APPROVAL SHEET

Title of Dissertation: Building Practical, Efficient, and Reliable Fault-tolerant Distributed Systems

Name of Candidate: Chao Liu
Doctor of Philosophy, 2021

Dissertation and Abstract Approved:

Alan T. Sherman
Professor
Department of Computer Science and Electrical Engineering

Date Approved: _____

NOTE: *The Approval Sheet with the original signature must accompany the thesis or dissertation. No terminal punctuation is to be used.

ABSTRACT

Title of dissertation: BUILDING PRACTICAL, EFFICIENT, AND RELIABLE FAULT-TOLERANT DISTRIBUTED SYSTEMS

Chao Liu, Doctor of Philosophy, 2021

Dissertation directed by: Professor Alan T. Sherman
Computer Science and Electrical Engineering

Building a real-world distributed system should consider a range of fundamental design objectives, including fault tolerance, reliability, performance, and scalability. Modern distributed systems require tolerance to any kind of service disruption, whether caused by a simple hardware fault or by a large-scale disaster. Famous systems like ZooKeeper, Google file system (GFS) and Bigtable are designed to tolerate benign faults. Byzantine fault-tolerant (BFT) state machine replication (SMR) is regarded as an ideal candidate that can tolerate arbitrary faulty behaviors, and can be applied to multiple real-world systems. BFT SMR as one of consensus protocols is the core of blockchain to provide agreement services, and its efficiency highly affects the performance of a blockchain system. This dissertation presents three fault-tolerant distributed systems by leveraging novel BFT protocols, practical cryptographic schemes and libraries, efficient and scalable system designs, modern programming language, and complete and detailed evaluations and deployments. Besides fault tolerance, this dissertation also presents different approaches to build distributed systems toward high performance, scalability, and usability.

In this work, we first focus on constructing distributed systems with asynchronous BFT protocol. Asynchronous BFT protocols such as HoneyBadgerBFT and BEAT can achieve only static security. Unfortunately, the weaker static model of security does not capture the power of several types of attackers. We develop two protocols EPIC and HALE to defend against adaptive corruption, where the adversary can corrupt the replicas at any moment during the execution of the protocol. Meanwhile, when there is no contention or contention is rare among correct replicas, it is necessary for correct replicas to terminate fast so that performance can be improved. We develop MiB, a novel and efficient asynchronous BFT framework using new distributed system constructions as building blocks. We also systematically carried out experiments for asynchronous BFT protocols with failures and evaluated their performance in various failure scenarios. Finally, we present Chios, an intrusion-tolerant publish/subscribe system which protects against Byzantine failures. Our publish/subscribe system achieves decentralized confidentiality with fine-grained access control and strong publication order guarantees.

BUILDING PRACTICAL, EFFICIENT, AND RELIABLE FAULT-TOLERANT DISTRIBUTED SYSTEMS

by

Chao Liu

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, Baltimore County in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2021

Advisory Committee:
Professor Alan T. Sherman, Chair/Advisor
Dr. Haibin Zhang, Co-advisor
Dr. Sisi Duan, Co-advisor
Professor Mohamed Younis
Associate Professor Ting Zhu

Acknowledgments

First and foremost, I would like to thank my advisor, Prof. Alan T. Sherman, for providing guidance throughout my final year at UMBC. His professional expertise and insights helped me diving into the voting world. His passion about research and life had a profound impact on my critical thinking, collaborative attitude, and communication. I'm so fortunate and happy working with the VoteXX team. It's an honor to work with and learn from the creative, responsible, and talented team members.

I would like to express my sincere thanks to my two co-advisors, Dr. Haibin Zhang and Dr. Sisi Duan. They are not only the outstanding advisors who provide insightful guidance in research, but also great mentors who are always willing to listen and help whenever needed. I do not think I could reach my destination without their continued support in this tough journey.

Thanks also go to the other members of my committee: Prof. Mohamed Younis and Prof. Ting Zhu for being on my doctoral dissertation committee. I am really honored to have them on my committee.

I would like to thank my friends and lab mates – Yusen Wu, Xin Wang, Rui Jin, Hao Chen, Shuai Xu, Cyrus Bonyadi and many more whom I cannot list all here.

It is a pleasure to thank my mother and father, who gave me everything to support me to reach this stage.

I offer my regards and blessings to my wife who supported me in all respects.

She has always been behind me.

Thanks to everyone who help me to complete this work!

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Modern society has been growing increasingly dependent on real-world networked computer systems. A distributed system is a collection of multiple computers (processors) working closely together to solve a single problem. Distributed systems take many forms and cover a variety of system architectures. To build a distributed system, a range of fundamental design objectives should be considered, including fault tolerance, performance and cost, availability, flexibility and extensibility, and scalability.

Fault tolerance is one of the hot topics in modern distributed system. A fault-tolerant system can experience failure (or multiple failures) in its components, but still continue operating properly. Crash fault tolerance assumes that servers (processors) fail in a silent manner and never send incorrect messages. When considering more powerful adversaries or attacks, Byzantine fault tolerance makes it possible to design systems that are resilient against arbitrary faults and attacks. An increasing number of practical systems use BFT, such as permissioned blockchains [9, 13, 74], firewalls, SCADA systems [63, 109], Boeing 777 aircraft information management system [172], and Boeing 777 flight control system [165]. Byzantine fault tolerance is also considered in SpaceX Dragon. [1]

Even though BFT has been researched more than thirty years, it is still a chal-

[1] https://en.wikipedia.org/wiki/Byzantine_fault

lenge to design and deploy it in modern distributed systems. Existing requirements on permissioned blockchain push research on BFT protocols. Facing the challenge of new problems in distributed systems, this dissertation proposes novel BFT protocols and system designs toward building real-world distributed systems.

1.1 Dissertation Statement

I adopt the following dissertation statement:

Real-world fault-tolerant distributed systems can be designed and implemented in a more efficient, secure, scalable, practical, and reliable fashion by leveraging novel BFT protocols, practical cryptographic schemes and libraries, efficient and scalable system designs, modern programming languages, and complete and detailed evaluations and deployments.

The dissertation statement brings out two things. First, building a real-world distributed system is complex and challenging, and a range of fundamental design objectives should be taken into account. I will show our various design strategies for different required tasks in different chapters. Second, multiple techniques can be used to design and implement distributed systems. To design a system that satisfies detailed requirements and solves the issues we proposed, I will present the methods and techniques and mix and match them in different chapters.

1.2 Challenges

There are many different concepts associated with distributed systems, including distributed file systems, distributed pervasive systems (e.g., sensors, mobile devices), distributed Web-based systems etc. All of these systems have been built to fulfil the following objectives [60, 136, 161]:

- Fault tolerance and failure management,
- Scalability,
- Security,
- Heterogeneity,
- Performance, and
- Transparency.

This dissertation focuses on fault tolerance of distributed systems and also seeks methods to improve the performance, guarantee the security, and maintain scalability of the distributed systems.

The existing weakly synchronous protocols such as PBFT can not be scalable. An adversary can halt the consensus resulting in the PBFT achieving zero throughput. Practical asynchronous BFT with modern framework, such as HoneyBadgerBFT [130] and BEAT [70], has been extensively studied in recent years and can scale up to 100 replicas. However, these protocols can defend only against static corruption, where the adversary is restricted to choose its set of corrupted replicas at the start of the protocol and cannot change this set later on. To capture general attacks and real threats where the adversary can choose its set of corrupted replicas

at any moment during the execution of the protocol, based on the information gathered by the adversary, it is challenging to design adaptively secure asynchronous BFT protocols. Asynchronous BFT protocols are complex, consisting of different distributed components (RBC, ABA, and crypto). It is necessary to understand the performance bottleneck(s) for asynchronous BFT. HoneyBadgerBFT, BEAT, and Dumbo [85] assume optimal resilience with $3f + 1$ replicas (where f is an upper bound on the number of Byzantine replicas). It is interesting to ask whether more efficient protocols are possible by relaxing the resilience level. Meanwhile, these recent BFT protocols evaluated their performance under failure-free scenarios. It is unclear if these protocols indeed perform well during failures and attacks. Existing pub/sub systems have no strong fault tolerance (BFT) guarantees. It is challenging to define the properties of a BFT and confidentiality-preserving pub/sub system, covering strong access control and message ordering guarantees, in the sense of cryptography and reliable distributed systems. It is desirable to evaluate and compare our system with some real-world systems systematically, such as Kafka, Kafka with passive replication, and Hyperledger Fabric modules.

1.3 Contributions

This research makes several valuable contributions to the field of distributed systems. It focuses on developing new BFT protocols and system designs for new problem models.

In the first portion of our research, we develop EPIC and HALE to build practi-

cal asynchronous BFT protocols with adaptive security in the computational setting (where the adversary is limited to polynomial time) and the stronger information-theoretic model (where the adversary is unbounded). In the computational model, we provide EPIC using adaptively secure key generation and common coin protocols. In the information-theoretical model, we provide HALE leveraging the classic local coin protocol of Bracha. HALE is more robust than EPIC and does not need distributed key generation. We first characterize efficient BFT protocols using corruption models (adaptive vs. static corruptions). We discuss how to achieve adaptive security for various candidate asynchronous BFT protocols. In this work, we also clarify three "understandings", understanding the performance bottlenecks for asynchronous BFT, understanding the impact of cryptography in asynchronous BFT, and understanding benefits and drawbacks of common coins and local coins.

In the second portion of our research, we design new distributed system primitives with suboptimal resilience, including new RBC constructions and ABA combinations. In particular, we provide an erasure-coded version of IR RBC using Merkle tree and provide a learner-version of RBC (where some replicas are passive learners). We formally prove the correctness of the new RBC constructions. We build a highly flexible MiB [120] framework allowing mixing and matching different RBC and ABA primitives. We designed experiments for asynchronous BFT protocols in failure and attack scenarios. This is the first systematic evaluation for these recent asynchronous BFT protocols using the ACS framework.

Then, in the third portion of our research, we designed, implemented, and evaluated Chios [67], a Byzantine fault-tolerant (BFT) pub/sub system with fine grained

access control and strong reliability, without sacrificing the decoupling property of pub/sub. We formally define the properties of a BFT and confidentiality-preserving pub/sub system, covering strong access control and message ordering guarantees, in the sense of cryptography and reliable distributed systems. Chios is the first pub/sub system achieving decentralized and fine-grained access control as well as publication total order. Chios is versatile and modular, supporting three additional and fully-fledged pub/sub instances designed to meet different goals.

1.4 Roadmap

The remainder of this dissertation is organized as follows. Chapter 2 shows the background and related work. Chapter 3 presents our work, EPIC and HALE, which achieves adaptive security. Chapter 4 presents our efficient asynchronous BFT protocol with more replicas. Chapter 5 presents our work, Chios, for intrusion-tolerant and confidentiality-preserving publish/subscribe messaging. Chapter 6 concludes this dissertation and discusses future work.

Chapter 2

Background and Related Work

In this chapter, we present the background and related work of Byzantine fault tolerance, Byzantine reliable broadcast, and asynchronous binary agreement. The first and second chapters are constructed on the modern asynchronous common subset framework so that we review it here without showing the same content in different chapters.

2.1 Byzantine Fault Tolerance

The reliance on online service accessible on the Internet demands highly-available systems that provide correct service without interruptions. Byzantine fault tolerant (BFT) protocols are used to build replicated services. Lamport et al. [115] defined the concept of the Byzantine fault tolerance system. This system should satisfy two properties:

- If all correct servers input the same value, they produce the same output;
- If the input message is correct, then all correct servers use the value as input and use it to produce the output.

BFT should provide both safety and liveness properties [113].

- **Liveness**: ensures that all requests from correct clients are eventually executed.
- **Safety**: ensures that requests are executed sequentially under a single schedule

consistent with the order seen by clients.

PBFT [51] proposed by Castro and Liskov is a well-known consensus protocol to cope with Byzantine systems and is regarded as the baseline for almost all BFT protocols published afterward. Zyzzyva [107] employs speculation in the execution of state updates, allowing a high throughput pipeline of state-machine replication. The Next 700 BFT Protocols [81] provide a principled approach to view-change in BFT protocols. Their approach switches not only leaders, but also entire regimes, to respond to adaptive system conditions. BFT-SMART [35] is an original open-source library, which was developed in Java and implements a protocol for SMR similar to PBFT. Hotstuff [167] uses a threshold signature scheme to reduce communication complexity and Facebook is building Libra [29] on top of its variant. RBFT [25] uses multiple concurrent instances of PBFT to detect a slow master instance and triggers a leader replacement through PBFT's complex view change. Many BFT protocols have been proposed, such as Upright [55], ABFT [64], ByzID [66], BChain [68], hBFT [69], FastBFT [121].

The BFT consensus algorithms such as scalable BFT [30], practical BFT (PBFT) [51], Zyzzyva [107], are used in consortium blockchains.

2.2 Byzantine Reliable Broadcast

In distributed systems, a set of processes can use the powerful primitive Byzantine reliable broadcast (RBC) to agree on a message from a designated sender, even if some processes (including the sender) are Byzantine. The RBC is appealing since

this primitive can be used in a completely asynchronous environment and it can be implemented in important applications such as payment systems.

RBC protocols are starting with the algorithm of Bracha and Toueg [37, 39] and then Bracha described the first RBC protocol for asynchronous and fully connected reliable networks (i.e., networks where each process is able to communicate with any other in the system and where messages cannot be lost). Guerraoui et al. [83] proposed the scalable RBC. They achieve a scalable solution by relying on stochastic samples instead of quorums, where samples can be much smaller than quorums. In dynamic RBC [82], a process can enter and leave the system at any time without requiring consensus. RBC [138] is an efficient algorithm and proved to have optimal bit complexity. The RBC [62] implements the algorithm with only two communication steps, two message types, and $n^2 - 1$ protocol messages but is a weaker $t-$resilience, namely $t < n/5$.

2.3 Asynchronous Binary Agreement

Byzantine agreement (BA) is one of the most fundamental problems in distributed computing and cryptography. In this problem, a set of n parties, each holding an input v_i, aims to agree on a value v by jointly running a distributed protocol. The BA is appealing because it is used as a building block to design more complex systems and regarded as strong consistency guarantees, e.g. databases, replicated services, or secure voting mechanisms.

Lamport et al. [115] first introduced the problem of BA. The BA protocol has

since been extensively studied for almost four decades under various assumptions. The asynchronous binary agreement (ABA) does not require a global clock shared among the parties, so it is more preferable to its synchronous counterparts.

Consensus, however, is impossible to solve deterministically in asynchronous systems (FLP result [78]). Randomization is a solution to circumvent the FLP result, which is based on a random operation, tossing a coin (0 or 1 with equal probability). Ben-Or [31] and Rabin [147] provide two different and classic approaches to coin tossing.

- **Local coin (Ben-Or)**: each node tosses a coin locally.
- **Common coin/shared coin (Rabin)**: a common/shared coin gives the same values to all nodes.

The local coin approach is simpler but needs to terminate in an expected exponential number of rounds. The common/shared coin approach requires an additional coin sharing scheme but can terminate in an expected constant number of rounds.

There are multiple version of ABA protocols proposed by Mostefaoui et al. However, the efficient ABA [134] is reported with liveness problem [158]. Cachin et al. [47] proposed a method that overcomes the problem and maintains the simplicity of the original approach of ABA [134]. The binary consensus problem allows a process to agree only on a single binary value. To allow processes to agree on arbitrary values, there exist many reductions to multi-value consensus work [135].

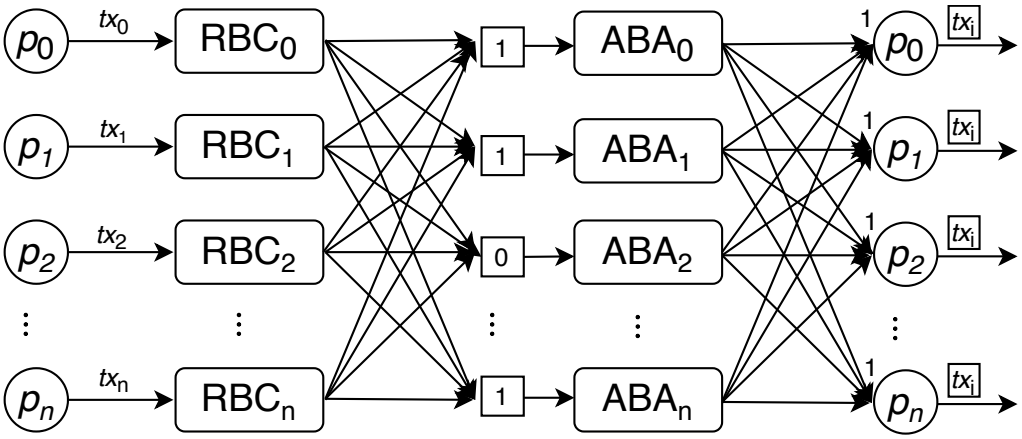


Figure 2.1: The ACS consensus workflow.

2.4 Review of Efficient Asynchronous BFT Protocols

We consider the ACS (asynchronous common subset) framework for asynchronous BFT. In the framework, servers propose a subset of transactions in their transaction pool and deliver the union of the transactions in the agreed-upon vector. The framework was used by Ben-Or et al. [32], CKPS [43], SINTRA [45], HoneyBadgerBFT [130], BEAT [70], and Dumbo [85].

Figure 2.1 reviews the framework for HoneyBadgerBFT and BEAT (with threshold encryption ignored). The protocols proceed in epochs. Each epoch consists of two phases: an RBC phase and an ABA phase. In the RBC phase, each replica proposes a subset of transactions from its transaction pool (a proposal) and uses RBC to broadcast the transactions. In the ABA phase, replicas run n parallel ABA instances, the i-th of which is used to agree on whether the proposal from replica p_i has been RBC-delivered.

In Figure 2.1, replicas p_i ($i \in [0..3]$) propose transactions tx_i, respectively. When a replica delivers a value from an RBC instance $j \in [0..3]$, it inputs 1 to

the j-th ABA instance. HoneyBadgerBFT and BEAT follow Ben-Or et al. [32] and ask each replica to abstain from proposing 0 until $n - f$ ABA instances have been delivered by the replica, which guarantees system throughput.

Also for high throughput, HoneyBadgerBFT ensures that each replica proposes mostly disjoint sets of transactions. Thus, replicas in HoneyBadgerBFT propose randomly selected transactions. To prevent an adversary from censoring a particular transaction, HoneyBadgerBFT requires replicas to use threshold encryption to encrypt transactions. After delivering transactions in ciphertext, replicas collectively decrypt them.

HoneyBadgerBFT uses the bandwidth-efficient AVID broadcast for RBC [46] and the MMR protocol for ABA [134]. BEAT is a family of five asynchronous BFT protocols, providing a series of improvements to HoneyBadgerBFT. In particular, the baseline protocol in BEAT (hereinafter BEAT) eliminates the usage of pairing-based cryptography and leverages more efficient threshold cryptography using elliptic curves. Other protocols in BEAT mainly explore the performance impact using different RBC or information dispersal protocols.

Chapter 3

How to Achieve Adaptive Security for Asynchronous BFT?

3.1 Overview

State machine replication (SMR) [151, 114] is a proven software technique to enable fault-tolerant and highly available services in critical distributed systems (e.g., Google's Spanner [41], Apache ZooKeeper [90]).

Byzantine fault-tolerant (BFT) SMR is the only known software solution for masking arbitrary failures and malicious attacks. BFT has been regarded as the model for building *permissioned blockchains*, where the distributed ledgers (i.e., replicas) know each other's identities but may not trust one another. BFT can also be used to improve the performance and deal with the lack of finality for *permissionless blockchains*, where enrollment is open to anyone and nodes may join and leave dynamically. These permissionless blockchains using BFT are also known as *hybrid blockchains* (e.g., [18, 65, 75, 104, 105, 126, 143, 168]).

BFT protocols can be roughly divided into three categories according to their timing assumptions: *asynchronous*, *synchronous*, or *partially synchronous* [72]. In asynchronous BFT, neither safety nor liveness relies on timing assumptions. In synchronous BFT systems, both safety and liveness may be violated if the synchrony assumption fails to hold. Partially synchronous BFT systems never violate safety but they achieve liveness when the network behaves synchronously only. (It was

demonstrated in [130] that PBFT [50] would achieve zero throughput against an adversarial asynchronous network scheduler, echoing the celebrated FLP impossibility result [78].) For this reason, asynchronous BFT protocols are inherently robust against timing, performance, and denial-of-service (DoS) attacks and (arguably) the most appropriate solutions for mission-critical blockchain applications.

Due to its inherent robustness, asynchronous BFT (and atomic broadcast) have been extensively studied [19, 32, 43, 45, 59, 70, 112, 130, 133, 149]. Most notably, two recent asynchronous BFT systems, HoneyBadgerBFT [130] and BEAT [70], have comparable performance as partially synchronous BFT protocols and can scale to 100 replicas. In particular, HoneyBadgerBFT is an efficient asynchronous protocol with a modern implementation and a scalable real-world deployment, while BEAT offers various performance improvements for different application scenarios.

Despite the impressive performance and robustness for HoneyBadgerBFT and BEAT, the protocols have several major issues.

Static vs. Adaptive corruptions. Depending on how the adversary decides to corrupt parties, there are two types of corruptions for BFT protocols:

- *Static corruptions*, where the adversary is restricted to choose its set of corrupted replicas at the start of the protocol and cannot change this set later on.
- *Adaptive corruptions*, where the adversary can choose its set of corrupted replicas at any moment during the execution of the protocol, based on the information it accumulated thus far (i.e., the messages observed and the states of previously corrupted replicas).

There is a strong separation result: statically secure protocols are not necessarily adaptively secure [48, 61]. HoneyBadgerBFT and BEAT (and a prior system SINTRA [45]) defend against static adversary only. The reason is that these protocols rely heavily on efficient but statically secure threshold cryptography. The situation is in contrast to that of partially synchronous BFT protocols, most of which achieve adaptive security [24, 35, 50, 56, 68, 81, 89, 107, 129].

Computational vs. Information-theoretic security. Depending on the capacities of the adversary, BFT protocols can be in one of the following two models:

- *Computational security*, where the adversary is restricted to probabilistic polyn omial-time (PPT).
- *Information-theoretic security*, where the adversary is unbounded.

There are good reasons why one may favor information-theoretic security over computational security. Information-theoretically secure constructions do not rely on any cryptographic assumptions, while computationally secure constructions assume the hardness of some intractability problems (e.g., RSA, Diffie-Hellman). These mathematical problems may, in the future, be proven to be "easy," or be (partly) broken by newly developed cryptanalysis techniques, or become trivially solvable due to some technological breakthrough (e.g., quantum computer). We stress that favoring information-theoretic model is not merely a theoretical concern: it is a common consensus in the cryptography community to minimize use of cryptographic assumptions.

Understanding the (exact) performance bottleneck(s) for asynchronous

BFT. Asynchronous BFT protocols are complex, consisting of different distributed components and cryptographic building blocks. Both HoneyBadgerBFT and BEAT use reliable broadcast (RBC), asynchronous binary agreement (ABA), and threshold cryptography. Duan et al. [70] showed in BEAT that asynchronous BFT can perform rather differently if using different RBC protocols: Bracha's broadcast [37] results in a latency-optimized asynchronous BFT, AVID broadcast [46] leads to bandwidth-efficient and high-throughput ones, and AVID-FP [88] and its variant can be used to build bandwidth-optimal asynchronous BFT storage. In this paper, we find that not just RBC, the other two building blocks — ABA and threshold cryptography — can impact the performance significantly.

The case of ABA. Both BEAT and HoneyBadgerBFT utilize an ABA protocol proposed by Mostefaoui, Moumen, and Raynal [134] (MMR ABA). The MMR ABA protocol is known as the most efficient ABA protocol that terminates in two rounds in expectation (completing within $O(r)$ rounds with probability $1 - 2^{-r}$). In each round, the MMR protocol has two or three steps. (In contrast, the entire PBFT protocol has three steps only.)

The situation is exacerbated by a liveness issue recently reported [4]. Specifically, the MMR protocol assumes perfect random coins completely independent of the state of all correct nodes when they query the coin. The property is not guaranteed by any existing cryptographic common coin protocols. A malicious network scheduler can keep correct nodes entering the next round with inconsistent values, causing the protocol not to terminate. The best known solution to date is to use Cobalt ABA [127] which has one additional step for *each* round. The added cost,

by percentage, could be significant, considering the MMR ABA terminates in two rounds in expectation and in each round there are only two or three steps.

While theoretically the resulting asynchronous BFT has much more rounds than PBFT, it was demonstrated that some ABA protocols may terminate faster than expected [132, 133]. The experimental result, however, was reported for ABA protocols terminating in an expected exponential number of rounds and for settings where the total number of replicas is at most ten. It is still unclear how the ABA protocols that use common coins and terminate in expected constant rounds and the ABA protocols that use local coins and terminate in expected exponential rounds perform with more replicas. More importantly, it is interesting to evaluate the performance of asynchronous BFT using these ABA protocols for settings with more replicas.

The case of cryptography. It is well known that cryptography can be vital to the performance of BFT protocols. It was originally believed that signature-free BFT protocols such as PBFT are (much) more efficient than BFT protocols using signatures. It was later reported (e.g., BFT-SMaRt [35]) that with modern infrastructures, BFT protocols using signatures can be comparable to those without signatures. Some recent protocols such as SBFT [80] using more expensive threshold signatures were also shown to be both efficient and scalable.

The situation for asynchronous BFT is arguably more complicated. First, HoneyBadgerBFT and BEAT use two (instead of one) threshold cryptographic primitives — threshold encryption and threshold pseudorandom function (PRF). Both protocols use them extensively. Second, the way of using threshold cryptography

for them is quite different from that of partially synchronous BFT protocols. The threshold cryptographic operations in these asynchronous BFT protocols are evenly distributed among n replicas. The two features make it difficult to predict the exact bottlenecks for asynchronous BFT. In particular, if one aims to build a BFT protocol by modifying two primitives, one would have to implement and evaluate multiple protocols.

Besides, most known adaptively secure cryptographic schemes are known to be (much) more expensive than their statically secure ones [117, 118, 122].

It is unclear if asynchronous BFT protocols using adaptively secure schemes can be practical. This question has to be answered via rigorous implementation and extensive evaluation.

3.1.1 Our Contribution

Characterizing BFT protocols. We first characterize efficient BFT protocols using corruption models (adaptive vs. static corruptions). We discuss how to achieve adaptive security for various candidate asynchronous BFT protocols.[1]

EPIC. In the computational model, we provide EPIC that uses an adaptively secure key generation and secure common coin protocols.

EPIC takes a new approach to adaptively secure asynchronous BFT. First, EPIC uses the LM-LJY adaptively secure threshold PRF scheme [117, 122] for common coins. Second, EPIC uses the Cobalt ABA protocol [127] which resolves

[1] We discuss protocols published after the conference version [119] of this submission that merely considers the computational security model.

the liveness issue of HoneyBadgerBFT and BEAT. Last, EPIC uses a hybrid method of the random transaction selection (as used in HoneyBadgerBFT and BEAT) and the FIFO transaction selection (as used in CKPS [43] and SINTRA [45]), eliminating the use of expensive threshold encryption.

Both HoneyBadgerBFT and BEAT use threshold PRF and threshold encryption and require a trusted dealer to generate the cryptographic keys for individual replicas. The key generation procedure for HoneyBadgerBFT and BEAT is thus centralized. In contrast, most of the efficient BFT protocols rely on authenticated channels or digital signatures and do not need a trusted setup. It is desirable for efficient asynchronous BFT protocols to support a distributed key generation. In fact, we should build asynchronous BFT protocols with distributed key generation that is secure against adaptive corruptions.

EPIC instantiates and implements the distributed key generation protocol [117] which is also secure against adaptive corruptions. In comparison, HoneyBadgerBFT and BEAT do not have distributed key generation protocols, and most of the multi-party computation protocols simply do not instantiate ideal broadcast channels. We view this contribution as an important one, as our implementation and evaluation will help understanding the difficulty of distributed key generation and guiding the parameter selection for these protocols.

EPIC relies on an adaptively secure common coin protocol which uses a pairing-based assumption [117]. The adaptively secure common coin protocol is the only known such protocol. The corresponding mathematical problem is weaker than well-studied problems such as RSA. Admittedly, if, in the future, the problem is proven

to be not as difficult as one might thought (which is likely), the architecture that EPIC uses is no longer viable (unless one could find a new construction for adaptively secure common coin protocol based on a different cryptographic assumption). HALE, which we introduce right below, does not have the potential drawback.

HALE. In the stronger information-theoretical model, we provide HALE. HALE aims to "revive" the original idea of Ben-Or, Kemler, and Rabin [32]: their ACS construction was designed to work in the information-theoretic setting from the perspective of theoretical feasibility. HALE, on the other hand, aims to be practical and HALE relies on three novel ideas with the first one being our new transaction selection process (the same one as in EPIC). For ABA, HALE uses Bracha's ABA protocol that leverages local coins instead of common coins [37]. For RBC, one can choose the well-known Bracha's broadcast protocol [38] that is secure against adaptive corruptions.

HALE just needs the minimum assumption that there exist authenticated channels. Very importantly, HALE does not rely on threshold cryptography and therefore does not need the costly distributed key generation phase.

Understanding the performance bottlenecks for asynchronous BFT. As our new protocol modifies almost all building blocks for asynchronous BFT (including ABA, threshold PRF, and threshold encryption) but RBC, evaluating which component dominates the performance bottleneck is a difficult task. We therefore mix and match different building blocks to implement four asynchronous BFT protocols and evaluate their performance difference. Besides, to understand the

cryptographic overhead, we implemented our baseline protocol with static security and our systems with adaptive security. Our approach complements BEAT which essentially evaluated the performance difference using different RBC protocols.

Understanding the impact of cryptography. In the literature of asynchronous BFT protocols, it does seem that they adopt quite different cryptographic primitives. HoneyBadgerBFT uses some pairing-based cryptography (that is slightly outdated) and Dumbo [85] uses the same cryptographic libraries with no modifications. BEAT, on the other hand, strives to use standard elliptic curve cryptography with standard 128-bit security. As EPIC uses pairing-based cryptography, we want to use the well-received and the-state-of-the-art pairing types, and meanwhile, understand the exact cryptographic overhead differences among various pairing curves (some of which are more efficient but less secure).

Understanding benefits and drawbacks of common coins and local coins. It was shown that Bracha's ABA protocol [37] may exhibit better performance than the ABA protocol of Cachin, Kursawe, and Shoup [44] in the LAN environment where the total number of replicas is at most ten [132].

Compared to ABA protocols, BFT protocols are much more complex. It is hard to predict how BFT protocols would perform using common-coin vs. local coins. In particular, it is interesting to find if local-coin based BFT protocols may have some benefits compared to well studied common-coin based BFT protocols.

Practicality of EPIC and HALE. We show that EPIC is slightly slower than asynchronous BFT protocols with static security if the network size is small; how-

ever, if the network size grows larger, EPIC is not as efficient as those with static security. Besides, EPIC achieves its peak throughput when the network size is small, but even with 31 replicas, EPIC can still achieve throughput of 10,000 tx/sec for transactions of size 250 bytes.

We also show while in most scenarios, HALE is less efficient than EPIC, HALE is reasonably fast. HALE achieves 42,000 tx/sec and 3,400 tx/sec for the 4-server setting in the LAN and WAN environments, respectively. Remarkably, HALE is more efficient than EPIC in the LAN setting when the number of replicas is smaller than 16.

3.2 Related Work

BFT assuming partial synchrony. Efficient partially synchronous BFT has been extensively studied [24, 27, 35, 56, 68, 80, 81, 89, 107, 129, 167]. Even for partially synchronous BFT protocols focusing on robustness [24, 56], their performance can drop 78% - 99% in the presence of Byzantine replicas and/or clients [25]. It is demonstrated that PBFT would achieve zero throughput against an adversarial asynchronous scheduler [130].

Asynchronous binary agreement (ABA). ABA was introduced independently by Ben-Or [31] and Rabin [147]. ABA is a fundamental primitive to build most complex distributed system protocols [32, 42, 43, 45, 59, 131, 133]. For this reason, a significant number of ABA protocols have been proposed [34, 38, 44, 49, 79, 127, 134, 147, 154, 155, 159, 171]. Cachin, Kursawe, and Shoup (CKS) [44] proposed an

efficient ABA which achieves optimal resilience and runs in $O(n^2)$ message complexity. The CKS ABA heavily uses RSA-based dual-threshold signatures [152] which are computationally expensive. Mostefaoui, Moumen, and Raynal (MMR) [134] presented the first signature-free ABA that has the same message complexity as the CKS ABA [44]. The MMR ABA is used in HoneyBadgerBFT and BEAT. It is reported that the MMR ABA, however, has a liveness issue, if being instantiated using any existing cryptographic coin flipping protocols [4].

Asynchronous atomic broadcast and BFT. In an atomic broadcast, a broadcaster (one of the replicas) broadcasts messages to all replicas, and all replicas should deliver messages in the same order. Instead, BFT state machine replication specifies clients and replicas, and all replicas delivers client messsages in the same order. We do not distinguish Byzantine atomic broadcast and BFT and collectively call them BFT.

Asynchronous BFT protocols, such as SINTRA, HoneyBadgerBFT, and BEAT, follow the asynchronous common subset (ACS) framework [32, 43] which can be realized using RBC and ABA. The underlying ABA protocols are efficient, terminating in an expected constant number of rounds.

RITAS is a stack of randomized distributed protocols defending against Byzantine failures [133], RITAS consists of an efficient atomic broadcast implementation of Correia, Neves, and Verissimo [58]. While the protocol theoretically terminates in an expected exponential number of rounds, it was demonstrated that the protocol in practice may execute in only a few rounds for certain conditions. The RITAS protocol is shown to be efficient for a cluster of ten replicas.

Recently, Abraham, Malkhi, and Spiegelman provided an asynchronous BFT protocol that reduces message complexity to $O(n^2)$ [20], utilizing a similar workflow used in HotStuff [167]. Besides its higher latency, the protocol has theoretically lower throughput than HoneyBadgerBFT and BEAT which can select random transactions for high throughout.

Asynchronous hybrid BFT protocols. KS [112] and RC [149] are asynchronous hybrid BFT protocols guaranteeing both safety and liveness under asynchronous environments. Both protocols have an optimistic BFT protocol under "normal" circumstances (where there is no failure or the primary is correct) and a pessimistic BFT protocol under "rare" circumstances (e.g., asynchrony). KS [112] and RC [149] use PBFT-like protocols during normal operations and randomized asynchronous BFT for recovery in case of failures or asynchrony. KS proceeds in epochs and uses Bracha's broadcast [37] during the normal-operation phase, just like in PBFT. It suggests using randomized Byzantine agreement for backup and delivers some requests for liveness. It has the same efficiency as PBFT during gracious execution. RC replaces the reliable broadcast primitive in KS using consistent broadcast, a weaker primitive. RC is the first BFT protocol (atomic broadcast) with the message complexity only $O(n)$ in its normal case, while all other BFT (atomic broadcast) protocols have the message complexity $O(n^2)$. The improvement comes at the cost of more expensive recovery phase using heavy public-key cryptography. There is no implementation for either KS or RC.

Asynchronous MPC. Lu et al. [123] recently provided the first robust asyn-

chronous multi-party computation system with guaranteed output delivery using HoneyBadgerBFT. The protocol achieves static security.

3.3 System and Threat Model

BFT. We consider a Byzantine fault-tolerant state machine replication (BFT) protocol, where f out of n replicas can fail arbitrarily (Byzantine failures) and an adaptive adversary can coordinate faulty replicas. The adaptive adversary can choose its set of dishonest parties at any moment during the execution of the protocol, based on the messages transmitted and the internal states of previously corrupted replicas. The BFT protocols considered in this paper tolerate $f \leq \lfloor \frac{n-1}{3} \rfloor$ Byzantine failures, which is optimal.

When considering the information-theoretic setting, we assume there exist authenticated channels, a minimum assumption that nodes can use to authenticate with each other.

A replica *delivers operations*, each *submitted* by some client. The client should be able to compute a final response to its submitted operation from the responses it receives from replicas. We use operations, (client) requests, and transactions (blockchain terminology) interchangeably. Correctness of a BFT protocol is specified as follows.

- **Agreement**: If any correct replica delivers an operation m, then every correct replica delivers m.
- **Total order**: If a correct replica has delivered an operation m with a sequence

number, and another correct replica has delivered an operation m' with the same sequence number, then $m = m'$.

- **Liveness**: If an operation m is submitted to $n - f$ correct replicas, then all correct replicas will eventually deliver m.

The liveness property has been referred to by other names (e.g., "fairness" [43], "censorship resilience" [130]).

Timing assumption. We can roughly divide BFT protocols into three categories according to their timing assumptions: *asynchronous*, *synchronous*, or *partially synchronous* [72]. An asynchronous BFT system makes no timing assumptions on message processing or transmission delays. If there is a known bound on message processing delays and transmission delays, then the corresponding BFT system is synchronous. Partially synchronous BFT lies in-between: messages are guaranteed to be delivered within a time bound, but the bound may be unknown to participants of the system or system designers.

Asynchronous BFT protocols are inherently more robust than other BFT protocols. Due to the celebrated FLP impossibility result [78] which rules out that deterministic protocols reach consensus in fully asynchronous environments, asynchronous BFT protocols must rely on randomization and be probabilistically live. We consider purely asynchronous systems making no timing assumptions on message processing or transmission delays. We assume synchrony for the distributed key setup phase, which is a one-time event. We will discuss the implication of the system choice.

3.4 Primitives and Building Blocks

This section reviews the cryptographic and distributed systems building blocks for EPIC.

Threshold pseudorandom function (PRF). We describe threshold PRF with a decentralized key generation (e.g., [117]).

A (t, n) threshold PRF scheme for a function F consists of the following algorithms (FGen, Eva, Vrf, FCom).

- An interactive key generation algorithm FGen involves n players $p_1, \cdots, p_n$. Each player p_i takes as input common public parameters, a security parameter l, the number n of total servers, and threshold parameter t. The output of the protocol is (pk, vk, sk), where pk is the public key, vk is the verification key, and $sk = (sk_1, \cdots, sk_n)$ is a list of private keys. Both pk and vk are known to anyone, and p_i only obtains sk_i.
- A PRF share evaluation algorithm Eva takes a public key pk, a PRF input m, and a private key sk_i, and outputs a PRF share σ_i.
- A share verification algorithm Vrf takes as input the verification key vk, a PRF input m, and a PRF share σ_i, and outputs a single bit.
- A combining algorithm FCom takes as input the verification key vk, a PRF input m, and a set of t valid PRF shares, and outputs a PRF value σ.

We require the threshold PRF value to be unpredictable against an adversary that controls up to $t-1$ servers. We also rely on an additional uniqueness property, which guarantees that for a given public key pk, there exists exactly one valid

signature on each message m. One can consider both static and adaptive adversaries just as in BFT protocols. In the adaptive corruption model, the adversary can corrupt players and query signing oracles at any moment of the protocol, based on the information collected so far.

Byzantine reliable broadcast (RBC).

In RBC, a sender (one of the replicas) sends a message to all other replicas. An asynchronous RBC protocol [114] satisfies the following properties:

- **Agreement**: If two correct replicas deliver two messages m and m' then $m = m'$.
- **Totality**: If some correct replica delivers a message m, all correct replicas deliver m.
- **Validity**: If a correct sender broadcasts a message m, all correct replicas deliver m.
- **Integrity**: Every correct replica delivers a message m from sender p at most once. If p is correct, then m was previously broadcast by p.

Bracha's broadcast [37] is a well-known implementation of RBC. To broadcast a message m, its communication complexity is $\mathcal{O}(n^2|m|)$. Cachin and Tessaro [46] proposed an erasure-coded RBC (AVID broadcast) reducing the bandwidth to $\mathcal{O}(n|m|)$. EPIC is compatible with any RBC and implements AVID broadcast as in HoneyBadgerBFT and BEAT.

Asynchronous binary agreement (ABA). In an ABA protocol, each replica has a binary value as an initial input v_{input} (also known as a *vote*). ABA allows replicas to agree on the value of a single bit and deliver the value. ABA should satisfy the

following properties:

- **Validity**: If all correct replicas have the same input value v, correct replicas will deliver v.
- **Agreement**: If a correct replica delivers v and another correct replica delivers v', then $v = v'$.
- **Termination**: All correct replicas eventually deliver a binary value with probability 1.

An ABA protocol proceeds in rounds, where for a round r, a replica has an input est_r. ABA protocols considered in the paper have an expected constant number of rounds. In each round, there are a small number of steps (two to four steps for the ABA protocols we consider), and replicas query the common coin (realized using threshold cryptography) and decide to either terminate the protocol or propose some values for the next round.

3.5 Problems and Technical Overview

Figure 2.1 reviews the framework for HoneyBadgerBFT and BEAT.

3.5.1 Characterizing BFT Using Corruption Models

We characterize existing BFT protocols using corruption models. For static security, the adversary needs to decide which replicas to corrupt before the execution of the system, whereas for adaptive security, the adversary can adaptively choose which replicas to corrupt, based on information the adversary has accumulated thus far.

Common-coin asynchronous BFT. Efficient asynchronous BFT systems, such as SINTRA, HoneyBadgerBFT, and BEAT, use the ACS framework and rely on cryptographic common coins.

SINTRA uses the RSA-based dual threshold signature scheme of Shoup [152] and the Diffie-Hellman problem based threshold PRF scheme of Cachin, Kursawe, and Shoup [44] implemented using finite fields. HoneyBadgerBFT uses the pairing-based threshold encryption of Baek and Zheng [97] and the pairing-based threshold signature of Boldyreva [36]. BEAT uses the threshold encryption scheme of Shoup and Gennaro [153] and the threshold PRF scheme of Cachin, Kurasawe, and Shoup, both of which are implemented using elliptic curves. All of the above threshold cryptographic schemes are proven secure against static corruptions only. Therefore, the corresponding asynchronous BFT systems achieve static security.

A recent protocol, Dumbo [85], refines the framework of Cachin, Kursawe, Petzold, and Shoup [43]. It was shown in Dumbo [85] that Dumbo has better performance than HoneyBadgerBFT in WAN environments.

Unfortunately, in each epoch, Dumbo uses $n^3 + 12n^2$ threshold signatures which are based on pairing operations. As in HoneyBadgerBFT, Dumbo uses some outdated pairing types. One would need to use well-received pairing types that achieve standard 128-bit security. Note one pairing operation with such a level of security is about 10 times slower than the conventional elliptic curve operation. In order to make Dumbo work in the adaptive security model, one would be have to replace all the threshold signatures in the static security model with ones with adaptive security. This replacement potentially makes the approach prohibitively

expensive.

Local-coin asynchronous BFT. RITAS is a stack of randomized distributed protocols defending against Byzantine failures [133] in the adaptive corruption model. It comprises an efficient atomic broadcast protocol of Correia, Neves, and Verissimo (CNV) [58]. Instead of using common coins, the CNV protocol relies on local coins. While the protocol terminates in an expected exponential number of rounds, it was demonstrated that the protocol in practice may execute in only a few rounds for certain conditions. In a different scenario, the classic Bracha's ABA protocol [37] was shown to have better performance than the ABA protocol of Cachin, Kursawe, and Shoup [44] in the LAN environment where the total number of replicas is at most ten [132].

The CNV atomic broadcast protocol is much more bandwidth and round expensive than both HoneyBadgerBFT and BEAT. This expenses make it theoretically less efficient in a scalable WAN environment. No known experimentation was conducted for the CNV protocol with more than ten replicas.

Partially synchronous BFT. Most of the existing BFT protocols in partially synchronous environments [24, 35, 56, 81, 89, 107, 129] achieve adaptive security. Protocols such as SBFT [80] and HotStuff [167] rely on statically secure threshold signatures and achieve static security only.

Committee-based (BFT) protocols. Some scalable hybrid blockchain protocols [144] or Byzantine agreement protocols [103] do not use threshold cryptography but involve the selection of a small committee among all replicas for consensus,

which makes adaptive security a non-trivial task for them. This situation is not our concern, as we study conventional BFT protocols where all replicas, not just a fraction of them, need to participate in the consensus process.

3.5.2 Achieving Adaptive Security for EPIC

EPIC follows HoneyBadgerBFT and BEAT and uses a novel combination of new primitives to achieve adaptive security.

As we mentioned above, both HoneyBadgerBFT and BEAT use threshold common coin and threshold encryption schemes which are statically secure. Intuitively, to achieve adaptive security, one would have to replace both statically secure primitives using adaptively secure ones.

First, we adopted and implemented the adaptively secure threshold PRF scheme of Loss and Moran [122], which is built from the adaptively secure threshold signature scheme of Libert, Joye, and Yung [117].[2] The adaptively secure threshold PRF scheme (hereinafter the LM-LJY threshold PRF) requires four pairing computation for signature verification, twice more expensive than the threshold PRF scheme in HoneyBadgerBFT which requires two pairing computation. The LM-LJY scheme is much more expensive than the pairing-free threshold PRF scheme in BEAT. It is natural to explore the performance penalty of using adaptively secure threshold PRF protocol.

[2]Specifically, Loss and Moran [122] proved that the signature scheme of Libert, Joye, and Yung satisfies the uniqueness property. It is therefore trivial to derive a threshold PRF scheme in the random oracle model.

To handle threshold encryption, one could use an adaptive secure one as well. To our knowledge, the scheme of Libert and Yung [118] is the most efficient threshold encryption scheme. The scheme, however, relies on a bilinear group of composite order, which is much less efficient than a regular, prime-order bilinear group [84]. We take a different approach without using threshold encryption. In our approach, replicas maintain a transaction buffer. Replicas select a random subset of T transactions in plaintext for most epochs. They periodically switch to select the first T transactions in their buffer. Doing so can keep the efficiency as HoneyBadgerBFT and BEAT, while ensuring any transaction cannot be censored for too long.

On the one hand, EPIC eliminates the usage of any threshold encryption scheme, thereby potentially improving the performance of asynchronous BFT. On the other hand, the periodical switch for selecting transactions in a FIFO manner may reduce performance. Therefore, we must experimentally verify which of the above two factors will dominate the performance overhead.

Loss and Moran [122] claimed that they obtained the first adaptively secure ABA protocol running in $O(n^2)$ communication complexity by using the LM-LJY threshold PRF to obtain common coins for the MMR ABA. The scheme, unfortunately, has the same liveness issue as reported [4]. In contrast, EPIC combines the LM-LJY threshold PRF and the Cobalt ABA to obtain an adaptively secure ABA protocol running in $O(n^2)$ communication complexity.

As the LM-LJY threshold PRF scheme is the only threshold cryptographic scheme used in EPIC, we just need to build a decentralized key generation protocol for the LM-LJY threshold PRF scheme. A decentralized key generation protocol

has already been described in the same paper by Libert, Joye, and Yung [117]. The key generation algorithm assumes a broadcast channel and private and authenticated pairwise channels. Most cryptographic and multi-party computation systems assuming broadcast channels simply use best-effort broadcast (see [87] and references therein) and therefore do not provide the fault tolerance needed. We provide a concrete instantiation using Bracha's broadcast [38] and an implementation for the protocol. Our key generation process works in synchronous environment, tolerating up to $n/2$ Byzantine faulty replicas.

3.5.3 Achieving Adaptive Security for HALE

HALE aims to "revive" the original idea of Ben-Or, Kemler, and Rabin that ACS works in the information-theoretic setting. We carefully build HALE so it can be instantiated efficiently. In contrast to BEAT, HALE relies on three novel ideas with the first one being our new transaction selection process (the same one as in EPIC). For ABA, HALE uses Bracha's ABA protocol that leverages local coins instead of common coins [37]. Local coin based ABA does not rely on any cryptographic assumptions. To ensure its correctness, one just needs the authenticated channel assumption. For RBC, to enforce strict information-theoretic security, one should choose one that is secure against unbounded adversaries; for instance, the well-known Bracha's broadcast [38] is indeed secure against adaptive corruptions.

The CNV atomic broadcast protocol (implemented in the RITAS framework) also works in the information-theoretic setting but it has a significantly large number

of steps even in the normal scenarios. The feature makes it less efficient even in the failure-free scenario. Instead, jumping ahead slightly, HALE can achieve high throughput in the LAN setting and reasonable throughput in the WAN setting when n is small; HALE can even outperform EPIC in the LAN setting when n is smaller than 16.

3.6 EPIC

In this section, we describe the design of EPIC. EPIC follows the ACS framework and has an RBC phase and an ABA phase. Different from HoneyBadgerBFT and BEAT, EPIC achieves adaptive security and decentralized key generation. Figure 4.1 describes the EPIC protocol using RBC and ABA in a black-box manner.

We begin with a high-level overview. The protocol proceeds in epochs numbered by s (initialized as 0). In each epoch, replicas select a subset of transactions as a proposal from their transaction buffer and agree on a set of transactions containing the union of the proposals of at least $n - f$ replicas. Let B be the batch size of the transactions for an epoch. In an epoch, each replica proposes transactions of size $b = \lceil B/n \rceil$ (the batch size for a replica). In the RBC phase, replicas use RBC to broadcast the proposals. In the ABA phase, n parallel ABA instances are run. The i-th ABA instance is used to agree on whether the transactions from replica p_i have been delivered in the RBC phase. If a correct replica p_j terminates the i-th ABA instance with 1, the transactions from p_i are delivered. Otherwise, the transactions will not be included. We follow Ben-Or et al. [32] (and HoneyBadgerBFT

Initialization
  buf $\leftarrow \emptyset$ {transaction buffer}
  B {batch size}
  μ, δ {parameters for transaction selection}
  $s \leftarrow 0$ {epoch number}
  i {replica id}
  $\{\text{RBC}_j\}_n$ {n RBC instances where j is the sender of RBC_j}
  $\{\text{ABA}_j\}_n$ {n ABA instances}
epoch ***s***
  if $s = 0, \cdots, \mu - 1 \bmod (\mu + \delta)$
    let *value* be a random selection $\lceil B/n \rceil$ of transactions for the first B elements in buf
  else let *value* be the first $\lceil B/n \rceil$ transactions in buf
  input *value* to RBC_i
  upon delivery of $value_j$ from RBC_j
    if ABA_j has not yet been provided input, input 1 to ABA_j
  upon delivery of 1 from ABA_j and $value_j$ from RBC_j
    $output \leftarrow output \cup value_j$
  upon delivery of 1 from at least $n - f$ ABA instances
    for each ABA_j instance that has not been provided input
      input 0 to ABA_j
  upon termination of all the n ABA instances
    deliver *output*
  $s \leftarrow s + 1$

Figure 3.1: EPIC algorithm for p_i.

and BEAT), ensuring at least $n - f$ ABA instances terminate with 1, and thus the union of the transactions from at least $n - f$ replicas are delivered. To this goal, every replica abstains from proposing 0 until $n - f$ ABA instances have been delivered by the replica. Each ABA instance terminates with probability 1/2 for each round. As EPIC must wait for all ABA instances to finish, the expected running time of EPIC is $O(\log N)$.

As in HoneyBadgerBFT and most BEAT instances, EPIC uses an adaptively secure RBC — AVID broadcast [46]. In the following, we specify other building blocks for EPIC:

- the transaction selection approach in EPIC,
- the decentralized key distributed algorithm for EPIC,
- EPIC's ABA protocol—the Cobalt ABA, and
- the adaptively secure common coin protocol from the LM-LJY threshold PRF scheme.

Transaction selection for EPIC. EPIC uses a novel transaction selection approach which is distinguished from prior asynchronous BFT protocols such as SINTRA, HoneyBadgerBFT, and BEAT.

In SINTRA, replicas maintain a log of transactions according to the order they are received and replicas select as input the first subset of transactions in the transaction buffer. We call the approach FIFO selection. The approach can be easily shown to achieve liveness but leads to low system throughput. This slowdown is because the transactions delivered are unions of the transactions selected by replicas, and replicas tend to select the same transactions in each epoch.

In HoneyBadgerBFT (and BEAT), replicas propose randomly selected sets of transactions to improve throughput. Doing so directly causes a liveness issue, as a network adversary can censor certain transactions so that they will not be delivered. Thus, HoneyBadgerBFT (and BEAT) choose to use threshold encryption to avoid censorship. In their approach, replicas first encrypt the proposals and then decrypt them collectively when transactions in ciphertext are delivered. We call this approach ETD (standing for "encrypt-then-decrypt"). HoneyBadgerBFT uses the pairing-based threshold encryption scheme of Baek and Zheng, while BEAT uses

the threshold encryption scheme of Shoup and Gennaro [153]. Both schemes are efficient but only statically secure.

In EPIC, we take a different approach without using threshold encryption. We ask replicas to select random transactions *in plaintext* for most epochs (e.g., 4/5 of the total epochs) and periodically switch to the FIFO selection. The strategy is performed deterministically for all replicas.

More formally, let s be the epoch number initially numbered 0. Let μ and δ be two system parameters determining how often the protocol switches between the two modes. In EPIC, replicas can first perform random selection for μ epochs and then the FIFO selection for δ epochs. Our approach can be generalized to many other scenarios, as long as the switching strategy is deterministic (and known to all replicas), and the system can perform the FIFO selection for a non-negligible fraction of epochs (to ensure liveness).

Our transaction selection approach can keep the efficiency of HoneyBadgerBFT and BEAT, while avoiding censorship. Since our approach can be somewhat viewed as a hybrid of HoneyBadgerBFT and SINTRA, we call it HYB (standing for "hybrid").

The HYB approach eliminates the use of (expensive) threshold encryption scheme, which would improve efficiency. The periodical transaction switch to the FIFO selection may reduce performance. Besides, the approach provides trade-offs between latency and throughput. Roughly, if μ is reasonably larger than δ, the system favors throughput over latency, and otherwise the opposite is the case.

The Cobalt ABA. HoneyBadgerBFT and BEAT use the MMR protocol in the

ABA phase. It was reported that the MMR protocol is not live if being instantiated using any existing cryptographic common coin protocols [4]. Essentially, the MMR protocol makes a strong common coin assumption which existing cryptographic common coin protocols fail to satisfy. Besides, the MMR protocol in HoneyBadgerBFT and BEAT achieve static security due to the use of statically secure common coin protocols.

EPIC thus enhances the ABA choice in HoneyBadgerBFT and BEAT in two aspects. First, we use the Cobalt ABA protocol [127] instead of the MMR ABA to resolve the liveness issue. Second, we instantiate the Cobalt ABA protocol using the adaptively secure LM-LJY threshold PRF scheme.

The MMR and Cobalt ABA protocols are illustrated in Figure 3.2, where the Cobalt ABA includes the boxed code but the MMR ABA does not include. Specifically, the MMR protocol has two to three steps in each round. In the first step, all replicas broadcast their input. If a replica receives $f + 1$ matching input value that is different from its input, it triggers the second step by broadcasting the value. In the last step, if a replica receives $2f + 1$ matching value v, it broadcasts an $\mathsf{aux}(v)$. Next, if a replica receives $2f + 1$ $\mathsf{aux}()$ messages, it either uses the only available binary value from the $\mathsf{aux}()$ messages or the common coin value to enter the next round. The liveness issue for MMR protocol is due to the usage of the cryptographic common coin. The adversary (and network scheduler) can learn the value of the common coin and *manipulate* the sequence of messages received by other replicas to make the protocol never terminate. To solve the issue, the Cobalt ABA protocol introduces one more step in each round. In the Cobalt ABA protocol,

each replica needs additionally to broadcast the values received in the aux() step. We use the Cobalt ABA to obtain an adaptively secure ABA protocol running in $O(n^2)$ communication complexity.

The additional step in the Cobalt ABA protocol can be significant, as the Cobalt ABA only has three or four steps in each round and it may run in several rounds (two on average). On the other hand, it was shown that some ABA protocols may terminate faster than expected (in a small-scale setting) [133, 132]. It is natural to ask what the performance penalty of both BEAT and EPIC using the Cobalt ABA would be.

Initialization
  $r \leftarrow 0$ {round}
  $est_0 \leftarrow v_{input}$ {set input for round 0 to initial input}
round r
  broadcast bval(est_r) {broadcast input}
  upon receiving bval(v) from $f+1$ replicas
    if bval(v) has not been sent, broadcast bval(v)
  upon receiving bval(v) from $2f+1$ replicas
    $bin\_values \leftarrow bin\_values \cup \{v\}$
  wait until $bin\_values \neq \emptyset$ {move to the second step}
    broadcast aux(v) where $v \in bin\_values$
  upon receiving $n-f$ aux() such that the set of values $vals$ the messages is a subset of bin_values
    broadcast conf$_r$($vals$) {move to the third step}
  upon receiving $n-f$ conf$_r$() such that the set of values $vals$ is a subset of bin_values
    $c \leftarrow Coin()$ {obtain common coin}
    if $vals = \{\rho\}$
      $est_{r+1} \leftarrow \rho$
      if $\rho = c$, **deliver** ρ {terminate the protocol}
    else $est_{r+1} \leftarrow c$ {enter the next round}
  $r \leftarrow r+1$

Figure 3.2: The MMR and Cobalt ABA protocol. The Cobalt ABA includes the boxed code, while the MMR ABA does not.

Distributed key generation. EPIC uses one threshold cryptographic primitive

only, the LM-LJY adaptively secure threshold PRF. The threshold PRF scheme is the adaptively secure threshold PRF scheme of Loss and Moran [122], which is built from the adaptively secure threshold signature scheme of Libert, Joye, and Yung [117]. If the key generation for the threshold PRF is decentralized, so is EPIC.

In fact, a decentralized key generation protocol has been described in the same paper by Libert, Joye, and Yung [117], but no implementation is provided. The key generation algorithm assumes a broadcast channel and private and authenticated pairwise channels.

While most of existing distributed cryptographic and multi-party computation systems [87] relying on broadcast channels simply use best-effort broadcast, we provide a concrete instantiation using Bracha's broadcast [38]. Bracha's broadcast has three steps, achieving adaptive security in asynchronous environments.

It is straightforward to build a synchronous version for it. We describe the EPIC distributed key generation protocol in Figure 3.3.

Though a distributed key generation protocol should ideally work in asynchronous environments, our protocol works in synchronous environments only. We made the system decision, in part because the protocol is simpler to implement than existing asynchronous protocols [98, 106]. Another reason is that the key generation procedure is a one-time event, and replicas can set an adequately large timeout value to ensure safety. Note that key generation protocol can tolerate up to $n/2$ Byzantine failures, while an asynchronous key generation protocol only tolerates up to $n/3$ Byzantine failures.

Common public parameter setup: Let $\mathcal{BG} = (q, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$ be an asymmetric bilinear group, where $\mathbb{G}$, $\hat{\mathbb{G}}$, and $\mathbb{G}_T$ are cyclic groups of prime order q, g and $\hat{g}$ are generators for $\hat{\mathbb{G}}$, and $e\colon \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$ is an efficiently computable bilinear map.

• p_i chooses random polynomials for $k \in \{1, 2\}$: $A_{ik}[X] = a_{ik0} + a_{ik1}X + \cdots + a_{ikf}X^f$ and $B_{ik}[X] = b_{ik0} + b_{ik1}X + \cdots + b_{ikf}X^f$ of degree f. p_i runs RBC to broadcast $C_{ikd} = g^{a_{ikd}}\hat{g}^{b_{ikd}}$ for $d \in [0..f]$. p_i sends $\{A_{ik}(j), B_{ik}(j)\}_{k=1}^2$ to p_j for $j \in [1..n]$.

• p_i sends a complaint against p_j for any of the conditions:

- p_i received $\{A_{jk}(i), B_{jk}(i)\}_{k=1}^2$ from p_j and checks

$$g^{A_{jk}(i)}\hat{g}^{B_{ik}(i)} = \prod_{d=0}^{f} C_{jkl}^{i^d} \qquad \text{for} \quad k = 1, 2 \quad (1),$$

  but these equalities do not both hold,

- p_i did not receive values from p_j, or

- p_i received more than one set of values.

• Upon receiving a complaint from p_j, p_i runs RBC to broadcast $\{A_{ik}(j), B_{ik}(j)\}_{k=1}^2$ that satisfy (1).

• p_i marks p_j as disqualified if

- p_i received more than f complaints against p_j, or

- p_j answered a complaint with values that falsify (1).

p_i then builds the set of non-disqualified replicas Q.

• p_i computes $(pk, vk_1, \cdots, vk_n, sk_i)$ as follows:

- $$pk \leftarrow \prod_{i \in Q} C_{ik0},$$

- $$vk_j \leftarrow \left(\prod_{u \in Q} \prod_{d=0}^{t} C_{u1l}^{i^d}, \prod_{u \in Q} \prod_{d=0}^{t} C_{u2l}^{i^d} \right) \quad \text{for} \quad j \in [1..n]$$

- $$sk_i \leftarrow \left\{ \sum_{j \in Q} A_{jk}(i), \sum_{j \in Q} B_{jk}(i) \right\}_{k=1}^{2}$$

Figure 3.3: The distributed key generation algorithm FGen for $(f + 1, n)$ threshold PRF for replica p_i, where the output of p_i is $(pk, vk_1, \cdots, vk_n, sk_i)$. The algorithm is executed only once and tolerates $\frac{n}{2}$ corruptions. The first three steps are interactive, while the last two steps involve local computation only.

Common coin protocol. For the adaptively secure common coin protocol, we use the LM-LJY threshold PRF scheme of Loss and Moran [122] based on the adaptively secure threshold signature scheme of Libert, Joye, and Yung [117] (Figure 3.4). The threshold PRF scheme requires four pairing computation for signature verification. It is thus twice more expensive than the threshold PRF scheme in HoneyBadgerBFT and much more expensive than the pairing-free threshold PRF scheme in BEAT. The threshold LM-LJY PRF scheme provides adaptive security under the Symmetric eXternal Diffie-Hellman (SXDH) assumption.

Finally, we illustrate in Figure 3.5 the common coin protocol based on the LM-LJY $(f+1, n)$ threshold PRF scheme $(\mathsf{Eva}, \mathsf{Vrf}, \mathsf{FCom})$.

3.7 HALE: Achieving Adaptive Security using Local Coin

We study asynchronous BFT in the EPIC framework using ABA from local coins. Since ABA from local coins usually involves significantly more steps than those from common coins, it remains to see how the corresponding BFT protocols perform in the EPIC framework.

HALE follows the ACS framework and achieves adaptive security using local-coin based ABA protocols. Specifically, we use Bracha's ABA [37]. The pseudocode of Bracha's ABA is shown in Figure 3.6. We use $|\rho|$ to represent the number of delivered messages with value ρ. In addition, we add *vset* to represent valid binary values that can be accepted by a correct replica in each round. In the Bracha's ABA protocol, the key to correctness is that each replica accepts only *valid* values

Common public parameter setup: The same as Figure 3.3; besides, define two hash functions $H\colon \{0,1\}^* \to \mathbb{G}^2$, $H'\colon \mathbb{G}^2 \to \{0,1\}$.

$\mathsf{Eva}(pk, m, sk_i)$
  $(h_1, h_2) \leftarrow H(m)$
  parse sk_i as $\{A_k[i], B_k[i]\}_{k=1}^2$
  $z_i \leftarrow \prod_{k=1}^2 h_k^{-A_k[i]}$; $v_i \leftarrow \prod_{k=1}^2 h_k^{-B_k[i]}$
  return $\sigma_i \leftarrow (z_i, v_i)$.

$\mathsf{Vrf}(vk, m, vk_i, y_i)$
  parse σ_i as (z_i, v_i) and vk_i as (V_{1i}, V_{2i})
  $(h_1, h_2) \leftarrow H(m)$
  if $e(z_i, g) \cdot e(v_i, \hat{g}) \cdot \prod_{k=1}^2 e(h_1, v_{ki}) = 1$ **return** 1
  else return 0

$\mathsf{FCom}(vk, m, \{\sigma_j\}_{j\in S})$
  upon receiving $f+1$ valid PRF shares $\{\sigma_j\}_{j\in S}$
    for $j \in S$, parse σ_j as (z_j, v_j)
    $z \leftarrow \prod_{j\in S} z_i^{\Delta_{j,S}(0)}$; $v \leftarrow \prod_{j\in S} v_i^{\Delta_{j,S}(0)}$
    $\{\Delta_{j,S}$ denotes the Lagrange polynomial for $j \in S\}$
    return $\sigma \leftarrow H'(z, v)$.

Figure 3.4: The LM-LJY $(f+1, n)$ threshold PRF scheme $(\mathsf{Eva}, \mathsf{Vrf}, \mathsf{FCom})$.

upon receiving $Coin(\mathsf{sid})$
  $m \leftarrow \mathsf{sid}$
  $\sigma_i \leftarrow \mathsf{Eva}(pk, m, sk_i)$
  broadcast (i, m, σ_i)
upon receiving $f+1$ valid threshold PRF shares $\{\sigma_k\}_{k\in S}$ on m
  return $\sigma \leftarrow \mathsf{FCom}(vk, m, \{\sigma_k\}_{k\in S})$

Figure 3.5: The common coin protocol from the LM-LJY $(f+1, n)$ threshold PRF scheme, where sid is a session identifier and consists of an epoch number s, a round number r, and an ABA instance number $j \in [1..n]$.

in each step. Specifically, a value a replica accepts in each step must be congruent with the messages it received in the previous step/round. Therefore, to simplify the representation, we use *vset* to represent the *valid* values in each step. In the first phase of the first round, a correct replica can accept and deliver any values. In other phases and the first phase in other rounds, however, correct replicas accept a binary value only if it previously received the value from a sufficiently large fraction

Initialization
  $r \leftarrow 0$ {round}
  $est_0 \leftarrow v_{input}$ {set input for round 0 to initial input}
  $vset \leftarrow \{0, 1\}$ {valid binary values that will be accepted}
round r
  $\text{RBC}_0(est_r)$ {broadcast input using reliable broadcast}
  upon receiving $\text{RBC}_0()$ with $vals \in vset$ from $n - f$ replicas
    if $vals = \{\rho\}$, **deliver** ρ, $vset \leftarrow \{\rho\}$
    else
      $v = majority(vals)$
      $\text{RBC}_1(v)$ {move to the second step}
  upon receiving $\text{RBC}_1()$ with $vals \in vset$ from $n - f$ replicas
    if $|\rho| > n/2$, $v = \rho$, $vset \leftarrow \{\rho\}$
    else
      $v = \perp$, $vset \leftarrow \{0, 1\}$
    $\text{RBC}_2(v)$ {move to the third step}
  upon receiving $\text{RBC}_2()$ with $vals \in vset$ from $n - f$ replicas
    if $|\rho| \geq 2f + 1$, **deliver** ρ, $vset \leftarrow \{\rho\}$
    else if $|\rho| \geq f + 1$, $est_{r+1} = \rho$, $vset \leftarrow \{0, 1\}$
    else
      $c \leftarrow Random()$ {obtain a local coin}
      $est_{r+1} = c$, $vset \leftarrow \{0, 1\}$
  $r \leftarrow r + 1$

Figure 3.6: The Bracha's ABA protocol [37].

of replicas.

As shown in Figure 3.6, Bracha's ABA protocol has three phases. In each phase, each replica uses RBC to broadcast its value. In other words, each round involves 9 steps in total. In the first phase, every replica broadcasts its input value. If a replica receives values from $n - f$ replicas, it delivers a value ρ if ρ is the only value it has received. Otherwise, a replica uses the value from the majority of the $n - f$ replicas as input for the next phase. In the former case, a replica will only accept ρ value in the following phase. Otherwise, both 0 and 1 are considered *valid* which can be accepted in the next phase. In the second phase, if a replica receives

matching value ρ from more than $n/2$ replicas, it uses the value as input for the third phase. Otherwise, it broadcasts $\perp$ in the third phase, representing that it has previously received both 0 and 1 that might be valid. Finally, in the third phase, if a replica receives a value ρ from more than $2f+1$ replicas, it delivers ρ and will not accept any values other than ρ in the first phase of the following round. If a replica receives ρ from more than $f+1$ replicas, it uses ρ as input for the next round and considers both 0 and 1 as valid. Otherwise, a replica generates a random local coin, uses it as input for the next round, and considers both 0 and 1 as valid.

As Bracha's ABA relies on local coins, the expected rounds of termination is $O(2^n)$, which is significantly higher than $O(1)$ in ABA protocols that rely on common coins. Put in the ACS BFT framework, Bracha's ABA may not result in significantly longer latency or lower throughput, especially when there are no failures. This situation is mainly because Bracha's ABA may terminate in one round when a correct replica delivers $n-f$ $\text{RBC}_0(\rho)$. In other words, if all replicas use 1 as input for the ABA, the protocol will terminate in 9 steps (1 ABA round). In the ACS framework, replicas refrain themselves from using 0 for ABA instances before each replica terminates $n-f$ ABA instances with value 1.

Our implementation choice. Like all information-theoretically secure constructions, we need to consider how to instantiate concrete building blocks for HALE in a reasonable manner. For authenticated channels, the most natural method is to use message authentication codes (MACs), or digital signatures. For RBC, we decide to stick to the AVID broadcast. Note the AVID broadcast uses a collision-resistant hash function. Strictly speaking, one should not use hash functions, as hash

functions are cryptographic tools (though they are not based on some mathematical hardness problems). However, our thesis remains that they are simply instantiations of building blocks for HALE. If in the future the underlying MAC, digital signature, or hash function is found to be less intractable, one can easily replace it using one of secure schemes. There are numerous choices for them, as authentication and hash functions are basic tools in the information security field.

It is in sharp contrast to EPIC, which relies on a highly sophisticated cryptographic scheme—adaptively secure threshold PRF. So far, we have only one such realization and the realization is based on a non-standard pairing-based mathematical problem. Moreover, the realization is in the random oracle model (where the hash function is modeled a perfect random function), while our HALE instantiation is in the standard model.

3.8 Implementation

The entire EPIC library includes 13,000 lines of Python code, among which 900 lines of code are written for key distribution and 1,200 lines of code are used for evaluation. In total, we implemented four asynchronous BFT protocols summarized in Table 3.1. For both EPIC and EPIC-MMR, we use the BN256 pairing curve to understand the performance overhead incurred by threshold cryptography.

BEAT is our baseline protocol. We used the BEAT0 protocol, which leverages more efficient cryptography than HoneyBadgerBFT. While using BEAT0, we call our baseline protocol BEAT for simplicity.

Protocol	ABA	Common Coin	Transaction Selection	Liveness	Adaptive Security
BEAT	MMR [134]	CKS [44]	ETD	No	No
BEAT-Cobalt	Cobalt [127]	CKS [44]	ETD	Yes	No
EPIC-MMR	MMR [134]	LM-LJY [117, 122]	HYB	No	Yes
EPIC	Cobalt [127]	LM-LJY [117, 122]	HYB	Yes	Yes
HALE	Bracha's ABA	Bracha [38]	HYB	Yes	Yes

Table 3.1: The implemented asynchronous BFT protocols.

We use the BN256 curve [28] for the LM-LJY threshold PRF scheme. BN256 and BN254 curves achieve 110-bit security and have been widely used in many other BFT and blockchain systems (e.g., SBFT [80]). We modified the Charm Python library [22] to wrap a version of the relic C++ library [12] and then implemented the LM-LJY threshold PRF scheme using the modified Charm library.

To understand the overhead incurred by threshold cryptography, we also directly use Charm's PBC library to implement threshold PRF schemes two other pairing groups: the MNT224 (an asymmetric pairing group) and SS512 (a symmetric pairing group). Both groups have less than 100-bit security, which is below the well accepted level of security (and should not be directly used for real-world applications or critical infrastructure).

3.8.1 Average throughput vs. Total throughput.

To present our results better, we further distinguish total throughput TH in the conventional sense and the average throughput per replica AT. The latter is defined as the actual average delivered transactions per second proposed by each replica. In other words, the total throughput is the aggregation of average throughput per replica. Note that conventially BFT protocols under the partially synchrony model (e.g., PBFT [50]) only has total throughput. This difference is because contential BFT protocols are usually leader-based where the leader proposes transactions and other nodes decide whether they can reach a consensus about the results. Therefore, even when the network size f grows, in every epoch, only one batch of transactions are proposed. In comparison, in leaderless asynchronous BFT protocols, every node proposes a batch of transactions in each epoch. Therefore, the average throughput matches the throughput of leader-based protocols. In comparison, the total throughput of asynchronous protocols is the aggregated throughput, which potentially can grow as the network size grows.

Existing asynchronous BFT protocols report different throughput. Specifically, HoneyBadgerBFT and BEAT report different throughput numbers where HoneyBadgerBFT reported for TH (without considering overlapped transactions proposed by different replicas) and BEAT reported for AT. Although the (total) throughput represents the actual system throughput, we do find that both total throughput and average throughput are worth reporting. Specifically, the average throughput can better illustrate the performance downgradation when the network

size grows. In comparison, the total number of proposed transactions grows as the network size increases since replicas all propose transactions concurrently. Therefore, the total throughput does not necessarily downgrade as f increases.

3.9 Evaluation

Overview. We evaluate the protocols on Amazon EC2 using up to 91 virtual machines (VMs) in different regions across five continents. Each VM is the $t2.medium$ type with two vCPUs and 4GB memory, running Ubuntu 16.04. We evaluate the protocols in both LAN and WAN settings, where the VMs are launched in the same EC2 region in the LAN setting, and the VMs are evenly distributed in different regions in the WAN setting. We evaluate the protocols using different network sizes and batch sizes.

We evaluate the protocols with different network sizes. We use f to represent the network size, and the total number of replicas is $n = 3f + 1$. Recall B and $b = \lceil B/n \rceil$ are the batch size for the protocol and the batch size for transactions proposed by each replica, respectively. *All transactions are of size 250 bytes.*

When evaluating latency only, we have $b = 1$, where each replica proposes a single transaction. When evaluating throughput, we vary the size of b until the throughput reaches its peak and stabilizes.

The system throughput is evaluated according to the actual delivered transactions using real transaction buffers. In particular, overlapping transactions delivered are counted once. Unless stated otherwise, we let the transaction buffer at each

replica be $10 \cdot b$.

Our approach is more precise than HoneyBadgerBFT and BEAT. In their approaches, replicas use local random coins to generate independently distributed transaction sets from a large space, and the probability of any two sets being overlapping is negligible. Our approach is needed to understand the exact impact of using various transaction selection approaches.

We run each experiment five times, and each experiment runs 20 epochs. We then calculate the average results. For HYB in EPIC, we let μ be four and δ be one. Namely, we first run four epochs using random selections and then one epoch using the FIFO selection.

Our results show that EPIC achieves adaptive security with low overhead when f is small. In the WAN setting, EPIC has only 2%, 5%, 21% lower throughput than BEAT-Cobalt when $f = 1, 2, 5$, respectively. When f further grows, the performance overhead for EPIC compared with BEAT-Cobalt is significantly higher. When $f = 30$, EPIC achieves 68% lower peak throughput than BEAT-Cobalt.

Besides the conventional metrics (latency, throughput, and scalability), our evaluation also aims to identify the performance bottlenecks using a variety of experiments.

MMR ABA vs. Cobalt ABA. Both HoneyBadgerBFT and BEAT use the MMR ABA. The Cobalt ABA solves the liveness problem at the cost of an additional step in each round. We find that in all of our experiments, BEAT (using MMR) outperforms BEAT-Cobalt and EPIC-MMR outperforms EPIC (using Cobalt) in terms of both latency and throughput. We also find that the performance degradation

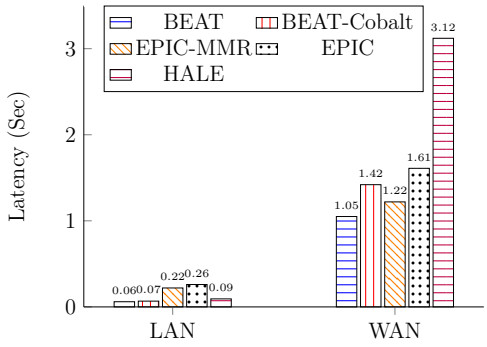


Figure 3.7: Latency for $f = 1$ in both LAN setting and WAN setting under no contention.

caused by the extra step in the LAN setting is small but certainly noticeable, while it becomes more visible in the WAN setting. This result is expected, as the network latency caused by the extra step has a more significant impact in the WAN setting.

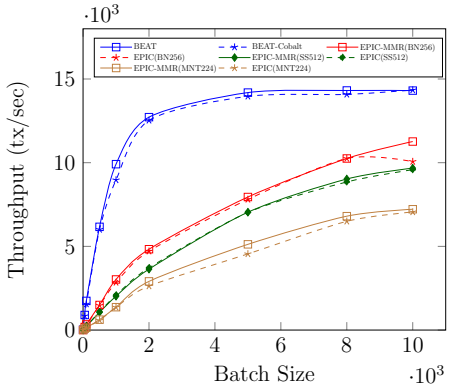


Figure 3.8: Throughput of BEAT, EPIC, HALE for $f = 1$ in the LAN setting.

Figure 3.8 and Figure 3.9 show the throughput of the BFT protocols for $f = 1$ in LAN and WAN environments, respectively. In the LAN setting, BEAT-Cobalt achieves 0.02% lower throughput than BEAT, and EPIC achieves 1% lower throughput than EPIC-MMR. In the WAN setting, BEAT-Cobalt achieves 2% lower throughput than BEAT and EPIC achieves 8% lower throughput than EPIC-MMR.

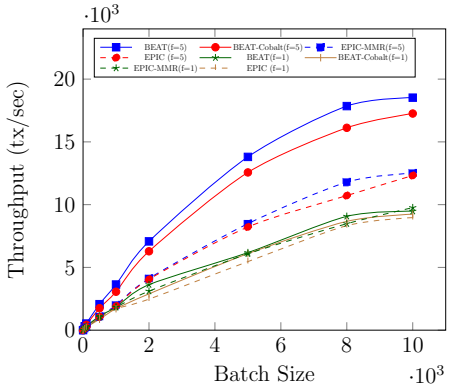


Figure 3.9: Throughput for $f = 1$ and $f = 5$ in the WAN setting.

As for latency, we show in Figure 4.9 BEAT-Cobalt has 16%-34% higher latency than BEAT and EPIC has 18%-31% higher latency than EPIC-MMR.

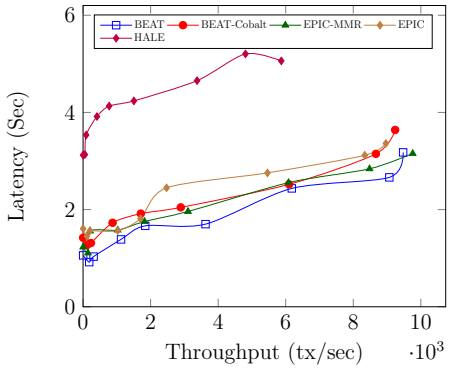


Figure 3.10: Latency vs throughput for $f = 1$ in the WAN setting.

Adaptive vs. Static security. We compare EPIC with BEAT-Cobalt, a live, asynchronous BFT protocol with static security. As shown in Figure 4.9-4.11, EPIC protocols consistently achieve lower throughput and higher latency than BEAT-Cobalt. This finding is due mainly to the fact that LM-LJY involves more expensive cryptographic computation. The performance difference in the WAN environment between the two protocols is relatively small: EPIC has 13% higher latency and

5% lower throughput than BEAT-Cobalt. In the LAN setting, the difference is considerably larger where the peak throughput of EPIC is 29% lower than that of BEAT-Cobalt. Besides, we show latency vs. throughput in Figure 3.10. We observe that for both BEAT and EPIC, their latency increases dramatically when the throughput is close to 9,000 tx/sec.

Performance breakdown for threshold cryptography. To understand the performance bottlenecks caused by the underlying threshold PRF components, we test the performance of various threshold PRF protocols, including CKS, LM-LJY, and Boldyreva's [36] (used in HoneyBadgerBFT). CKS uses NIST P256 curves, while the other two use BN256 pairing curves. We assess the latency of various schemes on a local machine (3.2GHz CPU, 16GB memory, Intel core i7) for $f = 1$ and $n = 4$. The result is summarized in Table 3.2. Among CKS, LM-LJY, and Boldyreva's, CKS has the lowest latency as it uses efficient regular elliptic curves. The performance breakdown result for the three threshold PRF schemes matches the system throughput result in Figure 3.8 and confirms the performance comparison result [70] between BEAT and HoneyBadgerBFT.

BFT using LM-LJY with different pairing curves. EPIC uses BN256 curves to implement the LM-LJY threshold PRF scheme. To understand the exact cryptographic overhead and impact of EPIC further, we additionally implement EPIC using three different pairing curves: EPIC (i.e., EPIC(BN256)), EPIC(SS512), and EPIC(MNT224). Moreover, for comparison, we also implement EPIC-MMR(BN256), EPIC-MMR(SS512), and EPIC-MMR(MNT224).

schemes	Eva	Vrf	FCom
CKS (NIST P256)	0.138	0.191	0.152
Boldyreva's (BN256)	0.233	4.190	0.123
LM-LJY (BN256)	1.849	8.071	0.231
LM-LJY (MNT224)	2.578	14.432	1.445
LM-LJY (SS512)	3.873	2.316	2.069

Table 3.2: Latency (ms) of different $(2, 4)$ threshold PRF schemes (Eva, Vrf, FCom).

We demonstrate in Table 3.2 the latency breakdown of the LM-LJY scheme using BN256, SS512, MNT224. We assess the throughput of EPIC and EPIC-MMR using the three curves and compare the performance with BEAT and BEAT-Cobalt in the LAN setting, the results of which are shown in Figure 3.8. Among all these protocols, the ones using BN256 curves achieve the highest throughput and the ones using MNT224 achieve the lowest throughput. For instance, the peak throughput of EPIC-MMR using MNT224 is 35% lower than that of EPIC-MMR using BN256. The peak throughput of EPIC-MMR using MNT224 curve is rather similar to that of EPIC which uses the Cobalt ABA and MNT224 curve. Among the three pairing curves, BN256 is not only the most secure one but also the most efficient one. Our evaluation thus validates our design choice of using BN256. Besides, the result in Table 3.2 explains (perfectly) the performance difference among the evaluated protocols.

Transaction selection. We evaluate the transaction selection approach (HYB) for EPIC in both LAN and WAN environments. In our experiments, we let δ be

one and run 100 epochs with different μ values. We run the experiments using $f = 1$ and varying b sizes. As we observe similar results for different b sizes, we selectively present the results for $b = 1000$ in Table 3.3. Our experiment shows our HYB strategy provides efficient trade-offs between latency and throughput: if μ is small, the system achieves lower throughput in both LAN and WAN settings.

μ	LAN	WAN	μ	LAN	WAN
2	5684.10	1147.02	3	6650.40	1341.90
4	7104.66	1433.63	5	7388.17	1490.97
10	7955.31	1605.40	15	8182.70	1651.24
20	8239.33	1662.64	50	8409.91	1697.20

Table 3.3: Throughput of EPIC in both LAN setting and WAN setting when $f = 1$ and $\delta = 1$. Each experiment is run for 100 epochs.

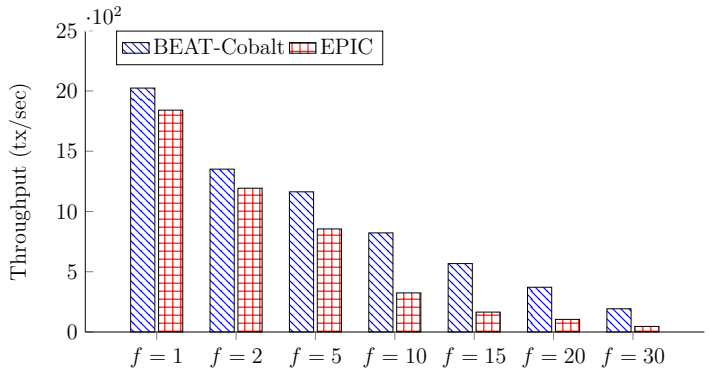


Figure 3.11: Average throughput per replica of BEAT-Cobalt and EPIC when $b = 5000$ in the WAN setting as f increases.

Scalability. We evaluate the scalability of all our implemented protocols by varying f from 1 to 30. We report the average throughput in Figure 3.11 and the (total)

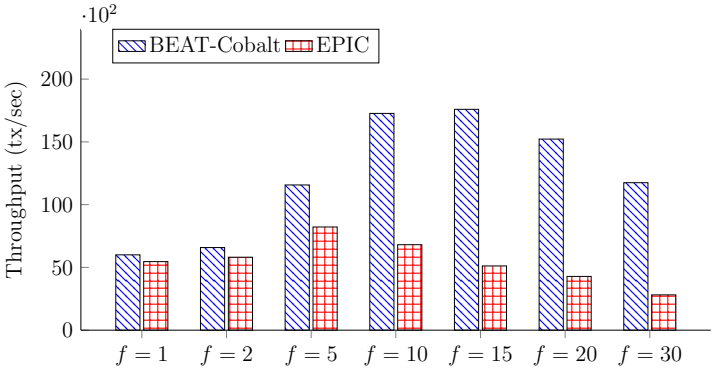
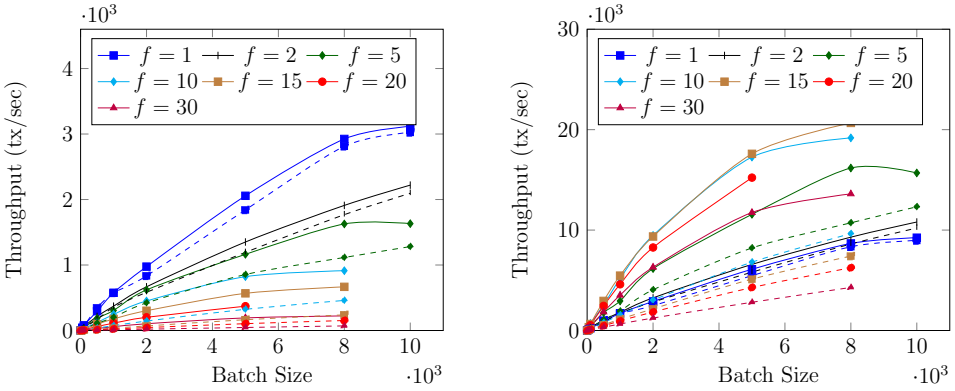


Figure 3.12: Throughput of BEAT-Cobalt and EPIC when $b = 5000$ in the WAN setting as f increases.



(a) Average throughput per replica for BEAT-Cobalt and EPIC in the WAN setting as f increases. BEAT-Cobalt and EPIC are represented in solid and dashed lines, respectively.

(b) Throughput for BEAT-Cobalt and EPIC in the WAN setting as f increases. BEAT-Cobalt and EPIC are represented in solid and dashed lines, respectively.

throughput in Figure 3.12 and Figure 4.11. The average throughput is the average of the actual delivered transactions per second of the replicas. The total throughput is the throughput in the conventional sense.

We evaluate the throughput by varying b and observe a similar trend in all protocols. We report the throughput of BEAT-Cobalt and EPIC as b increases in

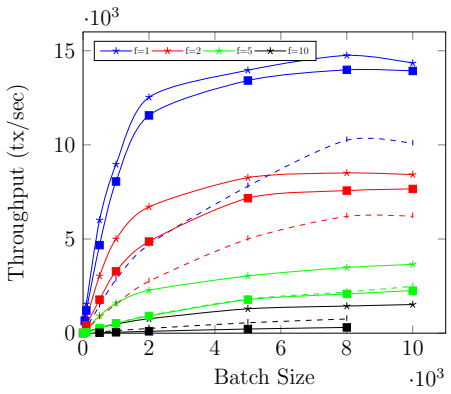
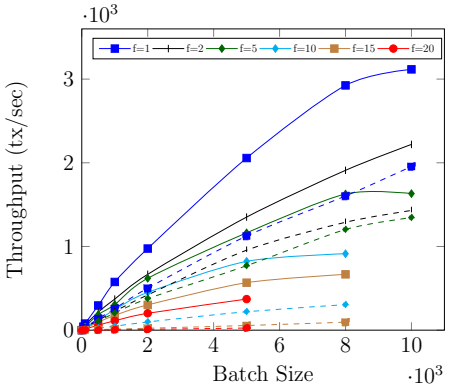
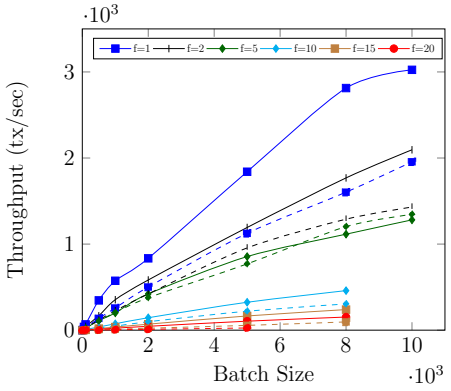


Figure 3.13: Average throughput for BEAT-Cobalt, EPIC, HALE in the LAN setting as f increases. BEAT-Cobalt, EPIC, HALE are represented in star, dashed and square lines, respectively.



(a) Average throughput for BEAT-Cobalt and HALE in the WAN setting as f increases. BEAT-Cobalt and HALE are represented in solid and dashed lines, respectively.



(b) Average throughput for EPIC and HALE in the WAN setting as f increases. EPIC and HALE are represented in solid and dashed lines, respectively.

Figure 4.11. We also report the average and total throughput for BEAT-Cobalt and EPIC for $b = 5000$. First, all the protocols achieve lower average throughput per replica when f grows. When $f = 1$, EPIC has 2% lower average throughput per replica than BEAT-Cobalt. When $f = 2$ and 5, EPIC has about 5% and 21%

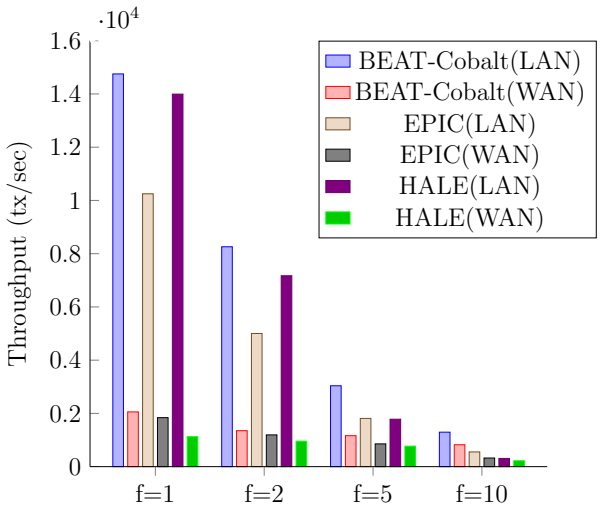


Figure 3.14: Average throughput for BEAT-Cobalt, EPIC and HALE in the LAN and WAN setting when $b = 5000$ as f increases.

lower average throughput per replica, respectively. But if f further increases, the difference for average throughput becomes significantly larger. (When $f = 10$, the peak (total) throughput of EPIC is around $10,000$ tx/sec.)

For the total throughput, however, we find that as f increases, the throughput for all asynchronous BFT protocols first increases and then decreases. This is (quite) expected, though it has been formally reported by HoneyBadgerBFT or BEAT. Indeed, when f first grows, the number of concurrently proposed transactions grows significantly, making the system throughput higher than that with smaller network sizes. When f grows further, the average throughput per replica becomes much smaller; even if the average throughput per replica gets multiplied by a large $dtx \geq (n - f)$ (the number of ABA instances that deliver 1), the total throughput remains smaller. In our experiments (using $b = 5000$), we observe that BEAT-Cobalt achieves the largest throughput when $f = 15$, and the throughput of

EPIC is the largest when $f = 5$.

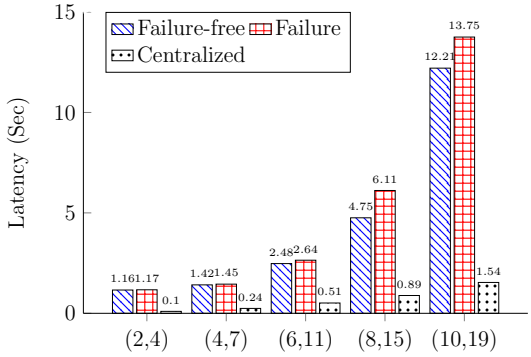


Figure 3.15: Latency of (t, n) distributed key generation vs. centralized key generation in the LAN setting.

HALE. We evaluate the throughput and scalability of HALE and compare the performance with BEAT-Cobalt and EPIC. We evaluate the throughput of the protocols in both LAN and WAN settings to understand the overhead of HALE fully, the asynchronous BFT protocol using local coins. Since we aim to present scalability (throughput as f increases), we present only the average throughput of the protocols where the total throughput is roughly average throughput times $n - f$.

We first compare HALE, EPIC, and BEAT-Cobalt in LAN settings and show the results in Figure 3.13. When f is small (in our experiments $f < 5$), the throughput of HALE is close to that of BEAT-Cobalt and is much higher than that of EPIC. Specifically, when $f = 1$ and $f = 2$, HALE achieves 38% and 22% higher average throughput than EPIC. This finding is expected since in a LAN environment, the network latency is low. Therefore, the communication overhead of HALE is low. Furthermore, HALE does not involve any threshold cryptography, which explains why the throughput of HALE is much higher than that of EPIC. The results roughly

match the observation from previous work [132].

When f increases, the performance of the three protocols is similar. This result is because the throughput of all protocols degrades as f increases. When f is greater than 5, the average throughput of EPIC is higher than that of HALE. This outcome is mainly because Bracha's ABA in HALE involves nine steps in each round, which results in significantly higher latency. When f is big, the communication overhead is high for HALE.

The performance in WAN environments is different. As shown in Figure 3.14(a), HALE achieves significantly lower throughput and scalability than BEAT-Cobalt in the WAN setting. When $f = 1$ and $f = 2$, HALE has about 37% and 35% lower peak average throughput compared to BEAT-Cobalt, respectively. On the other hand, HALE also achieves lower throughput than EPIC, as shown in Figure 3.14(b). When $f = 1$ and $f = 2$, HALE has about 35% and 31% lower throughput than EPIC. When f increases, the difference also becomes larger. This result is mainly because the Bracha's ABA in HALE creates high communication overhead in the WAN setting.

We also show the peak throughput of the three protocols in both LAN and WAN, as shown in Figure 3.14. The performance of all protocols in the LAN setting is significantly higher than that in the WAN setting. Even when $f = 10$, the LAN setting still has about 57%, 70% and 40% higher average throughput than the WAN setting for BEAT-Cobalt, EPIC and HALE, respectively. The peak average throughput for HALE where $f = 1$ is 13,999 tx/sec in the LAN setting and 1,125 tx/sec in the WAN setting. In other words, the peak total throughput is 42,000

tx/sec and 3,400 tx/sec in LAN and WAN settings, respectively.

Distributed key generation. Figure 3.15 summarizes the latency of our $(f+1, n)$ distributed key generation protocol in LAN environments. We evaluate the failure-free scenario, where all replicas are correct and the non-disqualified set includes all replicas, and the failure scenario, where there exists a single malicious replica. We also compare the two scenarios with a centralized key generation scenario where there is no interaction. Since the distributed key generation protocol runs in synchronous environments, we test only the optimal scenario where the timer equals the message processing and message transmission delays. Therefore, our evaluation result can be used to guide the timer setup for the protocol. One should setup much larger timers in practice. As shown in Figure 3.15, the distributed key generation incurs much higher latency compared to the centralized approach. The performance difference between the failure scenarios and the failure-free scenarios is noticeable but comparatively small.

3.10 Conclusion

We design and implement the first efficient asynchronous BFT protocols achieving adaptive security: EPIC (in the computational security model) and HALE (in the stronger information-theoretic security model). We evaluate both EPIC and HALE in both LAN and WAN environments. We show that EPIC is not much slower than asynchronous BFT protocols with static security; we also show while HALE is in general less efficient than EPIC, it outperforms EPIC in the LAN setting for $n \leq 16$.

Chapter 4

MiB: Asynchronous BFT with More Replicas

4.1 Overview

State machine replication (SMR) is a popular software technique achieving high availability and strong consistency guarantees in today's distributed applications (e.g., Apache ZooKeeper [90]), Google's Spanner [41]). Byzantine fault-tolerant SMR (BFT) is known as *the* model for permissoned blockchains, where the replicas need to authenticate themselves but do not necessarily trust each other. BFT is also used in permissionless blockchains (where nodes may join and leave the systems dynamically) to enhance the performance and achieve finality (e.g., [18, 104, 105, 126, 143, 168]).

Different from partially synchronous BFT protocols, asynchronous BFT protocols do not rely on any timing assumptions and are therefore more robust against performance, timing, and denial-of-service attacks. For this reason, many asynchronous BFT (atomic broadcast) protocols have been proposed [19, 31, 32, 43, 45, 59, 112, 133, 149]. In particular, several asynchronous BFT protocols proposed recently, HoneyBadgerBFT [130], BEAT [70], EPIC [119], and Dumbo [85], have comparable performance as partially synchronous BFT protocols (e.g., PBFT [50]) and can scale to around 100 replicas. These efficient protocols follow the asynchronous common subset (ACS) framework [31]. The ACS framework consists of a

reliable broadcast (RBC) component and an asynchronous binary agreement (ABA) component. HoneyBadgerBFT, BEAT, EPIC, and Dumbo all use RBC and ABA and are different only in concrete instantiations.

All these efficient asynchronous BFT protocols using RBC and ABA assume *optimal resilience*. Namely, if the system has n replicas, it tolerates $f < n/3$ Byzantine failures. Even in the best-case scenario, each asynchronous BFT epoch has at least six steps and the expected number of steps is much higher.[1] The situation is in sharp contrast to partially synchronous BFT protocols which usually have fewer steps (PBFT [50], for instance, terminates in three steps in the worst-case scenario).

We propose MiB, faster asynchronous BFT protocols with *suboptimal resilience*, including MiB5 (the case of $n \geq 5f+1$) and MiB7 (the case of $n \geq 7f+1$). Our technique is *generic* and can be applied to *all* of the four state-of-the-art asynchronous BFT protocols (HoneyBadgerBFT, BEAT, EPIC, and Dumbo). To illustrate our approach, we use BEAT as the underlying protocol, as BEAT is simpler than EPIC and Dumbo, and has the most efficient open-source implementation available [3]. (In fact, MiB can be based on any ACS instantiation or any asynchronous BFT using RBC and ABA.) For both MiB5 and MiB7, each epoch may terminate in as few as just *three* steps in the best-case scenario.

[1] In the best-case scenario, ACS includes a RBC phase with n parallel RBC instances and an ABA phase with n parallel ABA instances. RBC takes three steps whether using Bracha's broadcast [37] or the AVID broadcast [46]. The state-of-the-art ABA construction, the Cobalt ABA [127], has three or four steps in each round and the protocol may terminate in one or several rounds.

MiB techniques in a nutshell. At the core of MiB are new RBC constructions and new ABA combinations with suboptimal resilience, enabling faster termination and higher throughput.

For MiB5, we devise MBC, an erasure-coded version of Imbs and Raynal's RBC (IR RBC) [62] which terminates in two steps by requiring $n \geq 5f + 1$. MBC is bandwidth-efficient and step-optimal. In contrast, previous asynchronous BFT protocols use Bracha's broadcast or the AVID broadcast [46], either of which completes in three steps. MBC integrates the technique of AVID (using Merkle tree) and we formally prove the correctness of MBC. For MiB5, we instantiate the ABA construction in the ACS framework using a new combination of Bosco's weakly one-step ABA (W1S) [154] that requires $n \geq 5f + 1$ and the Cobalt ABA [127]. In ideal situations, the new ABA construction terminates in just one step; otherwise, it falls back to the state-of-the-art Cobalt ABA.

For MiB7, considering the step-optimal property of MBC, one may intuitively use MBC, even if there are more than $7f + 1$ replicas. We show that we can achieve better performance by asking a fraction of replicas to be passive learners rather than active RBC participants. Our new RBC construction, MBC with learners, or simply MBC-L, involves (much) fewer messages. We formally prove the correctness of MBC-L. For ABA, we combine Bosco's strongly one-step ABA (S1S) that requires $n \geq 7f + 1$ and the Cobalt ABA.

A (powerful) programming and evaluation platform. We build an asynchronous BFT programming and evaluation platform that fulfills two goals. First, the platform allows us to answer important research questions. For instance, how

could one be certain that our RBC and ABA choices would perform as expected? More importantly, do we have other combinations that would lead to BFT with better performance? The platform allows mixing and matching different RBC and ABA primitives. Our framework has a flexible but unified API and is highly expressive: in total, we design and implement seven fully-fledged asynchronous BFT instances using different RBC and ABA combinations.

Second, the platform allows us to perform experiments under failures and attacks. To the best of our knowledge, while state-of-the-art asynchronous BFT protocols, including HoneyBadgerBFT, BEAT, EPIC, and Dumbo, claim that they are more robust than partially synchronous BFT protocols in failure scenarios, no experiments are provided to validate the claims. The only framework we know that does so is RITAS [133], which performs evaluations for one specific asynchronous BFT protocol using less than 10 replicas in LANs. Our platform can, however, perform experiments under various failure scenarios (crash, Byzantine, attacks) and allow us to compare different BFT protocols in a unified framework and systematic manner.

Our contributions. We summarize our contributions in the following:

- We design new distributed system primitives with suboptimal resilience, including new RBC constructions and ABA combinations. In particular, we provide an erasure-coded version of IR RBC using Merkle tree and provide a learner-version of RBC (where some replicas are passive learners). We formally prove the correctness of the new RBC constructions.

- We build a highly flexible MiB framework allowing mixing and matching different RBC and ABA primitives. The framework consists of asynchronous BFT protocols described (MiB5 and MiB7) and five other variants. We provide meaningful tradeoffs among various situations, echoing the well-known claim that there is no "one-size-fits-all" BFT.
- We design experiments for asynchronous BFT protocols in failure and attack scenarios. This work is the first systematic evaluation for these recent asynchronous BFT protocols using the ACS framework.
- We have evaluated all seven MiB protocols and their competitors on Amazon EC2 with hundreds of experiments using up to 140 instances. We show that almost all MiB instances, in particular, MiB5 and MiB7, are much more efficient, in terms of both latency and throughput than their asynchronous BFT counterparts. Moreover, we show existing asynchronous BFT protocols, not just MiB protocols, are indeed robust against failures and attacks. We report many interesting results for different scenarios.

4.2 Related Work

BFT with suboptimal resilience. A number of BFT protocols assume $n > 3f+1$. For instance, Q/U requires $5f + 1$ replicas to tolerate f failures and achieves fault-scalability that tolerates increasing numbers of failures without largely decreasing performance [17]. BChain5 uses $5f + 1$ replicas to simplify the failure detection mechanism and remove the need for replica reconfiguration [68]. Zyzzyva5 uses

$5f+1$ replicas and trades the number of replicas in the system against performance in the presence of faults [107]. FaB Paxos [128] is an efficient partially synchronous BFT protocol using $5f+1$ replicas and having 3 communication steps per request.

Efficient asynchronous atomic broadcast and BFT. Most of the efficient asynchronous atomic broadcast (BFT) protocols follow Ben-Or's ACS framework [31], including SINTRA [45], HoneyBadgerBFT [130], BEAT [70], EPIC [119], and Dumbo [85]. They are different only in concrete instantiations. SINTRA, HoneyBadgerBFT, BEAT, and Dumbo achieve static security, where the adversary needs to choose the set of corrupted replicas before the execution of the protocol. In contrast, EPIC attains stronger adaptive security, where the adversary can choose to corrupt replicas at any moment during the execution of the protocol. Dumbo devises a new way of instantiating the ACS framework by using fewer ABA instances and achieves better performance. There are, however, efficient asynchronous BFT protocols that do not follow the ACS framework, including, for instance, RITAS [133]. DBFT [160] relies on an asynchronous framework but works in partially synchronous environments and is very efficient.

Asynchronous binary agreement (ABA) with optimal resilience. Beginning with Ben-Or [31] and Rabin [147], a significant number of ABA protocols have been proposed [34, 38, 44, 49, 79, 127, 134, 147, 154, 155, 159, 171]. Cachin, Kursawe, and Shoup (CKS) [44] proposed an ABA with optimal resilience and $O(n^2)$ message complexity. Mostefaoui, Moumen, and Raynal (MMR) [134] proposed the first signature-free ABA with the same message complexity as the CKS ABA [44].

The MMR ABA protocol was used by HoneyBadgerBFT, BEAT, and Dumbo. It was later reported that the MMR ABA has a liveness issue when being instantiated using any coin-flipping protocols known [4]. The Cobalt ABA protocol resolves the issue at the price of one more step for each round [127]. The Cobalt ABA is used in EPIC and an open-source implementation of BEAT [3]. Dumbo recently updated its ePrint version [86] by using the Cobalt ABA (code still unavailable).

ABA with suboptimal resilience. Assuming $n \geq 5f+1$, Berman and Garay [34] presented a common coin based ABA protocol that has only two steps in each round. The ABA protocol by Friedman, Mostefaoui, and Raynal (FMR) [146] extended BG and reduced the number of steps within a round to one. Song and van Renesse [154] proposed Bosco that terminates in one step in ideal situations, a protocol that we use in this paper.

4.3 System and Threat Model

We consider a Byzantine fault-tolerant state machine replication (BFT) protocol with n replicas, at most f of which may exhibit arbitrary behavior (Byzantine failures). Most of BFT protocols assume optimal resilience with $n \geq 3f+1$. In this work, we consider suboptimal resilience with $n \geq 5f+1$ and $n \geq 7f+1$.

In BFT, replicas deliver transactions (requests) submitted by clients and send replies to clients. A BFT protocol should satisfy the following properties:

- **Agreement**: If any correct replica delivers a transaction tx, then every correct replica delivers tx.

- **Total order**: If a correct replica has delivered transactions $\langle tx_0, tx_1, \cdots, tx_j \rangle$ and another has delivered $\langle tx_0^{'}, tx_1^{'}, \cdots, tx_{j'}^{'} \rangle$, then $tx_i = tx_i^{'}$ for $0 \leq i \leq min(j, j^{'})$.
- **Liveness**: If a transaction tx is submitted to $n - f$ replicas, then all correct replicas will eventually deliver tx.

According to the timing assumptions, BFT protocols can be divided into three categories: *asynchronous*, *synchronous*, or *partially synchronous* [72]. Asynchronous BFT systems make no timing assumptions on message processing or transmission delays. Synchronous BFT systems have a known bound on message processing delays and transmission delays. Partially synchronous BFT systems lie in-between: messages will be delivered within a time bound, but the bound may be unknown to anyone. Asynchronous BFT protocols are inherently more robust than other BFT protocols. We consider purely asynchronous systems making no timing assumptions on message processing or transmission delays.

4.4 Building Blocks

This section reviews the building blocks for MiB.

Erasure coding. An (m, n) maximum distance separable (MDS) erasure coding scheme can encode m data blocks (fragments) into n ($n \geq m$) coded blocks, and all blocks can be recovered from any m-size subset of coded blocks via a decode algorithm. We use MDS erasure coding by default.

Byzantine reliable broadcast (RBC). In RBC, a sender broadcasts a message to all other replicas in a group. An asynchronous RBC protocol satisfies the following

properties:

- **Validity**: If a correct replica p broadcasts a message m, then p eventually delivers m.
- **Agreement**: If some correct replica delivers a message m, then every correct replica eventually delivers m.
- **Integrity**: For any message m, every correct replica delivers m at most once. Moreover, if the sender is correct, then m was previously broadcast by the sender.

Asynchronous binary agreement (ABA). In ABA, each replica has a binary value $v \in \{0, 1\}$ (a vote) as the input, and correct replicas eventually deliver the same binary value as the output. ABA guarantees the following properties.

- **Validity**: If a correct replica delivers a value v, the v was proposed by at least one correct replica.
- **Agreement**: If a correct replica delivers v and another correct replica delivers v', then $v = v'$.
- **Termination**: All correct replicas eventually deliver a value with probability 1.
- **Unanimity**: If all correct replicas input the same initial value v, then a correct replica delivers v.

ABA protocols proceed in rounds, each of which includes several steps. We define one-step ABA as one ensuring one-step communication under the unanimity property, i.e., replicas terminate in one step if all correct replicas propose the same binary value [154].

4.5 Technical Overview

State-of-the-art asynchronous BFT protocols, such as HoneyBadgerBFT, BEAT, EPIC, and Dumbo, follow the asynchronous common subset (ACS) framework which includes a RBC phase and an ABA phase.

We consider asynchronous BFT protocols (collectively called MiB) using the ACS framework with *suboptimal resilience*. At the core of the MiB protocols are new RBC constructions and new ABA combinations with suboptimal resilience. These new RBC constructions and ABA combinations are rather generic and can be applied to any ACS instantiations. For illustrative purposes, we follow the BEAT workflow (as depicted in Figure 2.1): BEAT is slightly simpler than EPIC and Dumbo and has the most efficient open-source implementation available [3].

ABA with suboptimal resilience. HoneyBadgerBFT, BEAT, and Dumbo used the MMR ABA with optimal resilience [134]. The MMR ABA includes two to three steps in each round. The protocol, however, was found to have a liveness issue [4]. The Cobalt ABA protocol resolves the problem at the price of one more step for each round [127]. EPIC uses the Cobalt ABA, and some asynchronous BFT libraries (e.g., BEAT [3]) have updated their implementation using the Cobalt ABA.

For MiB, we use Bosco's one-step ABA protocols [154]: weakly one-step ABA (W1S) using $n \geq 5f + 1$ replicas and strongly one-step ABA (S1S) using $n \geq 7f + 1$ replicas. Both W1S and S1S terminate in as minimum as one single step (one round). W1S achieves this property when all replicas propose the same binary input (contention-free) and there are no faulty replicas (failure-free). S1S achieves

this property under the contention-free condition but does not assume a failure-free condition. Bosco's ABA needs to run a backup ABA protocol when the conditions are not satisfied. We thus use a combination of W1S and the Cobalt ABA for MiB5 (the case of $n \geq 5f + 1$) and a combination of S1S and the Cobalt ABA for MiB7 (the case of $n \geq 7f + 1$).

RBC with suboptimal resilience. We devise MBC, an erasure-coded version of IR RBC [62] which terminates in two steps by requiring $n \geq 5f + 1$. MBC is bandwidth-efficient and step-optimal. The low bandwidth property has shown to be extremely useful for the performance of HoneyBadgerBFT and more so for BEAT; it turns out that the step-optimal property also improves the system performance.

RBC with learners. When $n \geq 5f + 1$, we have already obtained a step-optimal RBC (MBC) which terminates in two steps. There does not exist a one-step RBC, no matter how many replicas one uses. While one may consider simply using MBC directly for the case of $n \geq 7f + 1$, we can actually do better.

We use the concept of *learners* (see, e.g., Paxos [116]) and propose *RBC with learners*. If there is a RBC that requires n_1 replicas and an ABA that requires n_2 replicas, where $n_2 > n_1$, then n_1 replicas are active replicas and $n_2 - n_1$ replicas are learners. The learners do not actively participate in RBC but only learn the results. When an active replica delivers a message, it forwards the message to all learners. A learner delivers a message when it receives $n_1 - f$ matching messages and then enters the ABA phase. Compared to regular RBC, RBC with learners reduces the number of messages transmitted. RBC with learners is a general primitive. The

specific MBC extension is called MBC-L, and the AVID extension is called AVID-L.

The MiB framework. To examine if the above new primitives or combinations could improve performance as expected, we build a highly modular and expressive framework. In such a framework, we mix and match various RBC and ABA primitives to design and implement seven asynchronous BFT protocols with suboptimal resilience. The framework is modular: various components are programmed to fit in a unified but flexible standard API (for varying f and n's). It is the framework that allows us to have a clear picture on the performance bottlenecks for all MiB instances, validate our theoretical design, and help find meaningful trade-offs.

4.6 MiB

4.6.1 MiB framework

This section describes the MiB protocols. In MiB, we propose new RBC primitives and new ABA combinations as building blocks. The building blocks can be applied to all asynchronous BFT systems using the ACS framework or any asynchronous BFT using RBC and ABA. To illustrate our approach, MiB is built on top of BEAT which has the most efficient open-source implementation available [3].

MiB protocols are different only in concrete RBC and ABA instantiations. So when describing MiB in general, we use RBC and ABA in a black-box manner. Figure 4.1 depicts the MiB framework which is the same as BEAT. MiB proceeds in epochs numbered by r (initially, 0). In an epoch, replicas choose a subset of transactions as a *proposal* from their transaction pool and agree on a set containing

the union of the proposals of at least $n - f$ replicas. We define B as the batch size of the transactions for an epoch; the batch size for a replica $b = \lceil B/n \rceil$. Replicas first run a RBC phase to broadcast their proposals. Then they run an ABA phase, where n parallel ABA instances are invoked. The i-th ABA instance agrees on whether the proposal of replica p_i has been delivered in the RBC phase. If a correct replica p_j terminates the i-th ABA instance with 1, the proposal from p_i is delivered. Otherwise, the proposal is not included. We ensure that at least $n-f$ ABA instances terminate with 1 and the union of the transactions from at least $n - f$ replicas are delivered. To achieve this condition, each replica abstains from proposing 0 until $n - f$ ABA instances have been delivered by the replica. As in BEAT, MiB uses threshold encryption to avoid transaction censorship and achieve liveness (the pseudocode of threshold encryption is not shown in Figure 4.1).

4.6.2 MiB5

For MiB5, we instantiate the ABA component in the ACS framework using a new combination of Bosco's weakly one-step ABA (W1S) [154] that requires $n \geq 5f + 1$ and the Cobalt ABA [127]. We also devise MBC, an erasure-coded version of IR RBC [62] which completes in two steps (optimal) and requires $n \geq 5f + 1$.

Weakly one-step (W1S). The state-of-the-art ABA for the $n \geq 3f + 1$ case, the Cobalt ABA, requires at least three steps in each round. For the case of $n \geq 5f + 1$, we use the Bosco's weakly one-step ABA protocol (W1S) [154].

W1S guarantees that if there are *no faulty* replicas and all replicas propose the

Algorithm: MiB Protocol (Replica p_i)

Initialization

let B be the batch size parameter
let $\mathsf{buf} \leftarrow \emptyset$ be a transaction buffer
let $\{\text{RBC}_j\}_{j \in n}$ and $\{\text{ABA}_j\}_{j \in n}$ be the j-th instance for RBC_j and ABA_j
let $output \leftarrow \emptyset$ be the output buffer
let $r \leftarrow 0$ be the epoch number

epoch r

select $b = \lceil B/n \rceil$ random transactions from the first B elements in buf as a proposal $value$
upon input value $value$
  input $value$ to RBC_i
upon delivery of $value_j$ from RBC_j
  if ABA_j has not yet been provided input, input 1 to ABA_j
upon delivery of 1 from ABA_j and $value_j$ from RBC_j
  $output \leftarrow output \cup value_j$
upon delivery of 1 from at least $n - f$ ABA instances
  for each ABA_j instance that has not been provided input
    input 0 to ABA_j
upon termination of all the n ABA instances
  deliver $output$
$r \leftarrow r + 1$

Figure 4.1: The MiB algorithm for p_i.

Algorithm: W1S/S1S

Initialization

$r \leftarrow 0$ {round}
v_p {input value}

round r

broadcast $\mathsf{bval}(v_p)$ {broadcast input}
upon receiving $\mathsf{bval}(v)$ from $n - f$ replicas
if more than $\lceil (n + 3f)/2 \rceil$ $\mathsf{bval}(v)$ messages contain the same value v
  deliver v {terminate the protocol}
if more than $\lceil (n - f)/2 \rceil$ $\mathsf{bval}(v)$ messages contain the same value v, and there is only one such value v
  $v_p \leftarrow v$
backup-ABA(v_p)

Figure 4.2: The algorithm for W1S and S1S.

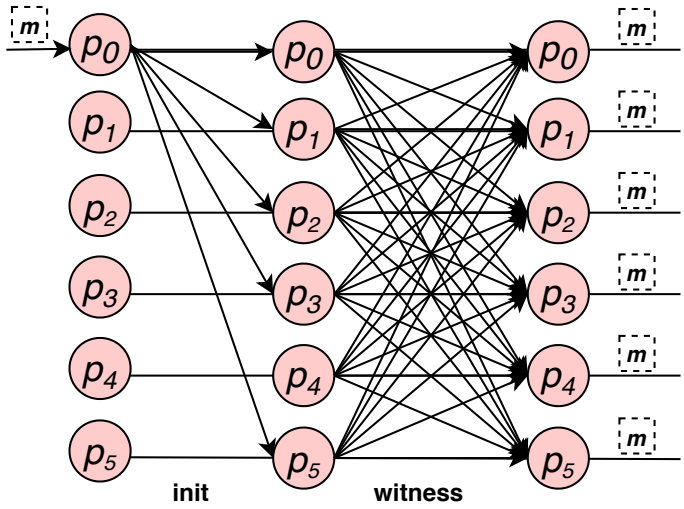


Figure 4.3: The MBC workflow, where p_0 broadcasts a message m and there are no faulty replicas.

same initial value v, then all the correct replicas deliver v and terminate the protocol in one step (one round). Figure 4.2 describes the pseudocode of W1S protocol. Each replica has a binary input v_p to the ABA. In the first step, each replica p_i broadcasts a message $\mathsf{bval}(v_p)$. A replica p_i waits for $\mathsf{bval}(v)$ from $n - f$ replicas (including itself). If more than $\lceil (n + 3f)/2 \rceil$ $\mathsf{bval}(v)$ messages include the same value v, a replica delivers v and terminates the protocol. If more than $\lceil (n - f)/2 \rceil$ $\mathsf{bval}(v)$ messages include the same v, a replica sets its local value to $v_p = v$. If the replica does not deliver any value in one step, it invokes a backup ABA protocol. In MiB5, we use the Cobalt ABA protocol [127] as the backup ABA protocol.

MBC. MBC is an erasure-coded version of IR RBC [62] which completes in two steps. MBC is bandwidth-efficient and step-optimal. We show the MBC workflow in Figure 4.3 and the MBC pseudocode in Figure 4.4.

As depicted in Figure 4.4, to broadcast a message m, a replica p_i applies the $(n - 2f, n)$ erasure coding scheme to generate n blocks, where the j-th block is

Algorithm: MBC (Replica p_i)

upon input(m) {if $p_{sender} = p_i$}
  let l_j be the j-th block of $(n-2f, n)$ erasure coding scheme applied to m
  h is the root of the Merkle tree for the $\{l_j\}_{j\in[0..n-1]}$ blocks
  send $\mathsf{init}(h, b_j, l_j)$ to each p_j, where b_j is the j-th Merkle tree branch

upon receiving $\mathsf{init}(h, b_i, l_i)$ from p_{sender}
  broadcast $\mathsf{witness}(h, b_i, l_i)$

upon receiving $n-2f$ valid $\mathsf{witness}(h, b_j, l_j)$
  interpolate $\{l'_j\}$ from the $n-2f$ blocks
  recompute Merkle root h' and if $h' \neq h$ then abort
  if $\mathsf{witness}(h, b_i, l_i)$ is not sent
    broadcast $\mathsf{witness}(h, b_i, l_i)$

upon receiving $n-f$ valid $\mathsf{witness}(h, b_j, l_j)$
  $m \leftarrow$ decode($\{l_j\}$) {from any $n-2f$ blocks}
  deliver(m)

Figure 4.4: The MBC protocol. A message (h, b_j, l_j) is valid, if b_j is a valid Merkle tree branch for the Merkle tree root h and the data block l_j.

denoted as l_j. The replica p_i then generates a Merkle tree for the n blocks. Finally, for j between 0 to $n-1$, replica p_i sends an $\mathsf{init}(h, l_j, b_j)$ message to the j-th replica p_j, where h is the root of the Merkle tree and b_j is the j-th Merkle tree branch. We say a message (h, b_j, l_j) is valid, if b_j is a valid Merkle tree branch for the Merkle tree root h and the data block l_j.

If a replica p_i receives an $\mathsf{init}(h, l_i, b_i)$ message, p_i broadcasts $\mathsf{witness}(h, l_i, b_i)$.

If a replica p_i receives $n-2f$ valid $\mathsf{witness}(h, l_j, b_j)$ messages, p_i interpolates all n blocks from $n-2f$ blocks, recomputes the Merkle tree root h'. If $h' \neq h$ and it has not broadcast any $\mathsf{witness}()$ message, it broadcasts $\mathsf{witness}(h, l_i, b_i)$. Otherwise, it simply aborts.

If p_i receives $n - f$ valid witness() messages, p_i recovers the original input m and delivers m.

As shown in Table 4.1, assuming the same n, MBC has fewer steps and fewer messages than AVID (used in HoneyBadgerBFT, BEAT, and Dumbo).

	resilience level	steps	number of messages
AVID	$n \geq 3f + 1$	3	$2n^2 + n$
MBC	$n \geq 5f + 1$	2	$n^2 + n$

Table 4.1: Comparison of the RBC algorithms.

Theorem 1 *The MBC protocol in Figure 4.4 is a reliable broadcast protocol.*

Proof: We prove the theorem from scratch (instead of using the proof of IR MBC in a black-box manner).

We first prove validity. If a correct sender broadcasts a message m, it will erasure codes the message m into n blocks $\{l_j\}_{j \in [0..n-1]}$ (where $n - 2f$ blocks are sufficient to recover m), and generates a Merkle tree proof (h, b_j) for each block l_j. Then the sender sends j-th block l_j and the corresponding proof (h, b_j) to the corresponding replica p_j. Upon receiving an init message, each replica will verify whether the message is valid and then broadcasts the witness messages to all replicas. Eventually, all correct replicas will receive $n - f$ witness valid messages. Since the sender is correct, the recomputed Merkle root must equal the agreed one. Any correct replica p_j can recover m using $n - 2f$ erasure coding scheme with matching root and then deliver m.

We now prove agreement. If some correct replica p_i delivers a message m with

some h, then the replica must have received $n - f$ valid witness messages with the matching root h. Among these $n - f$ replicas, at least $n - 2f$ replicas are correct. These correct replicas must have received the $\mathsf{init}(h, \cdot, \cdot)$ messages and must have sent $\mathsf{witness}(h, \cdot, \cdot)$ messages to all replicas. Therefore, all correct replicas will receive valid $n - 2f$ $\mathsf{witness}(h, \cdot, \cdot)$ messages. We claim that if p_i delivers a message with h, then except with negligible probability, any other correct replica will not abort (as the recomputed Merkle root $h' = h$). Otherwise, one can find an adversary attacking the Merkle tree (more concretely, attacking the collision resistance property of the underlying hash function of the Merkle tree). Therefore, all correct replicas will broadcast $\mathsf{witness}(h, \cdot, \cdot)$ and eventually all correct replicas will receive $n - 2f$ valid $\mathsf{witness}(h, \cdot, \cdot)$ messages. Again, according to the property of the Merkle tree, all correct replicas can recover and deliver the same m.

Finally, integrity holds by inspection of the protocol. This completes the proof of the theorem. □

4.6.3 MiB7

In MiB7, we combine Bosco's strongly one-step ABA (S1S) for $n \geq 7f + 1$ and the Cobalt ABA. We design a new RBC construction, MBC with learners, or simply MBC-L, to reduce the number of messages transmitted further.

Strongly one-step (S1S). The strongly one-step ABA (S1S) is another one-step algorithm for $n \geq 7f + 1$ in Bosco [154]. S1S runs the same algorithm as W1S but achieves different properties. If all correct replicas propose the same initial value v,

all correct replicas deliver v in one communication step. Namely, S1S guarantees one-step termination under contention-free situations; it does not require the failure-free conditions needed for the one-step termination in W1S. As in MiB5, MiB7 uses the Cobalt ABA as the backup ABA protocol.

MBC-L. When $n \geq 5f + 1$, MBC is already a step-optimal RBC. While intuitively for the case of $n \geq 7f + 1$, one could use MBC directly, we actually use *MBC with learners* (MBC-L). In such a primitive, some replicas are just learners instead of active replicas participating in the main broadcast process. Nevertheless, MBC-L remains a standard RBC satisfying all RBC properties.

We show in Figure 4.5 and Figure 4.6 the workflow and pseudocode of MBC-L, respectively. Assuming $n = 7f + 1$ replicas, we have $5f + 1$ replicas are active replicas and the other $2f$ replicas are learners. (We will describe the selection principle shortly.) To broadcast a message, a broadcaster p_i runs MBC to broadcast its input m among active replicas. When an active replica p_j delivers a message, it broadcasts a ready message to all learners. When a learner receives at least $4f + 1$ valid ready messages from active replicas, it recovers the value m and delivers it.

In the MiB7 environment, all replicas are active in the ABA phase. Namely, each replica votes for 1 for an ABA instance after it delivers a value in the RBC phase regardless of whether it is an active replica or a learner. Each replica waits until at least $n - f$ ABA instances terminate with 1 before invoking other ABA instances with 0 as input.

Our approach is generic. If there is a RBC that requires n_1 replicas (e.g., tolerating $f_1 = \lfloor (n_1 - 1)/3 \rfloor$ failures) and an ABA that requires n_2 replicas (e.g.,

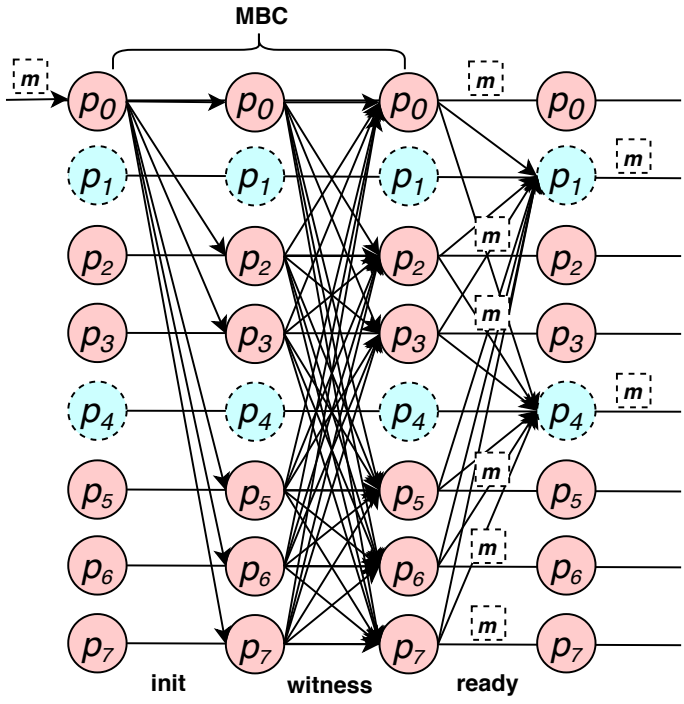


Figure 4.5: The MBC-L workflow. Replica p_0 is the broadcaster, replicas in solid circles are active replicas, and replicas in dashed circles are learners. Active replicas run the MBC protocol to deliver the message m and forward it to the learners once the message is delivered.

tolerating $f_2 = \lfloor (n_2 - 1)/5 \rfloor$ failures) where $n_2 > n_1$, then some $n_2 - n_1$ replicas are designated as learners who do not actively participate in RBC but later participate in ABA. The system tolerates f_2 failures. The n_1 replicas need to run RBC to deliver messages. When one of the n_1 replicas delivers a message, it forwards the message to all learners. A learner delivers a message when it receives $n_1 - f_2$ matching messages and then enters the ABA phase.

In the ACS framework, n parallel RBC instances are run concurrently. The principle of selecting active replicas is fairly arbitrary, as long as the system designer takes into account load balancing. For instance, in a system with replicas

Algorithm: MBC-L in MiB7 (Replica p_i)

upon input(m) {if $p_{sender} = p_i$}
  $active\_rep \leftarrow n_1 = 5f + 1$ active replicas
  let l_j be the j-th block of $(n_1 - 2f, n_1)$ erasure coding scheme applied to m
  h is the root of the Merkle tree for the $\{l_j\}_{j \in [0..n_1]}$ blocks
  send $\mathsf{init}(h, b_j, l_j, active\_rep)$ to each p_j, where b_j is the j-th Merkle tree branch

upon receiving $\mathsf{init}(h, b_i, l_i, active\_rep)$ from p_{sender}
  broadcast $\mathsf{witness}(h, b_i, l_i, active\_rep)$

upon receiving $n_1 - 2f$ valid $\mathsf{witness}(h, b_j, l_j, active\_rep)$
  interpolate $\{l'_j\}$ from the $n_1 - 2f$ blocks
  recompute Merkle root h' and if $h' \neq h$ then abort
  if $\mathsf{witness}(h, b_i, l_i, active\_rep)$ has not been sent
    broadcast $\mathsf{witness}(h, b_i, l_i, active\_rep)$

upon receiving $n_1 - f$ valid $\mathsf{witness}(h, b_j, l_j, active\_rep)$
  send $\mathsf{ready}(h, b_i, l_i)$ to $2f$ learners
  $m \leftarrow$ decode($\{l_j\}$) {from any $n_1 - 2f$ blocks}
  deliver(m)

upon receiving $n_1 - f$ valid $\mathsf{ready}(h, \cdot, \cdot)$ {learners}
  $m \leftarrow$ decode($\{l_j\}$)
  deliver(m)

Figure 4.6: MBC-L in MiB7. A message (h, b_i, l_i) is valid, if b_j is a valid Merckle tree branch for the Merkle tree root h and the data block l_j.

$\{p_0, \cdots, p_{n-1}\}$, the system designer can ask p_i ($i \in [0, \cdots, n-1]$) to select replicas $\{p_i, \cdots, p_{(i+5f) \bmod 7f}\}$ deterministically. One could ask replicas to randomly select $5f + 1$ replicas among all replicas, but the strategy is no better than the above-mentioned deterministic strategy that enables strictly even load balancing when n concurrent RBC instances are run.

While MBC-L has one more step than MBC, MBC-L in fact greatly reduces the number of messages in MiB7. As mentioned earlier, the total number of messages

in MBC is $n^2 + n$. Assuming $n = 7f + 1$, the total number of messages for MBC is $49f^2 + 21f + 2$. In MBC-L we use for MiB7, there are $2f$ learners. Each MBC-L instance involves $n-2f$, $(n-2f)^2$, and $2f(n-2f)$ messages in the init, witness, and ready steps, respectively. The total number of messages for MBC-L is $35f^2+17f+2$.

Theorem 2 *MBC-L in Figure 4.6 is a reliable broadcast protocol.*

Proof: Validity and integrity hold by inspection of the protocol. We focus on agreement, showing that if a correct replica p_i delivers m then a correct replica p_j eventually delivers m. We distinguish several cases:

Case 1: both p_i and p_j are active replicas. In this case, agreement follows trivially from that of MBC.

Case 2: p_i is an active replica and p_j is a learner. If p_i delivers a message m with some h, then according to the agreement property of the underlying MBC, all correct replicas will deliver m with the same h. These replicas will broadcast ready messages and eventually all learners will receive $n_1 - f$ valid ready messages. According to the property of the Merkle tree, all learners can recover and deliver the same m.

Case 3: p_i is a learner and p_j is an active replica. As p_i is an learner, it must have received $n_1 - f$ valid ready messages with matching h. This means that at least $n_1 - 2f$ active replicas have delivered the corresponding m. Due to the agreement property of underlying MBC protocol, all active replicas will deliver m.

Case 4: both p_i and p_j are learners. This case is similar to Case 3. Since p_i is an learner, it must have received $n_1 - f$ valid ready messages with the matching h.

Hence, at least $n_1 - 2f$ active replicas have delivered the corresponding message m. Due to the agreement property of underlying MBC protocol, all active replicas will deliver m. These replicas will send valid ready messages so that all learners will obtain $n_1 - f$ ready messages. The learners and active replicas agree on the same h and except with negligible probability, they will obtain the same m.

This completes the proof of the theorem. □

4.6.4 Other MiB Variants

We build a modular and expressive MiB framework, where we mix and match various RBC and ABA primitives to construct five additional asynchronous BFT protocols with suboptimal resilience. We summarize all MiB protocols in Table 4.2, including two MiB5 variants (MiB5a and MiB5b) and three MiB7 variants (MiB7a, MiB7b, and MiB7c). The framework allows us to validate our theoretical design and help identify meaningful protocol trade-offs among various combinations.

Note the idea of RBC learners in MBC-L applies to the AVID broadcast for both the case of $5f+1$ and $7f+1$. We show in Figure 4.7 and Figure 4.8 the workflow and pseudocode for the AVID-L protocol that we use for MiB5b. In AVID-L, some $3f+1$ replicas are active replicas and the rest of replicas are learners. A broadcaster p_i applies erasure coding to generate blocks for input m and broadcasts each block to the corresponding replica. Active replicas run AVID to deliver the input m. After the message is delivered, each active replica forwards learners the delivered message. A learner recovers the value m using blocks received and delivers m. Compared to

	resilience level	ABA	RBC
MiB5	$n \geq 5f + 1$	W1S	MBC
MiB5a	$n \geq 5f + 1$	W1S	AVID
MiB5b	$n \geq 5f + 1$	W1S	AVID-L
MiB7	$n \geq 7f + 1$	S1S	MBC-L
MiB7a	$n \geq 7f + 1$	S1S	AVID
MiB7b	$n \geq 7f + 1$	S1S	MBC
MiB7c	$n \geq 7f + 1$	S1S	AVID-L

Table 4.2: MiB protocols. RBC with -L labels are protocols with learners.

AVID, AVID-L has one additional step but less number of messages.

4.7 Implementation and Evaluation

Implementation. We build MiB from the open-source prototype of BEAT library [3] written in Python. We use the BEAT0 protocol (hereinafter BEAT for simplicity) as our baseline protocol. The MiB programming framework is modular, with a unified API encompassing eight protocols—BEAT and all seven MiB protocols summarized in Table 4.2. We use the zfec library [15] for erasure coding. Following BEAT, for all MiB instances, we use Shoup and Gennaro threshold encryption scheme [162] and the CKS threshold PRF [44] as the threshold encryption and the coin-flipping protocol, respectively. We use the prime256v1 curve with 128-bit security for the above two threshold cryptographic primitives. All the crypto algorithms are implemented using the Charm Python crypto library [22].

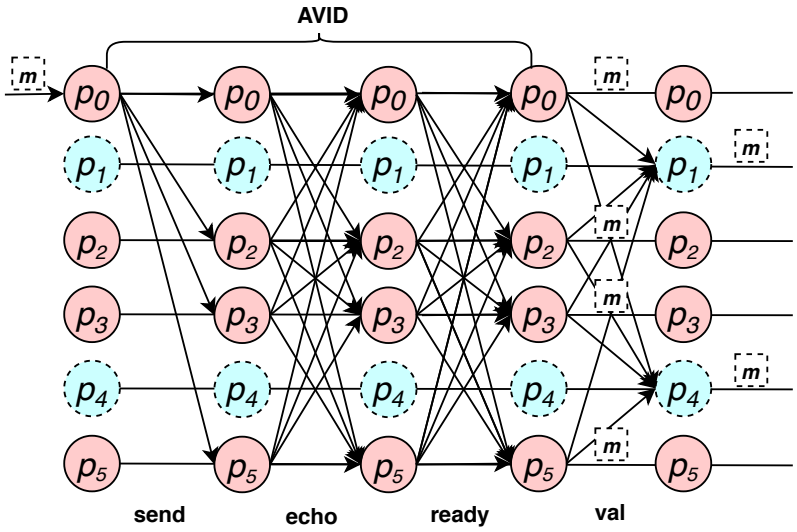


Figure 4.7: The AVID-L workflow. p_0 is the broadcaster. Active replicas are denoted with solid circles and learners are represented in dashed circles.

Evaluation overview. We deploy the MiB protocols and BEAT on Amazon EC2 and evaluate their performance using up to 140 instances distributed evenly in five continents. By default, we use the *t2.medium* type instances. Each instance has two virtual CPUs and 4GB memory. For one set of experiments, we also evaluate the peak throughput using *t2.micro* instances, each of which has one virtual CPUs and 1GB memory. The size of each transaction is 250 bytes. In every epoch, each replica proposes $b = \lceil B/n \rceil$ transactions, where B is the batch size.

We distinguish two scenarios: *the same n setting* and *the same f setting*. The same n setting allows evaluating the performance of all protocols with the same total number of replicas. In a system with n replicas, BEAT tolerates $\lfloor (n-1)/3 \rfloor$ failures, MiB5 and its variants tolerate $\lfloor (n-1)/5 \rfloor$ failures, and MiB7 and its variants tolerate $\lfloor (n-1)/7 \rfloor$ failures. In our evaluation, we choose $n = 16$, 31, and 46. For instance, when $n = 31$, BEAT tolerates 10 failures, MiB5 and its variants tolerate

Algorithm: AVID-L in MiB5b (Replica p_i)

upon input(m) {if $p_{sender} = p_i$}
  $active\_rep \leftarrow n_1 = 3f + 1$ active replicas
  let l_j be the j-th block of $(n_1 - 2f, n_1)$ erasure coding
  scheme applied to m
  h is the Merkle tree root of $\{l_j\}_{j \in [0..n_1]}$ blocks
  send $\mathsf{send}(h, b_j, l_j, active\_rep)$ to each p_j, where
  b_j is the j-th Merkle tree branch

upon receiving $\mathsf{send}(h, b_j, l_j, active\_rep)$ from p_{sender}
  broadcast $\mathsf{echo}(h, b_j, l_j, active\_rep)$

upon receiving $\mathsf{echo}(h, b_j, l_j, active\_rep)$ from p_j
  check if b_j is a valid Merkle tree branch for h and l_j
  check whether i is in $active\_rep$

upon receiving valid $\mathsf{echo}(h, \cdot, \cdot)$ from $n_1 - f$ distinct parties
  interpolate $\{l'_j\}$ from any $n_1 - 2f$ leaves received
  recompute Merkle root h' and if $h' \neq h$ then abort
  if $\mathsf{read}(h)$ has not been sent
    broadcast $\mathsf{ready}(h, active\_rep)$

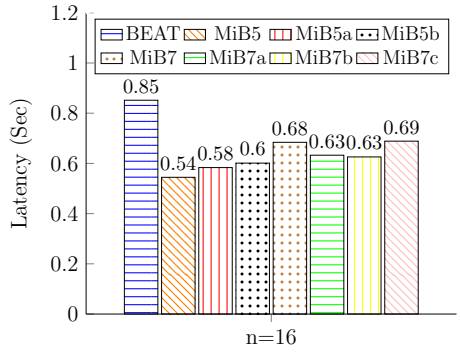
upon receiving $f + 1$ $\mathsf{ready}(h, active\_rep)$ from p_j
  if $\mathsf{ready}(\mathsf{h})$ has not yet been sent
    broadcast $\mathsf{ready}(h, active\_rep)$

upon receiving $2f + 1$ $\mathsf{ready}(h, active\_rep)$ from p_j
  send $\mathsf{val}(h, b_i, l_i)$ to $2f$ learners
  $m \leftarrow$ decode($\{l_j\}$)
  deliver(m)

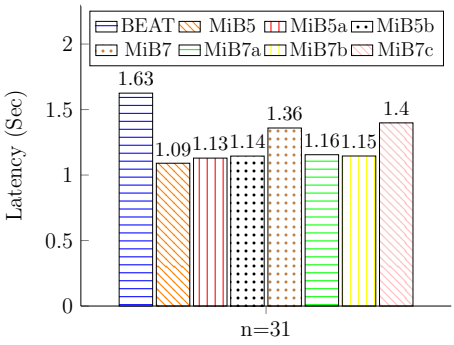
upon receiving $n_1 - f$ valid $\mathsf{val}(h, \cdot, \cdot)$ {learners}
  $m \leftarrow$ decode($\{l_j\}$)
  deliver(m)

Figure 4.8: AVID-L in MiB5b.

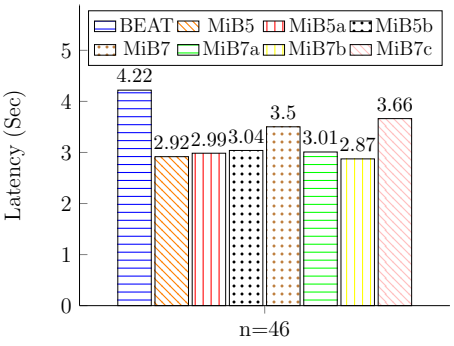
6 failures, and MiB7 and its variants tolerate 4 failures. The same n setting can provide guidance for selecting protocols when one has a fixed number of nodes for an application. For instance, if one has 31 nodes and is certain there would not be 6 failures, he or she may favor MiB5 over BEAT for performance considerations.



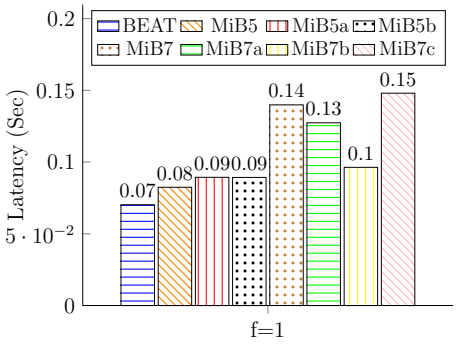
(a) Latency in WAN for $n = 16$.



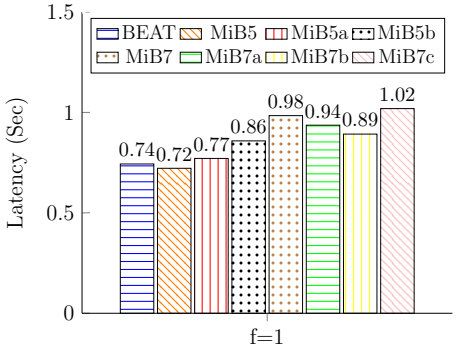
(b) Latency in WAN for $n = 31$.



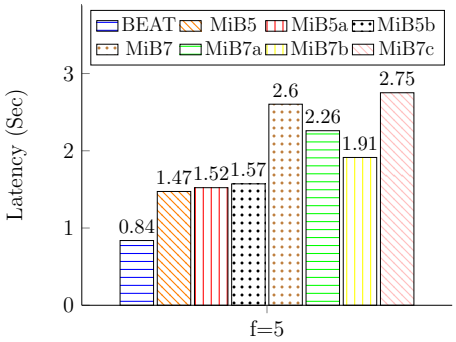
(c) Latency in WAN for $n = 46$.



(d) Latency in LAN for $f = 1$.

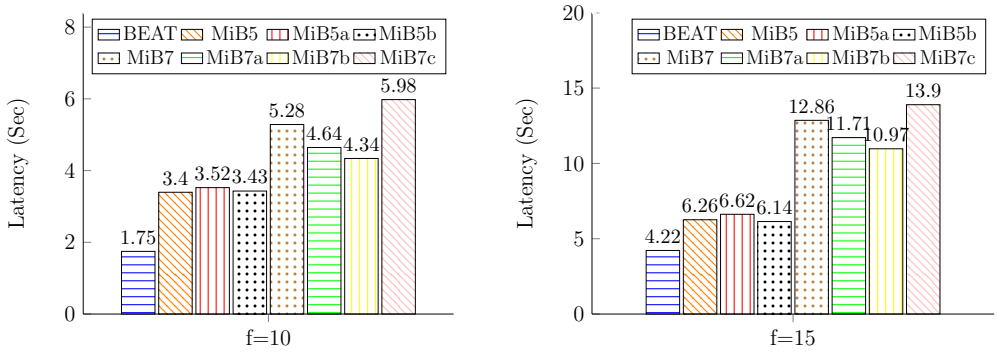


(e) Latency in WAN for $f = 1$.



(f) Latency in WAN for $f = 5$

The same f setting enables us to assess the performance for systems tolerating the same number of failures f. In this setting, the total number of replicas for BEAT, MiB5 and its variants, and MiB7 and its variants is $3f + 1$, $5f + 1$, and $7f + 1$, respectively. The evaluation for the same f setting is important for three reasons.



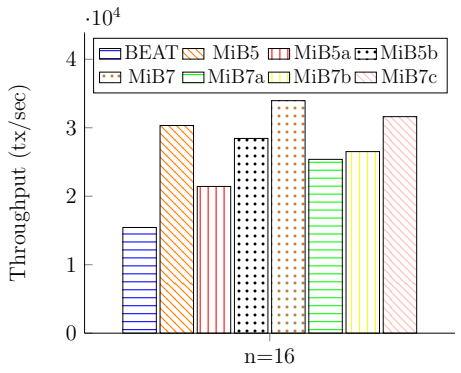
(g) Latency in WAN for $f = 10$

(h) Latency in WAN for $f = 15$

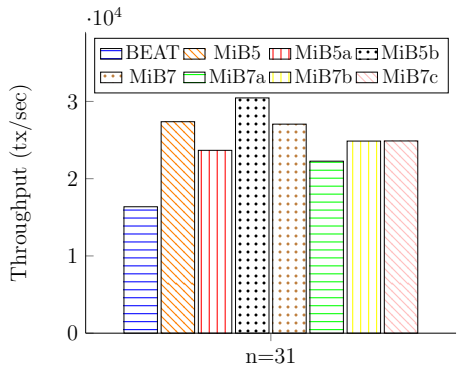
Figure 4.9: Latency of MiB protocols and BEAT in both LAN and WAN settings.

First, when one designs systems with a particular goal of tolerating some f failures, the evaluation can be directly used as a guideline. In many cases, one may not wish to adopt a system with fewer replicas, as other systems with more replicas may be more efficient. Second, it helps understand the performance difference among MiB5 and its variants, as well as the difference among MiB7 and its variants. Indeed, such an evaluation allows comparing protocols with different RBC and ABA components, thereby validating our theoretical design. Third, it enables us to analyze some corner cases: for instance, if one has 6 nodes, the system can only tolerate $f = 1$ failure, whether using BEAT or MiB5. The evaluation for the $f = 1$ case can help users to understand the trade-offs.

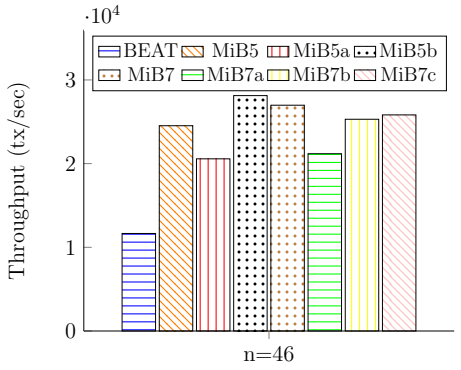
For some experiments, we vary the size of b from 1 (250 bytes per replica) to 10000 (2.38 MB per replica) to evaluate the throughput. We evaluate the latency when there is no contention, i.e., when $b = 1$. We evaluate the performance in the LAN settings (where the nodes are launched in the same EC2 region) and the WAN settings (where the nodes are evenly distributed in five continents).



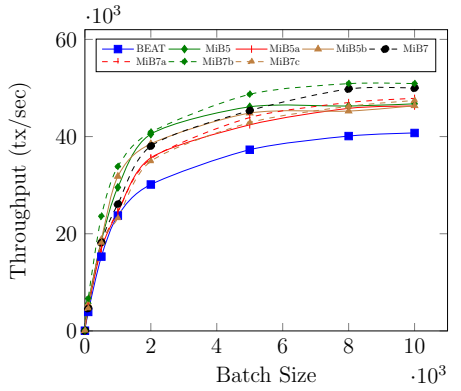
(a) Throughput for $n = 16$ and $b = 5000$.



(b) Throughput for $n = 31$ and $b = 5000$.



(c) Throughput for $n = 46$ and $b = 5000$.

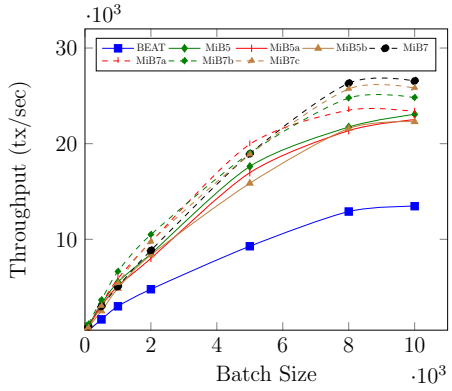


(d) Throughput in LAN for $f = 1$.

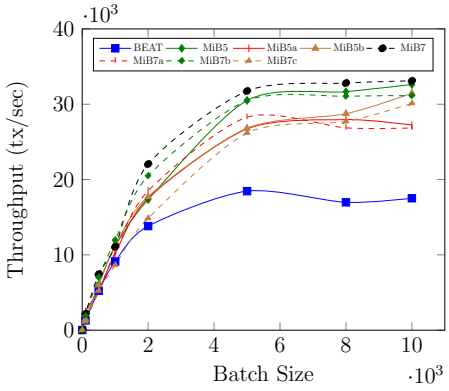
4.7.1 Latency

The same n. We first compare the latency of the protocols in the WAN setting when n is fixed. As shown in Figure 10(a-c), all MiB protocols have significantly lower latency than BEAT. This result is expected, since all MiB protocols terminate in fewer steps than BEAT (for both the best-case and average-case scenarios). For instance, when $n = 16$, the latency of MiB5 is 63.5% of that in BEAT. When we have a larger n, the difference between BEAT and MiB5 is more visible. For $n = 46$, the latency of MiB5 is 69.1% of that in BEAT.

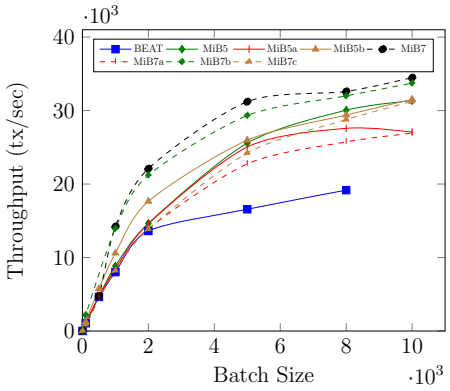
MiB7 and its variants have consistently higher latency than MiB5 and its



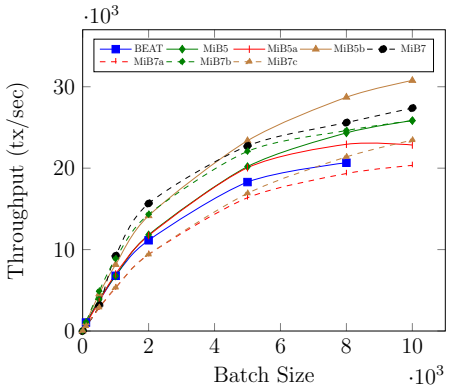
(e) Throughput in WAN for $f = 1$.



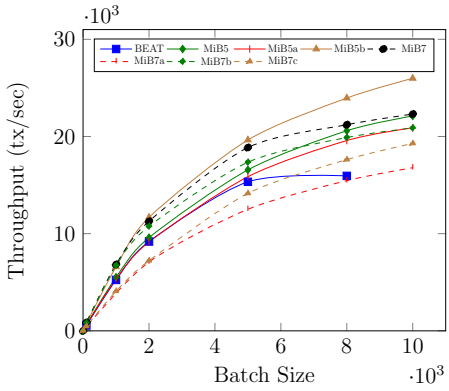
(f) Throughput in WAN for $f = 5$.



(g) Throughput in WAN for $f = 10$.



(h) Throughput in WAN for $f = 15$.



(i) Throughput in WAN for $f = 20$.

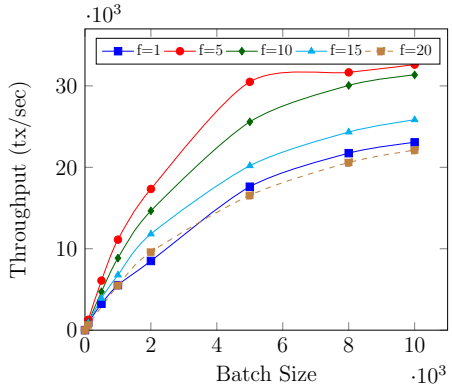
Figure 4.10: Throughput of MiB protocols and BEAT in the LAN and WAN settings.

variants. When $n = 16$, the latency of MiB7 is 25.9% higher than MiB5. This result is mainly because replicas need to collect more matching messages in both RBC

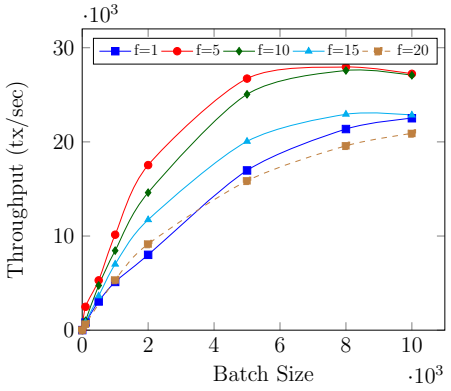
and ABA phases. MiB7 protocols need to collect increasingly more messages when n grows larger and the latency difference between the MiB7 instance and MiB5 is more significant.

We find that the two MiB5 variants have consistently higher latency than MiB5. This situation does not hold for the MiB7 variants. Interestingly, we observe that the result depends on the size of b. When b is small, both MiB7a and MiB7b have lower latency than MiB7 and MiB7c. Indeed, both MiB7 and MiB7c are RBC with learners involving an additional step for some replicas. When b grows larger, the network bandwidth consumption dominates the overhead, and correspondingly, MiB7 and MiB7c have lower latency.

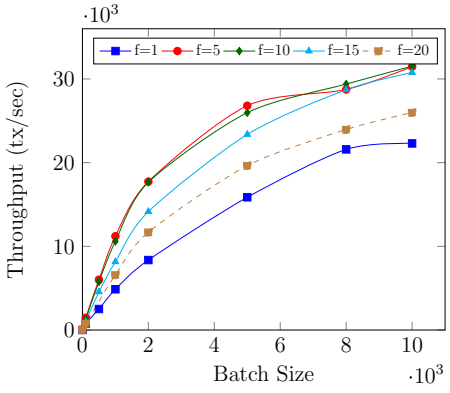
The same *f*. Figure 10(d-h) show the latency of the protocols in the LAN and WAN settings for the same f. Due to the upper bound on the number of EC2 instances we can launch in an EC2 region, we can evaluate the latency in the LAN setting only when $f = 1$. As shown in Figure 4.9(d), not surprisingly, the latency for all protocols is lower than the results in the WAN environment. BEAT has lower latency than other MiB protocols. Indeed, all MiB protocols have more replicas given the same f. For instance, when $f = 1$, MiB7 has 8 replicas, while BEAT has 4 replicas. Replicas in MiB7 and BEAT need to collect 7 and 3 matching messages in each RBC and ABA invocation, respectively. Thus, MiB7 and its variants have higher latency. The same result applies to the results in the WAN environment and when f grows larger. Specifically, all MiB7 protocols have higher latency than MiB5 and BEAT. MiB5 and its variants have higher latency than BEAT, except that MiB5a has lower latency than BEAT for the $f = 1$ case only.



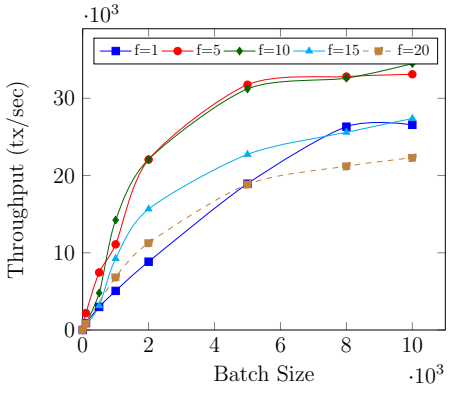
(a) MiB5.



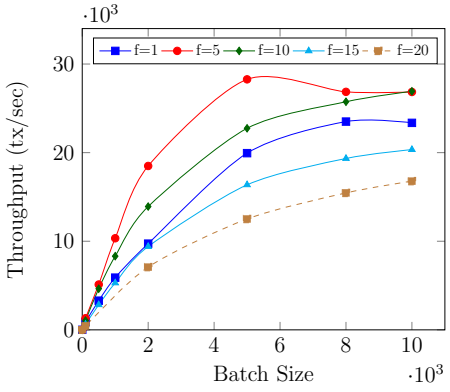
(b) MiB5a.



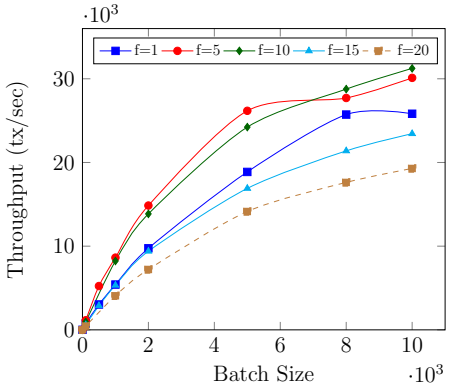
(c) MiB5b.



(d) MiB7.



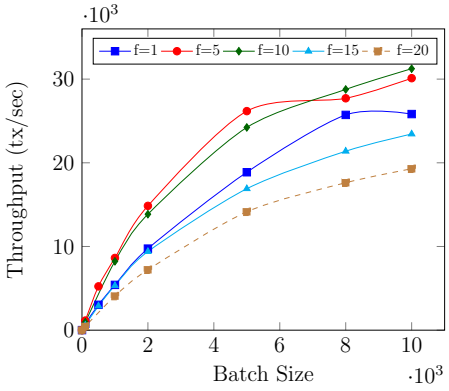
(e) MiB7a.



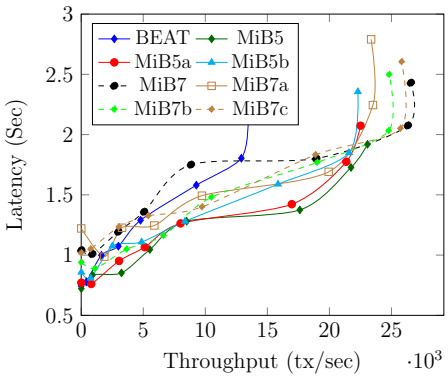
(f) MiB7b.

4.7.2 Throughput

The same n. We assess the throughput of the protocols using the same n. We fix the batch size to 5000 and evaluate the throughput. As shown in Figure 11(a-c), all



(g) MiB7c.



(h) Throughput *vs.* latency in WAN for $f = 1$.

Figure 4.11: (a-g) Scalability of MiB protocols; (h) throughput vs. latency in WAN. MiB protocols achieve significantly higher throughput than BEAT. As an example, when $n = 16$, the throughput of MiB5 is 96.5% higher than that of BEAT, while the throughput of MiB7 is 120.0% higher than that of BEAT. When $n = 46$, MiB5 and MiB7 achieve 110.4% and 131.5% higher throughput than BEAT, respectively. The performance improvement is mainly due to the step reduction in both the RBC and ABA components.

The same *f*. We report the throughput of the protocols with the same f from Figure 11(d-i). In both the LAN and WAN settings, when f is no greater than 10, all MiB protocols achieve consistently higher throughput than BEAT. Even when f is greater than 10, BEAT outperforms MiB7a and MiB7c only in some cases; other MiB protocols are consistently more efficient than BEAT. The performance difference is, in part, because MiB reduces the number of steps. Furthermore, given the same f, MiB protocols have a larger n and can propose more concurrent transactions. For instance, when $f = 1$ and $b = 5000$, the number of proposed transactions for BEAT is 33.3% and 50% less than MiB5 and MiB7, respectively. We also find that when

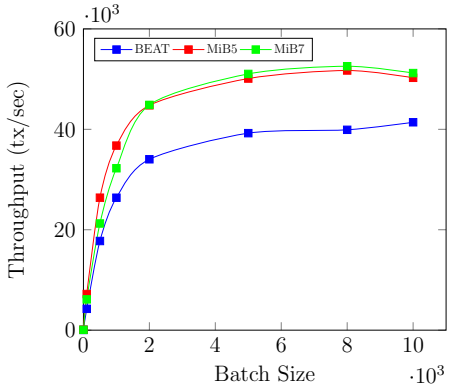
f grows larger (when $f \geq 15$), the network bandwidth consumption dominates the overhead and correspondingly the performance difference between BEAT and MiB protocols becomes comparatively small.

We also report in Figure 4.11(h) throughput vs. latency in the WAN setting for $f = 1$.

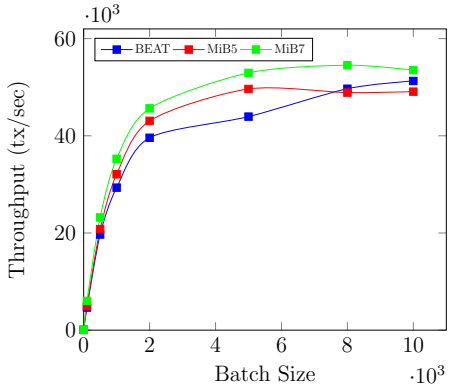
MiB5 *vs.* MiB7. We first report the throughput for the same n. When $n = 16$, MiB7 outperforms other protocols. When $n = 31$ and $n = 46$, MiB5b outperforms all other MiB protocols. This result is expected, since both MiB5b and MiB7 use learners to reduce the number of messages transmitted. Due to the use of learners, MiB7c also achieves higher throughput among the MiB variants. The reason why MiB7c achieves lower throughput than MiB5b and MiB7 is that AVID has one more step than MBC.

For all experiments, MiB5a and MiB7a achieve lower throughput than the other MiB protocols. Note MiB5a and MiB7a use AVID and do not use learners; AVID has one more step and more messages than MBC.

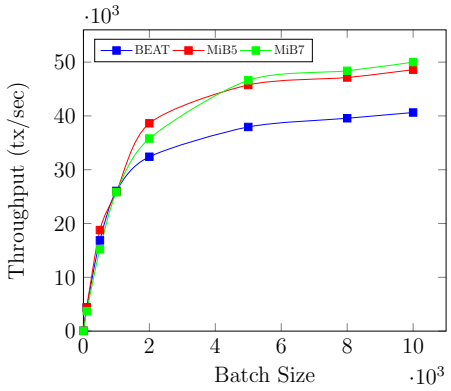
We also evaluate the performance using the same f. When $f = 1$ in the LAN environment, MiB7b outperforms other protocols. This situation is mainly because more transactions are proposed with a larger n. In contrast, in the WAN setting, MiB7 outperforms other protocols when $f \leq 10$ and MiB5b outperforms other protocols when $f \geq 15$. In particular, both MiB5b and MiB7 use learners to reduce the number of messages transmitted. Furthermore, MiB7 variants in general achieve higher throughput than MiB5 variants when f is smaller than 10 and lower throughput than MiB5 variants when f is greater than 10.



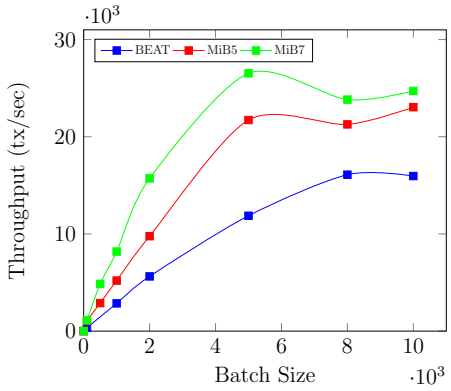
(a) Throughput in LAN for $f = 1$ in failure-free scenario.



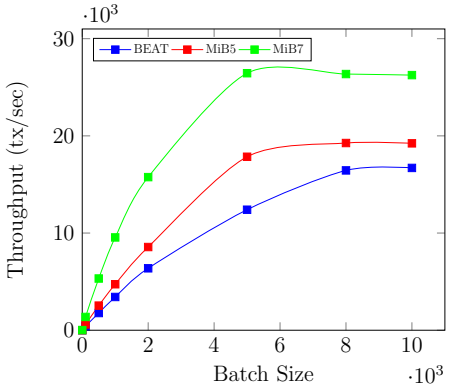
(b) Throughput in LAN for $f = 1$ in crash failure scenario.



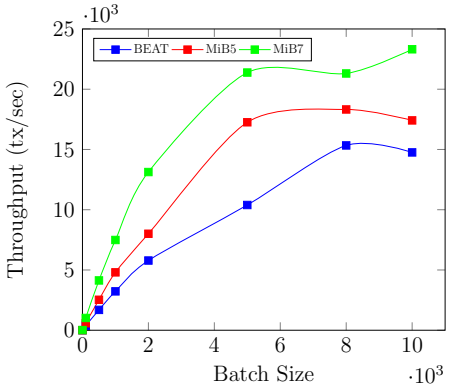
(c) Throughput in LAN for $f = 1$ in Byzantine scenario.



(d) Throughput in WAN for $f = 1$ in failure-free scenario.



(e) Throughput in WAN for $f = 1$ in crash failure scenario.



(f) Throughput in WAN for $f = 1$ in Byzantine scenario.

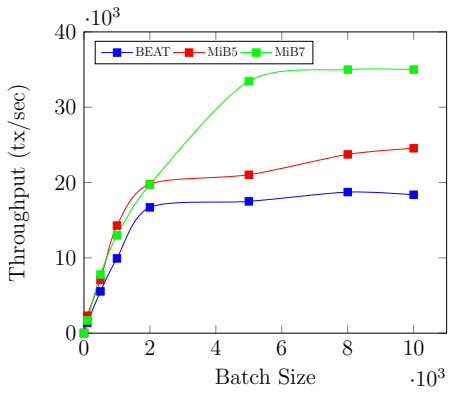
MiB with learners. MiB5b, MiB7, and MiB7c use learners and thus outperform other protocols (except for the $f = 1$ case). For instance, when $f = 1$ in the LAN setting, the peak throughput of MiB7b is 1.84% higher than MiB7. When f equals 5, 10, 15, and 20 in the WAN setting, the peak throughput of MiB7 is 6.3%, 2.2%, 5.9%, and 6.8% higher than MiB7b, respectively.

4.7.3 Scalability

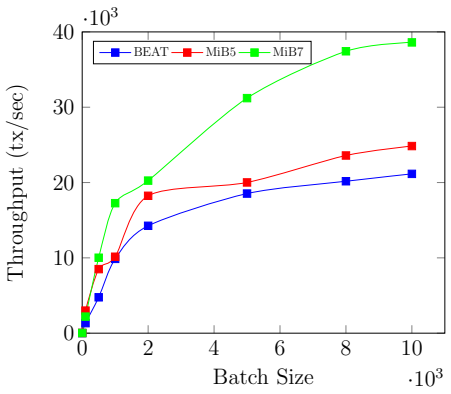
We evaluate the scalability of all our implemented protocols by varying f from 1 to 20 and varying b from 1 to 10000. When f increases from 1 to 20, the throughput first increases and then decreases. As illustrated in Figure 4.11, MiB protocols achieve their peak throughput around $f = 5$ and $f = 10$. The trend is very similar to that of BEAT (and EPIC): when f increases, the number of proposed transactions also increases so the throughput becomes higher; when f further increases, the network bandwidth becomes the performance bottleneck.

4.7.4 Performance under Failures

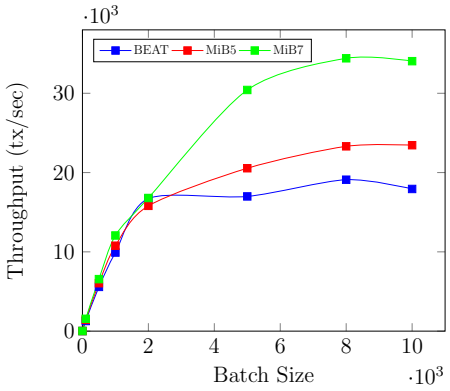
We now evaluate the performance of the protocols under failures. In Figure 4.12, we show the throughput of BEAT, MiB5 and MiB7 in three different scenarios: failure-free, crash failure, and Byzantine. In the crash failure scenario, we stop f replicas and run the protocols. In the Byzantine scenario, we chose to *simulate* the Byzantine behavior in the ABA phase instead of the RBC phase, because asynchronous RBC protocols are incredibly robust against Byzantine failures (see [57]). For the ABA



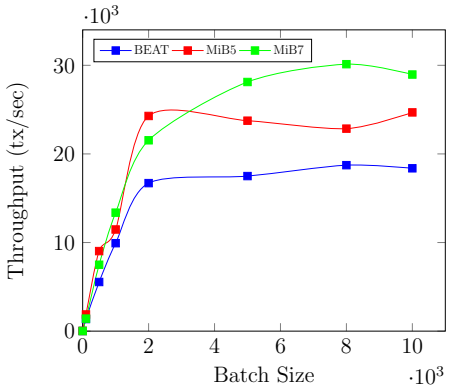
(g) Throughput in WAN for $f = 5$ in failure-free scenario.



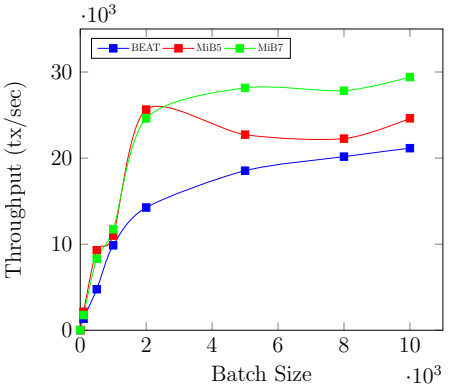
(h) Throughput in WAN for $f = 5$ in crash failure scenario.



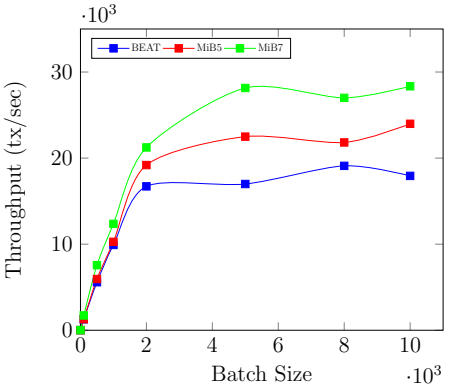
(i) Throughput in WAN for $f = 5$ in Byzantine scenario.



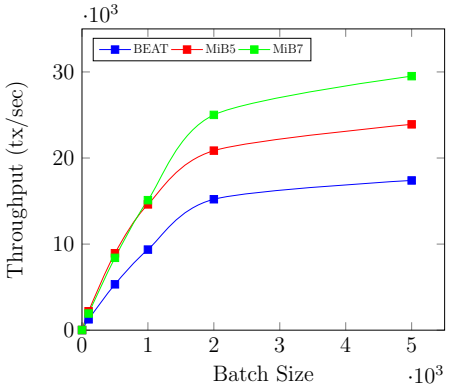
(j) Throughput in WAN for $n = 16$ in failure-free scenario.



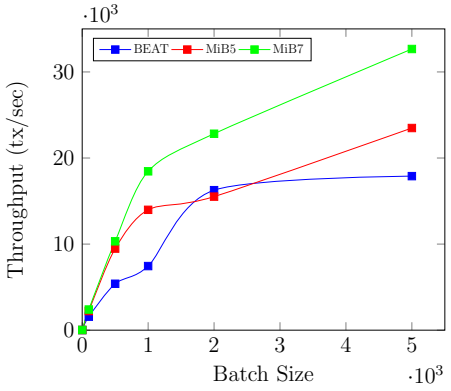
(k) Throughput in WAN for $n = 16$ in crash failure scenario.



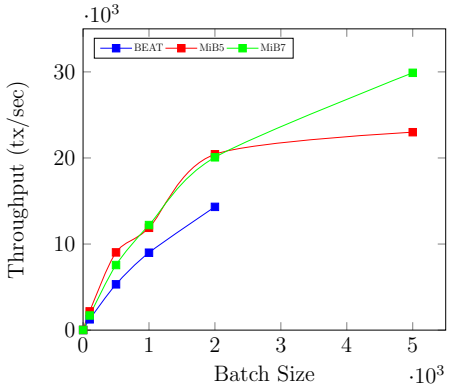
(l) Throughput in WAN for $n = 16$ in Byzantine scenario.



(m) Throughput in WAN for $n = 31$ in failure-free scenario.



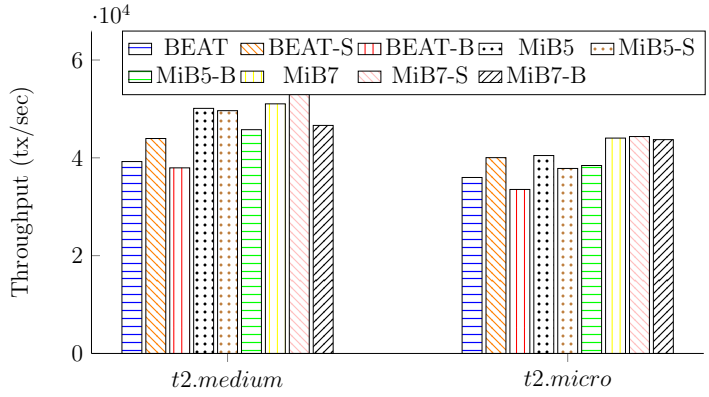
(n) Throughput in WAN for $n = 31$ in crash failure scenario.



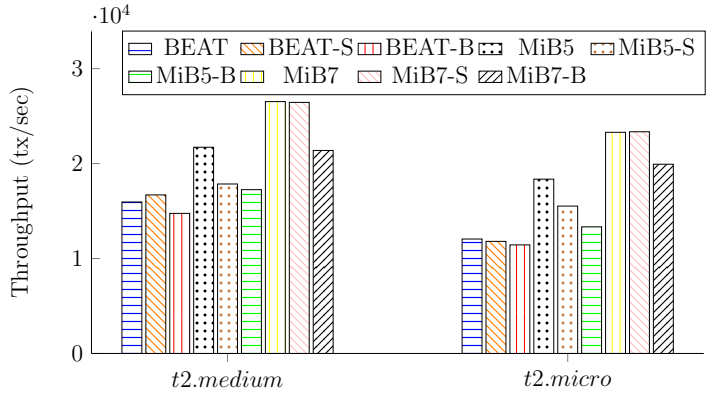
(o) Throughput in WAN for $n = 31$ in Byzantine scenario.

Figure 4.12: Throughput of BEAT, MiB5, and MiB7 in the LAN and WAN settings in failure-free, crash failure, and Byzantine scenarios.

phase, a Byzantine replica may exhibit either of the two following behaviors: not sending ABA proposals, or sending inconsistent proposals to different replicas. The former has been captured by the crash failure scenario. Hence, we focus on the latter one for Byzantine scenarios, where we let Byzantine replicas propose inconsistent values in the ABA phase. Note that inconsistent votes from Byzantine replicas may cause the protocols to terminate using more rounds, a strategy that might impact



(a) Throughput in LAN for $f = 1$ running on t2.medium and t2.micro instances.



(b) Throughput in WAN for $f = 1$ running on t2.medium and t2.micro instances.

Figure 4.13: Throughput of BEAT, MiB5, and MiB7 running on different hardware.

performance. Meanwhile, in this scenario, the ABA phase in both MiB5 and MiB7 would switch to a slower protocol, which may further reduce performance. We vary f from 1 to 10 in WAN to evaluate the performance. For $f = 1$, we also evaluate the throughput in LAN.

Failure-free vs. crash failure. All the three protocols achieve higher throughput in the crash failure scenario than that in the failure-free scenario. For instance, the throughput of BEAT in the crash failure scenario is 4.6%-24.5% higher than that

in the failure-free scenario. The throughput of MiB5 in the crash failure scenario is 0.3%-9.4% than that in the failure-free scenario. For MiB7, the throughput is 1.5%-10.6% higher in the crash failure scenario than the failure-free scenario. The results are expected, since the network bandwidth consumption is lower when f replicas crash compared to the failure-free scenario. Compared to BEAT, the throughput improvement under crash failures for both MiB5 and MiB7 are lower. This result is because MiB5 and MiB7 achieve one-step termination in the failure-free cases for over $n - f$ ABA instances. In contrast, in the crash failure scenario, both protocols switch to a slower backup ABA protocol under failures.

Failure-free vs. Byzantine. For all the experiments, all the three protocols achieve higher throughput in the failure-free scenario than the Byzantine scenario. The throughput of BEAT in the failure-free scenario is 1.9%-4.9% higher than that in the Byzantine scenario. For MiB5, the throughput is 2.9%-16.1% higher in the failure-free scenario. The throughput of MiB7 is 2.2%-6.0% higher in the failure-free scenario. The results are also expected since faulty replicas broadcast inconsistent messages to all replicas so the network bandwidth consumption is higher than that in the failure-free scenario. The degradation of performance under Byzantine failures for MiB5 and MiB7 is higher compared to that in BEAT. This result is because both protocols switch to a backup ABA under failures.

Performance using different hardware. We assess the performance of the protocols using both *t2.medium* instances and *t2.micro* instances. We show the peak throughput of BEAT, MiB5, and MiB7. For each protocol, we evaluate the peak

throughput under failure-free, crash failure, and Byzantine scenarios. For each protocol, we use protocol name to represent the performance in the failure-free scenario, -S to represent the performance of the crash failure scenario, and -B to represent the performance of the Byzantine scenario. We show the performance in Figure 4.13.

All the protocols achieve higher throughput using *t2.medium* instances. This is expected since all the computational operations are faster. In the LAN setting, the performance improvement BEAT in all three scenarios are consistency lower than that in the WAN setting. For instance, BEAT, BEAT-S, and BEAT-B achieve 9.0%, 9.8%, and 13.1% higher throughput using *t2.medium* than that in *t2.micro* in LAN, separately. In the WAN setting, the throughput are 32.3%, 41.5%, and 29.0% higher in the three scenarios. In contrast, the performance improvement for both MiB5 and MiB7 are in general higher in the WAN setting. For instance, MiB5 achieves 23.8% and 18.3% higher throughput using *t2.medium* in LAN and WAN, separately. MiB7 achieves 15.9% and 13.2% higher throughput using *t2.medium* in LAN and WAN, separately. This result is mainly because both MiB5 and MiB7 involve more replicas ($5f + 1$ and $7f + 1$) so the network bandwidth consumption dominates the overhead of the protocols.

4.8 Conclusion

We study two important directions in asynchronous BFT—BFT with suboptimal resilience and BFT performance under failures and attacks. This paper provides MiB, a novel and efficient asynchronous BFT framework using new distributed sys-

tem constructions as building blocks (including an erasure-coded version of IR RBC and a learner-version of RBC). MiB consists of two main BFT instances and five other variants. We design experiments with failures and systematically evaluate the performance of asynchronous BFT protocols (including MiB) in crash failure and Byzantine failure scenarios. Via a five-continent deployment using 140 replicas, we show the MiB instances have lower latency and much higher throughput than their asynchronous BFT counterparts, and moreover, asynchronous BFT protocols, including our MiB protocols, are indeed robust against failures and attacks.

Chapter 5

Intrusion-Tolerant and Confidentiality-Preserving Publish/Subscribe Messaging

5.1 Overview

Publish/Subscribe (pub/sub) is a popular messaging pattern allowing disseminating information from publishers to different subsets of interested subscribers via an overlay of brokers (servers). Publishers advertise information to the brokers and send publications as advertised. Subscribers express their interests for receiving a subset of publications by issuing subscriptions to brokers. Upon receiving publications from publishers matching the interests of subscribers, brokers send the corresponding publications to the interested subscribers.

One distinguishing feature of a pub/sub system is that it decouples publishers and subscribers in both time and space: publishers and subscribers do not need to know or synchronize with one another. This feature enables system flexibility and scalability. Pub/Sub systems are widely used in practice, such as Amazon SNS [1], AMQP [163], Apache Kafka [2], FAYE [7], Google Cloud Pub/Sub [8], and MQTT [11]. Pub/Sub serves as the core middleware for numerous applications, e.g., data collection and analysis, Internet-of-Things (IoT), network management and monitoring, streaming services.

Despite their popularity, existing pub/sub systems (built in both industry and academia) suffer reliability and confidentiality problems. Let us illustrate the issues with a health record exchange pub/sub system [91, 76], where the actors include patients and providers (physicians, hospitals, pharmacists), each of which can be publishers and subscribers. Publications may be medical files (e.g., reports, X-ray images) sent from patients or providers to patients or providers. Publications may also be new drug information and updates about the availability of facilities sent from providers to patients. For instance, an emergency unit receives a patient in critical conditions and disseminates the patient medical files as a publisher to various hospital units, while the hospital units may submit subscriptions (e.g., specialties, qualifications, schedule for patient admission and treatment sessions). As another example, a new-born is identified by a hospital for a rare dermatology disease. The hospital represents the new-born and the parents to send the medical images to some local expert dermatologists for timely treatment.

Consider another example of a market report notification system, where publishers are private sectors publishing paid market reports, and subscribers are investors who receive reports according to their interests (e.g., reports for certain categories, reports for specific periods). The brokers match publications with the interests and send the publications to interested (and paid) investors.

With these examples in mind, we now discuss the challenges of building intrusion-tolerant pub/sub systems.

Confidentiality and fine-grained access control (Or: Two-way information control). Publications in both examples (health records, private market reports)

need to be confidentiality-protected. In fact, confidentiality in pub/sub is strongly tied to *access control*, a process by which subscribers are granted access to certain publications based upon certain rules. The middleware community has long been expecting pub/sub systems where publishers can define by whom and how their data can be accessed, preferably not just role-based but also attribute-based.

For instance, the "ideal" situation for health record exchange is that publishers (patients and providers on behalf of patients) can decide by whom, when, and how their health records can be viewed or used. Patients should be able to decide which doctors can see their records, either exactly (by name), or those that meet certain criteria (e.g., "D.C. doctors", "more than 15-year practice in dermatology", "no malpractice history") [5]. For the market report example, publishers may enforce access control based on subscribers' qualifications, attributes, and if subscribers paid for the service (to the brokers).

Confidentiality-preserving pub/sub with fine-grained access control *enhances* the conventional pub/sub systems with *two-way* information control. In conventional pub/sub systems, subscribers can filter the information via subscriptions, but publishers cannot control who can receive the publications. The one-way information control is undesirable for applications such as cross-domain pub/sub systems where publications need to be protected (as shown in the health exchange and market report examples), most pub/sub systems in private corporate networks, and any IoT and big data applications where individual user data are sensitive.

Achieving the goal *securely*, however, is difficult. Existing pub/sub systems with confidentiality or access control either rely on non-cryptographic trusted do-

mains (an overly strong assumption), centralized architectures, and/or violate the decoupling feature of pub/sub systems [26, 77, 91, 92, 93, 102, 156, 157, 164, 170]. Building a decentralized pub/sub system with fine-grained access control is deemed to be a major open problem [141]. First, the approach to encrypting publications using the keys of subscribers does not work, because, due to the decoupling feature of pub/sub systems, publishers do not know the identities or keys of subscribers. Second, publishers cannot encrypt the data using the keys of brokers either, as brokers would know the publications in plaintext. Even if a single broker is compromised, all historical publications will be leaked.

Reliability. Another challenge of building intrusion-tolerant pub/sub systems is reliability under Byzantine (arbitrary) failures. Existing reliable pub/sub systems [52, 95, 100, 101, 148] only achieve weak reliability notions. One particular reliability notion is *publication total order* which guarantees subscribers should receive relevant publications in the same order. For instance, in the stock market, seeing a high price followed by a low price means something very different from seeing a low price followed by a high price; it is vital to ensure that all subscribers receive the price information in the same order. Publication total order would be easy to achieve if brokers use Byzantine fault-tolerant (BFT) state machine replication to maintain a total order of publications and ask subscribers to deliver publications according to the total order.

Even so, due to the two-way control (publication filtering via subscriber interests and publisher access control), not *all* publications will be sent to all subscribers. Therefore, the approach that brokers maintain a total order fail to work. In particu-

lar, subscribers do not know if they should wait for or skip publications with certain sequence numbers, as subscribers do not know if the corresponding publications are on the way or will never arrive.

Discussion. With the rise of blockchains, two pub/sub systems using blockchains (Hyperpubsub [139] and Trinity [148]) were proposed to defend against (Byzantine) failures. Both systems make a black-box usage of existing pub/sub systems and blockchain systems. Hyperpubsub uses Apache Kafka and Hyperledger Fabric [74], while Trinity combines MQTT and one of the four blockchains (Fabric, Tendermint [14], and test networks for Ethereum [6] and IOTA [10]). The two systems, however, suffer from at least three problems. First, both systems are essentially auditing systems using blockchains. The overall systems are not Byzantine fault-tolerant, as neither Kafka (only partially crash fault-tolerant) or MQTT (not fault-tolerant) can defend against Byzantine failures. Both liveness and safety are violated if any brokers of the two systems are compromised. Second, both Hyperpub and Trinity leveraging fully-fledged blockchains have demonstrated poor performance, because blockchains are essentially storage systems not designed for pub/sub systems, and many features of blockchains are not needed for pub/sub systems. Third, both Hyperpubsub and Trinity directly combine existing pub/sub and blockchains systems and therefore require a much larger number of nodes and resources than a blockchain system or a conventional pub/sub system.

Neither Hyperpubsub nor Trinity achieves confidentiality or publication total order, two goals we aim to address in this paper.

Our contribution. We design, implement, and evaluate Chios, a Byzantine fault-tolerant (BFT) pub/sub system with fine-grained access control and strong reliability, without sacrificing the decoupling property of pub/sub. Chios's security assumption is standard to BFT and threshold cryptography, i.e., an adversary cannot corrupt more than 1/3 of the total brokers. We summarize our contribution in the following:

systems	brief description	Byzantine publisher	Byzantine broker	confidentiality and access control	publication total order	publication liveness
Chios	BFT and confidentiality -preserving	●	●	decentralized; attribute-based	●	●
Kafka [2]	favor performance over reliability	○	○	○	○	○
AMQP [163]	"pub/sub for business"	○	○	use trusted virtual host; password for control	○	○
Hyperpubsub [139]	auditing system for Kafka	○	○	○	○	○
Trinity [148]	auditing system for MQTT	○	○	○	○	○
P2S [52]	crash fault-tolerant	○	○	○	◐	●
PubliyPrime [101]	Byzantine failure detection	○	●	○	◐	●
JM [95]	deconstructing BFT using a large number of nodes	●	●	○	○	●
IRC [93]	access control using ABE	●	●	centralized authority needed; expensive pairing-based crypto	○	○
EventGuard [156]	use trusted components	○	●	trusted nodes for confidentiality	○	○

Table 5.1: Characteristics of representative pub/sub protocols. ◐denotes partial support. P2S and PubiyPrime achieve weaker ordering guarantees than publication total order. (The formal definitions of publication total order and publication liveness are in Sec. 5.3.1.)

- We formally define the properties of a BFT and confidentiality-preserving pub/sub system, covering strong access control and message ordering guarantees, in the

sense of cryptography and reliable distributed systems.

- We demonstrate Chios is provably secure under our definitions by devising and extending cryptographic and reliable distributed system protocols (e.g., vector-label-input threshold encryption, broadcast encryption with decentralized key distribution). Chios is the first pub/sub system achieving decentralized and fine-grained access control as well as publication total order. We compare Chios with existing pub/sub systems in Table 5.1.
- Chios is versatile and modular, supporting three additional and fully-fledged pub/sub instances designed to meet different goals (e.g., different performance metrics, different application scenarios). This situation includes an instance that combines threshold encryption and broadcast encryption to enable more efficient and dynamic access control. For the instance, we also provide an optimized instantiation that is more efficient than a trivial instantiation. Both the general protocol and the instantiation use a novel approach to maintain the decoupling property of pub/sub.
- We implement and evaluate Chios, showing that all its variants are nearly as efficient as its unreliable (unreplicated) counterpart and existing pub/sub systems (Kafka and Kafka with passive replication) and orders of magnitude faster than blockchain-based systems (Fabric, Trinity, Hyperpubsub). None of existing pub/sub systems or blockchain-based systems achieve decentralized confidentiality or strong order guarantees.

5.2 Related Work

Fault-tolerant pub/sub. Most of industry pub/sub systems (Apache Kafka [2], FAYE [7], Google Cloud Pub/Sub [8], and MQTT [11]) do not have strong fault tolerance guarantees. For instance, Kafka is crash fault-tolerant for its controller part. For its broker components, most Kafka implementations are not fault-tolerant, though Kafka can be configured to use passive replication for weak fault tolerance. Pub/Sub systems with strong reliability have been mostly studied for the case of crash failures [52, 100, 169]. Only a handful of works consider a weaker subset of Byzantine failures [95, 101] and none of them achieve publication total order. Besides, PubliyPrime [101] does not handle Byzantine publishers or subscribers.

Pub/Sub with payload confidentiality. Confidentiality in pub/sub systems can be generally divided into two categories [141]: 1) confidentiality for publication headers and subscription constraints; 2) payload confidentiality (the ability to hide the payload of the publications, e.g., the patient health record). The confidentiality issue has become a major obstacle to wider adoption of pub/sub systems [141].

Chios addresses payload confidentiality but not confidentiality for publication headers or subscription constraints. Most prior pub/sub systems that handle payload confidentiality rely on overly strong "trusted domain" assumptions and do not maintain the decoupling feature of pub/sub systems that is essential to pub/sub system flexibility and scalability [26, 77, 102, 164, 170]. Srivatsa and Liu [156] devised EventGuard with many goals similar to ours. EventGuard, however, assumes a trusted service for confidentiality and authenticity.

Pub/Sub with access control (but no fault tolerance). While there are a number of pub/sub systems [91, 92, 93, 157] that use attribute-based encryption (ABE) [96] to achieve fine-grained access control, they all suffer from the following problems: 1) Efficient ABE schemes rely on relatively slow pairing-based cryptography. 2) All these systems use a trusted central authority which is a single point of failure. While the so-called *decentralized ABE* schemes exist [23], decentralization here actually means that *anyone* can serve as an ABE authority by creating a public key and issuing private keys to different users, but it does not mean that the keys are generated interactively among distributed nodes.

Reliable distributed systems with confidentiality. Several works achieve confidentiality in distributed file or storage systems that support *store* and *retrieve* operations [21, 46, 73, 94, 110, 111, 142]. In these systems, clients apply encryption, or secret sharing, to the data before the data is uploaded to the system.

Notably, Depspace [73] explores how to use publicly verifiable secret sharing and hash function to encrypt and locate client data, but it does not achieve linearizability.

AVID [46] suggests the use of threshold encryption to provide access control for Byzantine reliable broadcast and asynchronous verifiable information dispersal. AVID, however, considers a much simpler setting and does not have an implementation.

Yin et al. [166] built a BFT protocol which privately processes user data by separating agreement from execution and using threshold signatures. Assuming the same architecture, Duan and Zhang [71] provided a more efficient construction that

uses only symmetric encryption. Both protocols require a lot more nodes than a conventional BFT protocol.

Many recent works [54, 40, 108] explore how to perform private computation on blockchains using trusted execution environments (TEEs), e.g., Intel SGX.

These systems require trusting a single TEE vendor (e.g., Intel). Some cryptographic proposals use zkSNARKs [33] or multi-party computation [145] to achieve private computation. These approaches are limited in practice, as the cost to deal with *generic* operations is very high, and the throughput is low.

5.3 System and Threat Model

Background on pub/sub systems. Pub/Sub systems enable disseminating information from publishers (information sources) to subscribers (interested recipients) via an overlay of brokers (servers). Publishers advertise information to the brokers and send publications as advertised. Subscribers express their interests for receiving a subset of publications by issuing subscriptions. Brokers store subscriptions received from subscribers. Upon receiving matching publications from publishers, brokers send the corresponding publications to the interested subscribers. Besides storing subscriptions, brokers may maintain routing tables to deliver subscribers information.

The communication between publishers and subscribers is decoupled both in time and space. In particular, publishers and subscribers do not need to know or synchronize with one another. Indeed, direct communication among end-customers may

not be possible. The decoupling feature enables flexible and scalable information exchange and also avoids maintenance and charging difficulties for end-customers. Moreover, this allows anonymity between publishers and subscribers (assuming brokers are correct).

We consider *topic-based* pub/sub, which is dominant in industry pub/sub systems (e.g., Kafka, FAYE, MQTT, Amazon SNS, Google Cloud Pub/Sub). In topic-based pub/sub, a publication includes a *header* and a *payload*. The header contains the *topics* and their values (e.g., ID = "Alice", county = "Orange", price = "105"), while the payload contains the complete bulk data. Correspondingly, a subscription includes a set of *constraints* on the topics (e.g., ID = "Alice", county = "Franklin" or "Orange", price = 100). The brokers need to match publications against stored subscriptions according to the constraints of the topics ("equation," "and," "or" for topic-based pub/sub).

BFT. We consider BFT state machine replication (SMR) protocols, where f out of n replicas may fail arbitrarily (Byzantine failures) and a computationally bounded adversary can coordinate faulty replicas. A replica *delivers operations*, each *submitted* by some client. The client should be able to compute a final response to its submitted operation from the responses it receives from replicas.

5.3.1 Formalizing BFT Pub/Sub

Syntax. In our setting, publishers and subscribers are clients. Publishers can be subscribers and vice versa. We use brokers, servers, and replicas interchangeably.

We consider an overlay network, where brokers are connected in a complete graph.

A BFT pub/sub system consists of the following (possibly interactive) operations (reg, advertise, sub, pub, notify, read). An interactive registration algorithm reg is run by clients and brokers. Through the reg algorithm, new clients can be registered in the system and brokers can verify and store client (access) attributes (e.g., ages, certificates) enabling them to have access to publications in the future. For instance, a publisher may want only clients with certain attributes to see its publications. Clients should be able to register independently, and in particular, potential publishers and subscribers need not know one another. A client may not need to decide at this stage if the client would like to register as a publisher, a subscriber, or both, but rather may do this later via advertise and sub.

Publishers advertise to the replicas information that will be sent to all or a subset of clients. The advertise messages may be viewed as special publications. Subscribers send brokers subscriptions to express their interests via a sub operation. Brokers store subscriptions received from subscribers. Upon receiving matching publications from publishers via a pub operation, brokers send the corresponding publications to the interested subscribers via a notify operation. The read operation is similar to that of popular pub/sub systems (e.g., Kafka) and allows a client to read particular data of interests from brokers.

Operations (reg, advertise, sub, pub) change broker state and are collectively called write operations. Operations (notify, read) do not change broker state.

In our system, a publisher can send an encrypted publication together with access control rules ac to the system. We say a subscriber (a client) is *authorized* to

see a publication m, if the publisher submitting m has listed the subscriber in its access control rules ac.

Goals. The goal of our secure BFT pub/sub system is to achieve CIA (confidentiality, integrity, availability) against malicious brokers, publishers, and subscribers. As in a BFT system, we assume a strong adversary that can passively corrupt f out of n replicas and adaptively corrupt an unbounded number of clients. We divide the goals into confidentiality and reliability goals.

Confidentiality and access control goals. We provide a unified definition of security covering all confidentiality aspects (access control as specified by data providers and confidentiality for non-subscribers and brokers). Specifically, given a BFT pub/sub system, we associate the following game to an adversary $\mathcal{A}$ in Figure 5.1.

- $\mathcal{A}$ chooses to corrupt a fixed set of f brokers.
- $\mathcal{A}$ interacts with honest parties arbitrarily and chooses to corrupt clients adaptively.
- $\mathcal{A}$ selects two messages m_0 and m_1, an ac, and a unique tag tid that specifies an instance, and submits them to the encryption oracle for the system. $\mathcal{A}$ cannot corrupt any clients specified by ac (otherwise, $\mathcal{A}$ would have trivially won the game). The oracle randomly selects a bit b and computes an encryption c of m_b with ac and tid, and sends the ciphertext to $\mathcal{A}$.
- $\mathcal{A}$ interacts with honest parties arbitrarily subject only to the following two conditions that 1) $\mathcal{A}$ cannot ask the decryption oracle for the ciphertext c with ac and tid, and 2) $\mathcal{A}$ cannot corrupt any clients specified by ac.
- Finally, $\mathcal{A}$ outputs a bit b'.

Figure 5.1: We define the advantage of the adversary $\mathcal{A}$ to be the absolute difference between $1/2$ and the probability that $b' = b$.

Note it is easy to have a unique tid for a client operation (e.g., using a concatenation of the client identity cid and the timestamp of the operation ts). We comment that we do not need to additionally define decryption consistency (as in

threshold encryption), as this process is captured by Agreement 2 of the reliability goals (introduced below).

Our definition is easily shown to imply input causality (causal order) [150], which prevents the faulty replicas from creating an operation derived from a correct client's but that is delivered (and so executed) before the operation from which it is derived. The problem of preserving input causality was introduced in BFT atomic broadcast protocols by Reiter and Birman [150], later refined by Cachin et al. [43], and recently generalized by Duan et al. [71]. Preserving causal order equally makes sense in BFT pub/sub systems.

We do not aim to achieve confidentiality on publication headers or subscription constraints, although they need to be protected for some applications.

Reliability goals. We have the following reliability goals:

- **Agreement 1**: If any correct replica delivers a write operation m, then every correct replica delivers m.
- **Agreement 2**: If any correct subscriber delivers a publication p matching its subscription T, then every correct subscriber who has the same subscription T and has access to p delivers p.
- **Total Order 1**: If a correct replica has delivered write operations $m_1, m_2, \cdots, m_s$ and another correct replica has delivered $m'_1, m'_2, \cdots, m'_{s'}$, then $m_i = m'_i$ for $1 \leq i \leq min(s, s')$.
- **Total Order 2 (Publication total order)**: If a correct subscriber has delivered $p_1, p_2, \cdots, p_s$ for a subscription T and another correct subscriber has delivered

$p'_1, p'_2, \cdots, p'_{s'}$ for T, and if the two subscribers have the same access attributes, then $p_i = p'_i$ for $1 \leq i \leq min(s, s')$.

- **Liveness 1**: If a write operation m is submitted to $n - f$ correct replicas, then all correct replicas will eventually deliver m.
- **Liveness 2 (Publication liveness)**: If a publisher is correct and submits p matching a subscription T, then all correct subscribers that issued a subscription T and have access to p will eventually deliver p. If a subscriber issues a subscription T, then it will deliver all authorized publications matching T.
- **No Creation**: If a subscriber delivers a publication, then the publication was published by some publisher.
- **No Duplication**: A subscriber delivers no publications twice.

Agreement 1, Total Order 1, and Liveness 1 are properties for all write operations. The other properties are ones for pub/sub operations with respect to subscribers. We have considered access control when defining these properties. The properties can be easily simplified to work without considering access control.

Prior formalization on reliable pub/sub systems [52, 101, 148, 169] only consider a much smaller subset of properties we defined here. In particular, a weaker notion of publication total order was considered in several systems [52, 148, 169], where neither subscription restraints nor access control rules are considered, and total order is enforced among all publications across all subscribers. The weaker notion is immediately implied by the total order property of brokers (Total Order 1), as subscribers can directly deliver publications in the sequence number order deter-

mined by brokers. Moreover, in [169], Total Order 1 is not required, because they did not use a state machine replication approach.

No Creation and No Duplication have been previously formalized by Jehl and Meling [95] but with different names ("authentication" and "uniqueness").

5.4 The Chios System

Chios addresses two important problems in pub/sub systems, achieving decentralized, privacy-preserving pub/sub with fine-grained access control, and ensuring publication total order even with the two-way information control.

We first review threshold encryption. Then, we describe a toy protocol achieving all security goals except publication total order. Finally, we show our core protocol (Chios) achieving publication total order.

5.4.1 Review of VIL Threshold Encryption

Conventional labeled threshold encryption takes a single string as the label. We extend the primitive to support a vector of strings $L = (L_1, \cdots, L_s) \in \{0,1\}^{**}$ as labels. By a vector we mean a sequence of zero or more strings, and we let $\{0,1\}^{**}$ denote the space of all vectors. Our scheme supports an *arbitrary* number of vectors, each of which can be of *arbitrary* length.

Syntactically, a robust (t, n) VIL (variable-input-length) threshold encryption consists of the following algorithms. A probabilistic key generation algorithm TGen takes as input a security parameter l, the number n of total servers, and threshold

parameter t, and outputs (pk, vk, sk), where pk is the public key, vk is the verification key, and $sk = (sk_1, \cdots, sk_n)$ is a list of private keys. A probabilistic encryption algorithm TEnc takes as input a public key pk, a message m, and a vector label L, and outputs a ciphertext c. A probabilistic decryption share generation algorithm $\mathsf{ShareDec}$ takes as input a private key sk_i, a ciphertext c, and a label L, and outputs a decryption share τ. A deterministic share verification algorithm Vrf takes as input the verification key vk, a ciphertext c, a label L, and a decryption share τ, and outputs $b \in \{0, 1\}$. A deterministic combination algorithm Comb takes as input the verification key vk, a ciphertext c, a label L, a set of t decryption shares, and outputs a message m, or $\perp$ (a distinguished symbol).

Our VIL threshold encryption scheme, TDH2-VIL, extends the TDH2 threshold encryption by Shoup and Gennaro [153].

5.4.2 A Toy Protocol: Chios without Publication Total Order

System setup. We assume that the number of brokers is n, and f out of n brokers can fail arbitrarily (Byzantine failures). We set up an $(f + 1, n)$ VIL threshold encryption (TGen, TEnc, $\mathsf{ShareDec}$, Vrf, Comb) so that a public key pk and verification keys vk are associated with the system, while a secret key is shared among all brokers, with a broker i having a key sk_i for $i \in [1..n]$.

Publisher and subscriber registration. In Chios, communication among publishers and subscribers is decoupled both in time and space. Publishers and subscribers do not need to know or synchronize with one another. A client (a publisher

or a subscriber) registers with brokers using their attributes. During the registration, the brokers collectively verify and store client attributes. Chios runs BFT to ensure the registration information is consistent among brokers. More specifically:

- A client sends its attributes and the corresponding proof to brokers as a special registration operation.
- Upon receiving a registration operation, brokers verify the correctness of client attributes. Brokers discard the operation if the verification fails. (Note the verification of the client attributes can be done offline or online, as in PKI registration.) Brokers run the BFT protocol to assign a sequence number to the registration operation and store the operation in sequence number order. Brokers send replies signaling the success of registration.
- Upon receiving $f + 1$ matching replies, the client completes the registration.

Advertisements and subscriptions. During the advertisement process, publishers advertise to the system their publication scopes, and the brokers broadcast the type of events to all potential subscribers (who show an intent to receive subscriptions during the registration process or later via subscriptions). The $\mathsf{advertise}$ operation can be viewed as a special pub operation. During the subscription process, subscribers submit their subscriptions which are stored at the brokers. Advertisements and subscriptions are treated as BFT write operations that need to be ordered.

Publishing (with confidentiality and fine-grained access control). Let ts, op, o, $hr = [hr_1..hr_s]$, $\mathsf{ac} = [\mathsf{ac}_1..\mathsf{ac}_t]$, and p be the timestamp, the operation type

(pub), the executable operation o (which makes Chios stateful), the header, the access control policies, and the payload of a publication, respectively. The header hr consists of the topics of a publication and optionally additional associated-data that do not need to be privacy-protected. The approach provides fine-grained (per-publication) and attribute-based access control.

- A publisher cid takes as input ts, op, o, hr, p, and ac, and computes a threshold encryption ciphertext as follows. The vector of labels L for the client is of the form $(\mathsf{cid}, \mathsf{ts}, \mathsf{op}, hr, \mathsf{ac})$. The client cid takes as input the threshold encryption public key pk, L, and p, and outputs a labeled ciphertext $(L, c) \xleftarrow{\$} \mathsf{TEnc}(pk, p, L)$ using our vector-label-input threshold encryption. It sends brokers (L, c) as a BFT write operation.
- Upon receiving a client publication, brokers run the BFT protocol to order the publication (by assigning a sequence number to the publication), store the publication, and execute the associated operation o in sequence number order. The brokers send replies to the write request which may contain the executed result for the publisher.
- Upon receiving $f+1$ matching replies, the client completes the publish operation.

Notify. During the process, brokers enforce access control and send publications to authorized and interested subscribers. More specifically:

- Brokers decide authorized and interested subscribers for a publication (L, c) by matching publication topics with existing subscription constraints, checking access control policies associated with the publication, and checking global ac-

cess control policies already installed in the brokers. For authorized and interested subscribers, each broker $i \in [1..n]$ sends them its decryption share $\tau_i \xleftarrow{\$} \mathsf{ShareDec}_{sk_i}(L, c)$ and the sequence number sn assigned to the labeled ciphertext (L, c).

- Upon receiving $f + 1$ matching publications with *valid* decryption shares from the brokers with the same sequence number sn, a subscriber runs Comb to obtain the publication in plaintext and delivers it.

Read. As in Kafka, Chios can serve as a storage system and an authorized client can read stored data (publications) at brokers via engaging a protocol between the client and brokers.

- A client sends brokers a read request for a particular publication of the form (L, c).
- Upon receiving a read request, brokers decide if the client is authorized by checking access control policies associated with the publication. If the client is allowed to have access to the publication, each broker $i \in [1..n]$ sends the client its decryption share $\tau_i \xleftarrow{\$} \mathsf{ShareDec}_{sk_i}(L, c)$.
- Upon receiving $f + 1$ matching replies with *valid* decryption shares from the brokers with the same sequence number sn, the client runs Comb to obtain the publication in plaintext and delivers it.

The above system achieves all properties in Sec. 5.3.1 except publication total order.

5.4.3 Chios with Publication Total Order

Intuitively, to achieve publication total order, each subscriber needs to maintain a log of valid publications received and deliver them according to the sequence number order assigned by brokers; however, due to access control and subscriber interests, *not all* publications will be sent to all subscribers. Therefore, subscribers do not know if they should wait for or skip publications with certain sequence numbers.

To tackle the issue, we first require servers additionally to maintain topic-based sequence numbers in addition to the global sequence numbers. Doing so, however, does not suffice, as even if two subscribers have the same subscriptions, they may not receive the same publications due to the access control rules. We thus also require that servers send empty messages with sequence numbers to subscribers who are not authorized to receive the corresponding publications. This way, subscribers can safely skip empty publications and go ahead to deliver publications with larger publication sequence numbers.

We now describe in more detail how Chios achieves publication total order. As illustrated in Figure 5.2, we maintain two tables: a table for data blocks and a table for publication order indices. The data block table maintains all operations in the system, which are stored in the database. The publication order index table contains metadata of the data blocks and can be derived from the data block table. The index table is stored either in the database or in memory.

For each operation, we store the sequence number (sn), the client id (cid), the operation type (op), the message payload (p), timestamp (ts), access control rules

(ac), and the publication topics (tp). Certain fields in the data blocks can be NULL.

The publication order index table helps achieve topic-based total order (i.e., total order for the publications according to the topics). Specifically, for each topic, we maintain a simple data structure S-PS, where the S field consists of the sequence numbers of operations (sn, the same sequence numbers as in the data blocks table), and the PS field consists of the per-topic sequence numbers (ps).

sn	cid	ts	op	*p*	ac	tp
0	1000	4	pub	m_0	NULL	price="105"
1	1001	6	write	m_1	101	NULL
2	1000	10	pub	m_2	100,101	price="105", county="Orange"

Data Blocks

tp	S-PS
price="105"	0-0,2-1
county="Orange"	2-0

Publication Order Indices

Figure 5.2: Data blocks and the publication order indices.

The PS field contains incremental sequence numbers for a specific topic, ensuring there is no gap in the sequence numbers for operations with the same topic. For instance, as shown in Figure 5.2, in the data block table, operations with sequence number 0 and 2 are publications. There are two topics involved in the data block table: price = "105" and county = "Orange". Correspondingly, there are two topics in the publication index table. As both publications have the topic (price = "105"), the topic in the index table has two S-PS numbers: 0-0 and 2-1. The numbers 0 and 2 in the S field are the sequence numbers in the data block table, while the numbers 0 and 1 in the PS field are per-topic sequence numbers. Specifically, brokers distinguish three cases:

- For authorized and interested subscribers, each broker $i \in [1..n]$ sends them $(\mathsf{tp}, ps, \tau_i)$, where tp is the topic, ps is the topic sequence number, and $\tau_i \xleftarrow{\$} \mathsf{ShareDec}_{sk_i}$

(L, c) is the decryption share for broker i.

- For unauthorized and interested subscribers, each broker $i \in [1..n]$ sends them $(\mathsf{tp}, ps, \perp)$, where $\perp$ is a short distinguished symbol denoting an empty message payload (so that subscribers can safely skip the sequence numbers for a particular topic).
- For uninterested subscribers, brokers send nothing.

Each subscriber maintains a log of publications (either empty publications or publications in plaintext) for each topic tp. It delivers publications according to the ps order, and example of which is illustrated in Figure 5.3. More specifically,

- Upon receiving $f + 1$ matching publications of the form $(\mathsf{tp}, ps, \tau_i)$ from different brokers, a subscriber runs Comb to obtain a publication in plaintext p and stores p in Δ in its ps's position.
- Upon receiving $f + 1$ matching publications of the form $(\mathsf{tp}, ps, \perp)$ from different brokers, the subscriber directly skips the empty publication in the array Δ in its ps's position.
- The subscriber delivers a publication $p \in \Delta$ with a sequence number ps, if all publications with sequence numbers smaller than ps are either delivered (for non-empty publications) or skipped (empty publications).

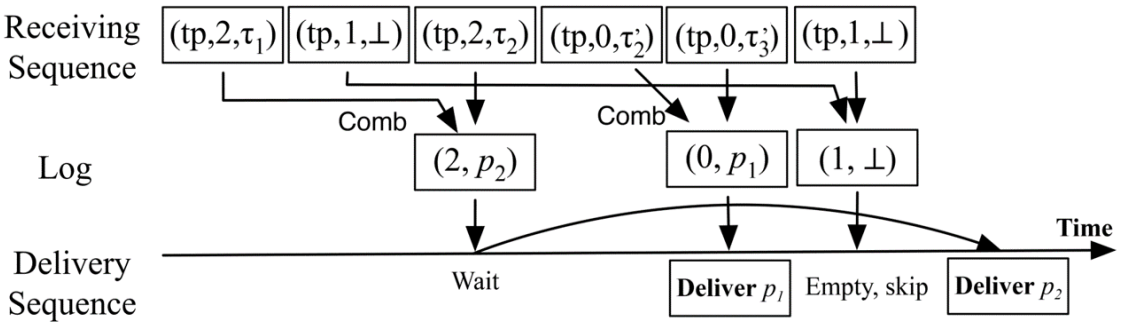


Figure 5.3: An example of how a subscriber delivers publications assuming $f = 1$ and $n = 4$. The subscriber receives a sequence of messages from BFT brokers and stores them in its buffer. It first receives $f + 1 = 2$ matching messages with $ps = 2$. It then runs Comb to obtain a publication p_2 in plaintext and stores in its log. The subscriber has to wait until publications with smaller sequence numbers (i.e., $ps = 0, 1$) have been dealt with. After the subscriber receives 2 matching messages with $ps = 0$, it runs Comb to obtain p_1 and delivers p_1. It then waits for messages with $ps = 1$. After the subscriber receives two empty messages for $ps = 1$, it directly skips the message and delivers message p_2 stored.

5.5 Implementation

Chios consists of a Java library and a Python library with about 30,000 lines of new code. We use BFT-SMaRt [35] written in Java as the underlying consensus engine, as BFT-SMaRt is "the most advanced and most widely tested implementation of a BFT consensus protocol" [99]. We use LevelDB [16] as the database. We extend the BFT-SMaRt library and implement a key-value store service. The Java library serves as an ordering service, which assigns a sequence number to a client operation.

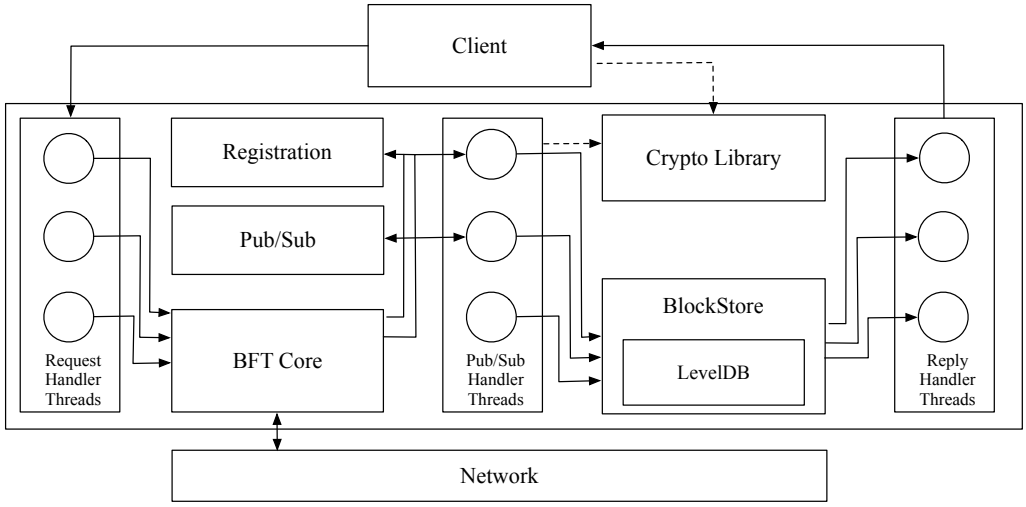


Figure 5.4: System architecture and message flow.

Then, we wrap the library in Python and develop all the core functionalities.

Figure 5.4 illustrates the system architecture and the message flow. The client operations are first handled through a request handler thread pool and the operations are then relayed to the BFT core. The BFT core batches concurrent client operations and assigns a sequence number to each operation. The ordered client operations are then processed by a pub/sub handler thread pool. Each thread processes a client operation at a time and outputs a reply according to the operation type. Chios uses a *batch-process, block-store* approach, where operations are batched according to a tunable parameter BlockSize, ordered, processed, and the results are stored in the database in blocks.

We use ECDSA for authentication and use SHA-256 as our hash function. We implement TDH2-VIL and threshold PRF [44] using the Charm Python library [22]. We use the NIST P-256 curve to provide 128-bit security.

We use AES and CBC with ciphertext stealing as our blockcipher and encryp-

tion scheme, respectively, to implement the NNL scheme [137].

5.6 Evaluation

Settings.

We deployed Chios on Amazon EC2 using up to 31 nodes for brokers and 25 nodes for clients (running up to 1,200 clients in total).

Each node, by default, is a compute-optimized *c5.2xlarge* type with 8 virtual CPUs (vCPUs) and 16GB memory. We also test the performance using a general-purpose *t2.medium* type with two vCPUs and 4GB memory to evaluate the performance on different hardware. We evaluate our protocols in both LAN and WAN settings, where the LAN nodes are selected from the same EC2 region, and the WAN nodes are uniformly selected from different regions.

We evaluate the protocols under different network sizes (number of replicas) and contention levels (number of concurrent clients). For each experiment, we use f to represent the network size, where $3f + 1$ brokers are launched in total. We use P, C, and B to represent the encryption-free module (Module 1), the threshold encryption module (Module 2), and the broadcast encryption module, respectively. Let $\mathsf{Mod} \in \{\mathsf{P}, \mathsf{C}, \mathsf{B}\}$ and let $\mathsf{op}(\mathsf{Mod})$ represent the operation op in the operation using the Mod module. For instance, $\mathsf{pub}(\mathsf{C})$ denotes pub operations for Module 2.

We examine the average latency under no contention where only one client sends a single operation to the servers. We examine the throughput under high contention of client requests. We evaluate the number of operations processed every

second for every 2,000 operations and use the average throughput of the entire experiment.

Overview. For the minimum one failure setting ($f = 1$), the Chios protocol with all desirable features (pub/sub, decentralized confidentiality, and fine-grained access control), achieves throughput of 45 kops/s for `pub` operations in LAN. To demonstrate Chios's performance rigorously, we first compare Chios Module `P` with five other pub/sub systems. Next, we evaluate the performance for different Chios Modules.

Comparison with five other pub/sub systems. We first compare Chios Module `P` with the following five systems, where Chios-Solo, Kafka, and Fabric-Solo are unreplicated systems, while Kafka-Rep and Fabric-Kafka are crash fault-tolerant systems:

- Chios-Solo. Unreplicated, single-node version of Chios.
- Kafka. As we summarized in Table 5.1 in Sec. 5.1, Kafka favors performance over reliability and does not achieve any security or reliability goals which we surveyed even in the crash failure model.
- Kafka-Rep. Kafka also supports passive (primary-backup) replication for its brokers with no total order guarantees.
- Fabric-Kafka [74]. Fabric is a popular permissioned blockchain system. Fabric currently does not protect against Byzantine failures. Fabric-Kafka uses the Zookeeper [90] system in Kafka to achieve consensus and is thus only crash fault-tolerant. Hyperpubsub is pub/sub auditing system using Fabric (with Raft [140])

and it is thus slower than Fabric-Kafka.

– Fabric-Solo [74] uses a single node for consensus and is thus not fault-tolerant. One Trinity instance [148] uses Fabric-Solo as its pub/sub auditing system and is slower than Fabric-Solo.

To evaluate the P module of Chios, we randomly assign topic number for publications during evaluation. We implement a read/write smart contract for Fabric and use the write operation for the write throughput. Our evaluation for throughput is standard: publishers send brokers operations, and we increase the number of publishers to obtain the peak throughput. We first find out the number of publishers when each system reaches peak throughput. To ensure a fair comparison, we evaluate the systems under the same *total* workload. Namely, the total number of operations sent publishers is the same for all systems. We let the size of all operations be 1kB and we utilize network sizes that tolerate one failure, i.e., four for Chios and three for Fabric and Kafka.

We report the throughput in LAN using 200 clients of the six systems in Figure 5.5. Chios Module P is as efficient as Chios-Solo and is only marginally less efficient than Kafka and Kafka-Rep. Chios is significantly more efficient than Fabric-Kafka and Fabric-Solo and thus even much more efficient than Hyperpubsub and Trinity.

It is unfair to compare Chios with Kafka with more nodes, as Kafka uses independent server instances for horizontal scalability.

Latency of Chios modules. We assess the latency in both the LAN and WAN

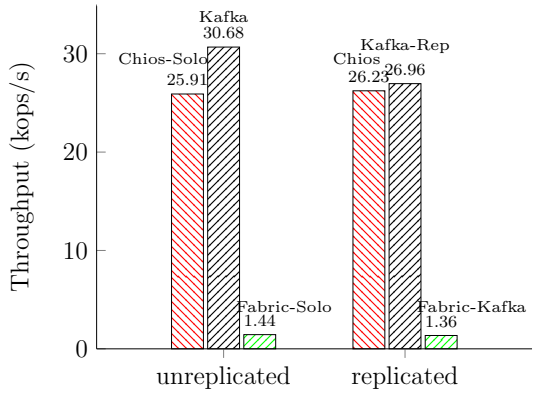


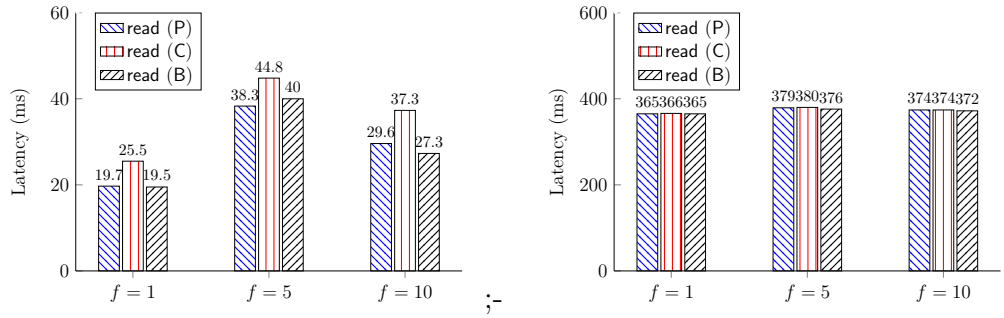
Figure 5.5: Throughput of Chios, Chios-Solo, Kafka, Kafka-Rep, Fabric-Kafka, and Fabric-Solo.
settings.

We let the BlockSize be one to understand the latency caused by the protocol itself.

In the LAN setting, the network latency is relatively small, so the overhead is more caused by the BFT agreement and execution of operations (e.g., verifying operation types, database interaction). In the WAN setting, the network latency causes more performance degradation than that in the LAN setting. For the threshold encryption and broadcast encryption modules, the latency evaluated includes the overhead of client-side encryption.

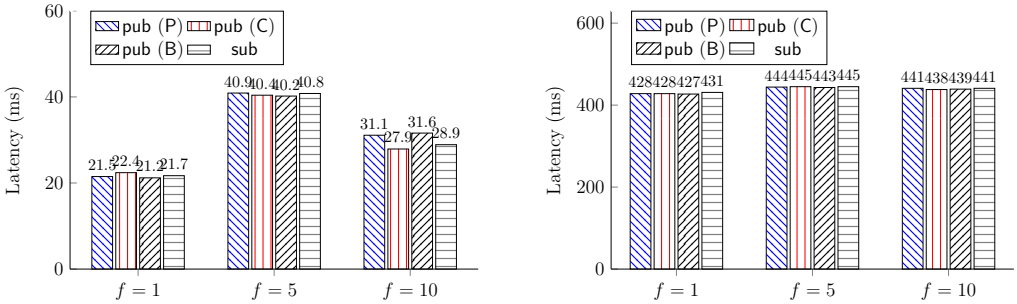
For read *operations.* We assess the latency for read operations in all three encryption modules as the network size increases. Figure 5.6(a) reports the latency for the LAN setting. As read(P) involves no encryption, it has the lowest latency among all three modules. For read(C), replicas verify the ac rules, decrypt the ciphertext, and send decryption shares to the clients.

Additional overhead is thus incurred. For read(B), we test the latency for the



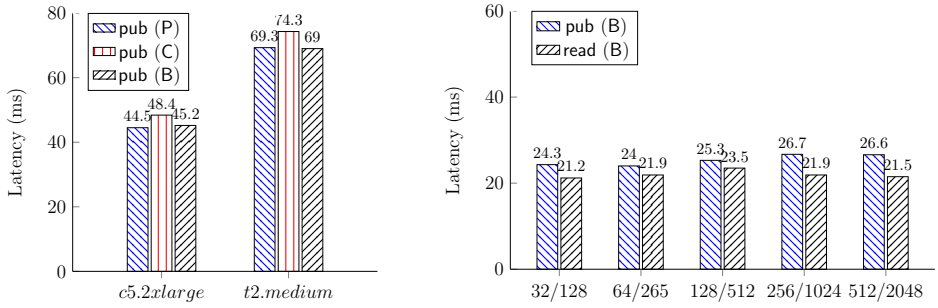
(a) Latency for read operations in the LAN setting with $f = 1, 5$ and 10.

(b) Latency for read operations in the WAN setting with $f = 1, 5$ and 10.



(c) Latency for pub operations in the LAN setting with $f = 1, 5$ and 10.

(d) Latency for pub operations in the WAN setting with $f = 1, 5$ and 10.

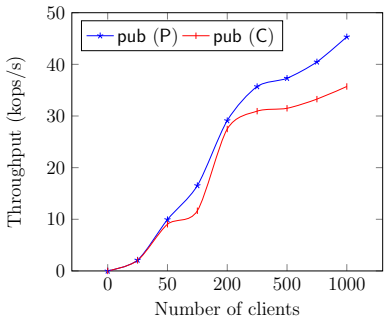


(e) Latency for pub operations in the LAN setting on different hardware with $f = 1$.

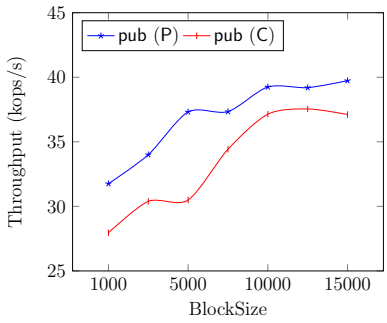
(f) Latency for read(B) and pub(B) operations in LAN with $f = 1$, where a/b represents that a out of b subscribers are revoked.

Figure 5.6: Latency of different Chios modules.

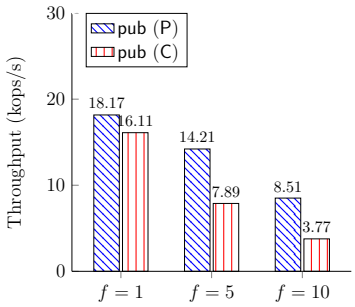
content distribution phase, as the key distribution phase needs to be done only once. The performance of read(B) is consistently better than that of read(C), as it uses symmetric cryptography only. In the WAN setting, the latency difference among



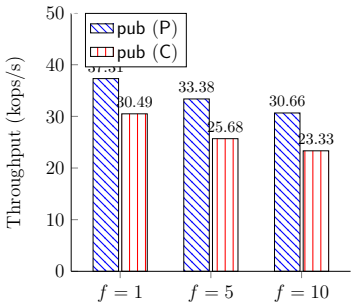
(a) Throughput in the LAN as the number of client threads increases.



(b) Throughput in the LAN as the size of block increases.



(c) Peak throughput in the WAN using 500 clients.



(d) Throughput in the LAN using 500 clients.

Figure 5.7: Throughput of different Chios modules.

the three modules is smaller, as shown in Figure 5.6(b), mainly because network latency dominates the overhead.

For pub *and* sub *operations.* We report their latency in Figure 5.6(c) and Figure 5.6(d). We also report the latency of pub using different hardware in Figure 5.6(e). We find that the latency for pub operations is higher than that of read operations. We also find that the latency difference for pub operations between the LAN and WAN settings is much higher than that for read operations. The findings are expected, as Chios implements the BFT read optimization which reduces much communication overhead.

Other operations. In Figure 5.6(f), we evaluate the performance of read(B) and

write(B) operations as the number of subscribers increases. In all these experiments, we randomly revoke 1/4 of the total subscribers. We find for both operations, the latency is steady, regardless of the number of subscribers. The reason is that the broadcast encryption module uses symmetric cryptography only for the content distribution phase.

Throughput of Chios modules. We evaluate the throughput of Chios with varying BlockSize in the LAN setting when $f = 1$. Figure 5.7(a) demonstrates the throughput when the BlockSize is the 5,000 and as the number of concurrent clients increases from 25 to 1,000. The system reaches peak throughput when the number of concurrent clients is larger than 800. The peak throughput that we observe for pub (P) is around 40 kops/s in LAN and 18 kops/s in WAN. We report the throughput when the total number of clients is 375 and as the BlockSize increases in Figure 5.7(b). We observe that the throughput becomes larger when the BlockSize increases; however, after BlockSize is larger than 10k, the throughput ceases to increase. In all experiments, the throuhgput for pub (C) is lower than that of pub (P) due to the cryptographic overhead.

We report the throughput for pub (P) and pub (C) using up to 31 servers and 500 concurrent clients for the LAN setting and the WAN setting, in Figure 5.7(d) and Figure 5.7(c), respectively.

For both the LAN and WAN settings, we find that the throughput for both modules degrade when the number of servers increases **(resembling that of BFT-SMaRt, the consensus engine for Chios)**, and the throughput for pub (C) degrades more significantly due to the cryptographic overhead.

5.7 Conclusion

We design and implement Chios, a highly efficient and intrusion-tolerant pub/sub system. Chios addresses two major challenges in pub/sub in terms of confidentiality and reliability: Chios achieves decentralized confidentiality with fine-grained and attribute-based access control and publication total order with two-way information control. Chios provides modular instances designed to meet different goals. Through extensive evaluation, we demonstrate Chios is efficient.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The dissertation's goal is to build the real-world distributed system focusing on different required tasks. We constructed three fault-tolerant distributed systems.

First, we proposed EPIC and HALE to achieve adaptive security. We designed these adaptive asynchronous BFT protocols and analysed them. Via a five-continent deployment on Amazon EC2, we showed EPIC is slightly slower for small and medium-sized networks than the most efficient asynchronous BFT protocols with static security. As the number of replicas is smaller than 46, EPIC's throughput is stable, achieving peak throughput of 8,000–12,500 tx/sec with a transaction size of 250 bytes. When the network size grows larger, EPIC is not as efficient as asynchronous BFT protocols with static security, with throughput of 4,000–6,300 tx/sec. We also show, while HALE is in general less efficient than EPIC, HALE is reasonably fast, achieving 42,000 tx/sec and 3,400 tx/sec for the 4-server setting in the LAN/WAN environments, respectively. Remarkably, HALE outperforms EPIC in LANs when the number of replicas is smaller than 16.

Next, We presented MiB, a novel and efficient asynchronous BFT framework. MiB consists of two main BFT instances and five other variants. As another contribution, we systematically designed experiments for asynchronous BFT protocols

with failures and evaluated their performance in various failure scenarios. We reported interesting findings, showing asynchronous BFT indeed performs consistently well during various failure scenarios. In particular, via a five-continent deployment on Amazon EC2 using 140 replicas, we showed the MiB instances have lower latency and much higher throughput than their asynchronous BFT counterparts.

We developed Chios, an intrusion-tolerant publish/subscribe system which protects against Byzantine failures. Chios is the first publish/subscribe system achieving decentralized confidentiality with fine-grained access control and strong publication order guarantees. This finding is in contrast to existing publish/subscribe systems achieving much weaker security and reliability properties. Chios is flexible and modular, consisting of four fully-fledged publish/subscribe configurations (each designed to meet different goals). We had deployed and evaluated our system on Amazon EC2. We compared Chios with various publish/subscribe systems. Chios is as efficient as an unreplicated, single-broker publish/subscribe implementation, only marginally slower than Kafka and Kafka with passive replication, and at least an order of magnitude faster than all Hyperledger Fabric modules and publish/subscribe systems using Fabric.

6.2 Future Work

This dissertation presents a few newest results on distributed system. We discuss below some topics for future research.

Dynamic membership in asynchronous BFT protocols. Although current asynchronous BFT protocols have many advantages and are appropriate for blockchain system where replicas can distribute in heterogeneous regions, but replicas cannot join and exit dynamically. The administrators have to stop the whole system to add or delete the replicas in the system. These protocols also take no measure to deal with those malicious replicas. It is worth studying how to design the dynamic asynchronous BFT protocols and make it practical.

Asynchronous BFT protocol in a nutshell? Asynchronous BFT is complex and includes several components: cryptographic scheme, RBC, and ABA. From 2016, the modern asynchronous BFT architecture is widely researched and several works were proposed, including HoneyBadgerBFT, BEAT, EPIC, Dumbo, Dumbo-mvba [125], Bolt-Dumbo [124]. Compared with the most famous partially synchronous BFT protocol, PBFT, which has only three steps, it is interesting to ask whether the performance of asynchronous BFT protocol can be improved.

BFT-SMaRt is a high-performance Byzantine fault-tolerant state machine replication library and it is already deployed in some real-world systems. Asynchronous BFT can be also implemented for example, BFT-SMaRt. It means that the asynchronous BFT library can be implemented in practice rather than in a one-time test.

BFT applications. This dissertation presents a complete and detailed system design for pub/sub system combined BFT with fine-grained access control. Compared with centralized system or weak fault-tolerant system, this approach gives us a great

guidance to employ the BFT as an oracle to achieve strong fault tolerance. Some applications can also take BFT into account.

Traditional anonymous communications cannot tolerate faulty nodes. They can detect failures when messages transmitting from senders to receivers using zero knowledge proof, trap messages or random partial checking. Meanwhile, most of the systems can only detect failures but how to penalize the faults is not considered. The nature of BFT is to tolerate malicious attacks and does not need to abort the whole system as most of anonymous systems do. However, BFT and anonymous communications have different goals and typologies. Is it possible to design an anonymous communication with BFT without affecting its efficiency? Low latency is significant for instant communication.

In the voting system, the election authority (EA) is one of the most important entities. However, the EA is centralized in some famous voting systems. Anonymous communications can be used to prevent liking a voter to their vote. However, the centralized EA can execute arbitrary attacks, for example, it can cancel some votes or abort the election. BFT can be used to tolerate faults even if some EA were faulty. It is worth studying whether it is possible to use the verification process to monitor the behavior of the EA [53]. Is it necessary to use BFT to tolerate a faulty EA?

Bibliography

[1] Amazon simple notification service (sns). https://aws.amazon.com/sns/.

[2] Apache kafka. https://kafka.apache.org.

[3] BEAT library. `https://github.com/fififish/beat`.

[4] Bug in ABA protocol's use of common coin. `https://github.com/amiller/HoneyBadgerBFT/issues/59`.

[5] Cms advances interoperability & patient access to health data through new proposals. https://edit.cms.gov/newsroom/fact-sheets/cms-advances-interoperability-patient-access-health-data-through-new-proposals.

[6] Ethereum: A secure decentralised generalised transaction ledger. In *Byzantium Version*.

[7] FAYE, simple pub/sub messaging for the web. https://faye.jcoglan.com.

[8] Google cloud pub/sub. `https://cloud.google.com/pubsub/`,.

[9] Hyperledger iroha. `https://github.com/hyperledger/iroha`.

[10] IOTA. In *https://www.iota.org/*.

[11] MQTT. www.mqtt.org/.

[12] Relic crypto library. `https://github.com/relic-toolkit`.

[13] Tendermint core. `https://github.com/tendermint/tendermint`.

[14] Tendermint core. In *https://github.com/tendermint/tendermint*.

[15] Zfec library. `https://pypi.python.org/pypi/zfec`.

[16] Leveldb, a fast and lightweight key/value database library by google. 2014.

[17] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie. Fault-scalable Byzantine fault-tolerant services. *ACM SIGOPS Operating Systems Review*, 39(5):59–74, 2005.

[18] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and A. Spiegelman. Solida: A blockchain protocol based on reconfigurable Byzantine consensus. In *OPODIS*, 2017.

[19] I. Abraham, D. Malkhi, and A. Spiegelman. Validated asynchronous Byzantine agreement with optimal resilience and asymptotically optimal time and word communication. *arXiv preprint arXiv:1811.01332*, 2018.

[20] I. Abraham, D. Malkhi, and A. Spiegelman. Asymptotically optimal validated asynchronous Byzantine agreement. In *Proceedings of the Symposium on Principles of Distributed Computing*, pages 337–346. ACM, 2019.

[21] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. Farsite: federated, available, and reliable storage for an incompletely trusted environment. In *OSDI*, 2002.

[22] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *J. Cryptographic Engineering*, 3(2):111–128, 2013.

[23] L. Allison and W. Brent. Decentralizing attribute-based encryption. In *Advances in Cryptology – EUROCRYPT*, 2011.

[24] Y. Amir, B. Coan, J. Kirsch, and J. Lane. Prime: Byzantine replication under attack. *IEEE Transactions on Dependable and Secure Computing*, 8(4):564–577, 2011.

[25] P.-L. Aublin, S. B. Mokhtar, and V. Quéma. RBFT: Redundant Byzantine fault tolerance. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*, pages 297–306. IEEE, 2013.

[26] J. Bacon, D. M. Eyers, J. Singh, and P. R. Pietzuch. Access control in publish/subscribe systems. In *Proceedings of the second international conference on Distributed event-based systems*, pages 23–34, 2008.

[27] J.-P. Bahsoun, R. Guerraoui, and A. Shoker. Making BFT protocols really adaptive. In *IPDPS*, pages 904–913. IEEE, 2015.

[28] P. S. L. M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In *Proceedings of the 12th International Conference on Selected Areas in Cryptography*, 2006.

[29] M. Baudet, A. Ching, A. Chursin, G. Danezis, F. Garillot, Z. Li, D. Malkhi, O. Naor, D. Perelman, and A. Sonnino. State machine replication in the libra blockchain. *The Libra Assn., Tech. Rep*, 2019.

[30] J. Behl, T. Distler, and R. Kapitza. Scalable BFT for multi-cores: Actor-based decomposition and consensus-oriented parallelization. In *10th Workshop on Hot Topics in System Dependability (HotDep 14)*, 2014.

[31] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In *PODC*, pages 27–30, 1983.

[32] M. Ben-Or, B. Kelmer, and T. Rabin. Asynchronous secure computations with optimal resilience. In *Proceedings of the 13th annual symposium on Principles of distributed computing*, pages 183–192. ACM, 1994.

[33] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. SNARKs for c: Verifying program executions succinctly and in zero knowledge. In *Annual cryptology conference*, pages 90–108. Springer, 2013.

[34] P. Berman and J. A. Garay. Randomized distributed agreement revisited. In *FTCS-23 The Twenty-Third International Symposium on Fault-Tolerant Computing*, pages 412–419. IEEE, 1993.

[35] A. Bessani, J. Sousa, and E. E. Alchieri. State machine replication for the masses with BFT-SMART. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 355–362. IEEE, 2014.

[36] A. Boldyreva. Threshold signatures, multisignatures and blind signatures based on the Gap-Diffie-Hellman-group signature scheme. In *PKC*, pages 31–46, 2003.

[37] G. Bracha. An asynchronous [(n-1)/3]-resilient consensus protocol. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 154–162. ACM, 1984.

[38] G. Bracha. Asynchronous Byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

[39] G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM (JACM)*, 32(4):824–840, 1985.

[40] M. Brandenburger, C. Cachin, R. Kapitza, and A. Sorniotti. Blockchain and trusted computing: Problems, pitfalls, and a solution for hyperledger fabric. 2018.

[41] M. Burrows. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 335–350. USENIX Association, 2006.

[42] C. Cachin, D. Collins, T. Crain, and V. Gramoli. Byzantine fault tolerant vector consensus with anonymous proposals. *arXiv preprint arXiv:1902.10010*, 2019.

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

[44] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in constantinople: Practical asynchronous Byzantine agreement using cryptography. *Journal of Cryptology*, 18(3):219–246, 2005.

[45] C. Cachin and J. A. Poritz. Secure intrusion-tolerant replication on the internet. In *DSN*, pages 167–176. IEEE, 2002.

[46] C. Cachin and S. Tessaro. Asynchronous verifiable information dispersal. In *SRDS*, pages 191–201. IEEE, 2005.

[47] C. Cachin and L. Zanolini. From symmetric to asymmetric asynchronous Byzantine consensus. *arXiv preprint arXiv:2005.08795*, 2020.

[48] R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively secure multi-party computation. In *STOC '96*, 1996.

[49] R. Canetti and T. Rabin. Fast asynchronous Byzantine agreement with optimal resilience. In *STOC*, volume 93, pages 42–51. Citeseer, 1993.

[50] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

[51] M. Castro, B. Liskov, et al. Practical Byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[52] T. Chang, S. Duan, H. Meling, S. Peisert, and H. Zhang. P2S: a fault-tolerant publish/subscribe infrastructure. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, pages 189–197, 2014.

[53] D. Chaum, R. T. Carback III, J. Clark, C. Liu, M. Nejadgholi, B. Preneel, A. T. Sherman, M. Yaksetig, and F. Zagórski. Votexx: A remote voting system that is coercion resistant. *UMBC Student Collection*, 2020.

[54] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 185–200. IEEE, 2019.

[55] A. Clement, M. Kapritsos, S. Lee, Y. Wang, L. Alvisi, M. Dahlin, and T. Riche. Upright cluster services. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 277–290, 2009.

[56] A. Clement, E. L. Wong, L. Alvisi, M. Dahlin, and M. Marchetti. Making Byzantine fault tolerant systems tolerate Byzantine faults. In *NSDI*, volume 9, pages 153–168, 2009.

[57] D. Collins, R. Guerraoui, J. Komatovic, P. Kuznetsov, M. Monti, M. Pavlovic, Y. Pignolet, D. Seredinschi, A. Tonkikh, and A. Xygkis. Online payments by merely broadcasting messages. In *DSN*, pages 26–38. IEEE, 2020.

[58] M. Correia, N. F. Neves, and P. Verissimo. How to tolerate half less one Byzantine nodes in practical distributed systems. In *SRDS*, pages 174–183. IEEE, 2004.

[59] M. Correia, N. F. Neves, and P. Veríssimo. From consensus to atomic broadcast: Time-free Byzantine-resistant protocols without signatures. *The Computer Journal*, 49(1):82–96, 2006.

[60] G. Coulouris, J. Dollimore, and T. Kindberg. Distributed systems, concepts and design, edition 4th, 2005.

[61] R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient multiparty computations secure against an adaptive adversary. In *EUROCRYPT*, 1999.

[62] I. Damien and R. Michel. Trading off t-resilience for efficiency in asynchronous Byzantine reliable broadcast. *Parallel Processing Letters.*, 26:8, 2016.

[63] A. Daneels and W. Salter. What is scada? 1999.

[64] A. S. de Sá, A. E. Silva Freitas, and R. J. de Araújo Macêdo. Adaptive request batching for Byzantine replication. *ACM SIGOPS Operating Systems Review*, 47(1):35–42, 2013.

[65] C. Decker, J. Seidel, and R. Wattenhofer. Bitcoin meets strong consistency. In *Proceedings of the 17th International Conference on Distributed Computing and Networking*, page 13. ACM, 2016.

[66] S. Duan, K. Levitt, H. Meling, S. Peisert, and H. Zhang. ByzID: Byzantine fault tolerance from intrusion detection. In *SRDS*, pages 253–264. IEEE, 2014.

[67] S. Duan, C. Liu, X. Wang, Y. Wu, S. Xu, Y. Yesha, and H. Zhang. Intrusion-tolerant and confidentiality-preserving publish/subscribe messaging. In *2020 International Symposium on Reliable Distributed Systems (SRDS)*, pages 319–328. IEEE, 2020.

[68] S. Duan, H. Meling, S. Peisert, and H. Zhang. BChain: Byzantine replication with high throughput and embedded reconfiguration. In *OPODIS*, pages 91–106, 2014.

[69] S. Duan, S. Peisert, and K. N. Levitt. hBFT: speculative Byzantine fault tolerance with minimum cost. *IEEE Transactions on Dependable and Secure Computing*, 12(1):58–70, 2014.

[70] S. Duan, M. K. Reiter, and H. Zhang. BEAT: Asynchronous BFT made practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 2028–2041. ACM, 2018.

[71] S. Duan and H. Zhang. Practical state machine replication with confidentiality. In *SRDS*, 2016.

[72] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.

[73] A. N. B. et. al. Depspace: A Byzantine fault-tolerant coordination service. In *EuroSys*, 2008.

[74] E. A. et. al. Hyperledger fabric: A distributed operating system for permissioned blockchains. In *Proceedings of EuroSys 2018 conference*, 2018.

[75] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse. Bitcoin-NG: A scalable blockchain protocol. In *NSDI*, pages 45–59, 2016.

[76] B. Eze, C. Kuziemsky, L. Peyton, G. Middleton, and A. Mouttham. Policy-based data integration for e-health monitoring processes in a B2B environment: experiences from canada. volume 5, pages 56–70. Multidisciplinary Digital Publishing Institute, 2010.

[77] L. Fiege, A. Zeidler, A. Buchmann, R. Kilian-Kehr, G. Mühl, and T. Darmstadt. Security aspects in publish/subscribe systems. In *Third Intl. Workshop on Distributed Event-based Systems (DEBS'04), Edinburgh, Scotland, UK*. IET, 2004.

[78] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. Technical report, Massachusetts Inst of Tech Cambridge lab for Computer Science, 1982.

[79] R. Friedman, A. Mostefaoui, and M. Raynal. Simple and efficient oracle-based consensus protocols for asynchronous Byzantine systems. *IEEE Transactions on Dependable and Secure Computing*, 2(1):46–56, 2005.

[80] G. Golan-Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. K. Reiter, D. Seredinschi, O. Tamir, and A. Tomescu. SBFT: A scalable and decentralized trust infrastructure. In *DSN*, pages 568–580, 2019.

[81] R. Guerraoui, N. Knežević, V. Quéma, and M. Vukolić. The next 700 BFT protocols. *ACM Transactions on Computer Systems*, 32(4):12:1–12:45, 2015.

[82] R. Guerraoui, J. Komatovic, P. Kuznetsov, Y.-A. Pignolet, D.-A. Seredinschi, and A. Tonkikh. Dynamic Byzantine reliable broadcast. In *24th International Conference on Principles of Distributed Systems (OPODIS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[83] R. Guerraoui, P. Kuznetsov, M. Monti, M. Pavlovic, D.-A. Seredinschi, and Y. Vonlanthen. Scalable Byzantine reliable broadcast (extended version). *arXiv preprint arXiv:1908.01738*, 2019.

[84] A. Guillevic. Comparing the pairing efficiency over composite-order and prime-order elliptic curves. In *ACNS*, 2013.

[85] B. Guo, Z. Lu, Q. Tang, J. Xu, and Z. Zhang. Dumbo: Faster asynchronous BFT protocols. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020.

[86] B. Guo, Z. Lu, Q. Tang, J. Xu, and Z. Zhang. Dumbo: Faster asynchronous BFT protocols. *IACR Cryptol. ePrint Arch.*, 2020:841, 2020.

[87] M. Hastings, B. Hemenway, D. Noble, and S. Zdancewic. SoK: General purpose compilers for secure multi-party computation. In *S&P*, 2019.

[88] J. Hendricks, G. R. Ganger, and M. K. Reiter. Verifying distributed erasure-coded data. In *PODC*, 2007.

[89] J. Hendricks, S. Sinnamohideen, G. R. Ganger, and M. K. Reiter. Zzyzx: Scalable fault tolerance through Byzantine locking. In *DSN*, pages 363–372. IEEE, 2010.

[90] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX annual technical conference*, volume 8. Boston, MA, USA, 2010.

[91] M. Ion, G. Russello, and B. Crispo. An implementation of event and filter confidentiality in pub/sub systems and its application to e-health. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 696–698, 2010.

[92] M. Ion, G. Russello, and B. Crispo. Supporting publication and subscription confidentiality in pub/sub networks. In *International Conference on Security and Privacy in Communication Systems*, pages 272–289. Springer, 2010.

[93] M. Ion, G. Russello, and B. Crispo. Design and implementation of a confidentiality and access control solution for publish/subscribe systems. volume 56, pages 2014–2037. Elsevier, 2012.

[94] A. Iyengar, R. Cahn, J. A. Garay, and C. Jutla. Design and implementation of a secure distributed data repository. In *14th IFIP Internat. Information Security Conf*, 1998.

[95] L. Jehl and H. Meling. Towards Byzantine fault tolerant publish/subscribe: A state machine approach. In *Proceedings of the 9th Workshop on Hot Topics in Dependable Systems*, pages 1–5, 2013.

[96] B. John, S. Amit, and W. Brent. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, 2007.

[97] Joonsang Baek and Yuliang Zheng. Simple and efficient threshold cryptosystem from the gap Diffie-Hellman group. In *GLOBECOM '03*, 2003.

[98] A. Kate, Y. Huang, and I. Goldberg. Distributed key generation in the wild. *IACR Cryptology ePrint Archive*, 2012.

[99] A. Kate, Y. Huang, and I. Goldberg. Distributed key generation in the wild. volume 2012, page 377. Citeseer, 2012.

[100] R. S. Kazemzadeh and H.-A. Jacobsen. Reliable and highly available distributed publish/subscribe service. In *2009 28th IEEE International Symposium on Reliable Distributed Systems*, pages 41–50. IEEE, 2009.

[101] R. S. Kazemzadeh and H.-A. Jacobsen. Publiyprime: Exploiting overlay neighborhoods to defeat Byzantine publish/subscribe brokers. 2013.

[102] H. Khurana and R. K. Koleva. Scalable security and accounting services for content-based publish/subscribe systems. In *SAC*, 2005.

[103] V. King and J. Saia. Breaking the $o(n^2)$ bit barrier: scalable Byzantine agreement with an adaptive adversary. *Journal of the ACM (JACM)*, 58(4):18, 2011.

[104] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *USENIX Security*, pages 279–296, 2016.

[105] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, and B. Ford. Omniledger: A secure, scale-out, decentralized ledger. *IACR Cryptology ePrint Archive*, 2017:406, 2017.

[106] E. Kokoris-Kogias, A. Spiegelman, D. Malkhi, and I. Abraham. Bootstrapping consensus without trusted setup: Fully asynchronous distributed key generation. *IACR Cryptology ePrint Archive*, 2019.

[107] R. Kolta, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: speculative Byzantine fault tolerance. *ACM Transactions on Computer Systems*, 27(4):7:1–7:39, 2009.

[108] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*, pages 839–858. IEEE, 2016.

[109] R. L. Krutz. *Securing SCADA systems*. John Wiley & Sons, 2005.

[110] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, et al. Oceanstore: An architecture for global-scale persistent storage. In *ASPLOS*, 2000.

[111] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, et al. Oceanstore: An architecture for global-scale persistent storage. volume 34, pages 190–201. ACM New York, NY, USA, 2000.

[112] K. Kursawe and V. Shoup. Optimistic asynchronous atomic broadcast. In *ICALP*, pages 204–215, 2005.

[113] L. Lamport. Proving the correctness of multiprocess programs. *IEEE transactions on software engineering*, (2):125–143, 1977.

[114] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[115] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. In *Concurrency: the Works of Leslie Lamport*, pages 203–226. 2019.

[116] L. Leslie. Paxos made simple. In *ACM Sigact News*, 2001.

[117] B. Libert, M. Joye, and M. Yung. Born and raised distributively: Fully distributed non-interactive adaptively-secure threshold signatures with short shares. *Theoretical Computer Science*, 645:1–24, 2016.

[118] B. Libert and M. Yung. Adaptively secure non-interactive threshold cryptosystems. In *ICALP*, 2011.

[119] C. Liu, S. Duan, and H. Zhang. EPIC: Efficient asynchronous BFT with adaptive security. In *DSN*, 2020.

[120] C. Liu, S. Duan, and H. Zhang. MiB: Asynchronous BFT with more replicas. *arXiv preprint arXiv:2108.04488*, 2021.

[121] J. Liu, W. Li, G. O. Karame, and N. Asokan. Scalable Byzantine consensus via hardware-assisted secret sharing. *IEEE Transactions on Computers*, 68(1):139–151, 2018.

[122] J. Loss and T. Moran. Combining asynchronous and synchronous Byzantine agreement: The best of both worlds. *IACR Cryptology ePrint Archive*, 2018:235, 2018.

[123] D. Lu, T. Yurek, S. Kulshreshtha, R. Govind, A. Kate, and A. Miller. HoneybadgerMPC and asynchroMix: Practical asynchronous mpc and its application to anonymous communication. In *CCS '19*, 2019.

[124] Y. Lu, Z. Lu, and Q. Tang. Bolt-dumbo transformer: Asynchronous consensus as fast as pipelined bft. *arXiv preprint arXiv:2103.09425*, 2021.

[125] Y. Lu, Z. Lu, Q. Tang, and G. Wang. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, pages 129–138, 2020.

[126] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena. A secure sharding protocol for open blockchains. In *CCS*, pages 17–30. ACM, 2016.

[127] E. MacBrough. Cobalt: BFT governance in open networks. *arXiv preprint arXiv:1802.07240*, 2018.

[128] J.-P. Martin and L. Alvisi. Fast Byzantine paxos. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 402–411, 2004.

[129] J.-P. Martin and L. Alvisi. Fast Byzantine consensus. *IEEE Transactions on Dependable and Secure Computing*, 3(3):202–215, 2006.

[130] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. The honey badger of BFT protocols. In *Proceedings of the SIGSAC Conference on Computer and Communications Security*, pages 31–42. ACM, 2016.

[131] H. Moniz, N. F. Neves, and M. Correia. Byzantine fault-tolerant consensus in wireless ad hoc networks. *IEEE Transactions on Mobile Computing*, 12(12):2441–2454, 2012.

[132] H. Moniz, N. F. Neves, M. Correia, and P. Veríssimo. Experimental comparison of local and shared coin randomized consensus protocols. In *SRDS*, pages 235–244, 2006.

[133] H. Moniz, N. F. Neves, M. Correia, and P. Verissimo. RITAS: Services for randomized intrusion tolerance. *IEEE transactions on dependable and secure computing*, 8(1):122–136, 2008.

[134] A. Mostefaoui, H. Moumen, and M. Raynal. Signature-free asynchronous Byzantine consensus with $t < n/3$ and $o(n^2)$ messages. In *PODC*, pages 2–9. ACM, 2014.

[135] A. Mostéfaoui and M. Raynal. Signature-free asynchronous Byzantine systems: from multivalued to binary consensus with $t < n/3$ messages, and constant time. *Acta Informatica*, 54(5):501–520, 2017.

[136] K. Nadiminti, M. D. De Assunçao, and R. Buyya. Distributed systems and recent innovations: Challenges and benefits. *InfoNet Magazine*, 16(3):1–5, 2006.

[137] D. Naor, M. Naor, and J. Lotspiech. Revocation and tracing schemes for stateless receivers. In *Annual International Cryptology Conference*, pages 41–62. Springer, 2001.

[138] K. Nayak, L. Ren, E. Shi, N. H. Vaidya, and Z. Xiang. Improved extension protocols for Byzantine broadcast and agreement. *arXiv preprint arXiv:2002.11321*, 2020.

[139] Z. Nejc and J. Kaiwen, Zhangand Hans-Arno. Hyperpubsub: a decentralized, permissioned, publish/subscribe service using blockchains: demo. In *Middleware*, 2017.

[140] D. Ongaro and J. Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319, 2014.

[141] E. Onica, P. Felber, H. Mercier, and E. Rivière. Confidentiality-preserving publish/subscribe: A survey. volume 49, pages 1–43. ACM New York, NY, USA, 2016.

[142] R. Padilha and F. Pedone. Belisarius: BFT storage with confidentiality. In *NCA*, 2011.

[143] R. Pass and E. Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *DISC*, 2017.

[144] R. Pass and E. Shi. Thunderella: blockchains with optimistic instant confirmation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–33. Springer, 2018.

[145] K. Patel. Secure multiparty computation using secret sharing. In *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*, pages 863–866. IEEE, 2016.

[146] A. Patra, A. Choudhary, and C. Pandu Rangan. Simple and efficient asynchronous Byzantine agreement with optimal resilience. In *PODC*, pages 92–101. ACM, 2009.

[147] M. O. Rabin. Randomized Byzantine generals. In *SFCS*, pages 403–409. IEEE, 1983.

[148] G. S. Ramachandran, K.-L. Wright, L. Zheng, P. Navaney, M. Naveed, B. Krishnamachari, and J. Dhaliwal. Trinity: A Byzantine fault-tolerant distributed publish-subscribe system with immutable blockchain-based persistence. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 227–235. IEEE, 2019.

[149] H. V. Ramasamy and C. Cachin. Parsimonious asynchronous Byzantine-fault-tolerant atomic broadcast. In *OPODIS*, 2005.

[150] M. K. Reiter and K. P. Birman. How to securely replicate services. In *ACM TOPLAS*, 1994.

[151] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.

[152] V. Shoup. Practical threshold signatures. In *EUROCRYPT 2000*.

[153] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *J. Cryptol.*, 15(2):75–96, Jan. 2002.

[154] Y. J. Song and R. van Renesse. Bosco: One-step Byzantine asynchronous consensus. In *International Symposium on Distributed Computing*, pages 438–450. Springer, 2008.

[155] T. Srikanth and S. Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2(2):80–94, 1987.

[156] M. Srivatsa and L. Liu. Securing publish-subscribe overlay services with eventguard. In *Proceedings of the 12th ACM conference on Computer and communications security*, pages 289–298, 2005.

[157] M. A. Tariq, B. Koldehofe, and K. Rothermel. Securing broker-less publish/subscribe systems using identity-based encryption. *IEEE transactions on parallel and distributed systems*, 25(2):518–528, 2013.

[158] P. Tholoniat and V. Gramoli. Formal verification of blockchain Byzantine fault tolerance. *arXiv preprint arXiv:1909.07453*, 2019.

[159] S. Toueg. Randomized Byzantine agreements. In *PODC*, pages 163–178. ACM, 1984.

[160] C. Tyler, G. Vincent, L. Mikel, and R. Michel. Dbft: Efficient leaderless Byzantine consensus and its application to blockchains. In *17th International Symposium on Network Computing and Applications*, 2018.

[161] M. Van Steen and A. Tanenbaum. Distributed systems principles and paradigms. *Network*, 2:28, 2002.

[162] S. Victor and G. Rosario. Securing threshold cryptosystems against chosen ciphertext attack. In *EUROCRYPT*, 1998.

[163] S. Vinoski. Total order in content-based publish/subscribe systems. In *Advanced message queuing protocol. IEEE Internet Computing*, 2006.

[164] A. Wun and H.-A. Jacobsen. A policy management framework for content-based publish/subscribe middleware. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 368–388. Springer, 2007.

[165] Y. Yeh. Safety critical avionics for the 777 primary flight controls system. In *20th DASC. 20th Digital Avionics Systems Conference (Cat. No. 01CH37219)*, volume 1, pages 1C2–1. IEEE, 2001.

[166] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin. Separating agreement from execution for Byzantine fault tolerant services. In *SOSP*, 2003.

[167] M. Yin, D. Malkhi, M. Reiterand, G. G. Gueta, and I. Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In *38th ACM symposium on Principles of Distributed Computing (PODC)*, 2019.

[168] M. Zamani, M. Movahedi, and M. Raykova. Rapidchain: A fast blockchain protocol via full sharding. In *CCS*, pages 931–948, 2018.

[169] K. Zhang, V. Muthusamy, and H.-A. Jacobsen. Total order in content-based publish/subscribe systems. In *ICDCS*, 2012.

[170] Y. Zhao and D. C. Sturman. Dynamic access control in a content-based publish/subscribe system with delivery guarantees. In *ICDCS*, 2006.

[171] P. Zielinski. Optimistically terminating consensus: All asynchronous consensus protocols in one framework. In *ISPDC*, pages 24–33. IEEE, 2006.

[172] R. Zurawski. *Industrial communication technology handbook, Second Edition.* CRC Press, 2014.