Artifact Evaluated NDSS
Available
Functional
Reproduced

Cross-Consensus Reliable Broadcast and its Applications

Yue Huang*, Xin Wang*§, Haibin Zhang†, Sisi Duan*‡§ ✉
*Tsinghua University
†Yangtze Delta Region Institute of Tsinghua University, Zhejiang
‡Zhongguancun Laboratory, Shandong Institute of Blockchains
§State Key Laboratory of Cryptography and Digital Economy Security
y-huang22@mails.tsinghua.edu.cn, wangxin87@tsinghua.edu.cn, bchainzhang@aliyun.com, duansisi@tsinghua.edu.cn
✉ Corresponding author

***Abstract*—Conventional Byzantine fault-tolerant protocols focus on the workflow within a group of nodes. In recent years, many applications of consensus involve communication across groups. Examples include communication between infrastructures running replicated state machine, sharding-based protocols, and cross-chain bridges. Unfortunately, little effort has been made to model the properties for communication across groups.**

In this work, we propose a new primitive called cross-consensus reliable broadcast (XRBC). The XRBC primitive models the security properties of communication between two groups, where at least one group executes a consensus protocol. We provide three constructions of XRBC under varying assumptions and present three applications for our XRBC protocols: a cross-shard coordination protocol via a case study of Reticulum (NDSS 2024), a protocol for cross-shard transactions via a case study of Chainspace (NDSS 2018), and a solution for cross-chain bridge. Our evaluation results show that our protocols are highly efficient and benefit different applications. For example, in our case study on Reticulum, our approach achieves 61.16% lower latency than the vanilla approach.

I. Introduction

Byzantine fault-tolerant (BFT) protocols that tolerate arbitrary failures is a fundamental building block in distributed computing and cryptography [1], enabling many applications such as blockchains and multi-party computation. Conventional BFT protocols only consider the communication among nodes in the *same* group. Little effort has been made to formalize the security properties of communication *across* groups, especially when some group has *built-in consensus* (the group executes a consensus protocol). In this work, we call the communication between different groups where at least one group has built-in consensus the *cross-consensus communication* problem. Cross-consensus communication is useful in a myriad of applications. In fact, many works informally study the problem as part of their solutions. Below, we summarize some representative scenarios.

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.230207
www.ndss-symposium.org

Communication between RSM. A lot of modern infrastructures use replicated state machine (RSM) to achieve high reliability of the services. Crash fault-tolerant consensus protocols (CFT) or Byzantine fault-tolerant protocols (BFT) are often used to build RSM. Examples include cloud coordination services [2], [3] and database systems [4], [5]. A line of work studies the communication between RSM in the client-server model [6], [7], where a request from the client may need multiple infrastructures to interact and each infrastructure runs a different consensus protocol.

Sharding-based blockchain/BFT. Sharding refers to an approach that divides the nodes in a system into multiple shards, each consisting of a (small) subset of nodes. Client requests (also called transactions) are processed by different shards concurrently so the system enjoys higher throughput and better scalability compared to non-sharding approaches. A typical workflow is that each shard runs a dedicated consensus protocol (e.g., PBFT [8]) for *intra-shard* transactions (transactions that only need to be processed by one shard). For *cross-shard* transactions that need to be processed by more than one shard, multiple shards need to communicate with each other to agree on the order of the transactions [9], [10], [11], [12]. Most sharding-based protocols in the Byzantine failure model assume that each shard is *correct*, i.e., in a partially synchronous setting, a correct shard does not have more than one-third faulty nodes. Recently, some sharding-based approaches also propose to handle *faulty* shards, where some shards may have more than one-third faulty nodes [13], [14]. To deal with the faulty shard, the approaches usually employ another shard (which is expected to be correct) to monitor the progress. We call such a protocol *cross-shard coordination* protocol. Communication between different shards can be frequent for coordination purposes.

Cross-chain bridges. The interoperability of blockchains is another example of cross-consensus communication [15]. Indeed, each blockchain has its own consensus protocol. When some transactions need to be processed by more than one chain (e.g., asset transfer from one chain to another), a dedicated protocol is needed to ensure the *atomicity* of the transaction such that either all chains commit the transaction or none of them commits the transaction. Multiple approaches can be used to instantiate cross-chain bridges. One type that involves cross-consensus communication is the *relay chain*, where a dedicated blockchain system is used to coordinate cross-chain transactions [16], [17], [18].

A. The Research Problem

While many previous works have cross-consensus communication as part of the solutions, the security properties of entire systems are often studied without treating cross-consensus communication as a building block. Take sharding-based BFT as an example; most approaches simply specify that "shard A sends some message v to another shard B [13]." A trivial instantiation is that all nodes in A send v to all nodes in B. To lower the communication, some work mentions that we can rely on the *leader* (i.e., a designated node) of A to send the message. Unfortunately, the correctness of such an approach is not that clear. First, without a clear formalization, we do not know what is correct. Second, one may argue that correctness means "*all correct nodes in* B *receive* v." However, if the leader in A is faulty, we cannot easily ensure that all correct nodes in B receive v.

Another notable example is that correct nodes in one group do not necessarily send matching messages to another group. For example, in Reticulum [19], nodes in one shard send their "votes" to another shard, and the votes are not necessarily matching. To ensure that the correct nodes in the other shard receive matching votes, expensive broadcast primitives are used, which might be an overkill.

Therefore, a long-overlooked research problem is:

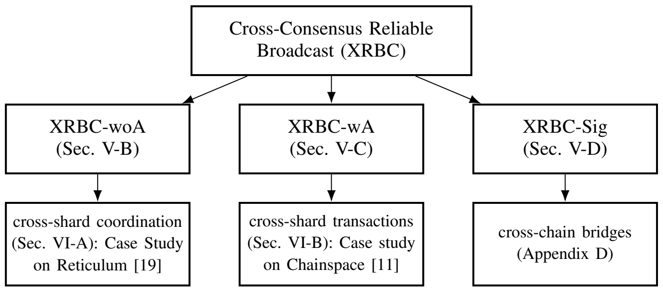
Can we formalize cross-consensus communication and build efficient constructions to improve upper-layer applications?

B. Our Contributions

A formal treatment of cross-consensus reliable broadcast (XRBC). We formalize the notion of *cross-consensus reliable broadcast*, abbreviated as XRBC (Sec. V-A). The idea is to model the communication between two groups of nodes: group A with m nodes and group B with n nodes. Among the two groups, one or both groups have built-in consensus, and nodes execute an atomic broadcast (ABC) [20] protocol[1]. In particular, nodes in A send cross-consensus messages to B. There are two cases: each cross-consensus message is a message delivered in ABC; the message is an intermediate message of ABC, e.g., a vote. In the former case, correct nodes in A always send consistent messages. In the later case, correct nodes do not always send consistent messages. XRBC guarantees that correct nodes in B always deliver the cross-consensus messages according to the same order.

Our XRBC primitive thus can be viewed as a reliable broadcast [21], [22], [23] across two groups, and we consider it a first-class primitive useful for multiple applications. Our constructions resemble the design of reliable broadcast protocols. However, the definitions are fundamentally different.

[1]ABC is *the* model for blockchains and also RSM, where all correct nodes agree on the sequence of delivered messages. It is only syntactically different from BFT-RSM (BFT for short) or CFT-RSM (CFT for short). Briefly speaking, ABC does not consider the execution of transactions and focuses on the consensus regarding the order of messages (or blocks in the context of blockchains).



Cross-Consensus Reliable Broadcast (XRBC)
XRBC-woA (Sec. V-B)
XRBC-wA (Sec. V-C)
XRBC-Sig (Sec. V-D)
cross-shard coordination (Sec. VI-A): Case Study on Reticulum [19]
cross-shard transactions (Sec. VI-B): Case study on Chainspace [11]
cross-chain bridges (Appendix D)

Fig. 1: Overview of the paper.

As a warm-up primitive, we also formalize *group-to-group parallel reliable broadcast* (Group-PRBC), i.e., parallel reliable broadcast instances between two groups (Sec. IV). We also provide a construction in the asynchronous network model (there does not exist an upper bound on the message processing and transmission delay) that largely resembles existing asynchronous consensus mechanisms [21], [22], [24], [25], [20]. We do not claim too much novelty in Group-PRBC and mainly present it as a baseline for the ease of understanding.

Practical instantiations of XRBC under different assumptions. We provide three practical instantiations of XRBC, as summarized below and also in Table I. All the protocols are studied in the asynchronous network model. We present three protocols, XRBC-woA, XRBC-wA, and XRBC-Sig, each under different assumptions. As we show later, we have identified interesting applications for each protocol, due to their differences in the underlying assumptions.

Applications of XRBC protocols. We show that our XRBC protocols can either be directly used or extended to benefit different applications, as summarized in Fig. 1.

- **Cross-shard coordination.** We show that the model of our XRBC-woA protocol can be used for cross-shard coordination protocols. We present a case study about Reticulum [19], a protocol in the synchronous setting (the message processing and transmission delay between any two correct nodes is bounded by a known Δ). We show that by formalizing the cross-consensus coordination protocol, one can simplify the proof of correctness. We then present a synchronous variant of XRBC-woA that lowers the communication complexity of Reticulum (see Table II).
- **Handling cross-shard transactions.** We show that we can use XRBC-wA or XRBC-Sig to simplify sharding-based protocols that handle cross-shard transactions. We present a case study about Chainspace [11] and show that our formalization makes it easier to justify the correctness.
- **Cross-chain bridge.** We use our XRBC-Sig protocol to build a middleware for the cross-chain bridge.

Implementation and evaluation. We implement and evaluate our protocols on Amazon EC2 using up to 182 instances. Our evaluation results show that all of our XRBC protocols

protocol	assumption	messages	communication	time
Group-PRBC (Baseline) (Sec. IV)	trusted PKI	$O(mn + n^3)$	$O(nmL + \kappa nm + \kappa n^2 + mn^2 \log n)$	$O(1)$
	none	$O(mn + n^3)$	$O(mnL + \kappa n^2 m + \kappa n^3)$	$O(1)$
XRBC-woA (Sec. V-B)	ABC in A; hash	$O(mn + n^2)$	$O(mnL + \kappa n^2)$	$O(1)$
XRBC-wA (Sec. V-C)	ABC in A and B; hash	$O(mn + \mathcal{M}_{\mathsf{ABC}})$	$O(mnL + \mathcal{C}^{\kappa}_{\mathsf{ABC}})$	$O(\mathcal{T}_{\mathsf{ABC}})$
XRBC-Sig (Sec. V-D)	proof of delivery in A; hash; ABC in A and B; trusted PKI	$O(n + \mathcal{M}_{\mathsf{ABC}})$	$O(nL + \kappa n + \mathcal{C}^{\kappa}_{\mathsf{ABC}})$	$O(\mathcal{T}_{\mathsf{ABC}})$

TABLE I: Comparison of the protocols presented in this work. m is the size of group A and n is the size of group B. L is the length of the cross-group/consensus messages. κ is the length of the security parameter (e.g., length of a digital signature or a hash). $O(\mathcal{T}_{\mathsf{ABC}})$ and $O(\mathcal{M}_{\mathsf{ABC}})$ are the time complexity and message complexity of ABC in B, respectively. $O(\mathcal{C}^{\kappa}_{\mathsf{ABC}})$ is the communication complexity of ABC for κ-bit inputs.

achieve low latency and decent throughput, making them suitable for all the abovementioned applications. For instance, in our case study on cross-shard coordination for Reticulum, XRBC achieves 61.16% lower latency than the approach by Reticulum.

II. Related work

Cross-consensus protocol. It was first mentioned in Aegean [6] that two RSMs need to communicate in some applications where some RSMs may need to send a specific request to another RSM. A recent work Picsou [7] formalizes the *cross-cluster consistent broadcast* (C3B) notion that models the communication between two groups of nodes and both groups have built-in RSMs. As Picsou is conceptually the closest one to our work, we summarize the main differences. First, the context is different. Namely, Picsou assumes both groups run RSM (i.e., ABC in our context). Such an assumption is the same as that in XRBC-Sig. However, our XRBC-woA and XRBC-wA only assume one of the groups has a built-in ABC. Second, the security definitions are fundamentally different. C3B focuses on the liveness property, i.e., one RSM eventually receives the message from another RSM. In contrast, we take a more systematic approach and consider the "safety" of the messages. Our notion can thus benefit more applications. Finally, the constructions are different. Picsou focuses on achieving low latency in the optimal case. In the optimistic case, one node in A communicates with one node in B. Meanwhile, PKI is assumed. In contrast, XRBC focuses on balancing communication and time complexity under different cryptographic assumptions. TrustBoost [26] provides a primitive called *cross-consensus communication (CCC)*. CCC also models the communication across groups, and TrustBoost uses CCC to build a combined ledger of multiple blockchains. CCC shares some similarities with our idea. The construction is similar to the trivial instantiation of XRBC that we present in Sec. V-A.

Reliable broadcast, Byzantine broadcast, and Byzantine agreement. Reliable broadcast is a fundamental primitive in distributed computing [21], [22]. Many efforts have been made to optimize the communication complexity [27], [23], [28]. It has already become a crucial building block for many applications [29], [30], [31], [25].

Byzantine broadcast (BB) is a *synchronous* variant of reliable broadcast. The security properties are very close to those for reliable broadcast. A synchronous BB can be achieved under $f < n$ [32]. One may relax the bound to $f < n/2$ or $f < n/3$, and the optimization of the communication complexity can be of independent interest [33]. Another synchronous notion is Byzantine agreement (BA, i.e., the consensus variant of Byzantine agreement where each node proposes a value and all nodes decide the same value). For a group of n nodes, the lower bound of synchronous BA is $f < n/2$ [34]. In this work, we consider synchronous BB and Byzantine agreement in our case study on the cross-shard coordination protocol.

Our XRBC primitive extends reliable broadcast from communication among the same group of nodes to communication across groups. Additionally, although our protocols presented in this work are asynchronous, they can be easily extended to synchronous ones. We show one example in our case study.

Cross-chain bridge. Many cross-chain solutions involve cross-consensus communication. One example is the relay chain, where a dedicated blockchain system is used to coordinate cross-chain transactions [16], [17], [18]. As the cross-chain transactions are *coordinated* by a blockchain that is already safe (i.e., no double spending) and live (i.e., transactions will eventually be processed), relay chains can handle a large volume of cross-chain transactions to ensure the atomicity of cross-chain transactions. Industrial examples include Cosmos [17], Polkadot [18], and CCIP by Chainlink [35]. Another example is the *sidechain* solutions for cross-chain transactions. A sidechain is a parallel chain to a blockchain (called mainchain), often used to improve the performance of the system [36]. While the mainchain does not need to be aware of the existence of the sidechain, a sidechain can be used for cross-chain transactions between two chains, using the so-called *two-way peg* solution [37], [38]. The technical report of R3 corda [39] mentions that two blockchains can be used as sidechains for each other to build a two-way peg. In both relay chain and sidechain-based solutions, communication between groups of nodes is involved. In this work, we present a solution that uses our XRBC protocol to build a lightweight cross-chain bridge. Our solution can be viewed as a two-way peg where two blockchains directly communicate with each other (serving as the sidechains of each other).

III. System Model and Building Blocks

We consider the communication between two groups of nodes, denoted as A and B. A has m nodes and B has n nodes, i.e., $|\mathsf{A}| = m$ and $|\mathsf{B}| = n$. Nodes in A are denoted as $\{P_1^{\mathsf{A}}, \cdots, P_m^{\mathsf{A}}\}$ and nodes in B are denoted as $\{P_1^{\mathsf{B}}, \cdots, P_n^{\mathsf{B}}\}$. We assume that nodes in A and B know the identities of each other. We assume the existence of point-to-point authenticated channels between each pair of nodes.

For the communication between A and B, one group is the *source*, and the other group is the *target*. Without loss of generality, we assume A is the source group.

When we present our protocols, we consider a fully asynchronous network, making no assumption about message processing and transmission delays. In one of our case studies, we study a synchronous variant of our protocol. In a synchronous network, the message processing and transmission are bounded by some Δ, and all nodes know the value of Δ.

Threat model. We consider Byzantine failures, i.e., arbitrary failures such as software bugs, hardware errors, and adversarial attacks. A non-faulty node is called a *correct* node. Unless otherwise mentioned, we assume that A has at most t faulty nodes and $m \geq 3t+1$. We assume that B has at most f failures and $n \geq 3f+1$. These assumptions are optimal in an asynchronous environment. We let $m = 3t+1$ and $n = 3f+1$ without loss of generality. One may relax the assumption on m and t to enable more applications. In one of our case studies, a synchronous network is considered. We assume $n \geq 2f+1$ and do not make assumptions about t.

As summarized in Table I, we make different assumptions for different protocols. For protocols using ABC in A or B as a black box, the threat model and assumptions are also related to the underlying ABC. For instance, for XRBC-woA, which assumes ABC in A, XRBC-woA is secure against an adaptive adversary (that corrupts nodes while the protocol is running) if the underlying ABC is adaptively secure.

Byzantine reliable broadcast (RBC). The RBC abstraction allows a sender P_s to reliably broadcast a message to a group of nodes. An RBC protocol is specified by *r-broadcast* and *r-deliver* such that the following properties hold:

- **Validity**: If a correct node P_s *r-broadcasts* a message v, then any correct node eventually *r-delivers* v.
- **Agreement**: If some correct node *r-delivers* a message v, then every correct node eventually *r-delivers* v.
- **Integrity**: Every correct node *r-delivers* some message v at most once. Moreover, if a node *r-delivers* a message v with sender P_s, then v was previously broadcast by node P_s.

Byzantine reliable agreement (RA). The RA abstraction is an *agreement* version of RBC [40]. The only difference is that each node holds an input. Without loss of generality, we consider RA among n nodes, among which at most f are Byzantine faulty. An RA protocol is specified by *r-propose* and *r-decide* such that the following properties hold:

- **Agreement**: Same as in RBC.
- **Validity**: If all correct nodes *r-propose* v, then eventually all correct nodes *r-decide* v.
- **Integrity**: If a correct node *r-decides* v, then at least $n-2f$ correct nodes *r-propose* v.

The names of the properties are slightly revised compared to the original RA notion [40]. As mentioned in [40], we can easily *transform* most RBC protocols into RA protocols.

Atomic broadcast (ABC). Atomic broadcast allows nodes to reach an agreement on the order of messages (values). It is also called *total-order broadcast* in the literature [1]. An atomic broadcast protocol Π is specified by *a-broadcast* and *a-deliver*. When a node is provided (by an adversary) with a queue of payload messages of the form $v \in \{0,1\}^*$, we say the node *a-broadcasts* the messages. Correct nodes should *a-deliver* the same sequence of messages in the same order. An ABC protocol should satisfy the following properties.

- **Safety**: If a correct node *a-delivers* a message v before *a-delivering* v', then no correct node *a-delivers* a message v' without first *a-delivering* v.
- **Liveness**: If a correct node *a-broadcasts* a message v, then all correct nodes eventually *a-deliver* v.

ABC is only syntactically different from RSM (and BFT). Namely, ABC focuses on the agreement on the order of messages and does not consider the execution of transactions.

Multivalued validated Byzantine agreement (MVBA). MVBA allows each node that has an input to agree on a value that satisfies a predicate Q known by all nodes [20]. MVBA is specified by *mvba-propose* and *mvba-decide* events. An MVBA protocol satisfies the following properties:

- **External validity**: Every correct node that terminates *mvba-decides* v such that $Q(v)$ holds.
- **Agreement**: If a correct node *mvba-decides* v, then any correct node that terminates *mvba-decides* v.
- **Integrity**: If all nodes follow the protocol, and if a correct node *mvba-decides* v such that $Q(v)$ holds, then some node *mvba-proposed* v such that $Q(v)$ holds.
- **Termination**: If all correct nodes are activated and all messages sent among correct nodes have been delivered, then all correct nodes *mvba-decide*.

The predicate of MVBA can be both signature-based or state-based. For a signature-based predicate, v is usually validated by some data (e.g., digital signatures) [20], [41]. For state-based predicate, v can be locally validated without requiring signatures [25]. State-based MVBA is equivalent to *index ACS* [40] where the predicate is modeled as the notion of *party validation*. In this work, we use the state-based MVBA/Index ACS in our warm-up protocol.

(Synchronous) Byzantine broadcast (BB). The BB abstraction (also abbreviated as BC) is a synchronous primitive. BB allows a sender P_s to reliably broadcast a message to a group of nodes. A BB protocol is specified by *b-broadcast* and *b-deliver* such that the following properties hold:

- **Validity**: If a correct node P_s *b-broadcasts* a message v, then any correct node *b-delivers* v.
- **Agreement**: If some correct node *b-delivers* a message v, another correct node *b-delivers* a message v', then $v = v'$.
- **Termination**: Every correct node *b-delivers*.

It is worth mentioning that for a group of n nodes, among which at most f are Byzantine faulty, a synchronous BB can be achieved under $f < n$ [32]. Meanwhile, building efficient BB with $f < n/2$ or $f < n/3$ is often of independent interest [33].

(Synchronous) Byzantine agreement (BA). A BA protocol is specified by *ba-propose* and *ba-decide*. Each node *ba-propose* a value v and all correct nodes *ba-decide* some value and then terminate the protocol. BA should satisfy the following properties:

- **Validity**: If all correct nodes *ba-propose* v, then all correct nodes *ba-decide* v.
- **Consistency**: If some correct node *ba-decides* v, another correct node *ba-decides* v', then $v = v'$.
- **Termination**: Every correct node *ba-decides*.

In a synchronous network, for a group of n nodes, the lower bound of synchronous BA is $f < n/2$ [34].

Digital signatures and hash function. We use digital signature and we require the conventional unforgeability property. We use a cryptographic collision-resistant hash function. Given a value v, we use $\mathsf{hash}(v)$ to denote the hash of the value.

IV. Warm-up: Group-to-Group Reliable Broadcast

As a warm-up primitive, we first present group-to-group parallel reliable broadcast (Group-PRBC) that models the communication between two groups, where neither of the two groups has to have built-in ABC. We also provide a practical asynchronous construction and discuss the complexities.

Group-to-group parallel reliable broadcast (Group-PRBC). Group-PRBC can be modeled as m parallel RBC instances. Namely, each node P_i^{A} in A reliably broadcasts a message v_i to all nodes in B, specified by the *pr-broadcast* event. Each node P_i^{B} in B *pr-delivers* a vector of messages $\boldsymbol{v}$. The j-th value in $\boldsymbol{v}$ is called the j-th slot, for which P_j^{A} is the designated sender. A Group-PRBC protocol should satisfy the following properties hold:

- **Validity**: If a correct node P_i^{A} in A *pr-broadcasts* a message v_i, then any correct node P_i^{B} in B eventually *pr-delivers* $\boldsymbol{v}[i] = v_i$.
- **Agreement**: If some correct node P_i^{B} in B *pr-delivers* a message $\boldsymbol{v}[j]$, then every correct node in B eventually *pr-delivers* $\boldsymbol{v}[j]$.
- **Integrity**: For any slot s, every correct replica *pr-delivers* $\boldsymbol{v}[s]$ at most once. Moreover, if a node *pr-delivers* a message $\boldsymbol{v}[s]$, then $\boldsymbol{v}[s]$ was previously *pr-broadcast* by node P_s^{A}.

The notion above is simply m parallel RBC instances. A protocol satisfying the notion above does not necessarily terminate. Indeed, conventional RBC does not guarantee termination if the sender is incorrect. Accordingly, any protocol satisfying the above properties may never terminate. We thus provide an additional termination property and also modify the integrity property as follows.

- **Termination**: If all correct nodes in A *pr-broadcast*, any correct node in B *pr-delivers* some $\boldsymbol{v}$ and $|\boldsymbol{v}| \geq m - t$.

```
The Group-PRBC Protocol

– Input: each node $P_i^{\mathsf{A}}$ in A pr-broadcasts $v_i$ and the inputs by
  correct replicas might not be the same
– Output: each node $P_i^{\mathsf{B}}$ in B pr-delivers a set of messages $\boldsymbol{v}$
  where $|\boldsymbol{v}| = m$
– Initialization: $W_i \leftarrow \perp$
Node $P_i^{\mathsf{A}}$
– query $\mathsf{RBC}_i$ and r-broadcast $v_i$ to all replicas in B
Node $P_i^{\mathsf{B}}$
– upon r-delivering $v_j$ in $\mathsf{RBC}_j$, set $\boldsymbol{o}_i \leftarrow o_j$ and $W_i[j] \leftarrow 1$
– wait until $|\boldsymbol{o}_i| \geq m - t$, set $W \leftarrow W_i$ and mvba-propose $W$
  to $\mathsf{MVBA_B}$
– // set the predicate of MVBA as follows: $Q(W) \equiv$ (there
  exist at least $m - t$ $\ell$ such that $W[\ell] = 1$) and (for any
  $W[\ell] = 1, W_i[\ell] = 1$)
Output condition
– wait until $W$ is mvba-decided in $\mathsf{MVBA_B}$, then for each
  $W[\ell] = 1$, wait until $\boldsymbol{o}[\ell] \neq \perp$ and then set $\boldsymbol{v} \leftarrow \boldsymbol{o}[\ell]$
– pr-deliver $\boldsymbol{v}$
```

Fig. 2: The warm-up Group-PRBC protocol. The procedures that make the protocol additionally satisfy termination and modified integrity are highlighted in blue.

- **Modified integrity**: Every correct replica *pr-delivers* $\boldsymbol{v}$ at most once. Moreover, if a node *pr-delivers* a message $\boldsymbol{v}$ and $\boldsymbol{v}[s] \neq \perp$, then $\boldsymbol{v}[s]$ was previously broadcast by node P_s^{A}.

A. The Warm-up Protocol

We present a Group-PRBC protocol in Fig. 2. The workflow that executes n parallel RBC instances is written in black and the procedures that make the protocol additional satisfy the termination property and the modified integrity property are highlighted in blue. For readers who are familiar with asynchronous common subset (ACS) protocols [42], [30], [25], [31], the Group-PRBC protocol is very similar to signature-free ACS protocols such as FIN [25]. Namely, as we consider an asynchronous environment, our protocol does not guarantee that the messages *pr-broadcast* by any correct nodes in A will be *pr-delivered* by correct nodes in B. The best we could do is that the values *pr-broadcast* by at least $m - t$ nodes in A can be *pr-delivered*, same as that for ACS.

Complexity. Group-PRBC involves m parallel RBC instances and one MVBA instance. Using the state-of-the-art RBC [23] and MVBA [25], [40], [43] constructions, the protocol achieves $O(1)$ expected time and $O(mn+n^3)$ messages if we consider a signature-based setting and $O(mn + n^2)$ messages if we consider a signature-based setting.

The communication complexity depends on the underlying RBC and MVBA constructions. Our protocol achieves $O(m\mathcal{C}^L_{\mathsf{RBC}} + \mathcal{C}^m_{\mathsf{MVBA}})$ communication, where $\mathcal{C}^L_{\mathsf{RBC}}$ and $\mathcal{C}^m_{\mathsf{MVBA}}$ denote the communication complexities of L-bit RBC and m-bit MVBA among n nodes, respectively.

- In the signature-free setting, we can use CCBRB [23] and FIN-MVBA [25] for our protocol to achieve $O(mnL + \kappa n^2 m + mn^2 + \kappa n^3) = O(mnL + \kappa n^2 m + \kappa n^3)$, where

L is the size of each node's *pr-broadcast* message and κ is the length of the security parameter.

- If we use threshold signatures (which additionally assume trusted PKI), we can use SigBRB [23] and Dumbo-MVBA [41] for our protocol to achieve $O(nmL + \kappa nm + mn^2 \log n + mn + \kappa n^2) = O(nmL + \kappa nm + \kappa n^2 + mn^2 \log n)$ communication.

V. Cross-Consensus Reliable Broadcast

We are now ready to discuss cross-consensus reliable broadcast. Compared to Group-PRBC, the main difference for cross-consensus reliable broadcast is that it is expected that the source group has built-in ABC (i.e., A executes an atomic broadcast protocol). XRBC ensures that regardless of whether correct nodes in A send consistent messages or not, correct nodes in B will deliver the same messages.

In this section, we first present the definition of our XRBC primitive, a first-class primitive that can be used in several applications. We then present three different constructions: XRBC-woA, XRBC-wA, and XRBC-Sig. The assumptions and motivating applications are summarized as below.

- XRBC-woA: B does not have built-in ABC. XRBC-woA offers flexibility since it does not assume ABC for B. Later, we show in Sec. VI that XRBC-woA has a unique application for cross-shard coordination. Namely, nodes in A send their intermediate consensus messages that might not be the same. Nodes in B only need to agree on whether a sufficiently large fraction of nodes in A send consistent messages.
- XRBC-wA: B has built-in ABC. XRBC-wA assumes built-in ABC of B. The main advantage is that in some scenarios (e.g., cross-RSM communication), node in B already execute an ABC protocol anyway. We can thus *reuse* the infrastructure to provide reliable cross-consensus communication.
- XRBC-Sig: The protocol run by A has proof of delivery, and B has built in ABC. An application scenario is a cross-chain bridge, where the proof of delivery is often provided by blockchain systems. In this way, we build a communication-efficient protocol.

As summarized in Table I, compared to the naive Group-PRBC construction, our protocols achieve improved communication, especially when $m < n$.

A. Defining Cross-Consensus Reliable Broadcast (XRBC)

XRBC. We define XRBC as follows. Each node P_i^{A} in A reliably broadcasts a message v_i to all nodes in B, specified by the *x-broadcast* event. Each node P_i^{B} in B *x-delivers* a message v. The message *x-broadcast* by correct nodes in A is also called a *cross-consensus* message.

We consider the scenarios where messages sent by nodes in A might or might not be consistent. XRBC ensures that even if the messages sent by correct nodes are not consistent, B will *x-deliver* cross-consensus messages in the same order.

The XRBC notion thus defines the order of messages *x-delivered* by correct nodes in B. In particular, an XRBC protocol is specified by *x-broadcast* and *x-deliver* such that the following properties hold:

- **Safety**: If any correct node in B *x-delivers* v before *x-delivering* v', then no correct node in B *x-delivers* v' without first *x-delivering* v.
- **Termination**: If at least $2t + 1$ correct nodes in A *x-broadcast* v, any correct node in B eventually *x-delivers* v.
- **Integrity**: Every correct node *x-delivers* v at most once. Moreover, if a correct node *x-delivers* a message v, then v was previously *x-broadcast* by correct nodes in A.

Our notion of XRBC does not assign a particular *order* to the cross-consensus messages. The actual instantiation of most ABC protocols, however, assigns a *number* to denote the order (e.g., height, sequence number). Without loss of generality, we use *epoch* number r to denote such an order.

Assumptions. Without loss of generality, we consider two setups for the ABC instantiation in A: the cross-consensus message v itself does not come with a *proof of delivery*; v is associated with a proof of delivery π (e.g., a threshold signature or an aggregate signature on v). In the former case, nodes in B cannot verify whether v is valid (i.e., v is a valid cross-consensus message from at least one correct node in A) upon receiving v. In the latter case, any node can immediately verify whether v is valid upon receiving (v, π).

We assume that all nodes in B expect to receive cross-consensus messages for each epoch and will process them accordingly based on the epoch number.

XRBC vs. Group-PRBC. The differences between our XRBC notion and our warm-up Group-PRBC notion are two-fold. First, XRBC favors the scenarios where messages sent by correct nodes in A are expected to be consistent. Second, we need to consider a sequence of messages. In this work, we clearly use the epoch number to denote the sequence of a message, although the epoch number is not exposed to the API of XRBC. As we will show shortly, our XRBC protocols can be more communication-efficient compared to our warm-up Group-PRBC protocol.

XRBC vs. RBC. Although our XRBC resembles the idea of RBC protocols, the definition of correctness is fundamentally different. We only keep the integrity property of RBC and expand it for XRBC. This is mainly because RBC is a *one-shot* primitive where one node *r-broadcast* a message to all nodes. In contrast, XRBC cares about the order of messages sent from one group to the other. The safety property is thus different. Meanwhile, our termination property resembles the validity property of RBC. We provide more discussion on this property in remark 2 below.

Remark 1. We do not assume the entire set of nodes in A and B has built-in ABC, mostly because the problem is reduced to ABC. Also, our XRBC notion can be extended to communication among more than two groups. The application scenario is the *multichain* infrastructure, such as Cosmos [17], in which some transactions might be processed by multiple chains. Our primitive can be extended to such a scenario for reliable communication across multiple groups.

Remark 2. The termination property we specify for XRBC requires at least $2f+1$ correct nodes in A to *x-broadcast* v. One question arises: Why not change the $2t+1$ threshold of the termination property to another value, e.g., $t+1$? In fact, if we assume that correct nodes in A always send consistent messages (e.g., messages *a-delivered* in ABC), there exists a trivial instantiation. Namely, correct nodes in A assign sequentially ordered epoch numbers to all cross-consensus messages, where the order is the same as that in ABC of A. Each correct node then sends the ordered cross-consensus messages to all nodes in B. For any correct node in B, it waits for $2t+1$ matching v from A for each epoch and then *x-delivers* v. If this were the case, we could lower the threshold of the termination property from $2t+1$ to $t+1$.

The case above does not capture the scenario where messages sent by correct nodes in A are not always consistent. For example, the messages sent by nodes in A might be some intermediate consensus messages. Jumping ahead, in the case study of Reticulum, nodes in A send their votes to nodes in B, and the votes by correct nodes are not always the same. The goal of Reticulum is to ensure that nodes in B only deliver some messages when a sufficiently large fraction of correct nodes in A send matching votes. We believe our relaxation of the requirements on messages from A enables more interesting applications. Furthermore, we can adapt the threshold or even adapt the termination property for different applications. For instance, in our case study of Reticulum, we change the termination property to unanimous voting.

Remark 3. In our work, we assume $m \geq 3t+1$ and termination requires that $2t+1$ correct nodes in A *x-broadcast* matching messages. We believe such requirements can be relaxed to enable more interesting applications.

B. XRBC-woA: XRBC without ABC in B

We first consider XRBC-woA, the model where B does not execute an ABC protocol. We show the pseudocode of our XRBC-woA protocol in Fig. 4 and the workflow in Fig. 3a. In particular, every node in B executes the protocol for an epoch r and waits for epoch r to output some value before proceeding to epoch $r+1$.

The protocol is built from only best-effort broadcast (where each node sends a message to all nodes) and reliable agreement (RA). First, every node in A sends a $(\textsc{Cross}, r, v_i)$ message to all nodes in B. Upon receiving $t+1$ matching $(\textsc{Cross}, r, v)$ messages, every node in B queries an RA instance $\mathsf{RA}_{r,\mathsf{B}}$ and then *r-proposes* v. After some value v is *r-decided*, each node in B *x-delivers* v and then proceeds to the next epoch.

Sketch of correctness and complexities. We show the proof in Appendix F-A and provide a sketch here. Safety and integrity of XRBC follow from the agreement and integrity properties of RA. For termination, if $2t+1$ correct nodes in A *x-broadcast* v, no correct node in B can receive $t+1$ matching v' such that $v' \neq v$. Thus, all correct nodes in B *r-propose* v. By the validity property of RA, all correct nodes *r-decide* v. All correct nodes in B then *x-deliver* v.

Our protocol consists of only best-effort broadcast and RA, so it achieves $O(1)$ expected time. In the protocol, every node in A sends a $(\textsc{Cross})$ message to all nodes in B and only nodes in B execute the RA instance. Our protocol thus achieves $O(mn+n^2)$ messages, as all known RBC/RA constructions achieve $O(n^2)$ messages.

We can replace the input to RA from v to $\mathsf{hash}(v)$ while correctness still follows. Namely, after each node *r-decides* $\mathsf{hash}(v)$, it waits for at least one $(\textsc{Cross}, r, v)$ such that $v = \mathsf{hash}(v)$ and then *x-delivers* v. The communication complexity of our protocol is thus $O(mnL+\mathcal{C}^{\kappa}_{\mathsf{RA}})$, where $\mathcal{C}^{\kappa}_{\mathsf{RA}}$ is the communication complexity of κ-bit RA protocol. Using our instantiation in Fig. 4, our protocol achieves $O(mnL+\kappa n^2)$ communication.

C. XRBC-wA: XRBC with ABC in B

Considering that B executes ABC, we can further simplify the protocol. Namely, we can use ABC in B to achieve an agreement for the *x-delivered* messages.

We show the pseudocode of XRBC-wA in Fig. 5 and the workflow in Fig. 3b. On top of XRBC-woA, we replace RA with ABC among nodes in B. We can also replace the input to ABC with $\mathsf{hash}(v)$.

Sketch of correctness and complexities. We show the proof in Appendix F-B. Briefly speaking, safety and integrity of XRBC-wA follow from the safety of ABC and the fact that no correct node *a-delivers* values twice in each epoch. For termination, correctness follows from the liveness of ABC.

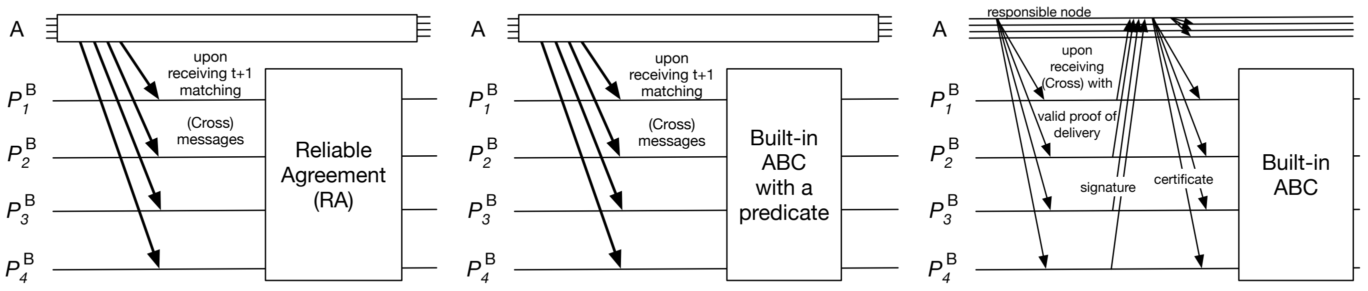
The complexities of our protocol in Fig. 5 depend on the ABC construction. Namely, the protocol achieves $O(\mathcal{T}_{\mathsf{ABC}})$ expected time, $O(mn+\mathcal{M}_{\mathsf{ABC}})$ messages, and $O(mnL+\mathcal{C}^{\kappa}_{\mathsf{ABC}}))$ communication, where $O(\mathcal{T}_{\mathsf{ABC}})$, $O(\mathcal{M}_{\mathsf{ABC}})$, and $O(\mathcal{C}^{\kappa}_{\mathsf{ABC}})$ are the time complexity, message complexity, and communication complexity of ABC in B, respectively.

D. XRBC-Sig: XRBC with Proof of Delivery

Motivation. Under the assumption that every cross-consensus message v can be associated with a proof of delivery π, we provide a construction as shown in Fig. 6. Our motivation is to simplify the workflow such that one node in A sends the cross-consensus message at a time. In this way, only $O(n)$ messages are needed (instead of $O(mn)$ in other XRBC protocols).

We define a notion of *responsible node for an epoch*. In particular, one designated node in A is supposed to send the cross-consensus message v together with the proof of delivery π to all nodes in B. We assume that all nodes in A are aware of the identity of the responsible node for each epoch r. The responsible node can be selected pseudorandomly via approaches such as VRF [44], [45], [46] and common coins [47], [30], [25].

A naive construction and the challenges. We show the pseudocode that is correct only when every responsible node is correct in black in Fig. 6. In each epoch, the responsible



(a) Our XRBC-woA protocol. (b) Our XRBC-wA protocol. (c) Our XRBC-Sig protocol.

Fig. 3: Overview of our approaches. Bold arrows denote messages sent by *all* nodes in A and regular arrows denote messages sent by one node.

XRBC-woA

- **Input**: each node P_i^{A} in A *x-broadcasts* v_i and the inputs by correct nodes are the same
- **Output**: each node P_i^{B} in B *x-delivers* v

Node P_i^{A}

- send a $(\textsc{Cross}, r, v_i)$ message to all nodes in B

Node P_i^{B}

- upon receiving $t+1$ $(\textsc{Cross}, r, v)$ messages from nodes in A, *r-propose* v to $\mathsf{RA}_{r,\mathsf{B}}$ among B

Output condition for epoch r

- wait until v is *r-decided* in $\mathsf{RA}_{r,\mathsf{B}}$ and *x-deliver* v

An instantiation of $\mathsf{RA}_{r,\mathsf{B}}$ adapted from Bracha's Broadcast [21], [22]

- upon v is *r-proposed*, send $(\textsc{Echo}, v)$ to all nodes in B
- upon receiving $n-f$ matching $(\textsc{Echo}, v)$ message, send $(\textsc{Ready}, v)$ to all nodes in B
- upon receiving $f+1$ matching $(\textsc{Ready}, v)$ messages and a $(\textsc{Ready}, v)$ message has not been sent, send $(\textsc{Ready}, v)$ to all nodes in B
- upon receiving $n-f$ $(\textsc{Ready}, v)$ messages, *r-decide* v

Fig. 4: The XRBC-woA protocol (XRBC protocol without ABC in B) for epoch r. The instantiation of $\mathsf{RA}_{r,\mathsf{B}}$ is the same as that by Das et al. [40]. One can replace the input to $\mathsf{RA}_{r,\mathsf{B}}$ with the hash of v to achieve lower communication.

XRBC-wA

- **Input**: each node P_i^{A} in A *x-broadcasts* v_i and the inputs by correct nodes are the same
- **Output**: each node P_i^{B} in B *x-delivers* v

Node P_i^{A}

- send a $(\textsc{Cross}, r, v_i)$ message to all nodes in B

Node P_i^{B}

- upon receiving $t+1$ $(\textsc{Cross}, r, v)$ messages from nodes in A, *a-broadcast* v
- set up a predicate in ABC as follows: an *a-broadcast* value v is valid after receiving $t+1$ $(\textsc{Cross}, r, v)$ messages and $(\textsc{Cross}, r-1, *)$ is *a-delivered*

Output condition for epoch r

- wait until *a-delivering* v in ABC and *x-deliver* v

Fig. 5: The XRBC-wA protocol (XRBC protocol with ABC in B) for epoch r. One can replace the input to ABC with the hash of v to achieve lower communication.

node sends a $(\textsc{Cross}, r, v_i, \pi)$ message to all nodes in B. After receiving a valid $(\textsc{Cross})$ message, every node in B sends a $(\textsc{Rep}, r, \mathsf{hash}(v), \sigma_i)$ message to the responsible node. Upon receiving $n-f$ valid $(\textsc{Rep})$ messages, the responsible node combines the signatures into a *certificate* σ and then sends a message $(\textsc{Confirm}, r, h, \sigma)$ to all nodes in B. Finally, nodes in B query ABC and *a-broadcast* h. After some value h is *a-delivered*, nodes obtain the corresponding v, *x-deliver* v, and enter the next epoch.

The challenge of the naive construction is to handle the case where the responsible node is Byzantine. Namely, the following scenarios may happen: 1) the responsible node does not send the cross-consensus message v to any node in B; 2) the responsible node sends v to some correct node(s) such that v is *x-delivered* but other nodes do not receive v.

To handle 2), we need to ensure that the responsible node in epoch $r' > r$ *help* nodes in B *x-deliver* v. We set up a pp field in the $(\textsc{Cross})$ message to do so. To handle 2), the nodes in B need to strictly follow the order of epoch numbers when they *a-broadcast* any messages in ABC. Finally, we set up a *Fetch* and a *Catchup* procedure.

The protocol. The pseudocode of our fully-fledged XRBC-Sig protocol is shown in Fig. 6 and the workflow is shown in Fig. 3c. We highlight those procedures to address the case of a Byzantine responsible node in blue.

For any node P_i^{A}, if P_i^{A} is a responsible node for epoch r, it first checks whether there exists a $Cer[r'] = \perp$ for any r' such that $r' < r$. If so, P_i^{A} adds the cross-consensus message in epoch r and the proof of delivery in pp. In this way, any potentially undelivered cross-consensus messages with proofs of delivery might be included in pp. Then, P_i^{A} sends a $(\textsc{Cross}, r, v_i, \pi, pp)$ message to all nodes in B. It then waits for $n-f$ $(\textsc{Rep})$ messages with matching signatures. After that, P_i^{A} combines the signatures into a certificate σ and sends a $(\textsc{Confirm}, r, \mathsf{hash}(v), \sigma)$ message to all nodes in A and B.

```
XRBC-Sig
– Input: each node $P_i^{\mathsf{A}}$ in A x-broadcasts $v_i$ and the inputs by correct nodes are the same
– Output: each node $P_i^{\mathsf{B}}$ in B x-delivers $v$
– Initialization: epoch $r$, certificate $Cer \leftarrow \perp$, last epoch $lr$

Node $P_i^{\mathsf{A}}$ in epoch $r$
– if $P_i^{\mathsf{A}}$ is responsible for epoch $r$
    – set $r' \leftarrow r-1$, while $Cer[r'] = \perp$
        – set $pp \leftarrow pp \cup (r', v_{r'}, \pi_{r'})$, where $v_{r'}$ is the cross-consensus message in epoch $r'$ and $\pi_{r'}$ is the PoD
        – $r' \leftarrow r' - 1$
    – send a (CROSS, $r, v_i, \pi, pp$) message to all nodes in B, where $\pi$ is the PoD for $v_i$ in epoch $r$
    – wait until receiving $n-f$ matching (REP, $r$, hash$(v), \sigma_j$) messages from B, let $\sigma$ be the certificate for the $n-f$ signatures, send (CONFIRM, $r$, hash$(v), \sigma$) to all nodes in A and B
    – upon receiving $f+1$ matching (FETCH, $lr, hr$) messages from B
        – for $r' \in [lr+1, hr-1]$, set $pp \leftarrow pp \cup (r', v_{r'}, \pi_{r'})$, where $v_{r'}$ is the cross-consensus message in epoch $r'$ and $\pi_{r'}$ is the PoD
        – send a (CATCHUP, $r, pp$) to all nodes in B
– for any $P_i^{\mathsf{A}}$ in epoch $r$
    – upon receiving a valid (CONFIRM, $r, h, \sigma$) from $P_j^{\mathsf{A}}$ such that $P_j^{\mathsf{A}}$ is responsible for epoch $r$
        – $Cer[r] \leftarrow (r, h, \sigma)$
        – for any $r' < r$, $Cer[r'] \leftarrow \perp$

Node $P_i^{\mathsf{B}}$
– upon receiving (CROSS, $r, v, \pi, pp$) message from $P_j^{\mathsf{A}}$
    – if the lowest $r'$ in $pp$ satisfies $r' > lr+1$, send a (FETCH, $lr, r'$) message to $P_j^{\mathsf{A}}$
    – wait until either condition is triggered: 1) (CATCHUP, $r, pp$) is received from $P_j^{\mathsf{A}}$ and then process $pp$ in the same way as below; 2) some value is x-delivered in epoch $r'$
    – for each $(r', v', \pi') \in pp$ (ordered by $r'$)
        – a-broadcast $(r', v', \pi')$ and wait until x-delivering $(r', v')$
    – send a (REP, $r$, hash$(v), \sigma_i$) to $P_j^{\mathsf{A}}$, where $\sigma_i$ is a digital signature for $(r, \text{hash}(v))$
– upon receiving (CONFIRM, $r, h, \sigma$) message from $P_j^{\mathsf{A}}$
    – if it has previously received a (CROSS, $r, v, \pi$) message such that hash$(v) = h$, a-broadcast $(r, v, \pi)$

Output condition for epoch $r$
– after $(r', v')$ has been x-delivered for any $r' < r$, wait until a-delivering $v$, x-deliver $v$ and set $lr \leftarrow r$
```

Fig. 6: The XRBC-Sig protocol (XRBC protocol with proof of delivery in A and ABC in B). The procedures that address incorrect responsible nodes are highlighted in blue. PoD for Proof of delivery.

If P_i^{A} is not a responsible node for epoch r, it waits for a certificate in the (CONFIRM) message and updates its $Cer[r]$.

For any node P_i^{B} in B, it waits for a valid (CROSS, r, v, π, pp) message from the responsible node in A. Then it processes the messages in the pp field. There are two cases: 1) the lowest epoch number r' for any cross-consensus messages in pp is the same as its last completed epoch lr; 2) the lowest epoch number r' in the pp field is higher than its last completed epoch. In the second case, P_i^{B} sends a (FETCH, lr, r') message to the responsible node in A. Then, P_i^{B} waits until one of the conditions is satisfied: it receives those cross-consensus messages between epoch $lr+1$ and $r'-1$ from the responsible node via a (CATCHUP) message; it has *x-delivered* some message for epoch r'. If P_i^{B} receives some pp via the (CATCHUP) or the (CROSS) message, it processes the cross-consensus messages according to the epoch numbers in ascending order. For each epoch r', P_i^{B} *a-broadcasts* (r', v', π'), where v' is the cross-consensus message and π' is the proof of delivery. Each node *a-broadcasts* a new message for epoch r' after it *x-delivers* some message for epoch $r'-1$. In this way, we can ensure that the cross-consensus messages are *x-delivered* in the same order, so the safety of XRBC is achieved.

Sketch of correctness. We sketch the correctness of XRBC-Sig. Safety and integrity follow those of ABC and the fact that correct nodes in B always *x-deliver* the messages according to the order of epoch numbers. We thus focus on termination.

Since there is only one responsible node for each epoch r, it is possible that no value is *a-delivered* and *x-delivered* for nodes in B. Our protocol uses the pp field to ensure that there exists a correct responsible node in some epoch $r' > r$. If the responsible node is selected, it will use the pp field to carry the *missing* cross-consensus messages. Correct nodes in B will eventually *x-deliver* the messages. Under the assumption that the responsible node is selected pseudorandomly, if a responsible node for epoch r is Byzantine and the cross-consensus message v is not *x-delivered*, with probability $\frac{m-t}{m}$, the responsible node in epoch $r+1$ is correct. If so, the responsible node in epoch $r+1$ will ensure that the correct nodes in B *a-broadcast* the cross-consensus message in epoch r. By the liveness property of ABC, all correct nodes in B *x-deliver* the message. In the worst case, t consecutive responsible nodes are faulty. The details of the proofs can be found in Appendix F-C.

Complexity analysis. The protocol involves a constant number of communication steps between the responsible node and the nodes in A. Following the sketch of correctness above, the time complexity of our protocol is $O(\mathcal{T}_{\mathsf{ABC}})$.

The message complexity of our protocol is $O(n + \mathcal{M}_{\mathsf{ABC}})$, as one node in A sends messages to nodes in B in each epoch.

The communication complexity of our protocol is $O(nL + \kappa n + \mathcal{C}^{\kappa}_{\mathsf{ABC}})$, under the optimization that the *a-broadcast* message is replaced by the hash of the message. Namely, this is because each responsible node is supposed to send a constant number of piggybacked cross-consensus values together with their proofs of delivery. While the proof of delivery can be instantiated in many different ways, the length of each proof can be as low as $O(\kappa)$ if we use threshold signatures.

Discussion. With our XRBC-Sig protocol, the complexity of the cross-consensus protocol does not depend on the size of the source group anymore. Note that we can further optimize the message complexity (and possibly the communication complexity) by additionally requiring one responsible node in B as the *receiver* of each cross-consensus message. However, this would increase the time complexity for the protocol. We provide additional discussion in Appendix A.

VI. Applications

We present three applications of our XRBC protocols. In this section, we present two sharding-based protocols. We discuss the third one (cross-chain bridge) in Appendix D.

A. Cross-shard Coordination

We present a case study about cross-shard coordination in Reticulum [19], a synchronous two-layer sharding protocol. We show that we can adapt our XRBC-woA protocol to the synchronous setting and provide a more communication-efficient cross-shard coordination protocol. Additionally, our protocol has fewer assumptions, as summarized in Table II.

protocol	**Reticulum** [19]	**ours**
assumptions	$n \geq 2f+1$; $n+m \geq 2(f+t)+1$	$n \geq 2f+1$
msg.	$O((m+n)^2)$	$O(mn+n^2)$
comm. (trusted)	$O(m\kappa(m+n)^2)$	$O(\kappa mn+\kappa n^2)$
comm. (bulletin)	$O(m\kappa(m+n)^3)$	$O(\kappa mn+\kappa n^2+n^3)$

TABLE II: Comparison of cross-shard coordination protocols. Here, "trusted" means that the protocol relies on threshold signatures and thus requires trusted setup. "Bulletin" means that the protocol assumes the standard public key infrastructure.

Cross-shard coordination in Reticulum [19]. Reticulum uses a two-layer structure: a layer with *process shards* and a layer of *control shards*. Process shards process the submitted transactions, and the control shards *manage* the process shards. Each process shard is managed by one control shard. The idea is to use small process shards so the performance is high when the process shard is failure-free. Meanwhile, large control shards are used to ensure the correctness of the system when the process shard has failures. Let the process shard be A (m nodes) and the control shard be B (n nodes). Reticulum assumes $n \geq 2f+1$ (which is optimal in the synchronous setting). Additionally, $n+m \geq 2(f+t)+1$.

In the protocol, a crucial phase is that each node in the process shard A sends its *vote* (a message with κ length) to all nodes in both A and the control shard B. The votes by correct nodes are not always consistent. To disseminate the vote, each node uses the $(\Delta+\delta)$-BB [33] protocol, a synchronous BB protocol that tolerates fewer than half Byzantine failures. The $(\Delta+\delta)$-BB protocol works roughly as follows (note that in our context, the sender is a node in A and the receivers are nodes in both A and B).

- (Propose). The sender P_s^{A} sends (v,σ) to all nodes in A and B, where σ is a signature for v.
- (Vote). Upon receiving a proposed value from P_s^{A}, send a vote together with (v,σ) to all nodes in A and B.
- (Commit and lock). If no equivocation of P_s^{A} is detected (informally, equivocation means that P_s^{A} sends two conflicting values in the propose phase) and $f+1$ matching votes are received, forward the votes to all nodes. Lock v in this case.
- (Byzantine agreement). Invoke a Byzantine agreement instance and use the locked value as input. Deliver the output of the Byzantine agreement protocol.

As there are m senders (from A) and $m+n$ receivers (both A and B), m BB instances are invoked so the protocol can be viewed as m parallel BB. The communication complexity is $O(\kappa(m+n)^3)$ under the trusted setup assumption (i.e., the votes are signed using threshold signature) and $O(\kappa(m+n)^4)$ under the standard PKI assumption. To avoid such a high communication overhead, Reticulum presents a simplified solution that *combines* the "vote", "commit and lock", and "Byzantine agreement" phases. Namely, in the propose phase, each node in A sends its proposed value to A and B. In the "vote" phase, every node aggregates the proposed values and then sends the vote messages to other nodes. In this way, the communication can be lowered to $O(m\kappa(m+n)^2)$ under the trusted setup assumption and $O(\kappa(m+n)^3)$ under the PKI assumption. The bottleneck under the trusted PKI assumption is the combined "vote" phase, where each node needs to send all the m received values. Meanwhile, the bottleneck under the PKI assumption is the "commit and lock" phase and the "Byzantine agreement" phase. We provide a detailed analysis in Appendix C.

Discussion. The simplified parallel BB does not achieve the security properties of m parallel BB instances. To see why, consider the fact that some node P_i^{A} in A is faulty. It can simply send different v to different nodes. Obviously, the votes by correct nodes are not matching. Eventually, all correct nodes will deliver $\perp$. In fact, some non-$\perp$ value is delivered only when *all* nodes in A are correct.

Such a design already inherently suits the needs of Reticulum as the goal is to use the above design as a *fast* path to improve the performance in the optimistic case. Namely, if all nodes in A send the same value v to both A and B, all nodes deliver v. Unfortunately, no formalization is given. Additionally, the communication cost is still high.

Synchronous XRBC-woA. We can use our XRBC-woA paradigm to make the cross-shard coordination protocol simpler and more communication-efficient. We provide a synchronous variant of our XRBC-woA protocol that achieves $O(\kappa mn+\kappa n^2)$ communication under the trusted setup assumption and $O(\kappa mn+\kappa n^2+n^3)$ under the PKI assumption.

We show the pseudocode of our synchronous XRBC-woA protocol in Fig. 7. Our protocol only assumes that $n \geq 2f+1$ (i.e., we do not need the $n+m \geq 2(f+t)+1$ assumption). To further suit the needs for Reticulum, we revise the termination property of XRBC as a new *unanimous termination* property. For this new termination property, we expose the epoch number r to the API.

- **Unanimous voting:** In some epoch r, if all nodes in A *x-broadcast* v, all correct nodes in A and B *x-deliver* v.

Other than the unanimous voting property, the safety and integrity are the same, except that all nodes in A and B *x-deliver* messages.

- **Input**: each node P_i^{A} in A *x-broadcasts* v_i and the inputs by the nodes might not the same
- **Output**: each node P_i^{A} in A and P_i^{B} in B *x-delivers* v

Node P_i^{A}

- send a (CROSS, r, v_i) message to all nodes in B

Node P_i^{B}

- upon receiving m (CROSS, r, v) messages from nodes in A, *ba-propose* v to $\mathsf{BA}_{r,\mathsf{B}}$ among B
- upon *ba-deciding* v' (v' might be $\perp$) in $\mathsf{BA}_{r,\mathsf{B}}$, send (NOTIFY, v') to all nodes in A and B

Output condition for epoch r

- upon receiving $f+1$ matching (NOTIFY, v) messages, *x-deliver* v

Fig. 7: The communication-efficient cross-shard coordination protocol from a synchronous variant of XRBC-woA protocol.

Compared to our XRBC-woA protocol presented in Sec. V-B, there are two main changes for the protocol in Fig. 7. First, we replace RA with BA since BA is a synchronous primitive. Second, to ensure that nodes in both A and B will *x-deliver* messages, all nodes in B send a (NOTIFY) message to A and B and every node *x-delivers* only after receiving $f+1$ matching (NOTIFY) messages.

Safety and integrity follow as BA is a synchronous variant of RA. Namely, BA ensures that all correct nodes in B decide the same values. As there are at most f faulty nodes in B, any correct node will *x-deliver* the same value.

For unanimous voting, if all correct nodes *x-broadcast* the same value, all correct nodes will *ba-propose* the same value in $\mathsf{BA}_{r,\mathsf{B}}$. According to the validity property of BA, all correct nodes *ba-decide* v in $\mathsf{BA}_{r,\mathsf{B}}$. Then, all correct nodes in A and B *x-deliver* v.

Assuming N is the total number of nodes, the state-of-the-art BA protocol achieves $O(\kappa N^2)$ communication under the trusted setup assumption [33] and $O(\kappa N^2 + N^3)$ under the PKI assumption [32]. Thus, our synchronous cross-shard coordination protocol achieves $O(\kappa mn + \kappa n^2)$ communication under the trusted setup assumption and $O(\kappa mn + \kappa n^2 + n^3)$ under the PKI assumption (see Appendix C for details).

B. Handling Cross-shard Transactions

Cross-shard transactions refer to the transactions that need to be processed by more than one shard. Many works present efficient solutions for handling cross-shard transactions [9], [10], [11], [12]. In this section, we present a case study on Chainspace [11]. We show that we can use our XRBC-wA and XRBC-Sig protocols to improve the time complexity or/and the communication complexity of Chainspace.

Cross-shard protocol in Chainspace [11]. Chainspace uses an *atomic commit* protocol (which can be viewed as a shard-level two-phase commit protocol [48]) to handle cross-shard transactions. Considering that A and B are two shards that need to process the cross-shard transaction tx, the protocol works roughly as follows.

- (Initial broadcast) The client sends tx to both A and B.
- (Sequence prepare) Nodes in each shard execute a conventional Byzantine fault-tolerant (BFT) protocol (e.g., PBFT [8] in Chainspace). After an agreement is reached locally, both A and B notify each other about the decision.
- (Process prepare) Nodes in each shard again execute a BFT protocol to agree on the notification result. After an agreement is reached, tx is committed. Another shard (called output shard) is employed to *finalize* the result.

The communication between different shards occurs at the "sequence prepare" phase, where *both* A *and* B *notify each other about the decision.*

The issue with a BFT-Initiator. Chainspace mentions that it can use a *BFT-Initiator* (e.g., leader of a shard) for sending the decision to other shards. In this way, the communication can be optimized. To address the case that the BFT-Initiator might be faulty, a two-phase process is used to recover from a faulty BFT-Initiator, as shown below.

- All nodes in each shard *monitor* whether the BFT-Initiator has correctly sent out the message.
- If the BFT-Initiator does not proceed according to the protocol, nodes may time out and act as BFT-Initiators.

The proof of Chainspace focuses on the correctness, where all nodes in A send messages to B, and discusses the correctness of the optimization. We argue that the correctness of the optimization might be trickier than expected. Consider the following scenario: after A agrees on the result "prepare" in the *sequence prepare* phase, nodes in A are expected to send a (Prepare) message to all nodes in B. Here, the BFT-Initiator is expected to send the message. We now consider the following scenario:

- An agreement is reached for nodes in A. The BFT-Initiator does not send the (Prepare) message to any node in B.
- Nodes in A are expected to *monitor* such a behavior and serve as BFT-Initiators if they time out.

Monitoring whether nodes in B have received the (Prepare) messages is highly challenging, if not impossible. This is because nodes in B are the only parties that will receive the messages. Without their feedback, nodes in A cannot know whether the BFT-Initiator has sent a (Prepare) message or not. This is similar to the challenges for our naive solution described in Sec. V-D. Meanwhile, the timeout mechanism might slow down the performance of the system under faulty BFT-Initiators.

We would like to emphasize that Chainspace is provably secure if *all* nodes in A send the messages to all nodes in B. In fact, we found that most protocols that use a cross-shard communication pattern largely ignore the details about the reliability of the communication [10], [9], [12]. For instance, Pyramid [10] and RapidChain [9] mention that some shard A sends a message to a shard B. We can only ensure that all nodes in B eventually receive a cross-shard message if all nodes in A send the message to all nodes in B. Such a paradigm has $O(Lmn)$ communication.

Thus, we claim that by modeling cross-consensus reliable broadcast as a dedicated primitive, it becomes easier to model the correctness of sharding-based protocols.

Cross-shard protocol from XRBC-wA and XRBC-Sig. We can use XRBC-wA or XRBC-Sig to instantiate "A sending a message to B." The protocol already ensures that B *x-deliver* the cross-shard messages in a total order (safety of XRBC), and they eventually receive the cross-shard messages (termination). We can use XRBC-wA to build a signature-free protocol so that neither A nor B needs to provide proof of delivery. Also, we can use XRBC-Sig to build a communication-efficient but signature-based protocol. It is worth mentioning that our XRBC-Sig protocol can be directly used in protocols like Chainspace and Pyramid to lower the communication complexity from $O(Lmn + \kappa mn)$ to $O(nL + \kappa n)$.

Note that several sharding-based protocols point out that showing proof of delivery is not necessarily an *advantage*. For instance, RapidChain [9] mentions that one major issue for Omniledger [12] needs to disseminate proof of delivery (called proof-of-acceptance in the paper), which incurs a large communication overhead. Thus, both XRBC-wA and XRBC-Sig are useful for sharding-based protocols.

VII. Evaluation

Implementation. We implement XRBC-woA, XRBC-wA, and XRBC-Sig in Golang [2]. We use gRPC as the communication library. We use HMAC to realize the authenticated channel, SHA256 as the hash function, and ECDSA as the digital signature scheme. For the ABC protocol in B, we use HotStuff [49]. Note that we can use any ABC protocols, e.g., variants of HotStuff that outperform HotStuff [50], [51], [52].

As XRBC is a new primitive, we do not have an existing work to compare. Thus, we use Group-PRBC as the baseline when assessing our XRBC protocols. We also assess the performance of our case study on Reticulum. In particular, we compare our protocol in Fig. 7 with the vanilla approach by Reticulum (presented in Sec. VI-A). For the underlying cryptographic primitives, we use the same approaches as our XRBC protocols to obtain a fair comparison. Our implementation of the protocols involves more than 11,000 lines of code.

Deployment considerations. Our XRBC protocols serve as dedicated protocols for reliable communication across two groups. In terms of deployment, our software can be built and used as a middleware. Since we already provide clear APIs (i.e., *x-broadcast* and *x-deliver*) for our protocols, our middleware can expose these two events to existing infrastructures (i.e., implementations of ABC). The integration is expected to be relatively straightforward.

Evaluation setup. We evaluate the performance on Amazon EC2 using up to 91 virtual machines (VMs). We use m5.xlarge instances. The m5.xlarge instance has four virtual CPUs and 16GB memory. We evaluate our protocols in both LAN and WAN settings. Unless otherwise mentioned, we deploy our protocols in the WAN setting, where replicas are evenly distributed across the following regions: us-west-2 (Oregon, US), us-east-2 (Ohio, US), ap-southeast-1 (Singapore), and eu-west-1 (Ireland). In some experiments, we evaluate the performance in the us-west-2 (Oregon, US) region.

We conduct the experiments with different network sizes and batch sizes. We use $m = 3t + 1$ and $n = 3f + 1$ in all experiments and use f and t to denote the numebr of faulty nodes. We use b to denote the batch size, where each replica sends b (Cross) messages one after the other. By default, each transaction has 250 bytes. In one of our experiments, we vary the transaction size to assess our improved XRBC-woA and improved XRBC-wA protocols.

Evaluation metrics. We repeat each experiment five times and report the average results. Throughput is measured as the number of non-repetitive (Cross) messages (i.e., one for each epoch) processed during the experiment. The latency is measured as the duration from the time each node in A sends a (Cross) message to the time nodes in B *x-deliver* the message in each epoch. If more than one (Cross) message is sent (i.e., $b > 1$), we report the average latency to process each message and call it *amortized latency*.

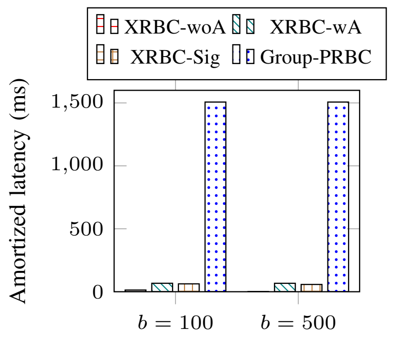
Most of our experiments focus on the latency of the protocols. In all the applications we have found for XRBC protocols, cross-consensus communication is usually used to communicate smaller messages (e.g., a hash). Our protocols can improve their latency and, accordingly, the performance of the entire system.

XRBC vs. the baseline protocol. We compare the latency and throughput of XRBC and Group-PRBC, our baseline protocol. We report the performance for $f = 10$, 20, and 30. For each f, we evaluate $b = 100$ and 500. As shown in Fig. 8a and Fig. 8b, the performance of XRBC protocols is consistently higher than that of Group-PRBC. We believe evaluating such a batch size is sufficient for XRBC protocols. As mentioned previously, for the applications we have identified for XRBC protocols, we do not need a larger batch size.

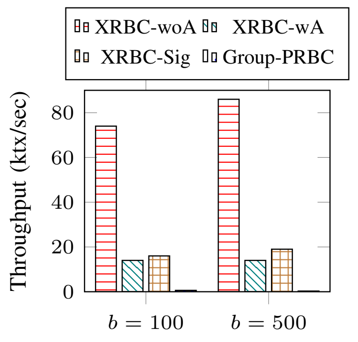
Latency of XRBC protocols. We assess the latency of our XRBC protocols for different f. As shown in Fig. 8c, we vary the batch size b and report the amortized latency. Among the three XRBC protocols, XRBC-woA consistently outperforms XRBC-wA and XRBC-woA. This is expected, as we use RA for nodes in B to agree on the order. RA is more latency-optimal (terminating in two communication steps) compared to the ABC approach we use (i.e., HotStuff).

Scalability of XRBC protocols. We show the performance of the protocols by varying f from 10 to 30. As shown in Fig. 8c, the latency of our protocol only slightly degrades as f grows. Even in the extreme case when $f = 30$, XRBC-woA achieves 12 ms latency. We believe this is because in our protocols, nodes in B only take the hash of the cross-consensus messages as input, making our protocol very lightweight.

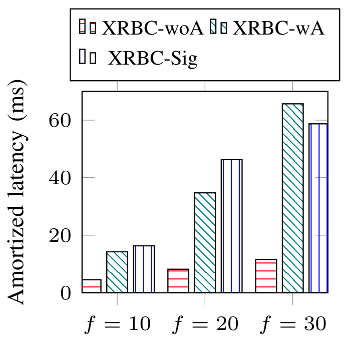
[2]Our codebase can be found at: https://doi.org/10.5281/zenodo.16945739 or https://github.com/DSSLab-Tsinghua/XRBC



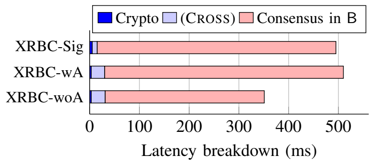
(a) Latency of XRBC and Group-PRBC for $t = 1$ and $f = 30$.



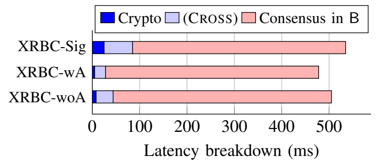
(b) Throughput of XRBC and Group-PRBC for $t = 1$ and $f = 30$.



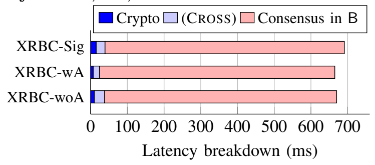
(c) Amortized Latency of XRBC-woA, XRBC-wA, and XRBC-Sig for $t = 1$ and $f = 10, 20, 30$.



(d) Latency breakdown of XRBC-woA, XRBC-wA, and XRBC-Sig for $t = 1$ and $f = 10$.

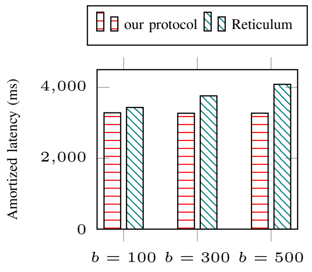


(e) Latency breakdown of XRBC-woA, XRBC-wA, and XRBC-Sig for $t = 1$ and $f = 20$.

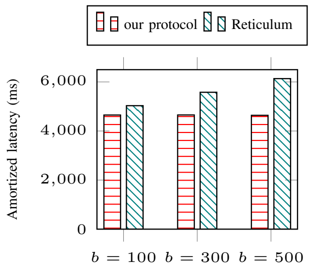


(f) Latency breakdown of XRBC-woA, XRBC-wA, and XRBC-Sig for $t = 1$ and $f = 30$.

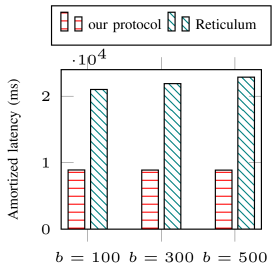
Fig. 8: Evaluation results of our XRBC-woA, XRBC-wA, and XRBC-Sig protocols.



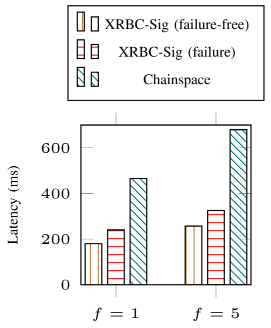
(a) Latency of our protocol and Reticulum for $t = 1$ and $f = 10$.



(b) Latency of our protocol and Reticulum for $t = 1$ and $f = 20$.



(c) Latency of our protocol and Reticulum for $t = 1$ and $f = 30$.



(d) Latency of XRBC-Sig and Chainspace for different f, where $t = 1$ and $b = 1$.

Fig. 9: Evaluation of our cross-shard coordination case study on Reticulum [19] (Sec. VI-A) and cross-shard transactions on Chainspace [11] (Sec. VI-B).

Latency breakdown. We report the latency breakdown of XRBC-woA, XRBC-wA, and XRBC-Sig. We assess the cases for $f = 10$, 20, and 30, and for $b = 1$. As shown in Fig. 8d-Fig. 8f, we break the latency of our protocol into three parts: Crypto, (CROSS) and consensus in B. Crypto denotes the time of cryptographic operations. (CROSS) denotes the latency of sending the (CROSS) message. Consensus in B denotes the amortized latency at B, i.e., RA in XRBC-woA, and ABC in XRBC-wA and XRBC-Sig. In all the cases, consensus in B is the bottleneck. This is mainly because (CROSS) only involves one step of communication between A and B. Our protocols involve the basic cryptographic operations such as digital signatures and hash, so the overhead is very low. Meanwhile, the latency for consensus in B is in general very close for the three protocols. In practice, we believe the main overhead of all XRBC protocols is thus the ABC approach in B.

n	b	$m = n$	$m = 4 < n$
$n = 31$	300	12.3	5.2
$n = 61$	300	25.0	10.2
$n = 91$	300	35.3	12.5

Fig. 10: Amortized latency of XRBC-woA for $m = n$ and $m < n$.

Latency with $m = n$ and $m < n$. We assess the performance of XRBC-woA when $m = n$ and $m < n$. As shown in Fig. 10, the latency of XRBC-woA is consistently lower when $m = n$. This is mainly because each node in B expects to receive much fewer matching messages to complete the protocol. This has validated our claim that our protocols have a higher performance gain when $m < n$.

Case study on cross-shard coordination comparison. We assess the latency of our protocol in Fig. 7 and compare it with the vanilla approach by Reticulum. As shown in Fig. 9a-Fig. 9c, compared to the vanilla approach, our protocol achieves 57.03%-61.16% lower latency for $f = 30$, 7.58%-24.38% lower latency for $f = 20$, and 4.37%-19.90% lower latency for $f = 10$. The results validate our claim that XRBC is a crucial building block that benefits upper applications.

Case study on Chainspace [11]. As mentioned in Sec. VI-B, our case study focuses on the claim that it is not straightforward to use the optimization of BFT-Initiator. To evaluate the situation, we implement the basic workflow as mentioned in Sec. VI-B. Namely, the BFT-Initiator in A sends a cross-consensus message to all nodes in B. Upon receiving such a message, each node in B sends a reply to all nodes in A. If a node in A does not receive a reply from $f + 1$ nodes in B within a certain time (200 ms in our experiment), it triggers a *view change* to elect a new BFT-Initiator.

We evaluate the latency of the scenario above under one faulty BFT-Initiator, and compare it with XRBC-Sig. The latency is the time from a transaction is sent to the time it is *x-delivered*. All experiments are conducted in the LAN setting by setting $b = 1$ and $t = 1$. As shown in Fig. 9d, the latency of XRBC-Sig is 48.4% and 52.0% lower than that of the Chainspace scenario for $f = 1$ and $f = 5$, respectively. Note that the latency of the Chainspace scenario is related to the actual timer. In contrast, our protocol does not need such a timer. Thus, our results show that our protocol can improve the performance in addition to making correctness more clear.

XRBC-Sig under failures. We also use Fig. 9d to show the performance of XRBC-Sig under failures. As seen in the figure, under the failure of the responsible node, the latency of XRBC-Sig is around 30% higher than that in the failure-free case.

VIII. Conclusion

We propose a new primitive called cross-consensus reliable broadcast (XRBC). We present three XRBC protocols under different assumptions and show their applications in sharding-based protocols and cross-chain bridges. Our experimental results show that all of our protocols achieve low latency and decent throughput.

Acknowledgment

This work was supported in part by the National Key R&D Program of China under 2022YFB2701700, the National Natural Science Foundation of China under 92267203 and 62272043, Beijing Natural Science Foundation under M23015, Yangtze Delta Region Institute of Tsinghua University, Zhejiang (No. LZZLX24F007), WeBank scholars program, and Tsinghua Shuimu Scholar program.

Ethics Considerations

This research is committed to the principles of research ethics. In particular, we adhere to the following principles:

- **Respect for persons**: No personal data was used in our research.
- **Beneficence**: We propose a new XRBC primitive and three concrete constructions: XRBC-woA, XRBC-wA, and XRBC-Sig. We show that these protocols benefit both sharding-based protocols and cross-chain bridge.
- **Justice**: We prove the correctness of all of our protocols. Any user's benefits are equally protected if the protocols correctly implemented.
- **Respect for law and public interest**: Our research complies with all relevant laws and regulations. Our methods and results are transparent, with no aspects concealed that might violate existing laws.

References

[1] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to reliable and secure distributed programming*. Springer Science & Business Media, 2011.

[2] M. Burrows, "The chubby lock service for loosely-coupled distributed systems," in *OSDI*, 2006, pp. 335–350.

[3] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems." in *USENIX annual technical conference*, vol. 8, no. 9. Boston, MA, USA, 2010.

[4] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild *et al.*, "Spanner: Google's globally distributed database," *TOCS*, vol. 31, no. 3, pp. 1–22, 2013.

[5] M. Štefanko, O. Chaloupka, B. Rossi, M. van Sinderen, and L. Maciaszek, "The saga pattern in a reactive microservices environment," in *ICSOFT 2019*, 2019, pp. 483–490.

[6] R. C. Aksoy and M. Kapritsos, "Aegean: replication beyond the client-server model," in *SOSP*, 2019, pp. 385–398.

[7] R. Frank, M. Murray, S. Gupta, E. Xu, N. Crooks, and M. Kapritsos, "Picsou: Enabling efficient cross-consensus communication," in *OSDI*, 2025.

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

[9] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: A fast blockchain protocol via full sharding," in *CCS*, 2018, pp. 931–948.

[10] Z. Hong, S. Guo, P. Li, and W. Chen, "Pyramid: A layered sharding blockchain system," in *INFOCOM*, 2021, pp. 1–10.

[11] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, and G. Danezis, "Chainspace: A sharded smart contracts platform," in *NDSS*, 2018.

[12] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger," in *S&P*, 2018, p. 406.

[13] M. Li, Y. Lin, J. Zhang, and W. Wang, "Cochain: High concurrency blockchain sharding via consensus on consensus," in *INFOCOM*. IEEE, 2023, pp. 1–10.

[14] B. David, B. Magri, C. Matt, J. B. Nielsen, and D. Tschudi, "Gearbox: Optimal-size shard committees by leveraging the safety-liveness dichotomy," in *CCS*, 2022, pp. 683–696.

[15] A. Augusto, R. Belchior, M. Correia, A. Vasconcelos, L. Zhang, and T. Hardjono, "Sok: Security and privacy of blockchain interoperability," in *S&P*, 2024, pp. 234–234.

[16] P. Team, "Polynetwork: An interoperability protocol for heterogeneous blockchains," 2020.

[17] J. Kwon and E. Buchman, "Cosmos whitepaper," *A Netw. Distrib. Ledgers*, vol. 27, pp. 1–32, 2019.

[18] G. Wood, "Polkadot: Vision for a heterogeneous multi-chain framework," *White paper*, vol. 21, no. 2327, p. 4662, 2016.

[19] Y. Xu, J. Zheng, B. Düdder, T. Slaats, and Y. Zhou, "A two-layer blockchain sharding protocol leveraging safety and liveness for enhanced performance," in *NDSS*, 2024.

[20] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols," in *Annual International Cryptology Conference*. Springer, 2001, pp. 524–541.

[21] G. Bracha, "An asynchronous [(n-1)/3]-resilient consensus protocol," in *PODC*. ACM, 1984, pp. 154–162.

[22] ——, "Asynchronous byzantine agreement protocols," *Information and Computation*, vol. 75, no. 2, pp. 130–143, 1987.

[23] N. Alhaddad, S. Das, S. Duan, L. Ren, Z. X. Mayank Varia, and H. Zhang, "Balanced byzantine reliable broadcast with near-optimal communication and improved computation," in *PODC*, 2022.

[24] M. Ben-Or, B. Kelmer, and T. Rabin, "Asynchronous secure computations with optimal resilience," in *PODC*. ACM, 1994, pp. 183–192.

[25] S. Duan, X. Wang, and H. Zhang, "Practical signature-free asynchronous common subset in constant time," in *CCS*, 2023.

[26] P. Sheng, X. Wang, S. Kannan, K. Nayak, and P. Viswanath, "Trustboost: Boosting trust among interoperable blockchains," in *CCS*, 2023, pp. 1571–1584.

[27] C. Cachin and S. Tessaro, "Asynchronous verifiable information dispersal," in *SRDS*. IEEE, 2005, pp. 191–201.

[28] S. Das, Z. Xiang, and L. Ren, "Asynchronous data dissemination and its applications," in *CCS*, 2021.

[29] S. Blackshear, A. Chursin, G. Danezis, A. Kichidis, L. Kokoris-Kogias, X. Li, M. Logan, A. Menon, T. Nowacki, A. Sonnino *et al.*, "Sui lutris: A blockchain combining broadcast and consensus," in *CCS*, 2024, pp. 2606–2620.

[30] H. Zhang and S. Duan, "Pace: Fully parallelizable bft from reproposable byzantine agreement," in *CCS*, 2022.

[31] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of BFT protocols," in *ACM CCS*, 2016, pp. 31–42.

[32] D. Dolev and H. R. Strong, "Authenticated algorithms for byzantine agreement," *SIAM Journal on Computing*, vol. 12, no. 4, pp. 656–666, 1983.

[33] I. Abraham, K. Nayak, L. Ren, and Z. Xiang, "Good-case latency of byzantine broadcast: A complete categorization," in *PODC*, 2021.

[34] A. Momose and L. Ren, "Optimal communication complexity of authenticated byzantine agreement," in *DISC*, 2021.

[35] "Cross-chain by chainlink," https://chain.link/cross-chain, 2025.

[36] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "Sok: Layer-two blockchain protocols," in *FC*. Springer, 2020, pp. 201–226.

[37] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón, and P. Wuille, "Enabling blockchain innovations with pegged sidechains," 2014.

[38] A. Singh, K. Click, R. M. Parizi, Q. Zhang, A. Dehghantanha, and K.-K. R. Choo, "Sidechain technologies in blockchain networks: An examination and state-of-the-art review," *Journal of Network and Computer Applications*, vol. 149, p. 102471, 2020.

[39] V. Buterin, "Chain interoperability," *R3 research paper*, vol. 9, pp. 1–25, 2016.

[40] S. Das, S. Duan, S. Liu, A. Momose, L. Ren, and V. Shoup, "Asynchronous consensus without trusted setup or public-key cryptography," in *CCS*, 2024.

[41] Y. Lu, Z. Lu, Q. Tang, and G. Wang, "Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited," in *PODC*, 2020.

[42] S. Duan, M. K. Reiter, and H. Zhang, "Beat: Asynchronous bft made practical," in *CCS*. ACM, 2018, pp. 2028–2041.

[43] B. Guo, Y. Lu, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Speeding dumbo: Pushing asynchronous bft closer to practice," *NDSS*, 2022.

[44] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *SOSP*. ACM, 2017, pp. 51–68.

[45] X. Wang, H. Wang, H. Zhang, and S. Duan, "Pando: Extremely scalable bft based on committee sampling," in *NDSS*, 2026.

[46] I. Abraham, T. H. Chan, D. Dolev, K. Nayak, R. Pass, L. Ren, and E. Shi, "Communication complexity of byzantine agreement, revisited," in *PODC*, 2019, pp. 317–326.

[47] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography," *Journal of Cryptology*, vol. 18, no. 3, pp. 219–246, 2005.

[48] P. A. Bernstein, V. Hadzilacos, N. Goodman *et al.*, *Concurrency control and recovery in database systems*. Addison-wesley Reading, 1987, vol. 370.

[49] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in *PODC*, 2019.

[50] M. J. Amiri, C. Wu, D. Agrawal, A. El Abbadi, B. T. Loo, and M. Sadoghi, "The bedrock of byzantine fault tolerance: A unified platform for BFT protocols analysis, implementation, and experimentation," in *NSDI*, 2024, pp. 371–400.

[51] X. Sui, S. Duan, and H. Zhang, "Marlin: Two-phase BFT with linearity," *DSN*, 2022.

[52] N. Giridharan, F. Suri-Payer, M. Ding, H. Howard, I. Abraham, and N. Crooks, "Beegees: Stayin' alive in chained BFT," in *PODC*. ACM, 2023, pp. 233–243.

[53] T. Xie, K. Gai, L. Zhu, S. Wang, and Z. Zhang, "Rac-chain: An asynchronous consensus-based cross-chain approach to scalable blockchain for metaverse," *ACM Transactions on Multimedia Computing, Communications and Applications*, vol. 20, no. 187, pp. 1–24, 2024.

Appendix A
Discussion

XRBC-woA vs. XRBC-wA. The advantage of XRBC-woA is that it always achieves $O(1)$ time, and the communication or message complexities do not depend on ABC in $\mathbb{B}$. Later, we show in Sec. VI that XRBC-woA has a unique application for cross-shard coordination. Meanwhile, the advantage of XRBC-wA is that if nodes in $\mathbb{B}$ execute an ABC protocol anyway, we do not need to implement RA on top of it. Namely, we can reuse the ABC implementation to process cross-consensus messages. This makes XRBC-wA a perfect solution for sharding protocols that handle cross-shard transactions.

Improving XRBC protocols using techniques of reliable broadcast (RBC). Since our XRBC primitive extends the notion of reliable broadcast, the communication of our protocols can be optimized using similar optimization techniques for reliable broadcast [23], [28]. For example, we can use erasure coding or online error correction code to improve the communication complexity of XRBC-woA and XRBC-wA by lowering the $O(mnL)$ term.

Appendix B
Additional Related Work

Cross-chain bridge. Many cross-chain solutions involve cross-consensus communication. One example is the relay chain, where a dedicated blockchain system is used to coordinate cross-chain transactions [16], [17], [18]. As the cross-chain transactions are *coordinated* by a blockchain that is already safe (i.e., no double spending) and live (i.e., transactions will eventually be processed), relay chains can handle a large volume of cross-chain transactions to ensure the atomicity of cross-chain transactions. Industrial examples include Cosmos [17], Polkadot [18], and CCIP by Chainlink [35]. Another example is the *sidechain* solutions for cross-chain transactions. A sidechain is a parallel chain to a blockchain (called mainchain), often used to improve the performance of the system [36]. While the mainchain does not need to be aware of the existence of the sidechain, a sidechain can be used for cross-chain transactions between two chains, using the so-called *two-way peg* solution [37], [38]. The technical report of R3 corda [39] mentions that two blockchains can be used as sidechains for each other to build a two-way peg. In both relay chain and sidechain-based solutions, communication between groups of nodes is involved. In this work, we present a solution that uses our XRBC protocol to build a lightweight cross-chain bridge. Our solution can be viewed as a two-way peg where two blockchains directly communicate with each other (serving as the sidechains of each other).

Appendix C
Analysis of the Complexities

Assuming N is the total number of nodes, the state-of-the-art BA protocol known so far achieves $O(\kappa N^2)$ communication under the trusted setup assumption [33] and $O(\kappa N^2+N^3)$ under the PKI assumption [32]. We now analyze the communication complexity of the combined $(\Delta+\delta)$-BB protocol used in Reticulum and our synchronous cross-shard coordination protocol.

Theorem 1. *The combined $(\Delta+\delta)$-BB protocol presented in Sec. VI-A achieves $O(m\kappa(m+n)^2)$ under the trusted setup assumption and $O(\kappa(m+n)^3)$ under the PKI assumption, where the length of input of each sender P_s^{A} is the same as the security parameter κ.*

Proof. In the "propose" phase, every node in A sends a value of length κ to A and B.

In the "vote" phase, under the trusted PKI assumption, each vote in the "vote" phase on a pair (v,σ) is a partial signature on the hash of (v,σ). In the "commit and lock" phase, the combined vote (of $f+1$ votes) is a threshold signature of length $O(\kappa)$. Under the PKI assumption, each vote in the "vote" phase still has length κ, but the combined $f+1$ votes now have length $\kappa(m+n)$.

Meanwhile, in the "vote" phase, every node needs to send all the received m values in the "propose" phase (the length of which is $m\kappa$).

Therefore, under the trusted PKI setting, the communication complexity is:

$$\begin{aligned}&\underbrace{\sum_{i=1}^{m} O(\kappa(m+n))}_{\text{propose}}+\underbrace{\sum_{i=1}^{m+n} O(m\kappa(m+n))}_{\text{vote}}\\&+\underbrace{\sum_{i=1}^{m+n} O(\kappa(m+n))}_{\text{commit and lock}}+\underbrace{O(\kappa(m+n)^2)}_{\text{Byzantine agreement}}\\=&O(m\kappa(m+n)^2)\end{aligned}\tag{1}$$

Under the PKI setting, the communication complexity is:

$$\begin{aligned}&\underbrace{\sum_{i=1}^{m} O(\kappa(m+n))}_{\text{propose}}+\underbrace{\sum_{i=1}^{m+n} O(m\kappa(m+n))}_{\text{vote}}\\&+\underbrace{\sum_{i=1}^{m+n} O(\kappa(m+n)(m+n))}_{\text{commit and lock}}+\underbrace{O(\kappa(m+n)^2+(m+n)^3}_{\text{Byzantine agreement}}\\=&O(\kappa(m+n)^3)\end{aligned}\tag{2}$$

□

Theorem 2. *Our synchronous cross-shard coordination protocol achieves $O(\kappa mn+\kappa n^2)$ communication under the trusted setup assumption and $O(\kappa mn+\kappa n^2+n^3)$ under the PKI assumption.*

Proof. In the "cross" phase, every node in A sends its value v (length κ) to all nodes in B. In the "notify" phase, every node in B sends a value v (length κ) to all nodes in A.

Therefore, under the trusted PKI setting, the communication complexity is:

$$\begin{aligned}&\underbrace{\sum_{i=1}^{m} O(\kappa n)}_{\text{cross}}+\underbrace{O(\kappa n^2)}_{\text{Byzantine agreement}}+\underbrace{\sum_{i=1}^{n} O(\kappa m)}_{\text{notify}}\\=&O(\kappa mn+\kappa n^2)\end{aligned}\tag{3}$$

Under the PKI setting, the communication complexity is:

$$\begin{aligned}&\underbrace{\sum_{i=1}^{m} O(\kappa n)}_{\text{cross}}+\underbrace{O(\kappa n^2+n^3)}_{\text{Byzantine agreement}}+\underbrace{\sum_{i=1}^{n} O(\kappa m)}_{\text{notify}}\\=&O(\kappa mn+\kappa n^2+n^3)\end{aligned}\tag{4}$$

□

Appendix D
Cross-Chain Bridge

As mentioned previously, cross-chain bridge is a solution for achieving blockchain interoperability [15]. Most existing solutions assume a partially synchronous model [53]. In this section, we present an efficient and asynchronous cross-chain bridge scheme built from XRBC-Sig. Our approach is compatible with any blockchains that use conventional ABC as its consensus mechanism. Additionally, our solution can also be viewed as a two-way peg [37], [38], where each chain is the sidechain of the other chain. For any cross-chain transactions, one chain is the *source* chain and the other is the *target* chain.

As shown in Fig. 11, we use XRBC-Sig as a middleware. The middleware is coupled with the *client* role (that submits the transactions to the system) of both the source chain and the target chain. We deploy the middleware on *every* replica

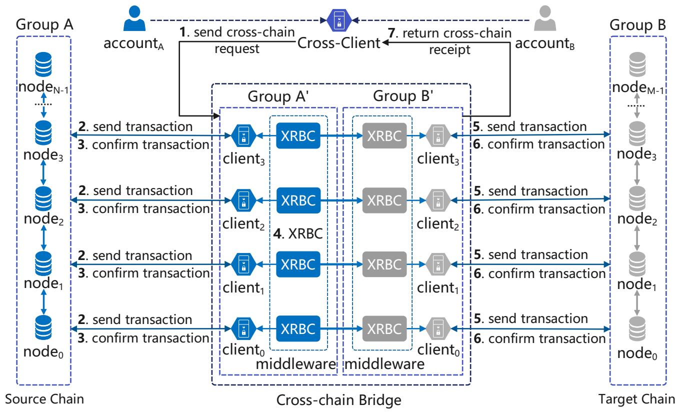


Fig. 11: A cross-chain bridge scheme from our XRBC-Sig protocol.

of both chains. Let A be the source chain and B be the target chain. The protocol roughly works as follow: 1) The middleware at A listens to the results of the source chain via the *client* module; 2) After a cross-chain transaction is created, the middleware queries XRBC-Sig and then sends the transaction to the XRBC-Sig module of replicas at B; 3) After the middleware at B receives the cross-chain transaction, it processes the transaction via the XRBC-Sig protocol. Whenever ABC at B needs to be queried, the middleware forwards the transaction to the *client* module of the target chain B. Finally, the transaction is *x-delivered* according to the protocol.

Our solution ensures the atomicity of cross-chain transactions. Namely, as long as a valid cross-chain transaction is created, the safety and integrity property of XRBC ensure that correct replicas in the target chain deliver the cross-chain transactions in the same order and the termination property ensures that the transaction will eventually be delivered by the target chain. In fact, our solution can be further extended to achieve chain unlinkability and cross-chain confidentiality [15] by integrating our solution with other cryptographic primitives.

As we will later show in our evaluation, our solution is highly efficient, thanks to the natural advantage of two-way peg. For instance, as only the source chain and the target chain are involved, the latency is already near-optimal.

Appendix E Proof of our Group-PRBC Protocol

Theorem 3. *Our Group-PRBC protocol (without the highlighted procedures in Fig. 2) satisfies validity, agreement, and integrity of Group-PRBC.*

Proof. After every correct node P_i^{A} *pr-broadcasts* v_i, it *r-broadcasts* v_i to all nodes in B using RBC. According to the validity property of RBC, validity of Group-PRBC holds. Similarly, according to the agreement property and integrity property of RBC, agreement and integrity of Group-PRBC hold. □

Theorem 4. *Our Group-PRBC protocol (with the highlighted procedures in Fig. 2) satisfies validity, agreement, termination, and modified integrity of Group-PRBC.*

Proof. Validity and agreement hold as every correct node P_i^{A} *r-broadcasts* its *pr-broadcast* value v_i. We now prove termination and modified integrity.

Termination. If all correct nodes in A *pr-broadcast* some value, according to the validity property of RBC, any correct node P_i^{B} in B will *r-deliver* some value in each RBC instance started by a correct node in A. As there are $m-t$ correct nodes in A, $|W_i| \geq m-t$ eventually holds for P_i^{B}. According to the protocol, every correct node P_i^{B} then *mvba-proposes* W_i to $\mathsf{MVBA_B}$.

We now show that the predicate $Q(W)$ eventually holds if W is *mvba-proposed* by a correct node. First, every correct node P_j^{B} *mvba-proposes* W after $|\boldsymbol{o}_j| \geq m-t$. Therefore, there exist at least $m-t$ ℓ such that $W[\ell]=1$. Second, if a correct node P_j^{B} sets $W[\ell]=1$, it has *r-delivered* some value in RBC_ℓ. The agreement property of RBC ensures that every correct node P_i^{B} eventually also *r-delivers* some value in RBC_ℓ and then sets $W_i[\ell] \leftarrow 1$. According to the termination property of MVBA, every correct eventually *mvba-decides*.

According to our protocol, each correct node waits until it *r-delivers* some value in RBC_ℓ and sets $\boldsymbol{o}[\ell] \neq \perp$. According to the external validity property of MVBA, we know that $Q(W)$ holds for P_i^{B}. According to the definition of the predicate, we know that $\boldsymbol{o}[\ell]$ eventually becomes non-$\perp$ in the above case. Every correct node eventually *pr-delivers* some value. Additionally, the predicate also requires that there exists at least $m-t$ ℓ such that $W[\ell]=1$. Accordingly, $|\boldsymbol{v}| \geq m-t$.

Modified integrity. According to the protocol, every correct node only *pr-delivers* once. If a node *pr-delivers* $\boldsymbol{v}[s] \neq \perp$, it has *r-delivered* some value in RBC_s. The integrity property of RBC ensures that $\boldsymbol{v}[s]$ was previously *r-broadcast* by P_s^{A}. □

Appendix F Proof of our XRBC Protocols

A. Proof of XRBC-woA

Theorem 5. *Our XRBC-woA protocol (Fig. 4) satisfies safety, termination, and integrity of XRBC.*

Proof. **Safety.** According to the protocol, every correct node completes the protocol for an epoch r before starting epoch $r+1$. As the messages *x-broadcast* by A are not repetitive, we show that if a correct node P_i^{B} *x-delivers* v in epoch r, a correct node P_i^{B} *x-delivers* v', $v=v'$. Namely, if $v \neq v'$, P_i^{B} *r-decides* v and P_j^{B} *r-decides* v', violating the agreement property of RA. Then, it is not hard to see that safety holds.

Termination. If every correct node in A starts epoch r, they all send $(\textsc{Cross}, r, v)$ for the same v (according to the assumption of XRBC). Therefore, any correct node P_i^{B} eventually receives $t+1$ matching $(\textsc{Cross})$ messages and then *r-proposes* v to $\mathsf{RA}_{r,\mathsf{B}}$. No correct node will *r-propose* some $v' \neq v$, as there are t faulty nodes in A and correct nodes send $(\textsc{Cross}, r, v)$.

According to the validity property or RA, all correct nodes *r-decide* v.

Integrity. According to the protocol, every correct node only *x-delivers* once in an epoch r. If a correct node P_i^{B} *x-delivers* v in epoch r, it has *r-decided* v in $\mathsf{RA}_{r,\mathsf{B}}$. According to the integrity property of RA, at least $n - 2f$ correct nodes *r-propose* v. As every correct node *r-proposes* v upon receiving $t+1$ $(\textsc{Cross}, r, v)$ messages and there are at most t faulty nodes in A, integrity of XRBC holds. □

B. Proof of XRBC-wA

Theorem 6. *Our XRBC-wA protocol with ABC in* B *(Fig. 5) satisfies safety, termination, and integrity of XRBC.*

Proof. **Safety.** According to the protocol, every correct node *x-delivers* v in epoch r after it *a-delivers* v. It will starts epoch $r+1$ before epoch r ends. Towards a contradiction, assume that a correct node P_i^{B} *x-delivers* v before *x-delivering* v' and another correct node P_j^{B} *x-delivers* v' before *x-delivering* v. According to the protocol, P_i^{B} *a-delivers* v before *a-delivering* v' and another correct node P_j^{B} *a-delivers* v' before *a-delivering* v, violating the safety property of ABC.

Termination. If every correct node in A starts epoch r, they all send $(\textsc{Cross}, r, v)$ for the same v (according to the assumption of XRBC). Therefore, any correct node P_i^{B} eventually receives $t+1$ matching $(\textsc{Cross})$ messages and then *a-broadcasts* v to ABC. According to the liveness property of ABC, all correct nodes eventually *a-deliver* v and then *x-deliver* v in epoch r.

Integrity. According to the protocol, every correct node only *x-delivers* once in an epoch r. If a correct node P_i^{B} *x-delivers* v in epoch r, it has *a-delivered* v in ABC. According to the predicate we set up for ABC, every correct node considers a *a-broadcast* value v valid after receiving $t+1$ $(\textsc{Cross}, r, v)$ messages. It is not too hard to see that at least one correct node in B has previously received $t+1$ $(\textsc{Cross}, r, v)$ messages. As there are at most t faulty nodes in A, v was previously *x-broadcast* by correct nodes in A.

□

C. Proof of XRBC-Sig

Theorem 7. *Our XRBC-Sig (Fig. 6) satisfies safety, termination, and integrity of XRBC.*

Proof. **Safety.** According to the protocol, every node P_i^{B} *x-delivers* some value in epoch r after (r', v') has been *x-delivered* for any $r' < r$. Towards a contradiction, assume that a correct node P_i^{B} *x-delivers* v before *x-delivering* v' and another correct node P_j^{B} *x-delivers* v' before *x-delivering* v. According to the protocol, P_i^{B} *a-delivers* v before *a-delivering* v' and another correct node P_j^{B} *a-delivers* v' before *a-delivering* v, violating the safety property of ABC.

Termination. If all correct nodes in A *x-broadcast* v for epoch r, a responsible node P_k^{A} will send a $(\textsc{Cross})$ message. There are two cases: 1) P_k^{A} is correct; 2) P_k^{A} is faulty. If P_k^{A} is correct, it will send a valid $(\textsc{Cross}, r, v, \pi, pp)$ message to all nodes in B. Every correct node in B will reply with a $(\textsc{Rep})$ message with a digital signature. As there are at least $n - f$ correct nodes in B, P_k^{A} eventually receives $n - f$ $(\textsc{Rep})$ messages and then sends a $(\textsc{Confirm})$ messages to both A and B. Accordingly, any correct node in A sets $Cer[r]$ as a non-$\perp$ value and any correct node B *x-delivers* (r, v).

If P_k^{A} is faulty, there are two different cases for B: 1) some correct node in B receives a valid $(\textsc{Confirm}, r, h, \sigma)$ message and a $(\textsc{Cross}, r, v, \pi)$ message such that $\mathsf{hash}(v) = h$; 2) no correct node in B receives a valid $(\textsc{Confirm})$ message. There are also two cases for A: 3) some correct node in A receives a valid $(\textsc{Confirm})$ message; 4) no correct node in A receives a valid $(\textsc{Confirm})$ message. We now show that for any combinations of the cases above, any correct node in B eventually *x-delivers* v.

Case 1) and 3). If some correct node in B receives a valid $(\textsc{Confirm}, r, h, \sigma)$ message and a $(\textsc{Cross}, r, v, \pi)$ message such that $\mathsf{hash}(v) = h$, the node will *a-broadcast* (r, v). According to the liveness property of ABC, correct nodes in B eventually *a-deliver* (r, v) and then *r-deliver* v.

Case 1) and 4). Same as above.

Case 2) and 3). If some correct node P_i^{A} receives a $(\textsc{Confirm})$ message in epoch r, it sets its $Cer[r]$ as a non-$\perp$ value. Without loss of generality, we discuss the case where P_i^{A} is the responsible for epoch $r+1$. The case where P_i^{A} is not responsible for epoch $r+1$ is the same as the combination of case 2) and 4) and we omit the detail here. As $Cer[r] \neq \perp$ for P_i^{A}, P_i^{A} only sends a $(\textsc{Cross}, r+1, v, \pi, \perp)$ message to B. In this case, correct nodes in B will send a $(\textsc{Fetch})$ message to P_i^{A}. Accordingly, P_i^{A} will send a $(\textsc{Catchup})$ message to all nodes in B. It is then not difficult to see that correct nodes will *a-broadcast* v. According to the liveness property of ABC, all correct nodes *x-deliver* v.

Case 2) and 4). Nodes will proceed to epoch $r+1$ in this case. If the responsible node P_i^{A} in epoch $r+1$ is correct, it will send the cross-consensus message for epoch r together with the proof of delivery in its $(\textsc{Cross})$ message. According to the protocol, correct nodes in B will parse the pp field as (r, v, π) and then *a-broadcast* v. According to the liveness property of ABC, all correct nodes *x-deliver* v. As there are $n - f$ correct nodes in B, it is not difficult to see that eventually some correct responsible node will ensure that the above happens so correct nodes eventually *x-deliver* v.

Integrity. According to the protocol, every correct node only *x-delivers* once in an epoch r. If a correct node P_i^{B} *x-delivers* v in epoch r, it has *a-delivered* v in ABC. According to the protocol, every correct node considers a *a-broadcast* value v valid if it has previously received a proof of delivery for v. Then, v was previously *x-broadcast* by correct nodes in A. □

Appendix G
Artifact Appendix

We propose a new primitive called cross-consensus reliable broadcast (XRBC). The XRBC primitive models the security properties of communication between two groups, where at least one group executes a consensus protocol. Our experimental results show that all of our protocols achieve low latency and decent throughput.

This artifact demonstrates how to reproduce the results presented in of Section VII of our paper. The experiments do not require any specialized hardware. Our test environment is a computer equipped with a 4-core CPU, 16 GB of RAM, 100 GB of storage, a 100 Mbps network connection, and the Linux operating system.

Note that all results reported in our paper require access to the Amazon EC2. In this artifact appendix, we present the workflow to reproduce scaled-down experiments using one machine. For readers who are interested in reproducing the results on EC2, please refer to the `https://doi.org/10.5281/zenodo.16945739` directory of our repository.

A. Description & Requirements

1) How to access:

1. Download the repository from `https://doi.org/10.5281/zenodo.16945739` (click the button "Download" and decompress "/xrbc-ae-25-revised.zip" manually or follow the command:
```
unzip xrbc-ae-25-revised.zip -d xrbc-ae-25
```
2. Enter the cloned repository directory (denoted as `$HOME`) and adding the execute permission.
```
cd xrbc-ae-25
```
3. Download all dependencies using scripts
```
bash autoEnv.bash
```

2) Hardware dependencies: The experiments do not require any specialized hardware. The experiments use only commodity hardware: a 4-core CPU, 16 GB RAM, 100 GB storage, and 100 Mbps network. We recommend using the Linux operating system.

3) Software dependencies: None.

4) Benchmarks: None.

B. Artifact Installation & Configuration

The artifact can be accessed by downloading the repository from a stable link. All the scripts, source codes, and sample output files can be accessed via the stable URL: https://doi.org/10.5281/zenodo.16945739.

C. Experiment Workflow

To simplify replication, we provide three test scripts—`autoE1.bash`, `autoE2.bash`, and `autoE3.bash`—that can be used to reproduce Claims C1, C2, and C3, respectively. The workflow proceeds as follows.

1. Download the repository from https://doi.org/10.5281/zenodo.16945739 and unzip the repository by:
```
unzip xrbc-ae-25-revised.zip -d xrbc-ae-25
cd xrbc-ae-25
bash autoEnv.bash
```
Upon successful installation, the terminal will display the log shown below
```
[SUCCESS] Installation verification completed
```
2. Run the experiments sequentially: E1, then E2, then E3.
```
bash autoE1.bash
bash autoE2.bash
bash autoE3.bash
```
3. Once all server and client nodes are running, the corresponding logs will appear in the terminal.
```
[INFO] Starting server node 35
2025/07/16 14:52:52 **Starting replica 35**
2025/07/16 14:52:52 Starting sender 35
2025/07/16 14:52:52 starting connection manager
2025/07/16 14:52:52 ready to listen to port
[SUCCESS] All server nodes have been started
[INFO] Waiting for server nodes to fully
start...
[INFO] Step 4: Starting client node [INFO]
Starting client: ./client 100 0 1 100
[SUCCESS] Client node has been started (PID:
29253)
```
4. After executing each experiment, inspect the per-node log located at
`$HOME:/var/log/[nodeid]/[ymd]_Eva.log`.
The log file is as follows.
```
2025/07/16 14:29:12 10, 1, 1, 10496: 79-10417, 0
```
5. The script automatically extracts all records from the log, computes average latency, and writes the results to the corresponding output files. An example of the E1 log is shown below.
```
[INFO] Processing results...
[INFO] Parsing log files from directories 4 to
35 for date: 20250716
[INFO] Result file will be written to:
../resultE1.txt
[SUCCESS] Results written to ../resultE1.txt
[INFO] E1: Number Of Nodes: 31, Number Of Nodes
(Group A): 4, Number of Request: 1, Average
Total Latency(ms): 11440, Average Cross Latency
(ms): 1135, Average Consensus in B Latency:
10305
[INFO] Averages calculated from 31 log files
[INFO] Test completed, cleaning up...
```
6. The results are written to `$HOME/resultE1.txt`, `$HOME/resultE2.txt`, and `$HOME/resultE3.txt`, respectively. An example of the E1 output is as follows.
```
E1:
Number Of Nodes (Group B): 31
Number Of Nodes (Group A): 4
Number of Request: 1
Average Total Latency(ms): 11440
Average Cross Latency (ms): 1135
Average Consensus in B Latency: 10305
```

D. Major Claims

- (C1): The main overhead of our XRBC protocol is incurred by the consensus in group B. This claim is validated by experiment (E1), which reproduces the results reported in Fig. 8d in the paper.

- (C2): Our XRBC protocol achieves higher performance when the number of nodes in Group A (i.e., m) is smaller than the number of nodes in Group B (i.e., n). **Note**: this claim is not included in our original submission. In response to the reviewers' suggestions, we have explicitly incorporated this claim into both our rebuttal and the revised manuscript.
- (C3): The latency of XRBC grows in a controlled and predictable manner as the number of faulty-tolerated nodes (i.e., as Group B's size and the fault bound f increase), with bounded per-node incremental overhead. This observation is substantiated by experiment (E3), which reproduces the results presented in Fig. 8c of the paper.

E. Evaluation

We conduct three types of experiments: XRBC latency breakdown (E1), latency comparison when $m < n$ (E2), and latency degradation as f grows (E3).

1) Experiment (E1): [10 computer-minute] We break down the latency of XRBC into two phases: *Cross* and *Consensus in B*. This experiment reproduces the results with $f = 10, t = 1, b = 1$, the same parameters used in Fig. 8d of the paper.

[How to:] run the experiment with the command under `$HOME` *:*

```
bash autoE1.bash
```

After the total experiment is completed, all data are reported in

```
$HOME/resultE1.txt
```

[Results] From the E1 test results (`resultE1.txt`), Consensus in B accounts for 90 % of total latency, while Cross adds only around 10 %, confirming Claim 1: Consensus in B dominates XRBC latency.

```
E1:
Number Of Nodes (Group B): 31
Number Of Nodes (Group A): 4
Number of Request: 1
Average Total Latency(ms): 11440
Average Cross Latency (ms): 1135
Average Consensus in B Latency: 10305
```

2) Experiment (E2): [10 computer-minute] To confirm that fewer Group-A nodes improve performance, we run E2 by setting the size of Group A as $m = 16$ and $m = 4$. Meanwhile, the size of Group B is $n = 31$.

[How to:] run the experiment with the command under `$HOME` *:* `bash autoE2.bash`

After the experiment is completed, all data are reported in

```
$HOME/resultE2.txt
```

[Results] The E2 results (`resultE2.txt`) show that lowering the size of Group A from 16 to 4 nodes improves the latency of the Cross phase by nearly a factor of three, thus validating Claim 2 that XRBC achieves better performance when $m<n$.

```
E2:
Number Of Nodes (Group B): 31
Number of Request: 1
Average Cross Latency (ms) when Number of Nodes
(Group A) is 16: 3094
Average Cross Latency (ms) when Number of Nodes
(Group A) is 4: 1135
```

3) Experiment (E3): [10 computer-minute] We rerun the experiment with $n = 31$ and $n = 91$, where n is the node count in Group B, to verify that latency degrades only marginally as n grows. Other parameters are fixed at $m = 4, b = 1$. which are the same configuration settings as Fig. 8c of the paper.

[How to:] run the experiment with the command under `$HOME` *:*

```
bash autoE3.bash
```

After the experiment is completed, all data are reported in

```
$HOME/resultE3.txt
```

[Results]

The E3 results (`resultE3.txt`) show that scaling Group B from 31 to 91 nodes increases XRBC latency in a controlled, predictable manner. When all nodes run on a single machine, the performance of the protocol depends on the hardware. Once the resources of the hardware is saturated, each added node contributes a limited and stable amount of extra latency, so total latency grows without superlinear escalation. Although absolute latencies vary with hardware, the same bounded per-node incremental pattern appears across runs (see `resultE3.txt`), matching the qualitative trend in Fig. 8c and supporting Claim C3.

```
E3:
Number of Nodes (Group A): 4
Number of Request: 1
Average Total Latency(ms) when Number of Nodes
(Group B) is 91: 16759
Average Total Latency(ms) when Number of Nodes
(Group B) is 31: 11440
```

F. Notes

If one uses IDE to run the scripts, executing `autoE1.bash`, `autoE2.bash`, and `autoE3.bash` may freeze the IDE due to the large computer resources taken to launch the experiments. After starting the bash script, wait for about two minutes. If the IDE is not responsible, please close the IDE and restart the experiments. As mentioned above, the data log in `resultE1.txt`, `resultE2.txt`, `resultE3.txt` is updated if the experiments are successfully launched.