Byzantine Protocols with Asymptotically Optimal Communication Complexity

Hanzheng Lyu[1,2], Shaokang Xie[2], Jianyu Niu[2(✉)], and Chen Feng[1]

[1] The University of British Columbia (Okanagan Campus), Kelowna, BC, Canada
hzlyu@student.ubc.ca, chen.feng@ubc.ca
[2] Southern University of Science and Technology, Shenzhen, China
12011206@mail.sustech.edu.cn, niujy@sustech.edu.cn

Abstract. Byzantine agreement protocols are essential components in distributed systems and hold significant relevance for blockchain networks. However, the communication complexity of these protocols remains a major obstacle when considering their application in large-scale blockchain systems. Recently, several elegant Byzantine protocols in the synchronous authenticated setting have been proposed to enjoy expected constant round complexity or optimal good-case latency. However, their overall communication complexity is still $\Omega(n^2\ell)$ bits for an ℓ-bit message to be agreed by a set of n replicas. This quadratic communication complexity makes them unsuitable for large-scale applications. In this paper, we systematically aim to reduce the communication complexity of these protocols. In particular, we show how these protocols can be extended to have a complexity of $O(n\ell + n^2\kappa)$ bits, where κ is determined by the security parameter λ. This communication complexity is optimal when ℓ reaches $\Omega(\kappa n)$.

Keywords: Blockchain · Byzantine agreement · State machine replication · Communication complexity · Round complexity

1 Introduction

Blockchain technology has revolutionized various sectors by enabling decentralized applications, smart contracts, and cryptocurrencies. Byzantine agreement (BA) is a fundamental problem in distributed computing that plays a crucial role in achieving consensus in blockchain networks. It enables a set of n replicas to reach a consensus on a value with up to f Byzantine replicas (which may behave arbitrarily). In particular, in the synchronous authenticated setting, BA can tolerate a minority of Byzantine faults (i.e., with consensus achieved as long as $f < n/2$) [12,19]. Because of the single agreed output for each running, BA is

This work was done in part while the first author was a visiting student at Southern University of Science and Technology (SUSTech). Jianyu Niu is affiliated with the Research Institute of Trustworthy Autonomous Systems of SUSTech.

H. Duan et al. (Eds.): SecureComm 2023, LNICST 567, pp. 247–264, 2025.
https://doi.org/10.1007/978-3-031-64948-6_13

also referred to as *one-shot* consensus [26]. The efficiency of a BA protocol is usually measured by two metrics: the round complexity (i.e., the number of rounds of communication before the protocol completes), and the communication complexity (i.e., the amount of information exchanged between replicas during the protocol).

One-shot consensus is not always enough. Applications like blockchains require replicas to make agreement on an ever-growing sequence of values (rather than a single value), which is also known as Byzantine state machine replication (SMR). To build Byzantine SMR, there are two approaches: running a series of one-shot BA instances and constructing an SMR protocol directly (without using any one-shot BA as a building block) [26]. The efficiency of a Byzantine SMR protocol can be measured by the communication complexity and the latency[1]. Another useful metric is the so-called good-case latency, which applies to the case when a stable honest leader stays in charge and drives consensus on many decisions [4,27]. By contrast, BA focuses more on the worst-case "latency" (i.e., the maximum number of rounds for completion). The reason is that BA requires all the replicas to terminate after agreeing on a single value (i.e., the termination property [2]), while in SMR protocols, replicas reach consensus on an ever-growing sequence of values. The difference between BA and Byzantine SMR also slightly influences our protocol design, which will be discussed in Sect. 4.

Recently, various elegant Byzantine protocols in the synchronous authenticated setting have been proposed to enjoy expected constant round complexity or optimal good-case latency [2–4]. Specifically, Abraham *et al.* proposed a synchronous BA protocol in [2], which can achieve an expected $O(1)$ round complexity and expected $O(n^2)$ communication complexity. We refer to this protocol as Sync BA in this paper. Later, a simple and practical synchronous Byzantine SMR, called Sync HotStuff [1], was proposed by Abraham *et al.* , which can achieve consensus with a latency of 2Δ in the steady state (where Δ is a synchronous message delay upper bound). In [27], Shrestha *et al.* proved a lower bound on the latency with optimistic responsiveness for Byzantine SMR and presented a protocol called OptSync to achieve the optimal responsiveness.

1.1 Motivation and Contributions

Despite significant advances in reducing round complexity and latency in Byzantine protocols, a pressing challenge remains: the expected communication complexity of these state-of-the-art solutions is still $O(n^2\ell)$ bits for an ℓ-bit message [2–4,27]. In large-scale distributed systems such as blockchains, the message size ℓ can be significantly large because of batched transactions. Thus, this quadratic communication complexity is not only a theoretical concern-it can translate into substantial delays, increased network load, and escalated operational costs, forming a critical performance bottleneck.

The root of this high communication complexity lies in the all-to-all message broadcasting employed by these protocols. More specifically, they predominantly rely on a designated replica, referred to as the leader, to initiate new proposals.

[1] SMR protocols sometimes have no clear round boundary between outputs.

Following this initiation, replicas are required to broadcast received proposals to all other participants in the system for an equivocation check, a crucial step to ensure the safety property—that is, to guarantee that honest replicas do not commit to two conflicting signed proposals from the leader.

In this paper, we aim to extend these Byzantine protocols for long messages with optimal communication complexity. Particularly, we achieve this by a simple observation: replicas can broadcast the signed hash digest of the leader's proposals (rather than the original proposal) to detect equivocation. As a result, replicas can first agree on a unique hash digest of a leader's proposal. Due to the collision-resistance property of cryptographic hash functions, there is only one message matching with the agreed hash digest. In addition, running these protocols ensures at least one honest replica to have the proposal (since at least one honest replica has to vote for the proposal). These guarantee that honest replicas can recover the right proposals from each other. But, as pointed in [24], directly applying this observation does not work here. This is because Byzantine replicas can each ask all honest replicas for the proposals, thereby incurring a communication complexity of $\Omega(n^2\ell)$. To address this issue, we leverage a simple but efficient leader-based recovery algorithm, which only has an expected $O(n\ell)$ communication complexity (to a constant hidden factor). Moreover, we take Sync BA in [2], Sync HotStuff in [1], and OptSync in [27] as three concrete examples to showcase our systematic approach described above.

While the use of hash digests in Byzantine protocols is not new, our paper contributes a unique, systematic approach that leverages this technique to substantially and optimally reduce communication complexity in modern contexts. We also address specific challenges that emerge when using hash digests in this manner and propose novel solutions, such as our leader-based recovery algorithm, that are central to the effectiveness of our approach.

Contributions. The contributions of this paper are as follows.

- We propose a paradigm to extend the Byzantine protocols with optimal communication complexity for long messages systematically. The overall communication complexity is reduced from $\Omega(n^2\ell)$ bits to $O(n\ell + n^2\kappa)$ bits.
- We instantiate the proposed paradigm with Sync BA protocol [2] to reduce its communication complexity from $\Omega(n^2\ell)$ bits to $O(n\ell + n^2\kappa)$ bits. We also give proof of its safety, termination, and validity.
- We instantiate the proposed paradigm with Sync HotStuff protocol [1] to reduce its communication complexity from $\Omega(n^2\ell)$ bits to $O(n\ell + n^2\kappa)$ bits. We also prove its safety and liveness.
- We instantiate the proposed paradigm with OptSync protocol [27] to reduce its communication complexity from $\Omega(n^2\ell)$ bits to $O(n\ell + n^2\kappa)$ bits. We also prove its safety and liveness.

Roadmap. Section 2 provides the system model, some necessary preliminaries, and goals. Section 3 introduces the framework to generalize the extensions of protocols. Section 4, 5 and 6 present the case study of Sync BA, Sync HotStuff, and OptSync, respectively. The related work is provided in Sect. 7, and the paper is concluded in Sect. 8.

2 System Model, Preliminaries and Goals

2.1 System Model

We consider a system with a set of n mutually-distrusting replicas, say $\mathcal{P} = \{P_1, \cdots, P_n\}$. We assume $n = 2f + 1$, and up to f replicas are Byzantine. In particular, we assume that all Byzantine replicas are controlled by a single adversary $\mathcal{A}_f$. The uncontrolled replicas by $\mathcal{A}_f$ are called honest replicas, who strictly follow the protocol. We assume that every pair of honest replicas is connected with an authenticated and reliable communication link. We consider a synchronous network model, i.e., there is a known delay bound Δ such that all message transmissions between two honest replicas arrive within the bound Δ.

2.2 Preliminaries

View and Leader Election. All the protocols studied in this paper run in views, and each view has a designated replica as the leader to initiate proposals. A leader for each view is randomly elected among all replicas, which can be realized by different cryptographic means (e.g., threshold coin-tossing scheme [8]) to generate the randomness needed.

Cryptographic Components. There exist standard digital signatures and public-key infrastructure (PKI) for all replicas. A message msg signed by the replica P_r can be denoted by $\langle msg \rangle_r = (msg, \sigma_r)$, where σ_r is the signed msg by replica P_r using its private key. A message can also be sequentially signed by multiple replicas, i.e., $\langle \langle msg \rangle_i \rangle_j = \langle msg, \sigma_i \rangle_j = (msg, \sigma_i, \sigma_j)$ where σ_i is a msg signed by P_i, and σ_j is (msg, σ_i) signed by P_j. There also exists a cryptographic hash function $H(\cdot)$ that can map input of arbitrary size to an output of fixed size. The hash function is collision-resistant, which means that the probability of the adversary $\mathcal{A}_f$ producing $m \neq m'$ such that $H(m) = H(m')$ is negligible.

Certificate. The certificate of a message msg (e.g., a block containing transactions) is proof that no less than $f+1$ replicas have signed msg, which is denoted as $C(msg)$. Here, a certificate could be implemented as a threshold signature [16] or an aggregated signature [7].

Normalizing the Length of Cryptographic Components. We normalize the length of cryptographic components by following [6,24]. We use λ to denote the security parameter, $\kappa_h = \kappa_h(\lambda)$ to denote the hash size, and $\kappa_s = \kappa_s(\lambda)$ to denote the (multi-)signature size. In addition, let $\kappa = \max(\kappa_h, \kappa_s)$; we assume $\kappa = \Theta(\kappa_h) = \Theta(\kappa_s) = \Theta(\lambda)$. Therefore, in the following analysis, we use the same parameter κ to denote the hash size and signature size for convenience. We assume that $\kappa < \ell$, where ℓ denotes the size of the messages being broadcast. This assumption is grounded in the practical context of [1,2,27], where message sizes are generally observed to be significantly larger than the security parameters required.

2.3 System Goals

Byzantine agreement (BA) and Byzantine State Machine Replication (SMR) protocols should satisfy the following security properties.

- **Byzantine agreement (BA).** A Byzantine agreement protocol should satisfy three properties: 1)**Termination:** Every honest P_i eventually outputs a value; 2)**Consistency:** The outputs of all honest replicas are the same; 3)**Validity:** if every honest P_i holds the same value v, then all honest replicas commit on v
- **Byzantine state machine replication (SMR).** A Byzantine SMR protocol should satisfy two properties: 1) **Safety:** Honest replicas do not output different messages at the same log position; 2) **Liveness:** Clients' request is eventually output by all honest replicas.

Communication Complexity. The communication complexity characterizes the expected number of bits sent among the honest replicas during the protocol. For the optimal Byzantine protocols against the minority, a lower bound of the communication complexity is $\Omega(\ell n + n^2)$, where the ℓn term represents a trivial lower bound that says all honest replicas have to deliver an externally valid value of ℓ bits in length, and the n^2 terms is a reflection of the lower bound of the message complexity [11]. In this paper, we aim to achieve $O(\ell n)$ commutation complexity by extending several state-of-the-art protocols.

3 The Framework in A Nutshell

In this section, we present a unified framework to generalize the extensions for these state-of-the-art BA and Byzantine SMR protocols. The framework contains two key phases: pre-process and recovery phases. In the pre-process, replicas agree on either the same hash value or $\perp$, while in the recovery phases, if replicas output the same hash value in the previous phase, they will eventually output the same message. In particular, we take the synchronous BA protocol in [2] as an example, by which we show how to extend the existing synchronous Byzantine broadcast primitive as shown in Fig. 1.

Pre-process Phase. The leader of view r is denoted as L_r. In the pre-process phase, the leader L_r broadcasts a new proposal m_i together with an additional signed message $\langle H(m_i), r\rangle_{L_r}$, where $H(m_i)$ is the hash value of the proposal m_i. When receiving the new proposal, replicas will vote for it and broadcast the signed message. Then, replicas run the left procedures of the BA protocol, and can eventually agree on the $H(m_i)$ or $\perp$. More formally, the pre-process stage must satisfy the following properties:

- Every honest replicas P_i eventually outputs the same hash digest h_i (termination).
- The output of all honest replicas are equal, i.e., $h_i = h'$ for some h' (consistency).

1) **Elect.** All replicas participate in the threshold coin-tossing scheme from [8], and a leader L is randomly elected.
2) **Sync status.** Replicas send their highest certificate C_v together with the msg v to the leader L.
3) **Sync replicas.** The leader L chooses the highest certificate C_v, and broadcasts the associated message v.
4) **Terminate.** Honest replica terminates the above process once they recover the message.

Fig. 1. The pseudocode for the recovery phase.

- If the sender is honest then $h' = H(m_i)$ (validity).
- As least one honest replica holds the message $m_i = h_i$ (recoverability).

The termination, consistency, and validity can be guaranteed by the used BA protocol. Besides, if a block is output by an honest replica, the block has one certificate C with the majority votes. This implies, at least one honest replica has signed for it. Besides, honest replicas only sign for a proposal only if they have received it from the leader. All of these guarantee that at least one honest replica has received the proposal and signed for it, which is also called the recoverability property.

Recovery Phase. The recovery extension guarantees that if no less than one honest replicas agree on the same $hash_i$, they will eventually recover the same proposal m_i. Specifically, replicas send their committed message to the leader, and the leader is responsible for sending out messages to replicas. More formally, the recovery stage must satisfy the following properties:

- Every honest replicas P_i eventually outputs a message m, which satisfies $m = H(m_i)$ (termination).
- The output of all honest replicas are equal, i.e., $m_i = m'$ for some m' (consistency).

First, according to the collision-resistant property of the cryptographic hash function, if all honest replicas agree on the same hash digest, the recovered message must be the same, i.e., consistency. Second, once an honest replica is elected as the leader, it will gather the proposals from the honest replicas and then broadcast them. It is easy to see that all honest replicas will terminate and output the same proposals, i.e., termination.

Also note that, with respect to the complexity of the final protocol, this special case optimally represents all four possible cases where the long-message protocol, as well as the short-message protocol, are either broadcast or consensus. This is because broadcast can be achieved with one single invocation of consensus.

Let k be the current view number and replica L be the leader of the current view. While in view k, a replica P_i runs the following protocol.

- **Setup (view 0).** Every replica P_i broadcasts its $\langle h_{v_i} \rangle_i$. The $f+1$ signatures from distinct P_i for the same value v form an initial certificate for $(h_v, 0)$.

0) **Elect.** A leader L_k for the current view k is elected and referred as L.
1) **Status.** Replica P_i sends $\langle k, \mathsf{status}, k_i, C_i(h_{v_i}) \rangle_i$ to L. At the end of this round, L accepts the highest certificate and sets $\mathsf{accepted}_L = (v_L, k_L, C_L) := (v_i, k_i, C_i(h_{v_i}))$. If no replica reports a certificate, L freely choose a v_L, and sets $\mathsf{accepted}_L := (v_L, 0, \perp)$.
2) **Propose.** L broadcasts $\langle \langle k, \mathsf{propose}, h_{v_L} \rangle_L, k_L, C_L \rangle_L$ together with v_L. At the end of this round, replica P_i sets $v_{L \to i} := v_L$ if $h_{v_L} = H(v_L)$ and $k_L \geq k_i$. Otherwise (leader is faulty), it sets $v_{L \to i} := \perp$.
3) **Commit.** If $v_{L \to i} \neq \perp$, P_i forwards $\langle k, \mathsf{propose}, h_{v_L} \rangle_L$ to all other replicas and broadcasts a $\langle k, \mathsf{commit}, h_{v_L} \rangle_i$ request. At the end of this round, P_i commits if and only if it receives $f+1$ matching commit requests and does not detect leader equivocation.
4) **Notify.** If P_i has committed on v at the end of the previous round, it sends a notification $\langle \langle \mathsf{notify}, h_v \rangle_i, C_i(h_v) \rangle_i$ to every other replica. At the end of this round, if P_i receives a notify message, it accepts the committed proposal. If P_i receives multiple notify messages, it accepts an arbitrary one. Lastly, P_i increments the view counter k and enters the next view.

Fig. 2. The Sync BA protocol under standard synchrony. The changed parts are denoted as the gray.

4 Case Study of Sync BA

In this section, we first present how to extend the Sync BA protocol proposed by Abraham *et al.* [2], then prove its security properties, and finally analyze its communication complexity.

4.1 Extending Sync BA

Each replica, denoted as P_i, maintains accepted proposals in the form of $\mathsf{accepted}_i = (v_i, k_i, C_i(h_{v_i}))$, where h_{v_i} is the hash digest of accepted proposal generated at view k, and $C_i(h_{v_i})$ represents the certificate for h_{v_i} (instead of $(v_i, k_i, C_i(v_i))$ used in [2]). Initially, P_i sets $\mathsf{accepted}_i := (\perp, 0, \perp)$. The protocol operates in views, each led by a designated leader. When a leader proposes a value v in view k, the proposal has rank k.[2] Its hash digest and rank can be written into a tuple (h_v, k). We first introduce the protocol, as illustrated in Fig. 2. In this protocol, each replica broadcasts a hash digest of a specific value instead of

[2] In [1], the term iteration is used instead of view. Here, we use the term view to be consistent with the other two extensions.

the raw value. The process consists of several rounds: first, a setup phase where replicas broadcast initial certificates; followed by leader election in Round 0; status reporting from replicas to the leader in Round 1; leader-initiated proposal broadcasting in Round 2; replica commitment and validation in Round 3; and finally, notification broadcasting in Round 4. For a detailed protocol description, please refer to [2].

4.2 Safety, Termination and Validity

In this section, we prove that the above extension of Sync BA satisfies safety, termination and validity properties.

Safety. In our extension, we mainly add the block hash h_v in the propose step and replace block content v with its hash h_v in the commit and notify step, while keeping other procedures unchanged. Thus, by the safety property of Sync BA (Theorem 2 in [1]), if two honest replicas first commit on h_v and h'_v, respectively, then $h_v = h'_v$. Due to the collision-resistant feature of the cryptographic hash function, there only exists a single value v for $H(v) = h_v$, and so, all honest replicas will eventually output the same value v.

Validity. Similarly, by the validity property of Sync BA protocol (See Theorem 4 in [1]), honest replicas will first commit on h_v if 1) every honest replica starts with the same input v, and then they all have an initial certificate C certifying h_v after the setup phase; and 2) no Byzantine replica has a certificate C' certifying $h'_v \neq h_v$. Besides, by the collision-resistant feature of the cryptographic hash function, honest replicas will output the same v.

Termination. Replicas iterate the above algorithm for k views. For each view, if the leader is honest, it will drive all replicas to output the same value, and BA successfully terminates. Specifically, if honest replicas have not committed on any value, the honest leader will enable honest replicas to commit on a valid value. Otherwise, the honest leader will collect the committed value from at least one honest replica at Round 1, and then synchronize the value with honest replicas that do not have it. The probability p of randomly electing an honest replica as the leader for a view is $p = f/(2f + 1)$. Thus, the failure probability p_f that honest replicas are not elected as leaders in the first k views is $p_f = p^k$, which decreases exponentially. When n is much larger than k, the above algorithm enables all replicas to terminate in constant rounds on average.

As said previously, BA requires all honest replicas to output values for termination. Thus, the above algorithm has to run k views to guarantee that there exists at least one honest leader, who can help honest replicas to either commit on a value or synchronize the committed value. However, when constructing the above BA into Byzantine SMR protocols, one BA instance can be completed once any honest replica has the proposal and commits on it. This is because the honest replica will report the committed value to a future honest leader, who will then initiate a new BA instance immediately. This observation will be used in our extensions of Byzantine SMR protocols. In fact, in Byzantine SMR

protocols, once an honest replica has committed on a value, all replicas will have an early termination and switch to the consensus process of the next value.

4.3 Communication Complexity and Round Complexity

Communication Complexity. First, in the setup phase, a replica has to broadcast a hash digest of its input value, so the total communication complexity of all replicas is $O(\kappa n^2)$. Next, each view has multiple rounds, and we analyze the communication complexity of them one-by-one. In Round 1, replicas involve in an all-to-one broadcasting of an ℓ-bit message, the associated certificate, and signature, thus the communication complexity is $O(\ell n + \kappa n)$. In Round 2, a leader has a one-to-all broadcasting of an ℓ-bit message, the associated certificate, and signature, thus the communication complexity is $O(\ell n + \kappa n)$. In Round 3 and 4, replicas have an all-to-all broadcasting of messages with κ-bit size, thus the communication complexity is $O(\kappa n^2)$. In general, the communication complexity of each view is $O(\ell n + \kappa n^2)$. Finally, the protocol runs the setup phase once and iterates for k views, so the overall communication complexity is $O(\ell n + \kappa n^2)$, when $k \ll n$.

Round Complexity. Sync BA terminates in expected 10 rounds, which is analyzed in [2]. Our recovery phase (see Fig. 1) adds extra 3 rounds. Sync status and Sync replicas will cause one extra round, respectively. Since the Elect step can happen in parallel to the pre-process phase, if the elected leader is honest, no extra round will be caused. If the elected leader is Byzantine, 2 extra rounds are expected to be added, since we have $n = 2f+1$. Therefore, the number of rounds caused by Elect is expected to be 1, and the extended Sync BA terminates in expected 13 rounds. Thus, the extended Sync BA does not increase the round complexity of $O(1)$.

5 Case Study of Sync HotStuff

In this section, we show how to extend Sync HotStuff in [1], then prove its security properties, and finally analyze the communication complexity.

5.1 Data Structrue

In Sync HotStuff, transactions are systematically aggregated into structures known as blocks. These blocks are then organized to form a sequential chain, wherein each block occupies a distinct position, referred to as its height. Specifically, a block B_k, located at height k, is structured as $B_k := (b_k, H(B_{k-1}))$, where b_k represents the proposed value at height k, and $H(B_{k-1})$ is the hash digest of its immediate predecessor block. The chain initiates with the first block B_1, which is distinct in that it has no predecessor, and is defined as $B_1 = (b_1, \perp)$. For a block to be deemed valid, it must satisfy two essential criteria: (i) its predecessor block must either be valid or be the null value $\perp$, and (ii) the proposed value it contains must not only fulfill application-level validity requirements but also maintain consistency across its ancestral chain [1].

Let v be the current view number and replica L be the leader of the current view. While in view v, a replica P_r runs the following protocol in the steady state.

1) **Propose.** If replica P_r is the leader L, upon receiving $C_v(h_{B_{k-1}})$ and B_{k-1}, broadcast $\langle \mathsf{propose}, h_{B_k}, v, C_v(h_{B_{k-1}})\rangle_L$ and B_k [a], where B_k extends B_{k-1}.
2) **Vote.** Upon receiving both $\langle \mathsf{propose}, h_{B_k}, v, C_v(B_{k-1})\rangle_L$ and block B_k from the leader, where B_k extends B_{k-1} and $h_{B_k} = H(B_k)$, if no leader equivocation is detected, forward the $\langle \mathsf{propose}, h_{B_k}, v, C_v(h_{B_{k-1}})\rangle_L$ to all other replicas, broadcast a vote in the form of $\langle \mathsf{vote}, h_{B_k}, v\rangle_r$ (instead of $\langle \mathsf{vote}, B_k, v\rangle_r$), set $\mathsf{commit\text{-}timer}_{v,k}$ to 2Δ and start counting down.
3) **(Non-blocking) Commit.** When $\mathsf{commit\text{-}timer}_{v,k}$ reaches 0, commit B_k and all its ancestors by broadcast $\langle \mathsf{commit}, h_{B_k}, v\rangle_r$.

[a] The original is the signed message $\langle \mathsf{propose}, B_k, v, C_v(B_{k-1})\rangle_L$.

Fig. 3. The Sync HotStuff steady-state protocol under standard synchrony. The changed parts are denoted as the gray.

5.2 Extending Sync HotStuff

Sync HotStuff runs in views, and each view has a delegated replica as the leader. Sync HotStuff contains two subprotocols, the steady-state protocol (in which a leader coordinates replicas to reach the consensus of values) and the view-change protocol (by which leaders are rotated). We first introduce the steady-state protocol, as illustrated in Fig. 3. In the protocol, the leader L initiates the process by broadcasting a block proposal through a message containing the hash of the block h_{B_k}, the view number v, and a certificate related to the predecessor block B_{k-1}. Upon receiving a valid proposal and ensuring no leader equivocation, a replica P_r broadcasts a vote $\langle \mathsf{vote}, h_{B_k}, v\rangle_r$ along with the block B_k. Once a replica votes for a block proposal B_k, it activates a commit timer ($\mathsf{commit\text{-}timer}_{r,v}$). If the replica remains in view-v and does not detect leader equivocation or undergo a view change within a time frame of 2Δ (where Δ is a specific time interval), the block B_k is committed. For a detailed protocol description, please refer to [1].

Next, we introduce the view-change protocol, which is introduced to maintain safety and liveness in the face of Byzantine faults, as illustrated in Fig. 4. If a leader fails to propose a block within a specific time frame, or if it equivocates by sending conflicting messages, replicas exit the current view and a new leader is elected. When a replica exits a view, it stops voting and cancels associated timers. After a waiting period, it selects the highest certified block, informs the new leader about the chosen block, and enters the next view. The new leader collects locked blocks from replicas, then broadcasts a new-view message with the highest certified block. Replicas receiving this message forward it and vote for the block if it has an equal or higher rank than their locked block. This view-change protocol ensures the system's resilience against Byzantine behavior and

Let L and L' be the leaders of views v and $v+1$, respectively. Each replica P_r runs the following steps.

i **Blame and quit view.** If fewer than m proposals trigger P_r's votes in $(2m+4)\Delta$ time in view v, broadcast $\langle \mathsf{blame}, v \rangle_r$. Upon gathering $f+1$ $\langle \mathsf{blame}, v \rangle$ messages, broadcast them, and quit view v. If leader equivocation is detected, broadcast the two equivocating messages $\langle \mathsf{propose}, h_{B_k}, v, C_v(h_{B_{k-1}}) \rangle_L$ signed by L, and quit view.

ii **Status.** Wait for 2Δ time. Pick a highest certified block $B_{k'}$, lock on $C_{v'}(h_{B_{k'}})$, send $C_{v'}(h_{B_{k'}})$ and $B_{k'}$ (if it knows $B_{k'}$) to the new leader L', and enter view $v+1$.

iii **New-view.** The new leader L' waits for 2Δ time after entering view $v+1$ and broadcasts $\langle \mathsf{new\text{-}view}, v+1, C_{v'}(h_{B_{k'}}) \rangle_{L'}$ and $B_{k'}$, where $B_{k'}$ is a highest certified block known to L'.

iv **First vote.** Upon receiving $\langle \mathsf{new\text{-}view}, v+1, C_{v'}(h_{B_{k'}}) \rangle_{L'}$, and $B_{k'}$, if $B_{k'}$ has a rank equal to or higher than r's locked block, forward $\langle \mathsf{new\text{-}view}, v+1, C_{v'}(h_{B_{k'}}) \rangle_L$ to all other replicas and broadcast $\langle \mathsf{vote}, h_{B_{k'}}, v+1 \rangle_r$.

Fig. 4. The Sync HotStuff view-change protocol under standard synchrony. The changed parts are denoted as the gray.

maintains the consistency and progress of the protocol. Detailed information can be found in [1].

5.3 Safety and Liveness

In this section, we prove that our extended protocol atop Sync HotStuff satisfies safety and liveness properties.

Safety. In Sync HotStuff, honest replicas were proved to commit on the same block in the same height (see Theorem 3 in [1]). Compared with Sync HotStuff, the block contained in the propose and view-change messages is replaced with its hash digest. Replicas broadcast these messages for equivocation check. Therefore, in our extension, honest replicas commit on the same hash h_B in the same height. In addition, due to the collision-resistant feature of the cryptographic hash function, there do not exist two different blocks B and B' at the same height such that $H(B) = H(B') = h_B$. Therefore, honest replicas will commit on the same block at the same height.

Liveness. First, as analyzed in Lemma 4 in [1], the blame condition guarantees that a Byzantine leader cannot stall progress. Since our extension does not change the blame condition, the liveness is maintained in the presence of Byzantine leaders. Second, there will eventually exist an honest leader. An honest leader can make every other honest replica vote for m proposals in $(2m+4)\Delta$ time, since it doesn't matter whether the vote is for a block or a hash (for the voting step). In addition, an honest leader does not equivocate. So no honest replica

will blame the honest leader, and all honest replicas will keep committing new blocks.

5.4 Communication Complexity

We analyze the communication complexity of the extended Sync HotStuff. In the steady-state protocol, in Step 1, a leader broadcasts a propose message with κ-bit size and a block with ℓ-bit size, thus the communication complexity for this step is $O(\ell n + \kappa n)$. In Step 2, there exists all-to-all broadcasting of leader's propose messages and replicas' vote messages, thus the communication complexity is $O(\kappa n^2)$. In Step 3, a replica may broadcast its commit message with κ-bit size, thus the communication complexity is $O(\kappa n^2)$. Overall, the communication complexity of the steady-state protocol is $O(\ell n + \kappa n^2)$.

In the view-change protocol, Step i has all-to-all broadcasting of the blame messages, thus the communication complexity is $O(\kappa n^2)$. Step ii involves all-to-one broadcasting of block with ℓ-bit size and certificate of κ-bit size, thus the communication complexity is $O(\ell n + \kappa n)$. In Step iii, there is one-to-all broadcasting containing a block with ℓ-bit size and new-view message with κ-bit size, thus the communication complexity is $O(\ell n + \kappa n)$. In Step iv, replicas forward the received new-view messages and vote messages, thus the communication complexity is $O(\kappa n^2)$. Overall, the communication complexity of the view-change protocol is $O(\ell n + \kappa n^2)$. Summing all of these, we have that the overall communication complexity of the extended protocol is $O(\ell n + \kappa n^2)$.

6 Case Study of OptSync

In this section, we show how to extend OptSync in [27] with optimal communication complexity. We first provide some additional preliminaries to understand OptSync.

6.1 Preliminaries

Chain Certificates. Chain certificates serve the purpose of comparing different chains received by replicas, and they come in two types: responsive certificates $C_v^{3/4}(h_{B_k})$ and synchronous certificates $C_v^{1/2}(h_{B_l})$.

Ranking Chain Certificates. When comparing chain certificates, we prioritize them based on views. If the first certificate has a smaller view, it's considered lower. Within the same view, we rank certificates by the height of their responsive certificates. If certificates share the same view and have a common responsive certificate (or none at all), we then compare them based on the heights of their synchronous certificates.

Tip of a Chain Certificate. The tip of a chain certificate refers to its highest block. For a given chain certificate $\mathcal{CC} = (C_v^{3/4}(h_{B_k}), C_v^{1/2}(h_{B_l}))$, if $C_v^{1/2}(h_{B_l}) \neq \perp$, the tip is defined as $H(B_l)$; otherwise, it is defined as $H(B_k)$.

Let v be the current view number and replica L be the leader of view v. While in view v, a replica P_r runs the following protocol.

1) **Propose.** If replica P_r is the leader L, upon receiving $C_v(h_{B_{k-1}})$, broadcast $\langle \mathsf{propose}, h_{B_k}, v, C_v(h_{B_{k-1}})\rangle_L$ and B_k[a], where B_k extends B_{k-1}.
2) **Vote.** Upon receiving the first proposal $\langle \mathsf{propose}, h_{B_k}, v, C_v(h_{B_{k-1}})\rangle_L$ and B_k with a valid view v certificate for a block at height $k-1$ (not necessarily from L) where B_k extends B_{k-1}, if no leader equivocation is detected, forward the proposal $\langle \mathsf{propose}, h_{B_k}, v, C_v(h_{B_{k-1}})\rangle_L$ to all replicas, broadcast a vote in the form of $\langle \mathsf{vote}, h_{B_k}, v\rangle_r$ (instead of $\langle \mathsf{vote}, B_k, v\rangle_r$), set commit-timer$_{v,k}$ to 2Δ and start counting down.
3) **(Non-blocking) Commit rules.** Replica P_r commits block B_k using either of the following rules if r is still in view v:
   (a) *Responsive commit.* On receiving $\lfloor 3n/4 \rfloor + 1$ votes for B_k, i.e., $C_v^{3/4}(h_{B_k})$, commit B_k and all its ancestors immediately. Abort commit-timer$_{v,k}$.
   (b) *Synchronous commit.* If commit-timer$_{v,k}$ reaches 0, commit B_k and all its ancestors.
4) **(Non-blocking) Blame and quit view.**
   - *Blame if no progress.* For $p > 0$, if fewer than p proposals trigger r's votes in $(2p+4)\Delta$ time in view v, broadcast $\langle \mathsf{blame}, v\rangle_r$.
   - *Quit view on $f+1$ blame messages.* Upon gathering $f+1$ distinct $\langle \mathsf{blame}, v\rangle_r$ messages, broadcast $\langle \mathsf{quit\text{-}view}, v, \mathcal{CC}\rangle_r$ along with $f+1$ blame messages where $\mathcal{CC}$ is the highest ranked chain certificate known to r. Abort all view v timers, and quit view v.
   - *Quit view on detecting equivocation.* If leader equivocation is detected, broadcast $\langle \mathsf{quit\text{-}view}, v, \mathcal{CC}\rangle_r$ along with equivocating proposals, abort all view v timers, and quit view v.

[a] The original is the signed message $\langle \mathsf{propose}, B_k, v, C_v(hB_{k-1})\rangle_L$.

Fig. 5. The OptSync steady-state protocol. The changed parts are denoted as the gray.

6.2 Extending OptSync

Extended OptSync executes in views, and each view has a leader. Similar to Sync HotStuff, OptSync contains two subprotocols, the steady-state protocol, as illustrated in Fig. 5, and the view-change protocol, as illustrated in Fig. 6. Detailed information can be found in [27].

We start with the steady-state protocol. In this protocol, the leader in a given view proposes a block, initiating a process where replicas vote on the proposal. Replicas validate the leader's proposal, vote for the block, and share the proposal with others. Blocks are committed based on a set of rules: if enough votes are received or when a synchronous commit timer expires. The protocol also includes mechanisms for detecting leader misbehavior, leading to a view-change triggered by blame messages from honest replicas. When a view change occurs, replicas

Let L and L' be the leaders of views v and $v+1$, respectively. Each replica P_r runs the following steps.

i **Status.** Wait for 2Δ time. Until this time, if a replica receives any chain certificates, the replica updates its chain certificate $\mathcal{CC}$ to the highest possible rank. Set lock_{v+1} to be the highest ranked chain certificate at the end of the 2Δ wait. Send $\langle \mathsf{status}, \mathsf{lock}_{v+1} \rangle_r$ and $B_{k'}$(if the replica knows $B_{k'}$ and $\mathsf{lock}_{v+1} = C_{v+1}(h_{B_{k'}})$) to L'. Enter view $v+1$.

ii **New-view.** The new leader L' waits for 2Δ time after entering view $v+1$. L' broadcast $\langle \mathsf{new\text{-}view}, v+1, \mathsf{lock}' \rangle_{L'}$ and $B_{k'}$, where lock' is the highest ranked chain certificate known to L' after this wait.

iii **First vote.** Upon receiving the first $\langle \mathsf{new\text{-}view}, v+1, \mathsf{lock}' \rangle_{L'}$ and $B_{k'}$, if $\mathsf{lock}_{v+1} \leq \mathsf{lock}'$, then broadcast $\langle \mathsf{new\text{-}view}, v+1, \mathsf{lock}' \rangle_{L'}$ and $\langle \mathsf{vote}, \mathsf{tip}(\mathsf{lock}'), v+1 \rangle_r$.

Fig. 6. The Optsync view-change protocol. The changed parts are denoted as the gray.

quit the current view and broadcast a message containing the highest-ranked chain certificate they know. This approach ensures regular block proposals and enables the protocol to maintain progress and security even in the presence of faulty leaders.

The view-change protocol ensures the replacement of a potentially faulty leader to maintain the system's liveness while upholding the safety of previously committed values. After quitting view v, a replica waits for 2Δ time and then enters view $v+1$. During this transition, the replica updates its chain certificate $\mathcal{CC}$ to the highest rank and sets lock_{v+1} to $\mathcal{CC}$. It shares lock_{v+1} and the corresponding block $B_{k'}$ with the new leader L'. Upon entering view $v+1$, leader L' waits for 2Δ time to receive status messages from honest replicas. Based on these messages, L' selects the highest ranked chain certificate lock' and creates a new-view message, which, together with the associated block, is sent to all honest replicas. Replicas receiving this message and block validate the certificate's rank, forwarding the new-view message and voting for it, ensuring a smooth transition.

6.3 Safety and Liveness

We prove that our extension to OptSync retains its security property. The proofs are similar to those in Sect. 5.3.

Safety. OptSync is proved to satisfy the safety, i.e., honest replicas do not commit conflicting blocks for the same height (see Theorem 14 in [27]). Since our extended protocol only adds the hash of block h_B in the propose step and replaces block B with its hash h_B in the vote and commit step, all honest replicas will commit on the same hash h_B for the same height. Since h_B corresponds to the unique block B, honest replicas will commit on the same block B for the same height.

Liveness. In our extended protocol, liveness is ensured as honest replicas consistently vote for, commit, and lock on block hashes (h_B) instead of the entire blocks (B). This adjustment does not impact the protocol's liveness property. In each view, a leader must propose at least p blocks within $(2p + 4)\Delta$ time to trigger votes from honest replicas. If a Byzantine leader equivocates or proposes fewer than p blocks, a view-change is initiated. Upon entering a new view, all honest replicas lock onto the highest certified chain. An honest leader extends upon the tip of the highest-ranked certified chain. Honest replicas vote for the new block since the leader's provided lock is equal to or greater than their own lock. Additionally, the honest leader consistently proposes at least p blocks, preventing the formation of a blame certificate that could trigger a view-change. As a result, all honest replicas continue to commit new blocks, ensuring the protocol's liveness property.

6.4 Communication Complexity

We analyze the communication complexity of the extended OptSync protocol. In the steady-state protocol, Step 1 has an one-to-all broadcasting containing a block with ℓ-bit size and a propose message with κ-bit size, thus the communication complexity is $O(\ell n + \kappa n)$. In Step 2, there is an all-to-all broadcasting of leader's propose messages and replicas' vote messages, thus the communication complexity is $O(\kappa n^2)$. In Step 3, replicas may conduct all-to-all broadcasting containing commit messages with κ-bit size, thus the communication complexity is $O(\kappa n^2)$. In Step 4, there exists all-to-all broadcasting of blame or quit-view messages with κ-bit size, thus the communication complexity is $O(\kappa n^2)$. In general, the communication complexity of the steady-state protocol is $O(\ell n + \kappa n^2)$.

In the view-change protocol, Step i has all-to-one broadcasting containing the block with ℓ-bit size and lock with κ-bit size, thus the communication complexity is $O(\ell n + \kappa n)$. In Step ii, the leader makes one-to-all broadcasting of a block with ℓ-bit size and lock with κ-bit size, thus the communication complexity is $O(\ell n + \kappa n)$. In Step iii, replicas conduct all-to-all broadcasting containing new-view and vote messages with κ-bit size, thus the communication complexity is $O(\kappa n^2)$. In general, the communication complexity of the view-change protocol is $O(\ell n + \kappa n^2)$. Summing all of these, we have that the overall communication complexity of the extended protocol is $O(\ell n + \kappa n^2)$.

7 Related Work

This work is closely related to the Byzantine agreement (BA) and Byzantine State Machine Replica (SMR) problems in the synchronous authenticated setting.

Byzantine Agreement. Byzantine Agreement was first introduced by Lamport *et al.* in [19,25]. Later, many studies have been conducted to reduce the communication complexity of Byzantine agreement. Dolev *et al.* in [10,12] presented a protocol with $O(\kappa n^3)$ communication complexity. Dolev and Reischuk [11]

proved that any deterministic protocol must incur $\Omega(n^2)$ communication complexity. Berman *et al.* [5] further reduced the communication complexity to $O(n^2)$. In [13], Fitzi *et al.* gave a lower bound on the communication complexity of $O(\ell n)$, where ℓ is the message length holds for BA extension. Hirt *et al.* in [18] presented an extension protocol for $t < n$ with optimal communication but non-optimal round complexity of $O(n^2)$. Ganesh *et al.* in [14,15] proposed some protocols, which are better in both round and communication complexity compared with protocols in [13,18,20]. Nayak *et al.* in [24] studied extension protocols of BA which achieve the best-possible communication complexity of $\Theta(n\ell)$. However, the protocols need to invoke BA oracle twice, which increases round complexity. Momose *et al.* [22,23] designed two protocols, which have a quadratic communication complexity with different trade-offs in resilience and assumptions. Abraham *et al.* proposed a synchronous BA protocol in [2], which can achieve an expected $O(1)$ round complexity and expected $O(n^2)$ communication complexity.

Byzantine SMR. In the past few years, several simple and elegant Byzantine SMR protocols are proposed. Hanke *et al.* in [17] proposed a Byzantine SMR protocol, called Dfinity protocol, which has an expected latency of 9Δ and quadratic communication complexity. Subsequently, Lyu *et al.* [21] managed to further reduce the latency by 10.7% under withholding attack. Chan *et al.* in [9] proposed PiLi, a synchronous Byzantine SMR that can achieve latency of 65Δ and quadratic communication complexity. Abraham *et al.* [1] presented Sync HotStuff, which can achieve near-optimal latency with $2\Delta + O(\delta)$ and communication complexity of $O(\ell n^2)$. In [27], Shrestha *et al.* proved a lower bound on the latency with optimistic responsiveness for Byzantine SMR and presented a protocol called OptSync to achieve the optimal responsiveness.

Summary. Although existing protocols made massive progress in optimizing communication complexity, they still suffer from high communication overhead, increased round complexity [24], or rely on stringent assumptions [22,23]. In contrast, our work reduces communication complexity to $O(n\ell + n^2\kappa)$ without compromising resilience or adopting restrictive assumptions.

8 Conclusion and Future Work

In this paper, we propose a paradigm to reduce the communication complexity of Byzantine protocols systematically. By instantiating this paradigm with three state-of-the-art Byzantine protocols, namely Sync BA, Sync HotStuff, and OptSync, we have demonstrated their potential for achieving significantly improved communication complexity. In particular, we have shown that these protocols can be extended to have a communication complexity of $O(n\ell+n^2\kappa)$ bits, where κ is determined by the security parameter λ. This communication complexity is optimal when ℓ reaches $\Omega(\kappa n)$, which makes our approach highly practical for real-world deployments. Our results provide a significant step forward in enhancing the scalability and performance of blockchain networks, facilitating their widespread adoption in various applications.

While our theoretical analysis suggests promising improvements in communication complexity, there are important avenues for future research. At this stage, our work is largely theoretical, and empirical validation under real-world conditions is essential. Additionally, we assume that $\kappa < \ell$, which is typical in practical scenarios, but may not always hold. Exploring the implications, i.e., $\kappa \geq \ell$, represents a key direction for future research.

Acknowledgement. We would like to thank the anonymous reviewers for their helpful comments. Jianyu Niu was partially supported by the Shenzhen Science and Technology Program under Grants RCBS20221008093248075 and JSGG20220831095603007. Chen Feng was partially supported by the NSERC Discovery Grant RGPIN-2016-05310 and CREATE grant 528125-2019.

References

1. Abraham, I., Malkhi, D., Nayak, K., Ren, L., Yin, M.: Sync HotStuff: simple and practical synchronous state machine replication. In: 2020 IEEE Symposium on Security and Privacy (SP), pp. 106–118 (2020)
2. Abraham, I., Devadas, S., Dolev, D., Nayak, K., Ren, L.: Synchronous byzantine agreement with expected $O(1)$ rounds, expected $O(n^2)$ communication, and optimal resilience. In: Goldberg, I., Moore, T. (eds.) FC 2019. LNCS, vol. 11598, pp. 320–334. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32101-7_20
3. Abraham, I., Nayak, K., Ren, L., Xiang, Z.: Byzantine agreement, broadcast and state machine replication with optimal good-case latency. arXiv e-prints (2020)
4. Abraham, I., Nayak, K., Ren, L., Xiang, Z.: Good-case latency of byzantine broadcast: a complete categorization. In: Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing, PODC 2021, pp. 331–341. New York, NY, USA (2021)
5. Berman, P., Garay, J.A., Perry, K.J.: Bit optimal distributed consensus. In: Baeza-Yates, R., Manber, U. (eds.) Computer Science, pp. 313–321. Springer, Boston, MA (1992). https://doi.org/10.1007/978-1-4615-3422-8_27
6. Bhat, A., Shrestha, N., Kate, A., Nayak, K.: RandPiper – reconfiguration-friendly random beacons with quadratic communication. Cryptology ePrint Archive, Report 2020/1590 (2020)
7. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 416–432. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-39200-9_26
8. Cachin, C., Kursawe, K., Shoup, V.: Random oracles in constantinople: practical asynchronous byzantine agreement using cryptography. J. Cryptol. **18**(3), 219–246 (2005)
9. Chan, T.H.H., Pass, R., Shi, E.: Pili : a simple , fast , and robust family of blockchain protocols (2019)
10. Dolev, D., Fischer, M.J., Fowler, R., Lynch, N.A., Strong, H.R.: An efficient algorithm for Byzantine agreement without authentication. Inf. Control **52**(3), 257–274 (1982)
11. Dolev, D., Reischuk, R.: Bounds on information exchange for Byzantine agreement. J. ACM (JACM) **32**(1), 191–204 (1985)

12. Dolev, D., Strong, H.R.: Authenticated algorithms for Byzantine agreement. SIAM J. Comput. **12**(4), 656–666 (1983)
13. Fitzi, M., Hirt, M.: Optimally efficient multi-valued Byzantine agreement. In: Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing, pp. 163–168 (2006)
14. Ganesh, C., Patra, A.: Broadcast extensions with optimal communication and round complexity. In: Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, pp. 371–380 (2016)
15. Ganesh, C., Patra, A.: Optimal extension protocols for Byzantine broadcast and agreement. Distrib. Comput. **34**(1), 59–77 (2021)
16. Gennaro, R., Goldfeder, S.: Fast multiparty threshold ECDSA with fast trustless setup. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 1179–1194. ACM (2018)
17. Hanke, T., Movahedi, M., Williams, D.: Dfinity technology overview series, consensus system. arXiv preprint arXiv:1805.04548 (2018)
18. Hirt, M., Raykov, P.: Multi-valued Byzantine broadcast: The $t < n$ case. In: Sarkar, P., Iwata, T. (eds.) Advances in Cryptology – ASIACRYPT 2014, ASIACRYPT 2014, LNCS, vol. 8874. Springer, Berlin (2014). https://doi.org/10.1007/978-3-662-45608-8_24
19. Lamport, L., Shostak, R.E., Pease, M.C.: The Byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)
20. Liang, G., Vaidya, N.: Error-free multi-valued consensus with Byzantine failures. In: Proceedings of the 30th annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, pp. 11–20 (2011)
21. Lyu, H., Niu, J., Gai, F., Feng, C.: Publish or perish: defending withholding attack in Dfinity consensus. In: 2021 17th International Conference on Mobility, Sensing and Networking (MSN), pp. 404–411. IEEE (2021)
22. Momose, A., Ren, L.: Optimal communication complexity of authenticated Byzantine agreement. arXiv preprint arXiv:2007.13175 (2020)
23. Momose, A., Ren, L.: Optimal communication complexity of Byzantine agreement, revisited. IACR Cryptol. ePrint Arch. **2020**, 1569 (2020)
24. Nayak, K., Ren, L., Shi, E., Vaidya, N.H., Xiang, Z.: Improved extension protocols for Byzantine broadcast and agreement. In: 34th International Symposium on Distributed Computing, DISC 2020, 12–16 October 2020, Virtual Conference. LIPIcs, vol. 179, pp. 28:1–28:17 (2020)
25. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2), 228–234 (1980)
26. Shi, E.: Streamlined blockchains: a simple and elegant approach (A tutorial and survey). In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11921, pp. 3–17. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34578-5_1
27. Shrestha, N., Abraham, I., Ren, L., Nayak, K.: On the optimality of optimistic responsiveness. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pp. 839–857 (2020)