stake (PoS) protocols [19], [24]. But this amount will be equal for every participant in the network. Since every participant has equal stake, the probability of being selected as the primary node or a member of the impetus committee is the same for every node. Alternatively, any node involved in equivocation can be blacklisted in the network.

3.2 Selecting Impetus Committee Members

Consider a set of n nodes and let $\mathcal{N} = \{i | \, 1 \leq i \leq n\}$ be their unique node ids, which for simplicity are assumed to be taken between 1 and n. Since all nodes are regular, $\mathcal{N}$ also denotes the set of regular nodes. Suppose that out of the n nodes at most f are faulty such that $f < n/3$; actually, we will assume the worst case $n = 3f + 1$.

Let $\mathcal{C} \subset \mathcal{N}$ denote the set of nodes in the impetus committee, such that $|\mathcal{C}| = c$, where $c \ll n$ is a predetermined number that specifies the size of the impetus committee (from now on $\mathcal{C}$ and impetus committee will be used interchangeably in the paper). The impetus committee $\mathcal{C}$ is formed by randomly and uniformly picking a set of c nodes out of n. In addition to the impetus committee members, there is a primary node picked randomly and uniformly out of the remaining $n - c$ nodes.

Since $f < n/3$, therefore on expectation, the impetus committee will have less than $c/3$ faulty nodes. Hence, $\mathcal{C}$ will likely have less than $c/2$ faulty nodes as well. Nevertheless, as the members of $\mathcal{C}$ are randomly picked, having $c/2$ or more faulty nodes in $\mathcal{C}$ are at long odds.

Moreover, the primary might be faulty as well. However, a view change can address primary as well as impetus committee failure. We carry on with analysis to precisely determine the likelihood of different scenarios for picking the members in $\mathcal{C}$. In the formation of $\mathcal{C}$ the number of possible ways to pick any specific set of c nodes out of n is $\binom{n}{c}$. The probability to pick exactly a correct nodes and b faulty nodes in $\mathcal{C}$, such that $a + b = c$, is $\frac{\binom{n-f}{a}\binom{f}{b}}{\binom{n}{c}}$. Therefore, the probability P_f of having at least $c/2$ faulty nodes ($b \geq c/2$) in $\mathcal{C}$ will be:

$$P_f = \sum_{b=\lceil c/2 \rceil}^{c} \frac{\binom{n-f}{a}\binom{f}{b}}{\binom{n}{c}}. \tag{1}$$

If $\mathcal{C}$ is unable to generate a block by the end of the timeout period, then $\mathcal{C}$ is replaced by another randomly chosen committee and a new primary through view change. Keeping the failure probability P_f constant, c grows sublinearly with n. Therefore, $c \ll n$ for large n.

View Change Probability due to Byzantine Behavior. View change can be triggered either by the failure of impetus committee $\mathcal{C}$ or failure of a primary. More specifically the two cases that can ultimately result in a view change are: (i) $b \geq c/2$, where b is the number of faulty nodes in $\mathcal{C}$ (this comes with probability P_f), or (ii) when $b < c/2$ and the primary node is faulty. For the latter case ii, since we choose the primary randomly from $n - c$ nodes if the total number of faulty nodes is $f < n/3$, then the probability of primary being faulty is at most $(f - b)/(n - c) < n/(3(n - c))$; hence, the probability that case ii occurs is less than $n(1 - P_f)/(3(n - c))$. Therefore, the probability P_v of having view change due to case i or ii is bounded by $P_v < n(1 - P_f)/(3(n - c)) + P_f$. Since P_f is approaching 0 and $n \gg c$, the upper bound of probability P_v can be approximated by $1/3$ asymptotically to the limit of n.

Probability of at Least One Correct Node in $\mathcal{C}$. For a view change to be initiated, it requires at least one correct node in $\mathcal{C}$. In the worst case, all the nodes in $\mathcal{C}$ are faulty, namely, $b = c$; this is the total failure scenario that does not allow view changes. We observe that the probability of not having any correct node in $\mathcal{C}$ for different values of n and c in our experiments is at most 3.8×10^{-22}. Consequently, the probability of avoiding total failure is extremely high. Since the total failure probability decreases rapidly as c increases, one can choose the value of c based on a target total failure probability.

4 The Protocol

The protocol begins with a client c sending its signed transaction $\langle REQUEST, o, t, c \rangle_c$ to the primary or broadcasting it to all the nodes, where o is the operation requested by the

Algorithm 1: Algorithm for primary node

1 **if** *There is* β *and* $\beta.s == HighApproval.s + 1$ **then**
2 **if** i *has the payload* m *for* β **then**
3 Generate block B from β with payload m
4 **end**
5 **else**
6 Request $Proposal$ for β from $\mathcal{C}$
7 **upon** *Receipt of* $Proposal$ **do**
8 Generate block B from payload m in $Proposal$
9 **end**
10 **upon** *Receipt of* $2f + 1$ *negative response* **do**
11 Generate block B and also attach $2f + 1$ negative responses for β
12 **end**
13 **end**
14 **end**
15 **else**
16 Generate block B
17 **end**
18 Broadcast B to the set $\mathcal{C}$ `// Pre-Prepare`
19 **upon** *receipt of* $2f + 1$ *valid* $Confirm$ *messages* **do**
  `// Commit block and increment height`
20 $\psi_i(B, s) \leftarrow true$
21 $s = s + 1$
22 Send $Response$ to clients
23 **end**

Algorithm 2: Node $i \in \mathcal{C}$

1 **upon** *receipt of valid* B **do**
2 Broadcast $Pre\text{-}Proposal$ message to $\mathcal{C}$
3 **end**
4 **upon** *receipt of* $\lfloor c/2 \rfloor + 1$ *valid* $Pre\text{-}Proposal$ *messages for* B **do**
5 Build the $Proposal$
6 Broadcast the $Proposal$ message to regular nodes (except the primary)
7 Send β to the primary
8 **end**
9 **check always for** *receipt of a valid* Γ_j *from regular node* j *or* Γ_p *from primary* **then**
10 Execute Algorithm 3
11 **end**
12 **check always for** *for* Γ_p *response* **then**
13 Forward the response to the primary
14 **end**
15 **if** *not received* B *by block timeout* **then**
16 Broadcast Γ_i to regular nodes
17 Accept messages from regular nodes to synchronize local history
18 **end**
19 **upon** *receipt of* $2f + 1$ *valid* $Confirm$ *messages* **do**
20 $\psi_i(B, s) \leftarrow true$
21 Broadcast $Approval$ message
22 Send $Response$ to clients
23 **end**
24 **check always for** *Receipt of first* $ConfV$ *before receiving* $Proof$ **then**
25 Broadcast $ConfV$
26 **end**
27 **check always for** *detecting proof of maliciousness:* E *complaint or* $f + 1$ Γ *complaints* **then**
28 Broadcast $proof$
29 **end**
30 **check always for** *Receipt of 2f+1 ConfV for the view change* **then**
31 Do not send Pre-Proposal/Confirm message anymore for this view
32 Build $ApproveV$ from $2f + 1$ ConfV messages
33 Broadcast the $ApproveV$ to regular nodes
34 **end**

client, t is the timestamp (primary uses t to order requests from client c). If the primary proposes a block containing a set of transactions, then clients send their transactions only to the primary. The primary will collect the transaction into a block and propose it to the network of nodes. Once the block is committed, nodes send a response (more information about the response message is provided in the next subsection) to each client confirming the execution of the respective transaction. If the client c does not receive a response within a specified time interval, it will broadcast its request to all the nodes. Each node relays the transaction to the primary and will expect the primary to propose this transaction within some time. If the transaction is not proposed, the primary will be considered faulty and a view change will be triggered.

The primary in the protocol may choose to broadcast the block of hashes of transactions (instead of transactions) to save bandwidth or increase throughput as done in [25]. In this case, the client has to broadcast the transaction to all nodes. The primary will then propose a block containing hashes of transactions. Since the hash size is often much smaller than the transaction size, a block will carry more hashes to increase the throughput. If the primary does not propose a transaction after the specified time, other nodes will relay the transaction to the primary similar to the previous client-node interaction. This is done to prevent faulty clients to trigger unnecessary view changes. Faulty clients may not send the transaction to the primary or the transaction might get delayed or lost. Therefore, other nodes need to relay the transaction to the primary. In case the primary still does not propose the transaction relayed to it by other nodes, it will be considered faulty and will be replaced. More discussion of this case is given in Section 7.

The basic operation of our protocol, Hermes, is presented in Algorithms 1, 2, and 4, which describe the normal execution between the impetus committee $\mathcal{C}$ and regular nodes. Algorithm 3 is executed by synchronization subprotocol. If normal execution fails, then our protocol switches to view change mode executing Algorithms 5, 6 and 7 to recover from failure. Note that the members of $\mathcal{C}$ and the primary also run themselves the protocols for regular nodes in normal mode. The protocol evaluation is presented in Section 7.

4.1 Happy Case Execution

The currently designated primary node p proposes a block by broadcasting a $Pre\text{-}Prepare$ message to $\mathcal{C}$ (Algorithm 1, line 18). A $Pre\text{-}Prepare$ message from primary p sends a newly created block $B = (\langle \text{“}Pre\text{-}Prepare\text{”}, v, s, h, d, o' \rangle_p, m)$ which contains the

Algorithm 3: Synchronization sub-protocol for node e

```
input : Γ
if e ∈ C then
    // If primary requests Proposal from previous view
    if Γ_p then
        if have Proposal for Γ_p then
            Send Proposal to the Primary
        end
        else
            // Send negative response
            Send F to primary
            Forward Γ_p to regular nodes
        end
    end
    if Γ_j & j ∉ C then
        Update node j by sending requested blocks
    end
end
else
    if Γ_i & i ∈ C then
        Send missing blocks to committee member i
    end
    if Γ_p then
        if have Proposal then
            Send Proposal to C
        end
        else
            Send F to primary
        end
    end
end
```

view number v, block sequence number s, transaction list m, its hash h, the previous blockhash d and optional field o' which can be used by the primary to send the proof $2f + 1$ of negative responses (F) during first epoch of its view (new primary in its first epoch might request the latest $Proposal$ from the previous view. If nodes do not have the $Proposal$ they will send negative response F). More details about this can be found in the subsection 4.3. Let $\rho = \langle \text{“}Pre\text{-}Prepare\text{”}, v, s, h, d \rangle_p$.

A node i in $\mathcal{C}$ begins $Pre\text{-}Proposal$ phase of the algorithm after receipt of a $Pre\text{-}Prepare$ message. Then, node i broadcasts a $Pre\text{-}Proposal$ message $\langle \text{“}Pre\text{-}Proposal\text{”}, v, s, h, i \rangle_i$ if it finds the $Pre\text{-}Prepare$ message to be valid. The validity check of the $Pre\text{-}Prepare$ message includes checking the validity of s, v, d, h and transactions inside m (Algorithm 2, lines 1-3). If node i receives $\lfloor c/2 \rfloor + 1$ $Pre\text{-}Proposal$ messages from other members of $\mathcal{C}$ for block B then the node i will successfully create a proposal block $(\langle \text{“}Proposal\text{”}, v, s, h, d \rangle_{\sigma_r}, B)$. This proposal block can be compressed into $(\langle \text{“}Proposal\text{”}, \rho \rangle_{\sigma_r}, m)$ and then node i will broadcast it to the regular committee members; σ_r aggregates the signatures of the $\lfloor c/2 \rfloor + 1$ members of $\mathcal{C}$ that contributed to the $Proposal$. Let $\beta = \langle \text{“}Proposal\text{”}, \rho \rangle_{\sigma_r}$ since the primary already has the payload m (from block B it already proposed), the impetus committee member i will only forward β to the primary instead of sending the whole $Proposal$ (Algorithm 2 lines $4 - 8$).

Algorithm 4: Regular node k

```
upon receipt of valid Proposal (valid β if k is also primary) from C do
    // Confirm proposal
    η_j(B, s) ← true
    Generate Confirm message and broadcast it to C
    Send Confirm message to primary
end
if timeout for a block or receipt of invalid block then
    Broadcast Γ ‖ E complaint to C
    if sequence number of block is out of order then
        Store the block locally and wait to fill the history gap
    end
end
upon receipt Approval message do
    if k ∈ C or k is primary then
        ignore
    end
    else
        foreach ordered block do
            // Commit the block at height s
            ψ_k(B, s) ← true
        end
        Send Response to clients
    end
end
// If a request for missing msg received
check always for receipt of a valid Γ_i ‖ Γ_p from i ∈ C then
    Execute Algorithm 3
end
// Initiate view change actions
check always for Receipt of Proof ‖ ConfV then
    Broadcast ConfV to C
end
check always for Receipt of a valid ApproveV for view change then
    // Transition to new view, based on common random number generation seed
    Randomly select members of C from N
    if k is not primary then
        Execute Algorithm 5 ‖ Algorithm 6
    else
        Execute Algorithm 7
    end
end
```

Upon receipt of a $Proposal$ message from $\mathcal{C}$, the regular nodes check if it is signed by at least $\lfloor c/2 \rfloor + 1$ members of $\mathcal{C}$. Regular nodes also verify the block by performing format checks and verification of each transaction against their history. If verification is successful, a regular node j sends back a signed $Confirm$ message $\langle \text{“}Confirm\text{”}, v, s, h, j \rangle_j$ to $\mathcal{C}$ and the primary (Algorithm 4, lines 1-5). The confirmation of a block B at height s by a node j is denoted by

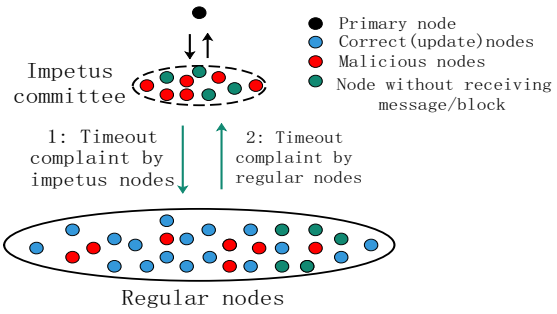


Fig. 2: Time out complaint

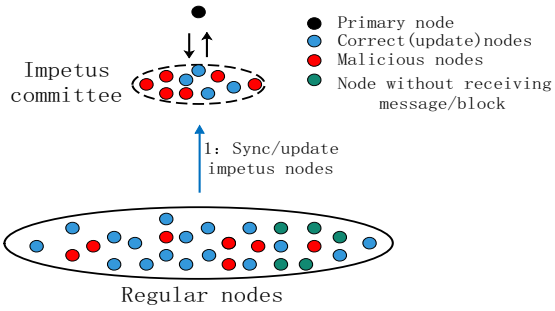


Fig. 3: Updating $\mathcal{C}$

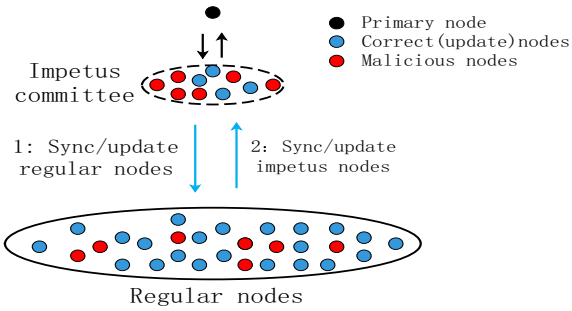


Fig. 4: Updating regular nodes

$\eta_j(B, s) \leftarrow true$.

Each member i of $\mathcal{C}$ aggregates $2f + 1$ signatures σ for $Confirm$ messages and then commits the block locally ($\psi_i(B, s)$ is used to show commit of a block B by a node i at the height s). The node i then broadcasts $Approval$ message $\langle$"$Approval$", $v, s, h\rangle_\sigma$ to regular nodes. Each member i of $\mathcal{C}$ will also send the $Response$ message ($\langle Response, s, v, r, t, i\rangle_i$) to each client confirming the execution of the respective transaction (with the timestamp t that the client submitted the transaction and its result r) that it contributed to the block B (Algorithms 1, lines 19-23 and Algorithm 2 lines $19-23$). Upon receipt of a valid $Approval$ message from $\mathcal{C}$ (for the current block in the sequence), the regular member k also commits the block ($\psi_k(B, s) \leftarrow true$) as shown in Algorithm 4, lines 12-22. Each node k (after committing the block) also sends a $Response$ message as shown in Algorithm 4 line 20.

4.2 Synchronization Sub-protocol

The impetus committee $\mathcal{C}$ might be faulty, when it has $\lceil c/2\rceil$ or more faulty nodes (shown in red in Figures 2, 3, 4). In such a case, the faulty members of $\mathcal{C}$ might attempt to not update $\zeta \leq f$ regular nodes without triggering view change; namely, not sending $Proposal$ and other messages to the ζ nodes. (In the upper bound of ζ, f may include at most $\lceil c/2\rceil - 1$ failed members of $\mathcal{C}$.) These ζ nodes (shown in green in Figure 2) may not be participating in the consensus process as they have not received messages from members of $\mathcal{C}$ (as majority of $\mathcal{C}$ have failed). Therefore, they will need to sync their history (download messages) with other nodes.

Suppose that node j is a regular node that needs to be synchronized ($j \in \zeta$) (download missing blocks). Let i be a member of $\mathcal{C}$ such that i has received a timeout complaint $\Gamma_j = \langle$"$Timeout$", $v, s', h', s'', h'', \tau, j, \rangle_j$ mentioning that j has not received blocks between the sequences s' and s'' with respective hashes h' and h''. The fields s', h' s'', h'' and τ are optional. The message type τ shows the type of message missing, e.g. a block or an aggregated view change message $\mathcal{Q}$ (will be discussed along with other message types in subsection 4.3). The view number v identifies the primary. If the requested block has not been sent previously, then the node i will forward missing messages for missing blocks as shown in Figure 4.

If node i times-out without receiving a valid expected message during the consensus process (as shown in the Figure 2) it will broadcast a complaint Γ_i to regular nodes. Consequently, a node i in $\mathcal{C}$ can recover a block by receiving it from regular nodes (as shown in the Figure 3). Thus, members of $\mathcal{C}$ and regular nodes synchronize their history (download missing blocks from each other) while keeping message complexity low (avoid quadratic message complexity).

When a new primary is elected it has to make sure that if a block is committed by at least a single correct node then it should be committed by at least $2f + 1$ nodes before proposing new blocks. In this case, the success of the primary is only guaranteed if equivocation is not performed by the previous primary along with the impetus committee. In the presence of equivocation by the primary as well as the impetus committee the protocol will fall back to R-safety. $\Gamma_p = \langle$"$Timeout$", $v', \beta\rangle_p$ is a specific type of timeout complaint by the new primary to request the latest uncommitted $Proposal$ (actually the payload m for β) from previous view. v' denotes the view of the current primary. A negative response from $2f + 1$ nodes for Γ_p request will prove that this $Proposal$ has not been committed by any node. If the primary receives a response for Γ_p it will re-propose the block. Thus, this additional recovery step after view change makes sure that safety is held after the view change. More details about the Γ_p will be discussed in the subsection 4.3.

Algorithm 3 is about the synchronization subprotocol. Lines 1-14 are executed if the node is a member of the impetus committee. In lines 1-10 node e responds to the Γ_p request from primary node. After sending the negative response to the primary, node e forwards Γ_p to regular nodes (Algorithm 3 line 8). In lines 11-13, the impetus committee member e responds to the request of a regular node j by sending the missing blocks to the node j. Lines 15-27 are executed if node e is not a member of the impetus committee. In lines 16-18, node e responds to an impetus member request/complaint Γ_i. In lines 19-26, node e responds to request Γ_p from the primary node.

The primary node might be malicious which can cause the impetus committee $\mathcal{C}$ to fail by not proposing a block. Another possible cause of failure can be the presence of majority of malicious nodes in $\mathcal{C}$. As a result, impetus committee nodes cannot collect at least $\lfloor c/2 \rfloor + 1$ signatures for proposed block. In a rare case, it is also possible that both the primary and the impetus committee $\mathcal{C}$ fail; in such a case the primary can collude with the $\mathcal{C}$ to perform block equivocation. The failure can be detected if $f + 1$ nodes send timeout complaint or a single node sends a complaint for equivocation by the primary and $2f+1$ nodes receive it. Upon detecting failure, Hermes will switch to view change mode to select new primary and impetus committee.

Algorithm 5: View change for regular node k

```
1  Send local history V_k to new primary
2  upon receipt of a valid Q message from a new committee
     member i do
3      Extract the most recent valid history from Q
4      if local history is not same as most recent history in
         Q then
5          Synchronize local history according to Q
6      end
7      Broadcast READY message (R_k) to new primary
8  end
9  check always for receipt of P from i ∈ C then
10     if P has at least 2f + 1 distinct READY messages
         then
11         return to Algorithm 4
12     end
13 end
14 check always for Γ_i where i ∈ C then
15     Execute Algorithm 3
16 end
```

4.3 View Change

The view change subprotocol in Hermes is different from conventional BFT-based protocols as it has an additional recovery phase to make sure if a block is committed by a single correct node just before view change, then it will be committed eventually by all other correct nodes (In the absence of equivocation). A view change can be triggered if there is sufficient proof of maliciousness of the primary or the impetus committee. The two types of complaint by a node i that can form a proof against the Byzantine primary are the Γ complaints (as discussed in subsection 4.2) as well as $E = \langle \text{“}Explicit\text{-}complaint\text{”}, v, \epsilon, i \rangle_i$ complaint, where ϵ determines the reason for the complaint which can be either an invalid block $Proposal$ or multiple $Proposal$s by the primary and/or $\mathcal{C}$ with the same sequence number. A $Proof$ is simply formed by $f + 1$ valid Γ complaints or a single valid E complaint. A single valid $Proof$ is required to trigger a view change.

During each epoch, a regular node waits to receive a proposed block from $\mathcal{C}$. If a regular node i does not receive the block after a timeout then it considers that $\mathcal{C}$ has failed and reports this to $\mathcal{C}$ (Algorithm 4, lines 6-7). If $f + 1$ nodes report a timeout, then there is at least one correct $\mathcal{C}$ member j that will broadcast the aggregated $f + 1$ timeout complaints (Γ) to all regular nodes (as $Proof$ in Algorithm 2, lines 27-29). Upon receipt of $f + 1$ Γ complaints the regular nodes send back a confirm view change ($\langle \text{“}ConfV\text{”}, \langle v, j \rangle_j, \langle Proof \rangle_{\sigma_v} \rangle$) message to $\mathcal{C}$, where σ_v is the aggregated signature from at least $f + 1$ complaints from Γ messages (Algorithm 4, line 26-28). If complaint is of type E then the confirm view will be of the form ($\langle \text{“}ConfV\text{”}, \langle v, j \rangle_j, \langle Proof \rangle_i \rangle$), where the node i will be the node that has complained. If an impetus committee member receives a $ConfV$ message and was not aware of the $Proof$ (therefore did not broadcast it to the regular nodes), it will also broadcast the $ConfV$ to the regular nodes (Algorithm 2 line 24-25). This step is added to prevent a Byzantine member in $\mathcal{C}$ from triggering the view change in a subset of regular nodes by sending the $Proof$ messages to a few regular nodes.

Algorithm 6: View change for node $i \in \mathcal{C}$

```
1  if i ∈ C then
2      check always for (Q ‖ P) from primary then
3          Forward (Q ‖ P) to regular nodes
4          if P has at least 2f + 1 distinct READY
             messages then
5              return to Algorithm 2
6          end
7      end
8  end
9  check always for Γ_j where j ∉ C then
10     Execute Algorithm 3
11 end
```

Algorithm 7: View change for new primary

```
1  upon receipt of V_i from node i do
2      Q ← Q ∪ V_i
3      if Q contains at least 2f + 1 histories then
4          Broadcast Q to C
5      end
6  end
7  Extract the most recent valid history
8  if local history is different from most recent history then
9      Synchronize local history V_i according to the Q
         from regular nodes
10 end
11 check always for receipt of R_i then
12     P ← P ∪ R_i
13     if P has accumulated at least 2f + 1 distinct
         READY messages then
14         Broadcast P to members of C
15     end
16 end
17 check always for history update request from regular
     node i then
18     if node j has the latest history/block and has not
         already sent it to i then
19         Send the blocks up to the latest block to
             regular node i
20     end
21 end
22 Return to Algorithm 1
```

Once a member i in $\mathcal{C}$ receives $2f+1$ $ConfV$ messages, then i triggers a view change by broadcasting message $(\langle \text{“}ApproveV\text{”}, \langle v \rangle_\sigma, \langle Proof \rangle_{\sigma_v}, \langle i \rangle_i, \rangle)$ to all regular nodes (Algorithm 2, lines 30-34), where $\langle v \rangle_\sigma$ is the aggregate signature for $2f+1$ $\langle v, j \rangle_j$ from $ConfV$ messages sent by a regular node j. Upon receipt of the $ApproveV$ message each regular node will begin the view change process (Algorithm 4, lines 29-36). In the view change, members of the new impetus committee $\mathcal{C}$ along with a new primary is selected by each node using a pre-specified common seed for pseudo-random number generation [26], which guarantees that every node selects the identical $\mathcal{C}$ and primary. In pseudo-random generator algorithms, a random result can be reproduced using the same seed. It is very important to make sure the seed for a random generator is not biased. In other words, malicious nodes cannot affect the process of key generation. If Byzantine nodes can influence the seed generation process, then they can control the pseudo-random generator. For example, if the hash of the latest committed block is used as the seed for the next primary and committee selection, a Byzantine primary will build a block in such a way that the next primary will also be a Byzantine primary along with the impetus committee with a majority of Byzantine nodes. In such a case, liveness cannot be guaranteed and Byzantine primaries can censor transactions.

Seed for the Pseudo-random Generation: There has been research on how multiple parties can agree on single or multiple random bits, using coin-flipping [27], [28], [29], [30], [31], to generate an unbiased, random, and unpredictable seed for a pseudo-random generator. More recently, random-beacons, which make use of (t, n)-threshold signature schemes and are closely related to coin tossing have been used as a source of randomness in consensus protocols [32], [33]. In a (t, n)-threshold signature scheme n parties jointly generate a public key also called the group public key. Moreover, each party holds an individual secret key share. It takes t out of n parties to generate the group signature that can be verified using the group public key. Each node in the threshold-based random-beacon signs a common string and broadcasts its signature share. Upon receiving at least t distinct threshold signature shares for the common string, each node builds a threshold signature. The threshold signature is verifiable against the group public key. The hash of the threshold signature is used as the seed for the pseudo-random generator. These methods satisfy liveness, bias-resistance, public verifiability, and unpredictability. In Hermes, we use the view number v as the seed for the pseudo-random generator. Since the view number increments during view change (and nodes agree on the view number), therefore it is alive and bias-resistance. The view number used as a seed in a pseudo-random generator also satisfies public verifiability. Therefore users can choose among different options for acquiring seeds for pseudo-random generation, based on their requirements and assumptions (i.e, if unpredictability is required or not).

The New Primary: After randomly choosing the new primary as well as the new impetus committee $\mathcal{C}$, each node sends a $ViewChange$ message $V_k = (\langle \text{“}ViewChange\text{”}, v+1, s', h, h^*, k \rangle_k, \sigma, \beta)$ to the new primary (Algorithm 5 line 1). The $ViewChange$ message has information about the latest block in the local history of the node k ($Approve$ message can be built from V_k). It includes the latest committed block sequence number s', block hash h, block hash h^* in β, incremented view number $v+1$, and signature evidence of at least $2f+1$ (σ) nodes that have confirmed the block (through $Confirm$ message). The β part of this message includes the latest $Proposal$ (that node k has voted for during the $Confirm$ phase) and its respective $Pre\text{-}Prepare$ message from the last primary (without its payload m) which was received by the node k from $\mathcal{C}$ (through a block $Proposal$ message). The inclusion of β in the view change message is used to ***recover*** the block (transactions) if less than $f+1$ correct nodes have committed the block. We will discuss more about this at the end of the current section. The information in V_k is used to determine whether nodes have different local histories, which in turn can allow them to synchronize their histories by getting the possible missing blocks from the new members of $\mathcal{C}$. Upon receipt of V_k from node k, the new primary extracts the parts h (hash of the latest committed block), s', and k and also checks the validity of σ and β. Out of the $2f+1$ nodes that contribute to σ, it is guaranteed that at least $f+1$ are correct nodes and at least one out of these $f+1$ correct nodes has the latest block (committed by at least $f+1$ nodes). Once the new primary receives $2f+1$ V_k messages, it aggregates them into $\mathcal{Q}$, which it broadcasts to members of $\mathcal{C}$ (Algorithm 7, lines 1-6) and then $\mathcal{C}$ will broadcast the message $\mathcal{Q}$ to all nodes (Algorithm 6). Upon receipt of $\mathcal{Q}$, node k makes sure that its history matches the history in $\mathcal{Q}$ (agreed upon by at least $f+1$ nodes) and if it does, node k sends back a $Ready$ message $R_k = \langle \text{“}Ready\text{”}, v+1, s', h, k \rangle_k$ to the new primary (Algorithm 5, lines 2-8). The primary will aggregate $2f+1$ $Ready$ responses into a single P message and broadcast it to $\mathcal{C}$ (Algorithm 7, lines 11-16) which will be forwarded to all the nodes (Algorithm 6). Upon receipt of P, node k is now ready to take part in new view (Algorithm 5, lines 9-13). If node k's history does not match that of $\mathcal{Q}$ it will synchronize its history (Algorithm 5, lines 4-6).

Recovery During View Change. Recall that we add a recovery phase to the view change in order to significantly reduce the latency. During this recovery phase, the Hermes protocol makes sure that if there is any block that has been committed by up to f correct nodes, it will be committed by all correct nodes eventually (S-safety). As stated, S-safety may not hold and the protocol may fall back to R-safety, in case of equivocation by the primary and impetus committee. But we will also show in subsection 5.2, that once $f-1$ Byzantine nodes are blacklisted, then the protocol will always exhibit S-safety. If there is β in $V_k \in \mathcal{Q}$ such that the sequence s of β is equal to the sequence of the highest $Approval$ message [1, 2], plus one ($\beta.s == HighApproval.s + 1$) , then the new primary has to propose the respective block B for the β. This means there may be a block at the sequence $\beta.s$ that has been committed by at most f nodes. Therefore, this block (β's block) needs to be recovered.

1. As stated, $Approval$ message can be built from each V_k in $\mathcal{Q}$.
2. $HighApproval = \underset{Approval}{\text{argmax}} \{Approval.s | Approval \in \mathcal{Q}\}$. In other words, $HighApproval$ is an $Approval$ message with highest sequence in Q.

If the new primary has already received the $Proposal$ or the block B for β it will re-propose it in the first epoch of its view. On the other hand, if the new primary does not have the complete $Proposal$ (including its payload m) for β it will request it from $\mathcal{C}$ using a Γ_p complaint. If an impetus committee member e has the $Proposal$ (payload m) it will send it back to the primary. If not, it will forward the Γ_p complaint to the regular nodes as well as sending a negative response $F = \langle \beta, false \rangle_e$ to the primary (Algorithm 3, lines 1-10). Regular nodes will also send back the $Proposal$ message for β if they have it to the impetus committee (which will forward it to the primary) or a negative response F to the primary. If the primary receives back the $Proposal$ it will re-propose the block. Else the primary will have to propose another block with $2f+1$ aggregated signatures for negative response F being attached to it. By attaching $2f+1$ aggregated signatures for negative response F to the block the primary proves that $Proposal$ for β was not committed by any correct node (Algorithm 1, lines 1-14). Therefore, if there is a β in $V_k \in \mathcal{Q}$ during the recent view change, then in the first epoch after the view change the new primary either has to propose the relative block for β or include $2f+1$ negative responses in the first block B that it proposes in the new view. If the primary fails to do so, a view change will occur (through E complaint).

View Change Complexity. The actual performance of view change mainly depends on the timeout values and the mechanisms for primary selection [17]. Since view-change messages are generally way smaller in size than the blocks, the message complexity during view change does not have significant effect on performance. For example, the quadratic message size of Q is about $\approx 2\%$ of the block size. On the other hand, the number of signatures processed by the nodes in critical-path during view change also has significant effect on performance. In Hermes (like other classic BFT variants), the number of signatures (authenticators) processed by a node during view change is $O(n^2)$. On the other hand, the authenticator verification complexity can be improved to $O(n)$ by using the technique introduced in Fast-HotStuff [34].

At least $2f+1$ $ViewChange$ messages $V_k = (\langle \text{“}ViewChange\text{”}, v+1, s', h, h^*, k \rangle_k, \sigma, \beta)$ are aggregated by the primary into $\mathcal{Q}$ and sent to nodes. Each view change message has an aggregated signature σ from $2f+1$ nodes, resulting in quadratic signature verification. But as stated, a verifying node only needs to verify two aggregated signatures (i.e., an aggregated signature of $\mathcal{Q}$ and an aggregated signature of the view change message with the highest view number). The verification of an aggregated signature for $\mathcal{Q}$ shows that $2f+1$ nodes have sent their $ViewChange$ messages, and the verification of $ViewChange$ with the highest view verifies the validity of the latest committed block in Q. Byzantine nodes cannot forge the latest committed block because it requires $2f+1$ signatures including at least $f+1$ signatures from honest nodes. If there is a block B that has been committed (by at most f honest nodes) but its proof of commitment is not included in Q, then there will be at least one view change message V that contains β for that block B. Block B will be recovered in the recovery phase as previously described (more details in Lemma 5). There is a possibility that some nodes send invalid $ViewChange$ messages. Those nodes can be blacklisted if they have properly signed the $ViewChange$ message. Below we describe a blacklisting mechanism for this equivocation fault. It can be generalized and used for any other Byzantine behavior where there is sufficient proof against malicious nodes.

4.4 Blacklisting Nodes that Perform Equivocation

As mentioned in subsection 4.3, a single explicit complaint by a node i ($E = \langle \text{“}Explicit\text{-}complaint\text{”}, v, \epsilon, i \rangle_i$) against the primary is enough to trigger a view change. This type of complaint made by only one node can prove that the primary is Byzantine. As stated previously, ϵ determines the reason for the complaint. ϵ can be a data structure with several fields and each field can be represented by another data structure specifying the proof for the complaint (i.e., equivocation, invalid block, etc.). Indeed, it is a design choice to decide which proofs can have their fields included in ϵ in addition to the equivocation. Currently, ϵ has a field $Eq = \langle hs, v, Ss \rangle$ for equivocation, where hs is a slice/array of equivocated block headers ρ (also containing primary signatures for each header) and Ss is the slice/array for aggregated signatures σ_c of the impetus committee. The signatures in hs and Ss are proof that the primary and at least one member of the impetus committee have signed equivocating blocks. If a node i detects equivocation, it sends an E complaint to the impetus committee which will eventually result in a view change as described in Subsection 4.3. Therefore, if any view change occurs due to the E complaint (no timeout), it proves without any doubt that the primary is malicious. As a result, the primary that has been changed due to the E complaint will be blacklisted by all honest nodes as described below.

After a view change due to equivocation (E complaint), each node expects the primary to propose a transaction to blacklist the previous primary (and other collaborators from the root committee) that performed equivocation. If the new primary is honest and has received the E complaint message during the view change process, then it will propose a block that includes a transaction (transaction also includes the proof ϵ) to blacklist the Byzantine nodes that were involved in equivocation. If the primary does not propose the transaction to blacklist the perpetrators of equivocation by the timeout period, then honest nodes forward the proof of equivocation ϵ to the primary and wait again for the proposal of the blacklisting transaction. If by the timeout an honest node does not receive a block containing a blacklisting transaction from the primary, it sends a timeout Γ complaint to the impetus committee. Since after a view change at least $f+1$ honest nodes are aware of ϵ, complaints from $f+1$ honest nodes will trigger a view change. After a view change, honest nodes wait again to receive blacklisting transactions from the primary, and this process repeats until the primary proposes blacklisting transactions in a block. Moreover, if a node is aware of the proof of equivocation ϵ against a node i and the node i has not been blacklisted yet but is randomly selected as the primary, then the honest node can send an E complaint to the impetus committee in order to trigger a view change.

There is also a possibility that a Byzantine primary of view v has been replaced through a Γ (timeout) complaint and later on a node j receives two equivocating block

proposals (due to network asynchrony). In this case, even though primary v is not a primary anymore, it should still be blacklisted. Therefore, node j will forward the E complaint with proof of equivocation ϵ to the impetus committee $\mathcal{C}$. The ϵ will be ignored if the primary has already been blacklisted. Otherwise, the impetus committee broadcasts the ϵ to all the nodes including the primary. The primary in charge receives ϵ and will then include the proofs in a transaction and add this transaction in the block proposal. The primary then proposes the block containing blacklisting transaction in addition to the normal transactions. Once the block is committed, the Byzantine nodes involved in equivocation will be blacklisted.

If this transaction regarding blacklisting a Byzantine primary is not proposed by the primary within some specific interval, then the honest nodes will issue timeout complaints, which will result in a view change. This will continue unless the Byzantine primary and its collaborators are blacklisted as discussed previously.

5 Analysis and Formal Proofs

In this section we provide proofs for R-safety, S-safety and liveness properties for Hermes.

5.1 Safety

Hermes has relaxed the agreement (safety) condition in the presence of equivocation faults. In the presence of equivocation in Hermes, a block is committed if at least $f+1$ correct nodes commit the block (R-safety). If $2f+1$ nodes commit a block it is guaranteed that at least $f+1$ of these nodes are correct. Due to this relaxation in the agreement condition the client also needs to wait for at least $2f+1$ $Response$ messages for its transaction to consider it committed (to make sure at least $f+1$ correct nodes have committed the block).

For simplicity we use $\mathcal{H}$ as the set of all correct nodes such that $\mathcal{H} \subseteq \mathcal{N}$ and $|\mathcal{H}| \geq 2f+1$ in the proof of correctness.

Lemma 1. *Hermes is R-safe during normal mode.*

Proof. Consider two different blocks B and B' with respective heights s and s' where each block has been committed by at least $2f+1$ nodes. It suffices to show that $s \neq s'$. Suppose, for the sake of contradiction, that during happy case execution (normal mode) both B and B' are committed at the same height ($s = s'$). Let $\mathcal{K}_1 \subseteq \mathcal{N}$ be the set of nodes that commit B such that $|\mathcal{K}_1| \geq 2f+1$ and for each $i \in \mathcal{K}_1$, $\psi_i(B,s) \leftarrow true$ (all members of $\mathcal{K}_1$ have committed B at height s). Similarly, let $\mathcal{K}_2 \subseteq \mathcal{N}$ be the set of nodes that commit B' such that $|\mathcal{K}_2| \geq 2f+1$ and for each $i \in \mathcal{K}_2$, $\psi_i(B',s') \leftarrow true$. Since $n \geq 3f+1$ and $f < n/3$, we get for $\mathcal{K}_1 \cap \mathcal{K}_2 = \mathcal{K}$ that $|\mathcal{K}| \geq f+1$. Hence, $\mathcal{K} \cap \mathcal{H} \neq \emptyset$. Therefore, there is an $i \in \mathcal{K} \cap \mathcal{H}$ such that $\psi_i(B,s) \leftarrow true$ and $\psi_i(B',s') \leftarrow true$ (that is, i committed both blocks). However, since $i \in \mathcal{H}$, i can only commit one block at any specific sequence, and hence, it is impossible that i executed both $\psi_i(B,s) \leftarrow true$ and $\psi_i(B',s') \leftarrow true$ with $s = s'$ and $B \neq B'$. Therefore, $s \neq s'$, as needed. □

Lemma 2. *Hermes is R-safe during view change.*

Proof. Consider the latest block B' at height s' that was committed by at least $2f+1$ nodes before the view change. We will show that B' will be included in the blockchain history after the view change. Let $\mathcal{H}_c \subseteq \mathcal{H}$ be the set of honest nodes that have committed B' (that is, $\psi_i(B',s') \leftarrow true$ for each $i \in \mathcal{H}_c$). Since the number of Byzantine nodes is f, we get that $|\mathcal{H}_c| \geq f+1$. At least a node $i \in \mathcal{H}_c$ must have reported a view change message V_i to the primary, such that V_i is aware of B' and s'. Since the primary node collects $2f+1$ view change messages into $\mathcal{Q}$, we get that $\mathcal{Q}$ contains at least $f+1$ view change messages from the nodes in $\mathcal{H}$ where at least one node is in $\mathcal{H}_c$ too. Hence, when a node j receives $\mathcal{Q}$ from the new primary, j will know that B' has been committed, a guarantee that B' is valid. Thus, block B' will be inserted into the local history of every node that receives $\mathcal{Q}$, and becomes part of the blockchain history. □

Therefore, we get the following theorem:

Theorem 3. *Under normal operation and a view change, Hermes provides relaxed safety guarantees (Hermes is R-safe).*

Proof. From Lemmas 1 and 2 we see that Hermes provides relaxed safety guarantees (Hermes is R-safe) during normal mode as well as view change mode. □

Lemma 4. *Hermes is S-safe during normal mode.*

Proof. We will prove this lemma by contradiction. Let us assume that during normal mode at least one correct node (say, node i) commits a block B at height s then we have $\psi_i(B,s) \leftarrow true$. This means there exists a set $\mathcal{K}_1 \subseteq \mathcal{N}$ such that $|\mathcal{K}_1| \geq 2f+1$ and for each $i \in \mathcal{K}_1$, $(\eta_i(B,s) \leftarrow true)$ (all members of $\mathcal{K}_1$ have confirmed B at height s). We also assume that there is another node j that has committed a block B' at height s' such that $s' = s$ ($\psi_j(B',s') \leftarrow true$). Therefore, there is another set $\mathcal{K}_2 \subseteq \mathcal{N}$ such that $|\mathcal{K}_2| \geq 2f+1$ and for each $i \in \mathcal{K}_2$, $(\eta_i(B',s') \leftarrow true)$. Based on $n \geq 3f+1$ and $f < n/3$, we get $\mathcal{K}_1 \cap \mathcal{K}_2 = \mathcal{K}$ such that $|\mathcal{K}| \geq f+1$. Hence, $\mathcal{K} \cap \mathcal{H} \neq \emptyset$. Therefore, there is an $i \in \mathcal{K} \cap \mathcal{H}$ such that $\eta_i(B,s) \leftarrow true$ and $\eta_i(B',s') \leftarrow true$ (that is, i confirmed both blocks). However, since $i \in \mathcal{H}$, i can only confirm one block at any specific level (height) during normal mode, and hence, it is impossible that i confirmed both $\eta_i(B,s) \leftarrow true$ and $\eta_i(B',s') \leftarrow true$ with $s = s'$ and $B \neq B'$. Therefore, $s \neq s'$. □

Lemma 5. *Hermes is S-safe during view change if there is no equivocation fault.*

Proof. Let us assume that a single node i has committed a block B at the height s ($\psi_i(B,s) \leftarrow true$) just before the view change occurs. That means there is a set $\mathcal{K}_1 \subseteq \mathcal{N}$ such that $|\mathcal{K}_1| \geq 2f+1$ and for each $i \in \mathcal{K}_1$, $(\eta_i(B,s) \leftarrow true)$. Therefore during view change there is another set $\mathcal{K}_2 \subseteq \mathcal{N}$ such that $|\mathcal{K}_2| \geq 2f+1$ and for each $i \in \mathcal{K}_2$, $V_i \in \mathcal{Q}$. As $n \geq 3f+1$, therefore we have $\mathcal{K}_1 \cap \mathcal{K}_2 = \mathcal{K}$ such that $|\mathcal{K}| \geq f+1$. Thus, $\mathcal{K} \cap \mathcal{H} \neq \emptyset$. Therefore, there is an $i \in \mathcal{K} \cap \mathcal{H}$ such that $\eta_i(B,s) \leftarrow true$ and $V_i \in \mathcal{Q}$. Since V_i contains β for block B, therefore the new primary has to re-propose block B in the first epoch of the new view in the height s as show in Algorithm 1 lines 1-18. □

5.2 The Equivocation Case

If the previous primary is malicious it can propose multiple blocks at the same height/sequence (just before the view change). Suppose the malicious primary proposes two blocks B' and B'' before the view change such that one of these blocks (say, block B') is committed by at most ξ number of nodes where $\xi \leq f$. In such a case it is not guaranteed that a block committed by ξ nodes can remain in the chain after the view change. Therefore, S-safety cannot be guaranteed. In this case, the protocol guarantees R-safety as long as $f-1$ Byzantine nodes are not blacklisted.

If $2f+1$ view change messages received by the new primary (correct) include a view change message V_k by the node k that contain the commit certificate ($Approve$ message with $2f+1$ signatures) for the block B' then it is guaranteed that the block B' will be re-proposed in the next epoch for the same height (not be revoked). But if the new primary collects the view change messages from another $2f+1$ nodes (not the f nodes that committed the block) then the new primary might either propose B' or B'' (randomly) in the next epoch. Therefore it is not guaranteed that the block B' that has been committed by at most f nodes (at the height h) will be re-proposed at the same height. This can happen because of the fact that out of these $2f+1$ nodes included in the $\mathcal{Q}$ prepared by the new primary at most f nodes can be Byzantine whereas at most another f might have received the block B''. At least one node can have the block B'. Therefore the new primary may receive two valid β messages, β' (for block B') and β'' (for block B'') from the nodes in the view change message (such that $(\beta'.s == \beta''.s == HighApproval.s + 1)$). The primary can re-propose any one of these blocks (B' or B'' for example) in the first epoch of the new view (if found to be valid). If block B' is proposed then it is fine as this block is committed by at most f nodes. But if B'' is proposed then the nodes that have committed the block B' will have to revoke the block B' and perform agreement on the block B'' at the same height/sequence. Block B' can be proposed in the next epoch if there is no common transaction between block B' and B''. If there is at least one common transaction among the blocks B' and B'' or some transactions in the block B' are invalid the primary can extract valid transactions from the block B' and propose them in another block. It should be noted that in such case the malicious primary and its culprits in the $\mathcal{C}$ can be punished by blacklisting and their stake in the network can be slashed.

Lemma 6. *After blacklisting $f-1$ Byzantine nodes, Hermes will guarantee S-safety.*

Proof. First, note that equivocation may break S-safety and cause the protocol to fall back to R-safety if there is a single Byzantine node in $\mathcal{C}$ in the presence of a Byzantine primary. In such a case, a Byzantine primary may propose two blocks for the same height, and at least one Byzantine node will vote for both of them. Since this equivocation can be detected, the primary and the node that has voted twice for two different blocks at the same height will be blacklisted. Thus, each time an equivocation breaks S-safety, at least two Byzantine nodes are blacklisted. There is also a possibility that a Byzantine primary equivocates but no node from the impetus committee collaborates in the equivocation. In that case, S-safety does not break. But the primary will be blacklisted. Hence, each time equivocation takes place, it will result in the blacklisting of perpetrator/s. Once $f-1$, nodes are blacklisted, then S-safety cannot be broken as there is no additional Byzantine node in the impetus committee to vote/sign for more than one proposed block at the same sequence. Hence, S-safety will be guaranteed. □

By proposing two equivocating blocks, at worst case there is a 50% chance that the S-safety might break (Assuming one of the two proposed blocks has been committed by at most f honest nodes) when the next primary is honest, as the next primary will randomly choose to re-propose one of the two blocks first (as discussed in subsection 5.2). There is also possibility that the primary does not receive any valid β for the equivocated block that has not been committed by any node, in that case S-safety will not break. It is guaranteed that the primary and its collaborator/s in the impetus committee will be blacklisted. To improve the probability of S-safety failure, a Byzantine primary and its collaborators in the impetus committee may try to propose more than two equivocating blocks. But in that case, more Byzantine nodes from the impetus committee have to collaborate. Hence, the number of Byzantine nodes to be blacklisted also increases (up to a maximum of $\lfloor c/2 \rfloor + 2$ in a single epoch). Once the number of Byzantine nodes gets $\lfloor c/2 \rfloor + 1$ in the impetus committee, then the Byzantine primary can propose more than $\lfloor c/2 \rfloor + 1$ equivocated blocks for the same sequence. In that case, $\lfloor c/2 \rfloor + 1$ Byzantine nodes from the impetus committee along with the primary will be blacklisted. But the probability of such an event is low. As a result, the Hermes design discourages Byzantine nodes from performing equivocation.

Similarly, blacklisted nodes will lose their stake in the network (if nodes have a stake in the network). Therefore, it is expensive for a Byzantine nodes to perform equivocation. Moreover, there is not enough incentive for a Byzantine nodes to perform equivocation as it will only cause processing delay in the network to recover from faults without affecting the client (without causing double spending).

As mentioned in subsection 5.2, in case of equivocation if less than $2f+1$ nodes (out of which at least $f+1$ are honest) have committed a block then this block might get revoked. But it should also be noted that unlike other forking protocols (Bitcoin, Ethereum, etc.) where a client has to wait for the confirmation time to make sure the block it has accepted will not be revoked, the revoking of block b has no affect on the client because a client will only consider a transaction in a specific block to be committed if and only if at least $2f+1$ nodes in the network commit that block. Thus, if a client considers a transaction to be committed then it has been committed by $2f+1$ nodes and it will not be revoked (as shown in Lemma 2). Therefore a Byzantine primary has no strong incentive to perform equivocation as a revoked transaction is not considered committed by the client.

5.3 Liveness

Hermes uses different means to provide liveness while keeping message complexity low (at $O(cn)$ messages). As the main communication among the nodes occur through

the $\mathcal{C}$, it is important that there must be at least one correct node in the $\mathcal{C}$ to guarantee liveness.

Another case that could potentially prevent liveness is when repeatedly selecting a bad $\mathcal{C}$ or malicious primary after a view change, which in turn triggers another view change, and this perpetuates without termination. However, as we show next, this extreme scenario may only occur with extremely low probability that quickly approaches 0. If the primary is malicious or the number of malicious nodes in the impetus committee $\mathcal{C}$ is at least $\lfloor c/2 \rfloor + 1$, then based on Algorithm 2 (lines 30-34), and Algorithm 4 (lines 29-36), a view change may occur. The probability P_v of such a bad event causing a view change is approximately a constant $1/3$ (Section 3). Treating each such bad event as a Bernoulli trial, we have that the bad events trigger consecutively κ view changes with probability at most P_v^κ, which quickly approaches 0 with exponential rate as κ increases linearly. Therefore, the probability of this scenario is negligible and does not affect liveness.

Additionally, a view change in Hermes also employ three techniques applied by the PBFT [5]. These techniques include: (1) exponential backoff timer for view change; (2) at least $f + 1$ complaints will cause a view change. These $f + 1$ complaints may be against the current view or higher views than the current view of a node. But the node will move with the changing the view of the smallest view of $f + 1$ complaints. This technique has been customized in Hermes with additional phases as described in 4.3. The first phase that has been added to Hermes include broadcasting $ConfV$ messages to regular nodes by impetus committee. The second phase include aggregation of at least $2f + 1$ $ConfV$ messages by the members of $\mathcal{C}$ into $ApproveV$ message and broadcasting it to regular nodes. These additional steps are added to make sure that at least $2f + 1$ nodes are aware of view change. And (3) faulty nodes ($f < n/3$) cannot trigger a view change.

It should be noted that even during a view change the performance of Hermes will not be affected by the slack primary. This is because the primary will send its message to the impetus committee and the impetus committee will forward the primary's message to regular nodes. Therefore view change messages from the primary will propagate proportional to the wire speed of prompt nodes.

6 Efficient Optimistic Responsiveness

Pipelining is an optimization technique that involves sending requests/messages back to back without waiting for their confirmation. This technique has been previously used in networking [35] as well as in consensus [17], [36] to improve performance. Therefore, as an optimization, the primary can propose a new block B' with sequence s after the $Pre - Proposal$ phase for block B with sequence $s - 1$ without waiting for the block B to get committed. The primary can wait only for majority of votes from the impetus committee to propose a new block. This technique decreases the wait time for the primary to propose the next block while making sure that with high probability the protocol will make progress.

In pipelined Hermes, we need to replace β with A_β, where A_β is an array of β in $ViewChange$ message V. As in basic Hermes the primary proposes the next block after it commits its parent block, therefore, there will be only one valid pending block in the network with β. But in case of pipelined Hermes, since the primary proposes a block after the $Pre - Proposal$ phase of its parent, by the end of a timeout a node might receive multiple uncommitted blocks. Therefore, the information about latest uncommitted blocks has to be included in A_β during the view change.

For state machine replication (SMR) in blockchains sending $Pre - proposal$ message, $Confirm$ message and committing a block during $Approval$ phase has to be done in order (serially) for each block. This means a node i will not confirm block B at the sequence s ($\eta_i(B, s) \leftarrow true$) before block B' with the sequence s' such that $s > s'$. Out of order messages can be cached and other operations, including signature verification and format checking, can be done concurrently. We leave more details on this to programmers who will be implementing this protocol.

7 Evaluations

We have implemented a prototype of optimized Hermes as well as chained HotStuff [17] (pipelined) using the Golang programming language. We chose pipelined HotStuff as it is a state-of-the-art consensus protocol. A variant of HotStuff called Libra-BFT [37] is also used by Facebook. We implemented HotStuff to have a similar code architecture with Hermes so that we can fairly compare both algorithms.

Experiments were conducted in the Amazon Web Services (AWS) cloud. Each node in the network was represented by an instance of type *t2.large* in AWS. Each *t2.large* instance has 2 virtual CPU (cores) and 8GB of memory. We recorded performance of Hermes as well as Hotsuff with network sizes of 40, 70, 100, 130, 160 and 190 nodes. The impetus committee size c for each network size n are 18, 27, 30, 33, 34 and 36 respectively. We adjusted values of n and c so that we can obtain the maximum total failure probability $\leq 3.8 \times 10^{-22}$ using Equation (1), when $b = c$ or all members of committee c is Byzantine. As mentioned previously, c grows sublinearly with n. For instance, as n grows from 40 to 190, c grows from 18 to 36 with the maximum failure probability less than 3.8×10^{-22}. Therefore, it still won't throttle the primary bandwidth for relatively large n.

Randomly generated transactions were used during experiments to transfer funds among different accounts. Transactions were included in a block (batch). We used different block sizes of 1MB and 2MB.

During our first evaluation all nodes had similar upload and download bandwidth of 1Gbps (128MBps). During our tests as shown in Figure 5 HotStuff performs better with small network size (40 and 70). This is due to the reason that when n is small, c ($c < n$) has to be larger to maintain a safe failure probability. Therefore, as the network size n grows larger the growth of c remains sub linear to the growth n while keeping safe failure probability. As a result, with larger n Hermes performs better and its performance is comparable to the performance of HotStuff. Whereas, in terms of latency as shown in Figure 6, Hermes outperforms HotStuff. Hermes has lower latency than HotStuff and variation in the latency of Hermes and HotStuff with different block sizes of $1MB$ and $2MB$ grow larger when $n > 70$.

Next we introduced slack primaries and then compared the performance of Hermes and HotStuff as shown in Figure

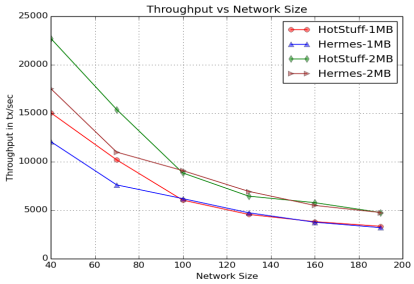


Fig. 5: Throughput with different network sizes when all nodes have higher bandwidth

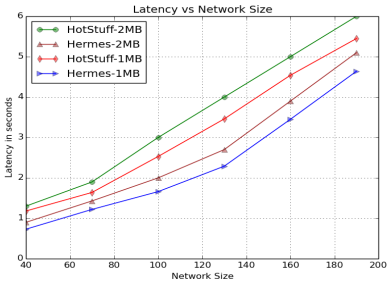


Fig. 6: Latency with different network sizes when all nodes have higher bandwidth

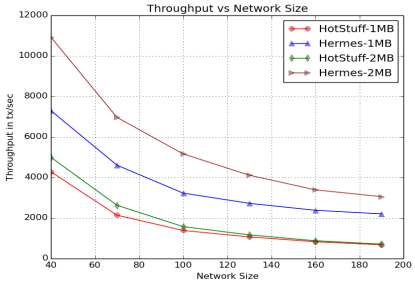


Fig. 7: Throughput when a slack primary is introduced

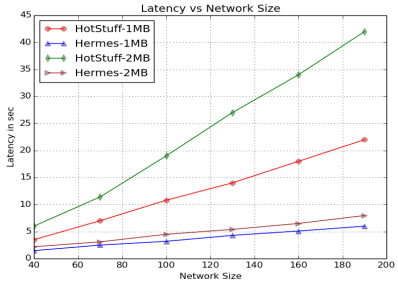


Fig. 8: Latency when a slack primary is introduced

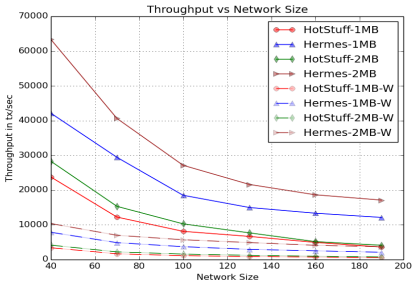


Fig. 9: Throughput with a slack primary and hash block

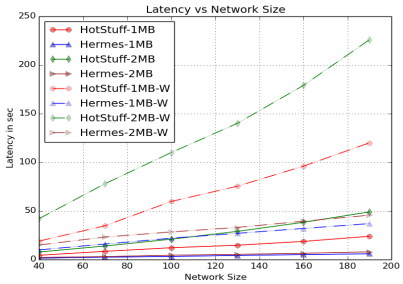


Fig. 10: Latency with a slack primary and hash block

7 and Figure 8.We configured the upload speed of slack primary to 246Mbps (roughly equivalent to 30MBps). Tests were rerun with block sizes of 1 MB and 2 MB. As shown in Figure 7, the throughput lead of Hermes grows from more than two times (when $n = 40$) to approximately three times when block size is 1MB and to four times when block size is 2MB (with $n = 190$). Similarly, Figure 8 shows that the latency of HotStuff grows very fast as the network size increases when the primary is slack. This means that clients have to wait for a longer time before they receive a response for their transaction from the network if the primary is slack.

It should be noted that if c is kept too large such that $c \approx n$, Hermes will lose its performance advantage with slack primaries. But the probability of total failure will still be close to zero.

Hash Blocks. It is also possible for a consensus protocol to use transaction hashes within a block instead of transactions [25]. This technique improves the performance as the hash size is often much smaller than the size of a transaction. Similar to pipelining optimization, this solution can also be adopted by BFT consensus protocols. Therefore, we also carried out experiments with blocks containing transaction hashes for Hermes as well as HotStuff with slack primary to observe the performance gain and limitations of hash blocks. It should be noted that the performance of this solution depends on the client as well as the network behavior. For example, if due to asynchrony in the network or maliciousness of a client, nodes do not receive transactions whose hashes are included in the block, then each node has to request it from the primary. Since it is guaranteed that an honest primary will provide missing transactions and this keeps message complexity and latency low. Figures 9 and 10 show the best-case performance with respect to the throughput and latency (shown in solid lines) and the worst-case performance of HotStuff and Hermes (shown in dotted lines) when the primary is slack. During the best case, a primary proposes hash blocks and all nodes have already received their respective transactions. Whereas during the worst case, nodes have not received any transaction and have to request transactions from the primary. To maintain the block size limit and avoid the network being overwhelmed, the primary sends transactions to each node in 1MB or $2MB$ chunks (depending on block size) until all requested transactions are sent to the requesting node (Hermes relays through the impetus committee). For example, when the block size is $1MB$ (each 1MB block contains transactions hashes instead of transactions), a node

requests transactions from the primary whose accumulated size is 3.5MB, then the primary sends transactions in three $1MB$ chunk (block-like data structure) and one 0.5 MB chunk.

Figures 9 and 10 show the best-case (presented by the solid line in different colors) and the worst-case (presented by the dotted line of the same color as the best case) throughput and latency values achieved by HotStuff and Hermes with block sizes of 1MB and 2MB under different network sizes (with the slack primary). For example, the throughput of Hermes with 1MB block size in the case of hash blocks will oscillate between solid blue curve (Hermes-1MB) and dotted blue curve (the W in "Hermes-1MB-W" stands for the worst case) in Figure 9. Similarly, the latency of Hermes-1MB in Figure 10 will oscillate between solid and dotted blue curves. Therefore, if we have honest clients and a stable network, using hash blocks will significantly improve performance. Whereas, in an unstable network or in the presence of malicious clients, protocol performance is adversely affected (especially the latency). Still, Hermes outperforms HotStuff in the cases when a slack primary is introduced (with or without hash blocks).

8 Related Work

The most relevant work to our protocol is Proteus [13] in which the primary proposes a block to a subset of nodes called the root committee. But after confirmation from the root committee the primary proposes the block to all nodes, which we have avoided in Hermes. Proteus has only focused on improving the message complexity. Moreover, the probability of view change in Proteus is higher (0.5) compared to Hermes (0.3) as the primary is chosen from the impetus committee. Proteus is only R-Safe whereas Hermes is R-safe and highly likely S-safe. To be S-safe Hermes has an additional recovery phase during its view change subprotocol. Proteus is a responsive protocol, whereas Hermes is efficiently optimistic responsive.

Random selection of subset of nodes in a BFT committee has also been used in Algorand [14], Consensus Through Herding [38], [15] and [16]. Algorand is a highly scalable BFT based protocol, but can tolerate 20 percent of Byzantine nodes in the network while providing probabilistic safety (depending on the size of committee and number of Byzantine nodes in the network). Consensus Through Herding [38] achieve consensus in a poly-logarithmic number of rounds. In [15], [16] protocols achieve Byzantine Agreement (BA) with sublinear round complexity under a corrupt majority. All of these protocols operate in a synchronous environment under an adaptive adversarial model. On the contrary, Hermes can operate in asynchronous environments and does not use the adaptive adversarial model.

HotStuff [17] is another BFT-based protocol linear message complexity $O(n)$. Hot-stuff has abolished the quadratic view change with the cost of an additional round. In HotStuff there can be multiple forks before the block gets committed. When a block gets committed, the blocks in its competitive forks will be revoked. This effectively causes wasted resources (bandwidth, processing power, etc.) that were consumed in proposing the revoked blocks. As the primary is changed with each block proposal, at most there can be f failed epochs (where no blocks are generated) out of n epochs. The rotating primary mechanism of HotStuff cannot resolve the problem of slack primary. Because whenever a slack primary is selected, there will be a sudden degradation in performance. Hence, protocol performance will be inconsistent. Libra-BFT is [37] based on HotStuff, but its synchronization module that is used to recover missing blocks has $O(n^2)$ message complexity.

There has been another line of work presented in PRIME [39], Aardvark [40] and RBFT [41]. In these papers the authors have tried to address the performance attack (delaying proposal) by a Byzantine primary by monitoring primary performance. When expectations fail, a primary is replaced by the network. But in their model, they have not considered the slack but honest nodes. That means slack primaries will also be treated as Byzantine primaries and will be replaced. Moreover, even a prompt primary might get replaced due to a network glitch. This will result in a higher number of expensive but unnecessary view changes. Moreover, if Proof-of-Stake (PoS) is used over BFT, then an honest but slack primary will seldom get a chance to propose a block, hence will not be able to increase its stake regularly, causing centralization of stake to prompt nodes. Furthermore, unlike Hermes, these solutions do not address a combination of problems which include message complexity, latency, and the slack primary. Hermes does not consider addressing the Byzantine primary performance attack. But performance monitoring module from [39] or [40] can be added to Hermes to address performance attacks.

9 Conclusion

In this paper, we proposed a highly efficient BFT-based consensus protocol for blockchains that addresses the problem of slack primaries. Furthermore, this protocol has lower latency and avoids all-to-all broadcast. To achieve this, the protocol relaxes strong safety conditions for equivocation faults. But, we also show that breaking strong safety in Hermes is expensive for a Byzantine primary. Breaking strong safety will result in blacklisting the nodes involved and their stakes get slashed. On the other hand, Breaking strong safety will only cause additional processing delay and will not result in a double-spending attack. Therefore, a Byzantine primary has no incentive to perform such an attack. As a result, the safety of Hermes is comparable to the safety of general BFT-based protocols.

Acknowledgment

This work was supported by Natural Sciences and Engineering Research Council of Canada (NSERC) CREATE 528125-2019, Joint NSERC Engage and Mitacs Grant 538022-19, NSERC Discovery Grant RGPIN-2016-05310 and the National Science Foundation grant: NSF CNS-2131538.

References

[1] Y. Kwon, J. Liu, M. Kim, D. Song, and Y. Kim, "Impossibility of full decentralization in permissionless blockchains," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, ser. AFT '19. New York, NY, USA: ACM, 2019, pp. 110–123.

[2] F. C. Gärtner, "Byzantine failures and security: Arbitrary is not (always) random," in *INFORMATIK 2003 - Mit Sicherheit Informatik, Schwerpunkt "Sicherheit - Schutz und Zuverlässigkeit", 29. September - 2. Oktober 2003 in Frankfurt am Main*, 2003, pp. 127–138.

[3] M. M. Jalalzai, G. Richard, and C. Busch, "An experimental evaluation of bft protocols for blockchains," in *Blockchain – ICBC 2019*, J. Joshi, S. Nepal, Q. Zhang, and L.-J. Zhang, Eds. Cham: Springer International Publishing, 2019, pp. 34–48.

[4] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, pp. 374–382, Apr. 1985.

[5] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, ser. OSDI '99. Berkeley, CA, USA: USENIX Association, 1999, pp. 173–186.

[6] M. Vukolic, "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication," in *Open Problems in Network Security - IFIP WG 11.4 International Workshop, iNetSec 2015, Zurich, Switzerland, October 29, 2015, Revised Selected Papers*, 2015, pp. 112–125.

[7] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 17–30.

[8] M. M. Jalalzai and C. Busch, "Window based BFT blockchain consensus," in *iThings, IEEE GreenCom, IEEE (CPSCom) and IEEE SSmartData 2018*, July 2018, pp. 971–979.

[9] J. Liu, W. Li, G. O. Karame, and N. Asokan, "Scalable Byzantine consensus via hardware-assisted secret sharing," *CoRR*, vol. abs/1612.04997, 2016. [Online]. Available: http://arxiv.org/abs/1612.04997

[10] R. Guerraoui, N. Knežević, V. Quéma, and M. Vukolić, "The next 700 bft protocols," in *Proceedings of the 5th European Conference on Computer Systems*, ser. EuroSys '10. New York, NY, USA: ACM, 2010, pp. 363–376.

[11] G. Golan Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. Reiter, D. Seredinschi, O. Tamir, and A. Tomescu, "Sbft: A scalable and decentralized trust infrastructure," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019, pp. 568–580.

[12] I. Eyal, A. E. Gencer, E. G. Sirer, and R. V. Renesse, "Bitcoin-ng: A scalable blockchain protocol," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. Santa Clara, CA: USENIX Association, 2016, pp. 45–59.

[13] M. M. Jalalzai, C. Busch, and G. G. Richard, "Proteus: A scalable BFT consensus protocol for blockchains," in *2019 IEEE International Conference on Blockchain (Blockchain 2019)*, Atlanta, USA, Jul. 2019.

[14] 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*, ser. SOSP '17. New York, NY, USA: ACM, 2017, pp. 51–68.

[15] T-H. Hubert Chan, Rafael Pass, and Elaine Shi, "Sublinear round Byzantine agreement under corrupt majority." [Online]. Available: http://elaineshi.com/docs/ba-cormaj.pdf

[16] I. Abraham, T.-H. H. Chan, D. Dolev, K. Nayak, R. Pass, L. Ren, and E. Shi, "Communication complexity of Byzantine agreement, revisited," 2018.

[17] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM PODC*, ser. PODC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 347–356.

[18] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, Jul. 1982.

[19] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," Jun 2016. [Online]. Available: http://atrium.lib.uoguelph.ca/xmlui/bitstream/handle/10214/9769/Buchman_Ethan_201606_MAsc.pdf

[20] H. Attiya, C. Dwork, N. Lynch, and L. Stockmeyer, "Bounds on the time to reach agreement in the presence of timing uncertainty," *J. ACM*, vol. 41, no. 1, p. 122–152, Jan. 1994.

[21] R. Pass and E. Shi, "Thunderella: Blockchains with optimistic instant confirmation," in *EUROCRYPT (2)*. Springer, 2018, pp. 3–33.

[22] A. Bessani, J. Sousa, and E. E. P. Alchieri, "State Machine Replication for the Masses with BFT-SMART," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2014, pp. 355–362.

[23] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *J. ACM*, vol. 35, no. 2, pp. 288–323, Apr. 1988. [Online]. Available: http://doi.acm.org/10.1145/42282.42283

[24] 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)*, May 2018, pp. 1545–1550.

[25] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, p. 398–461, Nov. 2002. [Online]. Available: https://doi.org/10.1145/571637.571640

[26] M. G. Luby and L. Michael, *Pseudorandomness and Cryptographic Applications*. Princeton, NJ, USA: Princeton University Press, 1994.

[27] M. Blum, "Coin flipping by telephone a protocol for solving impossible problems," *SIGACT News*, vol. 15, no. 1, p. 23–27, Jan. 1983. [Online]. Available: https://doi.org/10.1145/1008908.1008911

[28] M. Ben-Or and N. Linial, "Collective coin flipping, robust voting schemes and minima of banzhaf values," in *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, 1985, pp. 408–416.

[29] S. Popov, "On a decentralized trustless pseudo-random number generation algorithm," *Journal of Mathematical Cryptology*, vol. 11, no. 1, pp. 37–43, 2017. [Online]. Available: https://doi.org/10.1515/jmc-2016-0019

[30] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in constantinople: Practical asynchronous Byzantine agreement using cryptography," *J. Cryptol.*, vol. 18, no. 3, pp. 219–246, Jul. 2005.

[31] T. Moran, M. Naor, and G. Segev, "An optimally fair coin toss," in *Theory of Cryptography*, O. Reingold, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–18.

[32] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of BFT protocols," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 31–42. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978399

[33] T. Hanke, M. Movahedi, and D. Williams, "Dfinity technology overview series, consensus system," 2018.

[34] M. M. Jalalzai, J. Niu, C. Feng, and F. Gai, "Fast-hotstuff: A fast and resilient hotstuff protocol," 2020.

[35] V. N. Padmanabhan and J. C. Mogul, "Improving http latency," *Comput. Netw. ISDN Syst.*, vol. 28, no. 1–2, p. 25–35, Dec. 1995. [Online]. Available: https://doi.org/10.1016/0169-7552(95)00106-1

[36] N. Santos and A. Schiper, "Tuning paxos for high-throughput with batching and pipelining," in *Distributed Computing and Networking*, L. Bononi, A. K. Datta, S. Devismes, and A. Misra, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 153–167.

[37] M. Baudet, A. Ching, A. Chursin, G. Danezis, F. Garillot, Z. Li, D. Malkhi, O. Naor, D. Perelman, and A. Sonnino, "State machine replication in the libra blockchain," 2019.

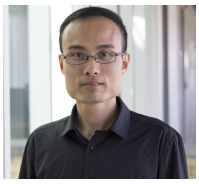
[38] T.-H. Hubert Chan, R. Pass, and E. Shi, "Consensus through herding," in *Advances in Cryptology – EUROCRYPT 2019*, Y. Ishai and V. Rijmen, Eds. Springer International Publishing, 2019, pp. 720–749.

[39] Y. Amir, B. Coan, J. Kirsch, and J. Lane, "Prime: Byzantine replication under attack," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 4, pp. 564–577, 2011.

[40] A. Clement, E. Wong, L. Alvisi, M. Dahlin, and M. Marchetti, "Making Byzantine Fault tolerant systems tolerate Byzantine Faults," in *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 153–168. [Online]. Available: http://dl.acm.org/citation.cfm?id=1558977.1558988

[41] P.-L. Aublin, S. B. Mokhtar, and V. Quéma, "Rbft: Redundant Byzantine Fault Tolerance," in *Proceedings of the 2013 IEEE 33rd International Conference on Distributed Computing Systems*, ser. ICDCS '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 297–306.

Mohammad M. Jalalzai is a postdoctoral researcher in Blockchain@UBC Cluster, under supervision of Dr. Chen Feng. He received his master's degree in computer science from Technical University of Berlin in 2010 and his PhD from Louisiana State University (LSU) in 2019. His research is mainly focused on Distributed Systems, more specifically on designing, implementing and testing secure, efficient and scalable Byzantine Fault Tolerant (BFT) consensus algorithms for blockchain networks. He is also interested in the intersection of machine learning and cyber security.



Chen Feng received the B.Eng. degree from Shanghai Jiao Tong University, China, in 2006, and the M.A.Sc. and Ph.D. degrees from The University of Toronto, Canada, in 2009 and 2014, respectively. From 2014 to 2015, he was a Postdoctoral Fellow with Boston University, USA, and EPFL, Switzerland.

He joined the School of Engineering, University of British Columbia (Okanagan Campus), Kelowna, Canada, in July 2015, where he is currently an Assistant Professor. He is a co-cluster lead of Blockchain@UBC and Principal's Research Chair in Blockchain-Empowered Digital Technology. He is interested in adapting new ideas and tools from information theory, coding theory, stochastic processes, and optimization to design better communication networks, with a particular emphasis on blockchain technology.

Costas Busch received the BSc degree in 1992 and the MSc degree in 1995 in computer science from the University of Crete, Greece. He received the PhD degree in computer science from Brown University in 2000. He is currently a Professor in the Division of Computer Science and Engineering at Louisiana State University. His research interests are in the following areas: theory of distributed computing, distributed algorithms and data structures, design and analysis of communication protocols for wireless, sensor, and optical networks, data directories for wireless sensor networks, data streaming algorithms, and algorithmic game theory. His research has been supported by the National Science Foundation.

Golden G. Richard III is a cybersecurity researcher and teacher and a Fellow of the American Academy of Forensic Sciences. He has over 40 years of practical experience in computer systems and computer security and is a devoted advocate for applied cybersecurity education. He holds a TS/SCI security clearance and supports NSA's CAE-CO internship program, teaching memory forensics, vulnerability analysis, and other topics to cleared interns. He is currently Professor of Computer Science and Engineering and Associate Director for Cybersecurity at the Center for Computation and Technology (CCT) at LSU. His primary research interests are memory forensics, digital forensics, malware analysis, reverse engineering, and operating systems. Dr. Richard earned his B.S. in Computer Science from the University of New Orleans and M.S. and Ph.D. in Computer Science from The Ohio State University. His first floppy drive cost 600 and required financing; despite that, he's still very much alive.

Jianyu Niu received the B.Eng. and the M.A.Sc. degrees from the Department of Electronics and Information, Northwestern Polytechnical University, China, in 2014 and 2017, respectively. He is currently pursuing the Ph.D. degree from the School of Engineering, The University of British Columbia, Kelowna, Canada. His research interests focus on wireless communication and blockchain systems including consensus and incentive design.