Chair of Network Architectures and Services
TUM School of Computation, Information and Technology
Technical University of Munich

Robustness and Performance of Fault-Tolerant Consensus

Richard Carl von Seck

Dissertation

Technische Universität München
TUM School of Computation, Information and Technology

Robustness and Performance of Fault-Tolerant Consensus

Richard Carl von Seck

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitz:		Prof. Dr. Andreas Herkersdorf
Prüfende der Dissertation:	1.	Prof. Dr.-Ing. Georg Carle
	2.	Prof. Dr. rer. nat. Hannes Hartenstein

Die Dissertation wurde am 18.02.2025 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 12.07.2025 angenommen.

Cataloging-in-Publication Data
Richard Carl von Seck
Robustness and Performance of Fault-Tolerant Consensus
Dissertation, Juli 2025
Network Architectures and Services,
TUM School of Computation, Information and Technology
Technische Universität München

ISBN 978-3-937201-84-9
ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)
DOI 10.2313/NET-2025-11-2
Network Architectures and Services NET-2025-11-2
Series Editor: Georg Carle, Technische Universität München, Germany

Abstract:

Construction of reliable distributed systems is subject to research for more than 40 years. Byzantine fault-tolerant (BFT) state machine replication (SMR) is an essential technique in this pursuit. While modern systems offer increasing performance and efficiency, BFT-SMR remains prohibitive for many use cases, due to considerable costs and overhead. This demand for optimization induced the creation of a plethora of approaches. The state of the art is complex. An abundance of architectures, tradeoffs, and scenario-specific behavior complicates choosing the right tool for the job. The necessary effort for conception, modification, proof, and implementation of BFT-SMR systems is considerable.

We present an alternative approach toward enhanced performance of practical BFT-SMR. By careful modification of BFT-SMR components – building blocks – it is possible to improve, potentially a range of, systems while preserving existing safety and liveness guarantees. We identify and study promising building blocks, frequently assumed in related work. We choose a state-of-the-art BFT-SMR system as representative example, studying candidate building blocks in its context. For each candidate, we analyze, model, implement, and experimentally quantify resulting behavior. In the scope of this thesis, we study the building blocks of (1) underlying networking and (2) underlying cryptographic primitives in more detail. Overall, we investigate two secure channel implementations, four transport protocols, and four signature schemes, while varying a range of BFT-SMR and protocol parameters, including additional metrics such as CPU load and bandwidth.

Our results demonstrate potential for both performance improvement and additional overhead, depending on building block, configuration, and scenario. For (1), we consider TCP to be the best default choice, except in edge cases. While secure channels incur slight overhead, their usage is functional. Under packet loss, we observe significant performance differences between protocols and configurations, with up to $\sim 20\%$ improvement for some scenarios. For (2), costly cryptographic base operations outweigh theoretical scaling benefits in small to medium-sized deployments. Threshold signature scheme architecture limits applicability for BFT-SMR. Semi-interactive schemes may be a promising alternative, assuming sufficiently performant presigning. Notwithstanding varying replication latency, comparable throughput can be achieved via e.g., request batching. Our study suggests potential advantages of threshold signature schemes in bandwidth-limited scenarios.

Optimization of BFT-SMR building blocks is a convenient approach, that potentially applies to a range of systems without affecting their proven guarantees. Our results demonstrate improvement potential, although the concrete impact varies, depending on building block, upper-layer system, and scenario. We contribute to BFT-SMR research in the chosen areas and extend the set of studied protocols and schemes in number and detail.

Kurzfassung:

Die Konstruktion von zuverlässigen Systemen ist seit mehr als 40 Jahren Gegenstand intensiver Forschung. State Machine Replication (SMR) mit byzantinischer Fehlertoleranz (BFT) spielt eine zentrale Rolle. Trotz steigender Effizienz moderner Systeme bleibt der Einsatz von BFT-SMR durch Mehrkosten in vielen Szenarien ungeeignet. Aus diesem Optimierungsbedarf entstand eine Vielzahl von Ansätzen. Der Stand der Technik ist komplex. Die Fülle an verfügbaren Architekturen, Zielkonflikten und szenariospezifischem Verhalten erschweren die Wahl einer geeigneten Lösung. Der mit Konzeption, Veränderung, Beweisen und Implementierung von BFT-SMR Systemen verbundene Aufwand ist signifikant.

Wir präsentieren einen alternativen Ansatz, mit dem Ziel verbesserter, praktischer BFT-SMR Systeme. Umsichtige Modifikation von BFT-SMR Komponenten – "Building Blocks" – ermöglicht die Optimierung von, potentiell einer Reihe von, Systemen, während bewiesene Safety und Liveness Garantien erhalten bleiben. Wir untersuchen Building Blocks, welche in zahlreichen Systemen vorausgesetzt werden. Als repräsentatives Beispiel und Studienkontext wählen wir ein einflussreiches BFT-SMR System. Wir analysieren, modellieren, implementieren und quantifizieren modifizierte Building Blocks experimentell. Im Rahmen dieser Dissertation untersuchen wir zugrundeliegende (1) Netzwerkkommunikation und (2) kryptographische Signaturschemata ausführlich. Wir begutachten zwei Implementierungen sicherer Kommunikationskanäle, vier Transportprotokolle und vier Signaturschemata, unter Variation einer Reihe von BFT-SMR- und Protokollparametern sowie Studie von erweiterten Metriken wie beispielsweise CPU-Last und Bandbreite.

Unsere Ergebnisse zeigen sowohl Verbesserungen als auch Einbußen, in Abhängigkeit von Building Block, Konfiguration und Szenario. In Kategorie (1) betrachten wir TCP als beste Standardwahl, außer in Grenzfällen. Trotz geringer Mehrkosten von sicheren Kanälen, ist ihre Nutzung zweckmäßig. Wir beobachten Performanzunterschiede zwischen Protokollen und Konfigurationen unter Paketverlust, mit Verbesserungen von bis zu $\sim 20\%$ in manchen Szenarien. In Kategorie (2) überwiegen teure, kryptographische Basisoperationen theoretische Skalierungsvorteile in kleinen und mittelgroßen Systemen. Die Architektur von Schwellwertkryptographie limitiert ihre Anwendbarkeit. Semi-interaktive Schemata sind eine mögliche Alternative, unter Annahme performanter Vorsignaturphasen. Trotz variabler Latenz ist ähnlicher Durchsatz, mittels beispielsweise Stapelverarbeitung, möglich. In Szenarien mit limitierter Bandbreite sind Schwellwertschemata potentiell vorteilhaft.

Optimierung von Building Blocks ist zweckmäßig, mit Wirkung auf, potentiell, eine Reihe von Systemen, ohne bewiesene Eigenschaften zu gefährden. Unsere Ergebnisse zeigen unterschiedliches Potential, in Abhängigkeit von Building Block, aufbauenden Systemen und Szenario. Wir leisten einen Beitrag zu BFT-SMR Forschung in den gewählten Bereichen und erweitern Menge und Detail der untersuchten Protokolle und Schemata.

Contents

II Optimization via the Underlying Network

III Optimization via the Underlying Cryptographic Primitives

IV Closure

List of Figures

List of Tables

1. Problem Statement & Goals

> *"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable."*
>
> Leslie Lamport, E-Mail to mailing list at Systems Research Center (1987) [1]

1.1 Introduction

Building reliable and secure computer systems is becoming ever more important, considering that almost all aspects of modern life are related to and directly affected by automated data processing and global interconnection through the Internet. Lamport explicitly characterized the negative impact potential of such a distributed system [1]. Construction of resilient systems is a core objective of distributed systems research for more than four decades [2]. By designing architectures in anticipation of faulty processes, a higher overall robustness can be achieved. A central concept to realize fault-tolerance is State Machine Replication (SMR) [3]. A target service is modeled as an abstract automaton and replicated to a number of independent processors. By redundant execution of the same operations on all processors and aggregation of computation results, the system is able to tolerate a minority of faulty processors. SMR systems are a promising approach to build reliable and secure infrastructure since the byzantine fault model (§2.1) includes robustness against malicious intelligence and equivocation. Naturally, this makes them suitable for scenarios with a need for high reliability, such as data centers, critical infrastructure, or transportation. At the heart of SMR lies the agreement on and synchronization of data between distributed processes. Since the inception of Bitcoin [4] in 2008, the rise of blockchains and cryptocurrencies sparked a wave of research, applying classical SMR methods for fault-tolerant agreement to the settlement of financial transactions [5].

Inherent challenges of building practical fault-tolerant systems are performance and scalability limitations. Redundant execution of a state machine requires additional hardware to set up and run. The agreement protocols typically incur a significant communication and processing overhead. In fact, the limitations on practical implementations through this overhead are so significant, that even modern systems, with explicit focus on scalability, struggle to maintain performance beyond hundreds or even tens of nodes [6–13].

Strategies to overcome these limitations and improve performance, scalability, or robustness are manifold (§4). Lower bounds on certain algorithm complexities or solvability in

the domain are known for decades [14, 15]. Hence, a significant amount of SMR optimization research is focused on the study of conceptual modifications to agreement algorithms or model assumptions for a specific use case or environment [16]. Relevant improvements in one parameter dimension are usually paid for with limitations in another dimension, or as formulated by Lamport: *"[...] there are certain fundamental costs required of any solution. The only way to reduce these costs is to reduce the degree of fault-tolerance – by either decreasing the number of faults or restricting the class of faults that can be handled."* [17].

1.2 Research Objectives

While SMR presents itself as a suitable solution to increase the robustness of arbitrary systems in theory, in practice, the operation and complexity costs are still high enough to prevent ubiquitous deployment. Thus, advances toward increased usability, performance, and scalability of practical SMR are in high demand. However, this goal is non-trivial. In over four decades of distributed systems research, a myriad of SMR protocols and optimizations have been proposed [6–13, 16, 18–38]. Still, fundamental complexities, overhead, and limitations remain. The objective of this thesis is to improve *practical* SMR systems. We aim to avoid increasing the existing complexity of the field by introducing even more conceptual tradeoffs on SMR protocol level. Thus, we ask the following research questions:

Q1 *How can the performance of contemporary, practical SMR be improved?*

We can detail **Q1** and divide it into the following sub-questions:

Q1.1 *Are fundamental improvements still possible?*

Adjustments and optimization of an algorithm or system by specialization or introduction of tradeoffs will most likely yield improvements in the special environment accounted for. Preferrable would be more *fundamental* improvements, that positively affect a larger group of (existing) protocols, scenarios, or tools.

Q1.2 *How can safety and liveness properties be preserved?*

Correct design and implementation of SMR algorithms is hard [16]. Usage of existing protocols with proven safety and liveness guarantees is generally preferable to development from scratch. However, modifications of the algorithm may invalidate proofs and guarantees. Is there a way to achieve (fundamental) optimization without endangering safety and liveness properties?

Q1.3 *Which system aspects should be investigated?*

Considering the directions and objectives derived from **Q1.1** and **Q1.2**, which aspects of an SMR system should be primarily considered for optimization efforts?

Now, we assume that we are able to successfully identify a selection of optimization approaches that satisfy the desirable properties discussed in context of **Q1**. In this case, the obvious question is how well these optimizations perform:

Q2 *How big is the impact of the identified optimization space?*

We note, that **Q2** applies to and should be evaluated for all identified optimization spaces individually. To prevent redundant formulation of the same question for multiple targets, we keep **Q2** independent of the optimization space here and answer the question in detail in the respective thesis part.

We can further detail **Q2** and divide it into the following sub-questions:

Q2.1 *What are necessary preconditions for benefits?*

Each optimization space is subject to a number of parameters, influencing behavior and performance. A configuration might not produce a positive effect in a given scenario. What are necessary preconditions for e.g., performance benefits to occur?

Q2.2 *How big are potential performance benefits?*

Considering a matching execution scenario and parameter configuration, improved performance may be possible. How significant are the potential performance benefits?

Resulting performance and limitations are then described for each of the identified spaces. Some results may apply to SMR systems in general, including systems limited to tolerance against less general fault classes such as crash faults. In the context of this thesis, we primarily focus on *practical* SMR systems aiming to tolerate arbitrary (byzantine) faults (§2.1), also known as systems with Byzantine Fault Tolerance (BFT).

1.3 Structure of this Thesis

We present discussion and answers to the posed research questions in their order of introduction. We first answer **Q1** in Part I, then we answer **Q2** for different building blocks by evaluation of different optimization approaches in Part II and III, and finally we conclude the thesis with a discussion of our research questions in Part IV.

Part I We answer **Q1** by proposing the optimization of BFT-SMR building blocks. Careful application of this method to existing systems and algorithms can preserve proven safety and liveness guarantees (**Q1.2**), while allowing for performance gains that are potentially applicable to a whole range of systems (**Q1.1**). We analyze state-of-the-art systems and identify commonly assumed building blocks (**Q1.3**) as candidates for optimization.

Part II We answer **Q2** for the impact of network transport optimization on leader-based BFT-SMR. First, typical assumptions, guarantees, and a selection of relevant network protocols are introduced. We choose a representative, state-of-the-art BFT-SMR system as a base and conduct a detailed analysis of the protocol and configuration space. This allows identification of optimization strategies and necessary preconditions (**Q2.1**). Finally, identified changes are implemented, and the impact is empirically evaluated through experiments (**Q2.2**).

Part III We answer **Q2** for the impact of variation of cryptographic primitives on leader-based BFT-SMR. Here, the primary focus lies on application of threshold cryptography and signature schemes. The use of threshold cryptography in BFT-SMR is motivated and relevant schemes and their architectures are presented. Using the representative BFT-SMR system from Part II, we conduct a detailed analysis of the protocol and optimization space in this context (**Q2.1**). A selection of threshold signature schemes is integrated into the system and the impact is empirically evaluated through experiments (**Q2.2**).

Part IV We conclude the thesis with a discussion of and answers to our research questions and a comparison to the state of the art.

1.4 Publications in the Context of this Thesis

The following publications were conducted in context of this thesis. At the beginning of each thesis part, we denote relations between contents of the respective chapters and already published results, as well as an overview of the author's contributions.

- Richard von Seck, Filip Rezabek, Benedikt Jaeger, Sebastian Gallenmüller, and Georg Carle, *BFT-Blocks: The Case for Analyzing Networking in Byzantine Fault Tolerant Consensus* in 2022 IEEE 21st International Symposium on Network Computing and Applications (NCA), volume 21, pages 35–44, 2022.
- Richard von Seck, Filip Rezabek, Sebastian Gallenmüller, and Georg Carle, *On the Impact of Network Transport Protocols on Leader-Based Consensus Communication* in Proceedings of the 6th ACM International Symposium on Blockchain and Secure Critical Infrastructure, BSCI '24, page 1–11, New York, NY, USA, 2025. Association for Computing Machinery.
- Richard von Seck, Filip Rezabek, and Georg Carle, *Thresh-Hold: Assessment of Threshold Cryptography in Leader-Based Consensus* in 2024 IEEE 49th Conference on Local Computer Networks (LCN), pages 1–8. IEEE, 2024.

2. Background: Distributed Systems

> *"[...] The goal of implementing completely fault-tolerant computing systems is unattainable. Fortunately, most applications do not require complete fault-tolerance. Rather, it is sufficient that a system work correctly provided that no more than some predefined number of failures occur [...]."*
>
> Richard D. Schlichting and Fred B. Schneider,
> Fail-Stop Processors: An Approach to Designing Fault-Tolerant Computing Systems (1983) [2]

Construction of distributed and reliable systems involves dedicated effort to ensure that the infrastructure runs as expected, even if operated under dynamic or adverse conditions. As outlined by Schlichting and Schneider [2], reliable operation under a predefined number of failures is typically considered sufficient. This robustness can be enabled by a set of core techniques and concepts, which we briefly introduce in this chapter. We first describe common formalizations of fault and synchrony models. Based on these models, we introduce replication as a mechanism for fault-tolerant operation. Finally, we illustrate a selection of distributed systems models and primitives, such as ordering, broadcast, and consensus, that can be used to implement replication. In distributed systems literature, algorithms and statements are typically made for and about independent components or agents that interact e.g., via message passing. Depending on the context, different terms such as *processors*, *processes*, or *replicas* for these components are used. In the scope of this thesis, we refer to these participants as *node*, *process* or *replica* interchangeably.

2.1 Fault Models

Let $\mathcal{N}$ be a set of *nodes*, where $n := |\mathcal{N}|, n \in \mathbb{N}$. Nodes are constructed from a specification and are assumed to behave accordingly. If a node is not behaving consistently with its specification [2, 3], we call this node *faulty*, *honest* otherwise. We denote the subset of faulty nodes as $\mathcal{F} \subseteq \mathcal{N}$, the number of faulty nodes as $f := |\mathcal{F}|, f \leq n$, the set of honest nodes as $\mathcal{H} := \{\mathcal{N} \setminus \mathcal{F}\}$, and the number of honest nodes as $h := (n - f) = |\mathcal{H}|$.

For the construction of fault-tolerant systems, there exist different metrics to quantify the resulting fault tolerance. Possible indicators include the number of failures within some time interval [2], the mean time between failures over a given time interval [3], and *t-fault tolerance* where a system can transparently tolerate up to t faults at the same time

without deviating from its specification [3]. The latter property is also referred to as *t-resilience* [15, 39, 40] and is one of the most popular metrics to quantify fault tolerance. After some fault occurs, typically a diagnostic and reconfiguration routine is assumed, which restores previously faulty nodes to continuously provide t-resilience [17].

Depending on the target scenario, different assumptions on potentially occurring faults or expected system behavior are made. Weaker assumptions typically allow for stronger faults; tolerating stronger faults requires more costly countermeasures. In the following, we briefly introduce the most prevalent fault models, Crash Faults (§2.1.1), Byzantine Faults (§2.1.2), and other fault models (§2.1.3).

2.1.1 Crash Faults

Upon encountering a *crash fault*, a node halts but does not execute any action that would otherwise violate its specification [2,41,42]. In particular, nodes are assumed to not recover on their own [42] and stay dead until diagnosis and reconfiguration is conducted. If a system is resilient against crash faults, we refer to it as offering Crash Fault Tolerance (CFT). Crash faults are a convenient model since crash fault resilience generally requires less resources than resilience against stronger faults [39]. Crash faults are assumed in prominent approaches toward fault-tolerance such as Paxos [43] or Raft [44]. From a rigorous perspective however, behavior of a real-world system is seldomly limited to crash faults in practice. Hence, crash fault behavior can be interpreted as a primitive to construct, rather than a guarantee to assume. Research provides constructions and characteristics of *fail-stop processors* [2, 41] that approximate crash fault behavior in practice.

2.1.2 Byzantine Faults

Dubbed after Lamport's [45] story about generals of the byzantine army in need of agreement, *byzantine faults* describe arbitrary behavior in case of a fault. No assumptions about faulty behavior are made. In particular, this includes equivocation, issuance of false statements of a node about itself and others, and malicious collusion between faulty nodes, rendering it a suitable model of potential malicious (attacker) intelligence. Hence, byzantine resilience has been proposed as a counter-mechanism against security incidents [46,47]. If a system is resilient against byzantine faults, we refer to it as offering BFT.

2.1.3 Other Fault Models

While we mostly focus on BFT systems, we briefly introduce other fault models for completeness. Many primitives or architectures aiming to provide fault tolerance assume reliable communication between nodes (§5.1). However, some works explicitly consider *omission faults* [18, 19, 48], where nodes fail to send or receive messages that are expected by their specification, or *timing failures* where a node violates a timing assumption placed on it [48]. Another branch of work uses trusted hardware components and Trusted Execution Environments (TEEs) to constrain failure behavior by e.g., implementing trusted append-only memory [49] or certified counter values [50]. A central goal is to reduce computational and communication costs and enhance robustness of fault-tolerant systems [51, 52].

2.2 Synchrony Models

Synchronization is a fundamental problem of distributed systems [53] and sometimes necessary to ensure correct execution or any progress at all [14]. Hence, for modeling and design it is functional to formalize the relative speeds of processes or transmission times. Assumptions on these characteristics are formalized as *synchrony models* in literature. We distinguish between and briefly introduce three prevalent models; Synchrony (§2.2.1), Asynchrony (§2.2.2), and the hybrid model of Partial Synchrony (§2.2.3).

2.2.1 Synchrony

In *Synchrony* there exist known, fixed upper bounds on process speeds (Φ) and transmission times (Δ) [18]. Another prevalent assumption is access to synchronized clocks with bounded clock drift [14, 42, 45]. A formal definition is given by Dolev et al. [40], where

Processing Synchrony:

> There is a constant $\Phi \geq 1$ s.t. in any time interval T where some $p \in \mathcal{N}$ takes $(\Phi + 1)$ steps, every honest $q \in \mathcal{H}$ must take at least one step.

Communication Synchrony:

> There is a constant $\Delta \geq 1$ s.t. every message is delivered within Δ real-time steps.

The authors also consider additional criteria for discussion of synchrony characteristics, such as message ordering and atomicity of send / receive operation pairs (§2.2.3).

Synchrony embodies strong assumptions on component and network behavior. Processes can rely on timing information to ensure algorithm correctness or progress. This simplifies implementation or is a prerequisite for problem solvability in the first place (§2.4.4).

However, ensuring synchrony in practical systems is non-trivial, or as Chandra and Toueg [42] put it: "*synchrony assumptions are at best probabilistic*". For the development of practical distributed systems, solutions developed against a weaker model with fewer assumptions and simpler semantics can be an advantage, e.g., in terms of portability. Furthermore, if application correctness relies on synchrony, even temporary absence of synchrony can jeopardize safety. For these reasons, weaker models are studied.

2.2.2 Asynchrony

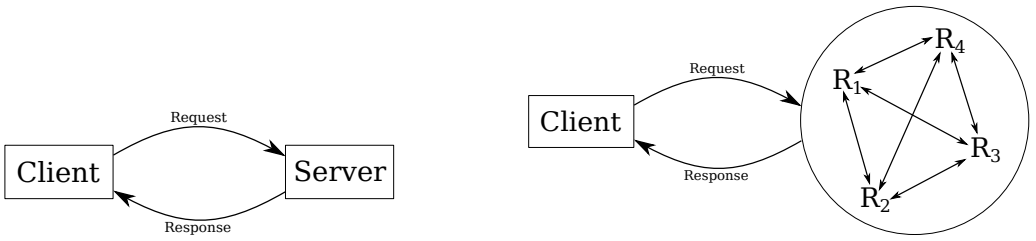
In *Asynchrony* there are no bounds on processing speed, message delay, clock drift, or any timing assumptions whatsoever [14, 18, 42]. Arguably, asynchrony is a more suitable model for real-world distributed system behavior. However, while distributed primitives capable of operating successfully under asynchrony are desirable, some problems are provably unsolvable under complete asynchrony [14, 18] (§2.4.4).

2.2.3 Partial Synchrony

As both synchrony and asynchrony have clear limitations in their applicability to real-world systems, research has devised hybrid synchrony definitions, or *Partial Synchrony*. One of the most prevalent partial synchrony models used for recent fault-tolerant systems are the definitions of Dwork et al. [18]. The authors assume *send* and *receive* as independent (not located in an atomic block) operations and define timing bounds located somewhere between complete synchrony and asynchrony. They provide two formalization approaches of partial synchrony as variations of communication and processor synchrony.

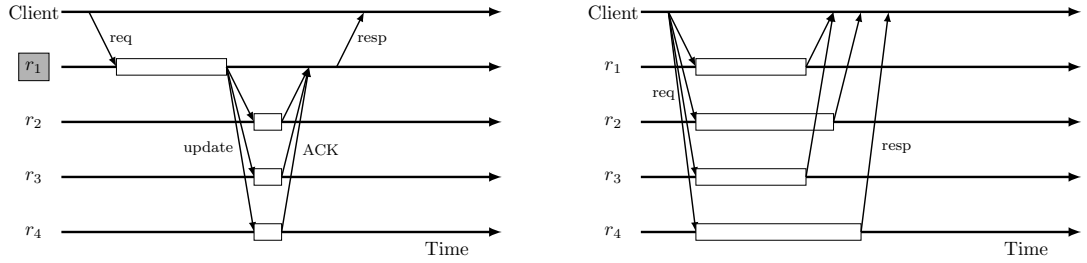
Let T be an interval of real time, R a run of the system describing all states, transitions, and exchanged messages for all $p \in \mathcal{N}$, and a communication bound $\Delta \in \mathbb{N}$. Then two possible definitions of *partial communication synchrony* are given by:

PS1.	$T := [1, \infty)$, $\forall R, \exists$ unknown Δ :	Δ holds during T
PS2.	$T' := [GST, \infty)$, $\forall R, \exists$ known Δ :	Δ holds during T'



(a) Without Replication (b) With Replication

Figure 2.1: Exemplary Client-Server Communication



(a) Passive Replication, Primary r_1 (b) Active Replication

Figure 2.2: Replication Types

where GST := Global Stabiliziation Time (GST). Replacing $\Delta \rightarrow \Phi$ in PS1/2, yields *partial processor synchrony* respectively. Hence, a system is partially synchronous if fixed upper bounds Δ, Φ exist at all times but are unknown a priori (PS1), or if fixed upper bounds Δ, Φ are known, but only hold after GST (PS2). The authors also make the point, that in practical systems $T' \sim \overline{T} := [GST, GST + L]$ might be sufficient for PS2, where L is some interval length necessary to make sufficient progress in an algorithm. In other words, there needs to exist a sufficiently long synchronous phase to make progress.

Dolev et al. [40] introduce granular definitions of partial synchrony based on five core system parameters, (1) processor synchrony, (2) communication synchrony, (3) message order synchrony, (4) availability of broadcast or point-to-point transmission, and (5) execution of send / receive as one atomic operation. Finally, Castro and Liskov [19] provide a definition of *weak synchrony*: Let m := message, t := time of $send(m)$, u := time of $deliver(m), t \leq u$, $delay(t) := u - t$, then $delay(t)$ does not grow faster than 2^t indefinitely.

We implicitly assume PS2 when referring to partial synchrony, if not stated otherwise.

2.3 Replication

Replication allows construction of fault-tolerant distributed systems. Consider a classical client-server scenario (Fig. 2.1a), where a client sends a request to a single server node and the server answers with a response. In order to address the single point-of-failure, the server node can be replaced by a set of *replicas* (Fig. 2.1b), redundantly storing necessary data and state. The replicas should fail independently to ensure the desired isolation [3].

2.3.1 Replication Types

Literature differentiates between two replication types, *passive* and *active* replication [54–56]. While hybrids are possible, we limit the discussion to these two approaches for brevity. In passive replication (Fig. 2.2a) a client only interacts with a single replica, the *primary*. Other replicas are called *backups*. Requested operations are only executed by this primary, which sends the execution result as an update to the backups. Once a sufficient number of

backups acknowledges the update, the primary responds to the client. In active replication (Fig. 2.2b), a client directly communicates with all replicas. Replicas assume equal roles in terms of verification, so the replicas execute the request independently and redundantly. Finally, the replicas respond to the client. A core problem of replication is ensuring consistency between replicas to prevent diverging state and conflicting responses. For passive replication, a view synchronous broadcast of update information to the backups is required, as well as a correct primary election facility (also referred to as *view change* [54]), e.g., realized by a group membership service [55].

2.3.2 State Machine Replication

As a variant of active replication, Schneider describes SMR as a method for implementation of fault-tolerant services [3]. Services are modeled as state machines, consisting of state variables and commands, offering deterministic execution that allows for state transition. By redundant and synchronized execution of a state machine on multiple replicas and combination of their output, resilience against a minority of faulty replicas can be achieved. If all honest replicas start in the same state and commands are reliably delivered to all replicas in the same order, all honest replicas are expected to yield the same result. More specifically, a causal (total) order should be established [3, 57] (§2.4.1). Depending on the scenario, different composing functions can be chosen. Voting [58] is a popular choice. Even though resilience against faulty clients is possible by e.g., client replication, literature often assumes honest clients or masking of client faults by another application layer.

Schneider decomposes the necessary replica coordination into two abstract requirements:

Replica Agreement:

> Every honest $q \in \mathcal{H}$ receives every client request.

Replica Order:

> Every honest $q \in \mathcal{H}$ processes incoming requests in the same relative order.

Depending on e.g., the target use case, fault model, and synchrony model, different distributed communication primitives (§2.4) can be employed to implement sufficient replica coordination. Crucially, possibility of construction, performance, and scalability of SMR for a target scenario depend on these chosen primitives and their complexities. Hence, in the pursuit of improving fault-tolerant systems, optimization of CFT and BFT-SMR has resulted in detailed study, optimization, and reformulation of the underlying primitives.

2.4 Primitives

Research provides constructions and proofs of communication primitives and attributes. We provide a brief overview of commonly referenced primitives in BFT-SMR research.

2.4.1 Ordering

Event ordering can be central to the correctness of a distributed algorithm. While several definitions of ordering properties exist [48, 53, 59, 60], we only introduce a subset, typically used in context of SMR. A simple ordering type based exclusively on single sender context is *FIFO Order* [48, 59]. Let $\mathcal{M}$ be the set of messages, messages $m, m' \in \mathcal{M}$, then

FIFO Order:

> If $q \in \mathcal{H}$ broadcasts m before m', no $r \in \mathcal{H}$ delivers m', unless it delivered m before.

Note that FIFO order does not consider context of some message $\overline{m}$ that was delivered by $sender(m)$, before m was broadcast. However, for some applications this context is necessary and stronger ordering guarantees (e.g., causal or total order) are required.

In a system with access to synchronized physical clocks, these clocks could be used to order events. However, network synchrony and clock synchronization are strong assumptions that are not necessarily guaranteed in real-life networks (§2.2.1). Lamport [53] provides approaches toward ordering without initially relying on physical clocks. First, Lamport asserts that a definition of a total order of events in a distributed system is not always directly possible. To nevertheless enable construction of a total order, (1) a partial order (*causal order*) on events is established, (2) *logical clocks* are constructed upon the partial order, (3) and an approach is described to break ties between concurrent events.

Causal Order:

Let each process $p \in \mathcal{N}$ consist of an ordered sequence of events $p := (e_1, \ldots, e_s)$, where $send(m)$ or $recv(m)$ of message m is considered to be an event. Then the "*happens before*" or "*causally affects*" relation ($\rightsquigarrow$) satisfies:

HB1.	Events $a, b \in p = (\ldots, a, \ldots, b, \ldots) \Rightarrow$	$a \rightsquigarrow b$
HB2.	$a := send(m), b := recv(m)$, for message $m \Rightarrow$	$a \rightsquigarrow b$
HB3.	$a \rightsquigarrow b \wedge b \rightsquigarrow c \Rightarrow$	$a \rightsquigarrow c$
HB4.	$a \not\rightsquigarrow b \wedge b \not\rightsquigarrow a \Rightarrow$	a, b are *concurrent*
HB5.	$\forall a \in p :$	$a \not\rightsquigarrow a$

An implementation of the partial ordering is given by logical clocks.

Logical Clock:

Let $C_i : e \rightarrow \mathbb{N}$ be the clock function of process p_i, returning a logical time for some event $e \in p_i = (e_1, \ldots, e_s)$.

Then the clock rules C1, C2

C1.	Events $a, b \in p_i = (\ldots, a, \ldots, b, \ldots) \Rightarrow$	$C_i(a) < C_i(b)$
C2.	$a = send(m) \in p_i, b = recv(m) \in p_j \Rightarrow$	$C_i(a) < C_j(b)$

culminate in implementation rules IR1, IR2:

IR1. Each p_i increments C_i between any two successive events

IR2. If $a = send(m) \in p_i$, then message m contains the timestamp $T_m = C_i(a)$.
Upon $recv(m) \in p_j$, p_j modifies its C_j from the old value C'_j s.t. $C_j \geq C'_j \wedge C_j > T_m$

Still, the causal order only guarantees a partial ordering of events. Hadzilacos and Toueg [59] provide a definition for a total order.

Total Order:

If $q, r \in \mathcal{H}$ deliver $m, m' \in \mathcal{M} : q$ delivers m before $m' \Leftrightarrow r$ delivers m before m'

To implement a total order ($a \approx\!\!\gg b$), Lamport [53] asserts that an arbitrary order between concurrent events can be defined, e.g., based on processor IDs. Let events $a \in p_i, b \in p_j$:

$$a \approx\!\!\gg b \Leftrightarrow (C_i(a) < C_j(b)) \vee (C_i(a) = C_j(b) \wedge p_i < p_j)$$

This arbitrary order can cause perception anomalies. A native total order can be established through use of sufficiently synchronized physical clocks [53]. An overview of other and more detailed ordering properties and their equivalence is given by Defago et al. [48].

2.4.2 Broadcast Primitives

A plethora of different broadcast primitives is described in literature, sometimes with slightly diverging definitions and guarantees [15, 42, 48, 59, 59, 61–63]. For brevity, we limit ourselves to primitives relevant in context of SMR, and generally only introduce the BFT variant of a primitive. Hadzilacos and Toueg [59] provide detailed definitions of the primitives, that we refer to in the following.

Reliable Broadcast:

Let $\mathcal{M}$ be the set of messages, message $m \in \mathcal{M}$. *Reliable Broadcast* offers primitives $broadcast(m)$ and $deliver(m)$. Assume that the sender of m is available as $sender(m)$.

$$\begin{aligned}
\text{Validity}: &\quad \forall q \in \mathcal{H} : q.broadcast(m) \Rightarrow && \text{eventually: } q.deliver(m) \\
\text{Agreement}: &\quad \forall q \in \mathcal{H} : q.deliver(m) \Rightarrow && \text{eventually, } \forall r \in \mathcal{H} : r.deliver(m) \\
\text{Integrity}: &\quad \forall m \in \mathcal{M}, s = sender(m), \forall q \in \mathcal{H} : \\
&\quad q.deliver(m) \text{ at most once }, && \text{and only if } s.broadcast(m)
\end{aligned}$$

No restrictions on the delivery order are imposed. In a BFT-SMR context, replicas delivering client commands in a different order can lead to state inconsistencies (§2.3.2). Hence, BFT-SMR systems typically employ a *Total Order Broadcast (TO-Broadcast)*. Note that literature partially uses the terms *Atomic Broadcast* [14, 42, 64] and TO-Broadcast interchangeably. As suggested by Defago et al. [48], we will use TO-Broadcast, as it unambiguously refers to the total ordering property.

TO-Broadcast:

Assume reliable broadcast as defined above. Now additionally require

$$\begin{aligned}
\text{Total Order}: &\quad \forall m, m' \in \mathcal{M}, \forall q, r \in \mathcal{H} : \\
&\quad q.deliver(m) \text{ before } q.deliver(m') \Leftrightarrow \quad r.deliver(m) \text{ before } r.deliver(m')
\end{aligned}$$

For BFT-SMR, the TO-Broadcast primitive should additionally ensure causal order [48, 59]. An extended overview of broadcast primitives with more fine-grained guarantees and requirements is provided by Hadzilacos and Toueg [59].

2.4.3 Leader Election

In the leader election problem, a group of processes aims to elect a single, functional leader from the group. Depending on the use case additional restrictions or guarantees may be present, e.g., process or communication faults. A leader election primitive can be a useful tool in distributed systems [65], e.g., for implementation of consensus (§2.4.4) in environments with failures [43, 62, 66], to implement view synchrony [54], optimize the efficiency of fault-tolerant systems [67] or distributed randomized algorithms, e.g., based on collective coin flipping [68]. A wide range of protocols has been proposed [65, 69–72]. A more detailed overview of literature is given by King et al. [69]. More recent work includes the analysis of game theoretic implications [73] as well as settings with additional privacy requirements, e.g., limiting knowledge of the leader identity to the elected leader itself [72].

2.4.4 Consensus and Agreement

Agreement between processes on a certain (set of) values is a central problem in distributed and cooperative systems. In the last decades, a significant body of work has studied a variety of agreement classes in different settings. For conciseness, we only introduce basic notions and variants that are most relevant to BFT-SMR, also omitting e.g., approximate agreement approaches [74].

Basic Definitions and Variants

A structured overview of central agreement definitions is provided by Fischer [15]. For fault-tolerant agreement, (multi-stage) majority voting is considered a central element. Fischer defines the following problems:

(Binary) Consensus:

Let processes $p_i \in \mathcal{N}$. Each p_i has its own input bit x_i. A protocol run ρ is *admissible* if it fulfills all necessary restrictions (e.g., in terms of safety). All honest $q \in \mathcal{H}$ are to agree on a common output bit y.

$$\begin{aligned} \text{Non-Triviality}: \quad & \forall y \in \{0,1\}, \exists\, x_i \wedge \text{admissible } \rho: && y \text{ is the output value} \\ \text{(Strong) Unanimity}: \quad & \forall i: x_i = x \in \{0,1\} \Rightarrow && y = x \end{aligned}$$

Interactive Consistency:

Consider the definitions from *Consensus* above, except all honest $q \in \mathcal{H}$ are to agree on a common output vector $\boldsymbol{y}$.

$$\text{(Strong) Unanimity}: \quad \forall j, q_j \in \mathcal{H}: \boldsymbol{y_j} = x_j$$

(Byzantine) Generals Problem:

A single distinguished process $l \in \mathcal{N}$ tries to send its input bit x to all other $p \in \mathcal{N}$. All $q \in \mathcal{H}$ are to agree on a common output bit y.

$$\text{(Strong) Unanimity}: \quad l \in \mathcal{H} \Rightarrow y = x$$

The Byzantine Generals Problem was initially studied and described in detail by Lamport et al. [45]. As outlined by Fischer [15], the generals problem can be interpreted as a special case of interactive consistency, since only the input value of a single process is of interest. Hence, a solution for interactive consistency also solves the generals problem, and n parallel invocations of a solution for the generals problem solve interactive consistency. Terminology varies in between publications, for example, other definitions of the consensus problem include arbitrary-length input values from each $p \in \mathcal{N}$ as well as additional termination properties [39, 42, 59, 62, 63, 75]. Correia et al. [63] refer to arbitrary-length consensus as *Multi-Value Consensus*. Sometimes, the term *Byzantine Agreement* is used to refer to (BFT) reliable broadcast [39], interactive consistency [39], or (multi-value) BFT consensus [76, 77]. We use the definition by Hadzilacos and Toueg [59] for byzantine faults:

(Multi-Value) Consensus:

Let the core primitives of consensus be *propose* and *decide*. Let processes $p_i \in \mathcal{N}$, each p_i has its own input value $v_i \in \mathcal{V}$, hence p_i *proposes* v_i. When some $p \in \mathcal{N}$ returns from the protocol with an output value ν, then p *decides* ν. If each correct $q \in \mathcal{H}$ proposes a value, then:

$$\begin{aligned} \text{Agreement}: \quad & q \in \mathcal{H} \textit{ decides } \nu \Rightarrow && \forall r \in \mathcal{H}: r \textit{ decides } \nu \\ \text{Integrity}: \quad & \text{if } \mathcal{N} \subseteq \mathcal{H}, q \in \mathcal{H} \textit{ decides } \nu \Rightarrow && \nu \text{ was } \textit{proposed} \text{ by some } r \in \mathcal{H} \text{ before} \\ \text{Termination}: \quad & \forall q \in \mathcal{H}: && q \text{ eventually } \textit{decides} \text{ exactly one value} \end{aligned}$$

In the dichotomy of *Safety* and *Liveness* properties by Alpern and Schneider [78], Agreement and Integrity can be interpreted as safety properties, while Termination poses as liveness property. Other consensus variants include the Asynchronous Common Subset (ACS) problem [77, 79], which extends the interactive consistency problem to asynchrony.

As outlined in §2.4.2, TO-Broadcast can be used to build BFT-SMR systems. Hadzilacos and Toueg [59] outline the relations between consensus and TO-Broadcast and demonstrate transformations between the primitives under different conditions. Bracha [76] advocates for the usage of reliable broadcast primitives and validation methods to construct consensus protocols by iterative limitation of adversarial behavior, and argues for the efficiency and simplicity of this approach.

Bounds and Limitations

Fischer, Lynch, and Paterson [14], also known as the FLP (impossibility) result, demonstrated the impossibility of solving consensus in completely asynchronous systems, that are subject to even a single crash fault. In pursuit of "circumventing" this result, research formulated models and solutions based on failure detectors [42], randomization [80], weaker problems [74, 81, 82], or stronger system assumptions such as partial synchrony (§2.2.3), typically forfeiting liveness in case of asynchrony but always guaranteeing safety [19].

Implementation of the introduced primitives can be conducted using different assumptions on fault model (§2.1), system timing (§2.2), and availability of cryptographic primitives. Lamport et al. [45] distinguish between *oral* messages, whose contents are fully under control of a sender or relaying node, and *signed* messages whose contents can be authenticated by using e.g., asymmetric cryptography. In any case, nodes are typically assumed to be able to reliably determine the immediate sender of any message [15, 45, 83].

Varying assumptions result in differences in the minimum number of honest nodes to tolerate a number of faulty nodes, as well as (message, signature, round) complexities of the algorithm. For synchrony, Lamport et al. [45] show that $n \geq 3f+1$ nodes are necessary to solve the byzantine generals problem, using oral messages. The authors also show an algorithm using signed messages that tolerates f faults for any $n \geq f+2$. Pease et al. [83] show that $n \geq 3f+1$ are necessary to solve interactive consistency using oral messages and that the problem is solvable for $n \geq f \geq 0$ using signed messages. An early overview of upper and lower complexity bounds is provided by Fischer [15], Freitas et al. [84] offer a recent survey of BFT-SMR algorithms, including complexities, and Hagit and Welch [85] provide a discussion of worst-case time complexity bounds for agreement algorithms.

In absence of synchrony, there are no reliable or known timing bounds. Bracha and Toueg [39] provide a method for designing consensus and broadcast protocols, using a threshold of process responses instead of timing. To prevent possible equivocation enabled by asynchrony, processes may not start or proceed unless presented with sufficient e.g., responses that guarantee (eventual) fitness/agreement of all correct processes in the future. Such a sufficiently large set of (correct) votes (or nodes) is also called a *quorum* [86]. The authors prove that for solving the byzantine generals problem $n \geq 3f+1$ nodes are necessary and sufficient in their model. Castro and Liskov [87] provide a more intuitive rephrasing of this method. In absence of timing bounds, the algorithm should proceed after establishing a quorum of $|\mathcal{H}| = (n-f) = 2f+1$ responses by honest processes. However, out of these $(n-f)$ received responses f might still originate from faulty processes, since responses by honest processes might simply be delayed. Hence, even if f out of $(n-f)$ received responses are faulty, the honest responses should outnumber the faulty ones:

$$\underbrace{f}_{\text{faulty responses}} < \underbrace{f+1}_{\text{honest responses}} = 2f+1-f = (n-f)-f \Rightarrow 3f < n$$

Symbol	Definition
$\mathcal{N}$	Set of replicas in the system
$\mathcal{F}$	$\mathcal{F} \subseteq \mathcal{N}$, Set of faulty replicas in the system
$\mathcal{H}$	$\mathcal{H} := \{\mathcal{N} \setminus \mathcal{F}\}$, Set of honest replicas in the system
n	$n := \lvert\mathcal{N}\rvert$, Number of replicas
f	$f := \lvert\mathcal{F}\rvert$, Number of faulty replicas
h	$h := (n - f) = \lvert\mathcal{H}\rvert$, Number of honest replicas

Table 2.1: Basic Notation used in this Thesis

2.5 Notation in this Thesis

For the rest of this thesis, we use notation as introduced in this section if not explicitly stated otherwise. An overview of the most relevant symbols is given in Tab. 2.1. We denote the set of natural numbers without 0 as $\mathbb{N} = \{1, 2, \ldots\}$. If 0 is to be included, we write $\mathbb{N}_0$ instead. We refer to inclusive intervals of integers $i, j \in \mathbb{Z}$ as $[i, j] = \{i, i + 1, \ldots, j - 1, j\}$. In the special case $i = 1, j > i$, we may write $[j] = \{1, \ldots, j\}$ for brevity. For intervals between real values $a, b \in \mathbb{R}$ we explicitly denote the value type before each usage in interval notation $[a, b]$. Since citations and references are also denoted using square brackets, an interval must be identified from context. Typically, we use interval notation to define variable value ranges $z \in [x, y]$. For more compact notation, we use "s.t." as an abbreviation of "such that" in some definitions.

Part I

Building Block Optimization as Method for BFT-SMR Improvement

3. Overview

"In distributed systems, suspicion, pessimism, and paranoia pay off."

Martin Kleppmann, Designing Data-Intensive Applications:
The Big Ideas behind Reliable, Scalable, and Maintainable Systems (2017) [88, p. 277]

Kleppmann [88, p. 277] asserts that work in the area of distributed systems benefits from special care and attention to detail. Of course, this does not only apply to the design of new algorithms and systems but also to the modification of existing ones.

As outlined in our research objectives (§1.2), we are interested in optimization strategies that *do not* affect existing safety and liveness proofs or introduce new tradeoffs or specialization on consensus algorithm level. In the optimal case, the strategies apply to a range of multiple, different BFT-SMR systems.

In Chapter 4 we analyze related work and categorize prevalent optimization techniques. We discuss the widespread introduction to tradeoffs and complexity and propose the optimization of BFT-SMR building blocks as an alternative improvement strategy.

In Chapter 5 we identify common assumptions on primitives or infrastructure that BFT-SMR system specifications frequently make, along with typically chosen implementations to realize the functionality. Afterward, we discuss where and why existing literature identifies bottlenecks in BFT-SMR systems. Based on these observations, we choose a set of promising BFT-SMR building blocks to investigate. Finally, we define an optimization methodology for further study and evaluation of the identified building blocks.

Chapter 6 complements the optimization methodology by choosing the BFT-SMR system *HotStuff* as representative framework to build on. We exemplify the impact of our optimization strategy using HotStuff. We discuss reasons for selection and introduce HotStuff in more detail. Finally, we list relevant limitations, that influence further study.

Relation to Existing Publications

These upcoming chapters contain content, published in the two papers [89, 90]:

1. Richard von Seck, Filip Rezabek, Benedikt Jaeger, Sebastian Gallenmüller, and Georg Carle, *BFT-Blocks: The Case for Analyzing Networking in Byzantine Fault Tolerant Consensus* in 2022 IEEE 21st International Symposium on Network Computing and Applications (NCA), volume 21, pages 35–44, 2022.
2. Richard von Seck, Filip Rezabek, Sebastian Gallenmüller, and Georg Carle, *On the Impact of Network Transport Protocols on Leader-Based Consensus Communication* in Proceedings of the 6th ACM International Symposium on Blockchain and Secure Critical Infrastructure, BSCI '24, page 1–11, New York, NY, USA, 2025. Association for Computing Machinery.

For improved readability, content from the two papers has been rearranged, reworked, and extended. Chapter 4 is a heavily reworked version from content of the introduction of [89] and the related work of [90]. Chapter 5 is an extended version of content from introduction, background, and building block analysis of [89]. Chapter 6 is an extended version of content from the introduction and HotStuff description of [89] and [90].

The following improvements and new contributions were added:

- §4: Extended and reworked related work corpus, added categorization and discussion
- §5.4: Added optimization methodology
- §6: Extended protocol description, detailed pipelining discussion, and limitations

Statement on Author's Contributions

The improvements listed above are the work of the author of this thesis. For the content on BFT-SMR optimization directions (§4) and building block analysis (§5) the author provided major contributions for the ideas and analysis.

4. BFT-SMR Optimization Directions

" The state of the art for BFT state machine replication is distressingly complex."

Ramakrishna Kotla et al., Zyzzyva: Speculative Byzantine Fault Tolerance (2007) [23]

4.1 Optimization Directions in Literature

We first provide a brief overview of prominent BFT-SMR optimization approaches, identifying relevant existing research directions. A plethora of work has been conducted with special focus on architecture and adjustments of the employed agreement algorithms.

Determinism

In general, we can distinguish between *deterministic* and *probabilistic* approaches to solve agreement. A recent comparative survey of both categories is presented by Freitas et al. [84]. The authors subsume that (1) deterministic algorithms can reach agreement in a fixed number of communication rounds, while they rely on (partial) synchrony to achieve liveness, and (2) probabilistic algorithms always operate in a varying number of communication rounds but also allow for agreement under asynchrony. Typically the randomization is achieved by some instance of collective coin flipping [68], where different processes generate the same random bits. A seminal approach proposed by Cachin et al. [20] uses threshold signatures as a distributed source of random bits.

Optimism

A series of works aims for optimization of the "good case", in which no or only few faults occur and where a costlier fallback path is triggered if (more) faults occur [8, 21, 23, 56, 91–97]. Other approaches explicitly focus on performance improvements in face of more frequently occurring faults, aiming to increase algorithm robustness [24, 26, 98–102].

Leaders and Communication Hierarchy

Proposed algorithms vary in terms of communication structure. While early algorithms for e.g., interactive consistency [83] involve all-to-all communication between replicas, a branch of work [7,8,19,31] introduces a replaceable *leader* role, reducing overall communication complexity. The exact behavior of the leader role can vary. In some approaches such

as PBFT [19] the leader essentially acts as primary but replicas still communicate in an all-to-all fashion and directly answer the client. In systems such as HotStuff [7] (§6), replicas receive requests directly from the client but agreement communication is exclusively conducted with the incumbent leader, reducing communication complexity in the good case. The extended responsibility of a leader is effectively an optimistic assumption on leader reliability and performance. The leader node becomes a potential performance bottleneck in terms of bandwidth or processing power (§18.6) and causes increased fallback costs if faulty replicas are elected. A recent line of work addresses these challenges by operating multiple, parallel leaders [9,32,37] or re-embracing leaderless architectures [35,83,103–105].

Communication overhead is further addressed by communication hierarchies or federation [28, 101, 106–108]. Another recent branch of work proposes a separation of transaction distribution from agreement via a *shared mempool* abstraction [10, 11, 13], to address bandwidth and processing limitations arising from client-replica communication.

Parallelism

Some protocols include optimistic strategies that allow progress on different abstraction levels in parallel. This includes *batched* processing of client requests [6,7,23,24,26,27,31,33], where replicas agree on multiple incoming transactions at once, reducing the agreement overhead per transaction. Parallelism on a higher abstraction level is done by *pipelining* of operations in the agreement protocol [7, 16, 30, 31, 33, 37]. One possible approach is view-pipelining [7], where a quorum of replicas votes on not only one but multiple (batches of) client requests in different phases of the consensus algorithm simultaneously (§6).

Trusted Components

Finally, a variety of research investigates applicability of TEEs to BFT-SMR. This includes compartmentalization of agreement protocols for additional robustness [52], simplification of the agreement protocol (complexity) by relying on TEE-secured counters and signatures, as well as studies of mixed fault models [51]. A recent overview of works in this direction is given by Gupta et al. [109].

4.2 Summary

There exists an abundance of research on BFT-SMR optimization with a significant amount of work dedicated to architectures and conceptual modifications. Kotla et al. [23] attest to "*distressing*" complexity as early as 2007. Guerraoui et al. [16] assess BFT-SMR systems to be "*notoriously difficult to develop, test and prove*" and suggest the impossibility of designing a "*one-size-fits-all*" BFT-SMR protocol. Finally, Singh et al. [110] attest to inherent tradeoffs in conceptual protocol design choices. Publication of new conceptual improvements, such as the shared mempool abstraction [10, 11, 13], is ongoing and further enhances BFT-SMR for certain scenarios. At the same time, increasingly better performance and explicitly ambitious goals of e.g., achieving "line rate" [111] for BFT-SMR protocols, require reliable and high-performing operation of all underlying components. This requirement materializes as new bottlenecks, e.g., in Chop Chop [111], where the authors explicitly consider modification of the underlying transport protocol due to state and tracking overhead for large numbers of short-lived connections at scale.

This development demonstrates a focus shift toward the study of BFT-SMR components as the next potential frontier for optimization. Considering the use cases and capital invested in or related to permissioned consensus systems, such as (1) the operation of (general purpose) blockchain or smart contract platforms [7,112–114], (2) integration with or extension of classical financial infrastructure as part of a Distributed Ledger Technology (DLT) deployment [115], or (3) usage of DLT for logistic tracing [116], the relevance of component study from economic and academic perspectives becomes increasingly evident.

4.3 Our Approach

From our perspective, optimization of components potentially yields additional benefits. Reconsidering our research questions **Q1.1** (fundamental improvements) and **Q1.2** (preservation of safety and liveness), we aim to avoid adding to the existing architectural complexity of BFT-SMR systems. Since modification of a provably secure algorithm may also invalidate the existing guarantees, we pursue a different approach. We are interested in *practical* solutions and improvements, relevant to real-world systems.

Hence, we propose the optimization of BFT-SMR *building blocks* as a way to work toward fundamental improvements on a range of practical systems. In this context, a *building block* denotes an independent, modular component used to implement a practical BFT-SMR system. In fact, literature considers certain classic assumptions explicitly as "building blocks" [117]. Safety and liveness guarantees can be preserved, by ensuring that the exposed component behavior is consistent with existing assumptions.

In its essence, the idea to leverage new technologies or components to build faster or more efficient BFT-SMR systems is not new, as our dedicated discussions of related work for the chosen building blocks demonstrate (§12, §19). However, preservation of BFT-SMR algorithm guarantees and interface during the process is not necessarily a priority in these works. Usage of another component technology is often published as part of a modified BFT-SMR algorithm, under a new name, adding yet another candidate to the space of available approaches. This practice does not only complicate identification of the right protocol for a target scenario, it also aggravates sensible comparison of systems with and without the new component, since the experiment environment and parameters can change significantly. We are convinced that a more systematic comparison of available building blocks across BFT-SMR algorithms is beneficial. While some candidate components do not naturally offer the same features (e.g., Message Authentication Codes (MACs) and digital signatures, §15) and may require architectural changes, others do not.

Meanwhile, some components are yet to be assessed in a level of detail that offers stronger insights into operation of practical systems. For example, we agree with recent research on longest chain consensus [118, 119] that a mismatch between the established, idealized network model and the challenges of practical BFT-SMR systems exists.

To subsume, we view our approach of *building block optimization* not as a fundamentally new paradigm, but as a helpful method to organize research on BFT-SMR in a way that can yield additional benefits. We are not aware of existing work that considers building block optimization for BFT-SMR as explicitly and for the reasons we do. However, we are convinced that the value of the method should be assessed by the results it yields for a range of building blocks. Depending on the respective building block, the amount, depth, and content similarity to related work varies. Hence, for a detailed discussion of tangible benefits and distinguishing features of our approach in a certain component space, we refer to the respective related work sections of our chosen building blocks (§12, §19).

In the following, we outline typical assumptions of BFT-SMR, detail commonly used implementations, and identify suitable building blocks (**Q1.3**) for further investigation.

5. BFT-SMR Building Block Analysis

> *" Modern [BFT-SMR] protocols involve about 20.000 lines of challenging C++ code encompassing synchronization, networking and cryptography. They are notoriously difficult to develop, test and prove."*
>
> Rachid Guerraoui et al., The Next 700 BFT Protocols (2010) [16]

Building block optimization may address the complexity and difficulty, that Guerraoui et al. [16] identify to be involved with development and modification of BFT-SMR protocols. We now aim to identify such building blocks. We discuss common assumptions in BFT-SMR systems, screen bottleneck analyses in existing literature, and finally define suitable building blocks for further optimization.

5.1 Common Assumptions

Typically, protocol designers resort to an established list of assumptions on the environment and primitives. An overview of the most prevalent assumptions is given in Tab. 5.1. These assumptions offer a more relaxed design space for protocol creation and simplify or primarily enable correctness proofs in the first place. In detail, the listed assumptions help to (I) mask network-related, partial, or complete transmission failures, (II) abstract the underlying network topology and potential influence by adversarial nodes by emulating a point-to-point connection between sender and receiver, and (III) reduce the required number of replicas to solve agreement (e.g., in the synchronous case) or increase the resilience of communication against malicious influence in non-point-to-point topologies.

Table 5.1: Prevalent Assumptions in BFT-SMR Systems

	Assumption	**Example Systems**	**Implementation**
I.	Reliable Communication	[6–8, 24, 27, 29, 31–38, 100]	TCP
II.	Point-to-point Interconnections	[6–8, 16, 19, 22, 24, 27, 29, 31–33, 35–38, 99, 100]	TCP / IP
III.	Authenticated Messages	[6–8, 16, 19, 22–27, 29, 31–38, 98–100]	Signatures / MACs / TLS

To fulfill the desired guarantees, a real-world system needs to choose a suitable architecture to implement the respective assumptions. We list the most common implementation choices for an assumption in Tab. 5.1. To implement Assumptions I and II, typically TCP/IP connections between replicas are used [6–8, 24, 27, 37, 99, 100]. Assumption II is usually not satisfied through actual, direct hardware interconnections (consider deployment in large-scale Wide Area Network (WAN) setups [100]), but only emulated through logical, overlay connections between any pair of nodes. Effectively, the traffic is still routed across a TCP/IP network. In addition to BFT-SMR message authentication for Assumption III (e.g., through cryptographic signatures), some systems employ secure channel implementations such as Transport Layer Security (TLS) as part of their network stack [6–8,24,100]. A detailed analysis and discussion of message authentication strategies is given in Part III.

5.2 Bottleneck Analysis in Literature

In the pursuit of improving BFT-SMR, existing literature analyzes and identifies a range of (potential) bottlenecks for different environments and configurations. Hence, we list the most common arguments and verdicts. In single-leader protocols, the leader is consistently identified as a potential system bottleneck [23, 31, 32, 37, 100, 120–126]. When discussing causes for this verdict, literature provides varying reasoning for different systems and environments. In high-performance-low-latency environments [123, 126, 127] as well as massively scaled scenarios [6, 100] the processing overhead of network communication is identified as bottleneck. Wang et al. [127], as well as Dang et al. [126] identify OS kernel processing to be the main source of overhead. Stathakopoulou et al. [100] assume leader bandwidth to be the first, and client authentication to be the second bottleneck.

Older protocols, which use asymmetric cryptography for signing replica votes, identify cryptographic processing cost as main overhead [128, 129]. But also newer works, which employ MAC-based authentication [19,23] and modern BFT-SMR optimization techniques such as batching, multiple leaders, or threshold signatures [7, 32, 99, 104], argue for cryptographic processing to be the main bottleneck. Other publications just identify a general CPU bottleneck, without further distinguishing a cause [87, 120, 130].

Summarizing the above insights, we see that except for the leader in leader-based protocols, a general bottleneck is not consistently identified. Instead, the primary bottleneck depends on the respective system architecture, environment, and configuration. Works focusing on large WANs or high-performance-low-latency environments tend to identify increased network overhead as central hindrance. The choice of protocol parameters can have a significant influence on the bottleneck location. As outlined by Stathakopoulou et al. [100], a smaller payload size per request shifts the bottleneck toward a CPU and verification load, while larger payload sizes may result in bandwidth limitations.

5.3 Building Block Identification

Based on identified common assumptions (§5.1) and bottleneck analysis in literature (§5.2) we now define BFT-SMR *building blocks* whose optimization we deem worthwhile.

Underlying Network Stack

Since reliable and point-to-point connections are one of the most widespread assumptions in BFT-SMR protocols, we conjecture that the design and implementation of the employed networking stack has significant impact on performance and robustness of the final system. While there exists dedicated work on datacenter-centric deployments that leverages e.g., special hardware communication primitives (§12), we deem a focus on general default

network stacks (such as TCP/IP) to be more promising. Decentralization [8, 100] and scalability [8, 11, 100, 131, 132] are common design goals of modern BFT-SMR systems. Hence, the chosen network stack implementation should be available to a large number of potentially heterogeneous nodes, as the commodity TCP/IP stack is. We identify the *underlying network stack* as the first building block. While all OSI layers below BFT-SMR semantics are relevant in practice, the core logic for typical BFT-SMR assumptions (Tab. 5.1) predominantly resides in the transport layer (reliable communication, point-to-point interconnections) and e.g., secure channels (authenticated messages). The wide range of features and modifiable parameters (retransmission logic, congestion control, ...) provides ample opportunity for study. Hence we place our focus mainly on the transport layer and networking-relevant secure channel semantics in this work (§9).

Cryptographic Primitives

Usage of cryptographic primitives in BFT-SMR can serve multiple purposes, such as message authentication for protocol message complexity reduction, additional provability of node actions for e.g., misbehavior detection and maintenance, as well as implementation of distributed randomness for asynchronous protocols (§4). Hence, their application is widespread through modern systems. We identify the employed *cryptographic primitives* as the second building block. Due to the plethora of possible use cases and tools, we focus on an evaluation of threshold signature schemes in context of this thesis. Threshold cryptography is proposed for many recent and state-of-the-art systems, including usage for both e.g., vote signatures and distributed randomness in BFT-SMR (§15.1).

Operating System Overhead

Some works identify operating system overhead as central hindrance in high-performance environments. Authors leverage data plane tools and hardware [124, 126, 133–135], communication offloading through Field-Programmable Gate Arrays (FPGAs) [122, 136], and Remote Direct Memory Access (RDMA) [127, 137] for direct performance improvements or simplification of the target agreement type by outsourcing parts of the problem space (e.g., message ordering) to network hardware [133]. We identify tools for reduction of general-purpose operation system overhead as a third building block. Our research effort in this area is still ongoing. Hence, we consider this third building block as future work.

5.4 Optimization Methodology

In order to evaluate the impact and utility of modifications or augmentations of BFT-SMR building blocks, we define a brief optimization methodology to follow for each area.

First, we review (1) existing work that is related to the target building block in context of BFT-SMR and (2) recent work on optimization of the building block independent of the BFT-SMR context. Since we are interested in the evaluation of a *practical* impact, we require a setup of real-world artifacts that can be modified and measured. A disadvantage of concrete implementations is a loss of generality since measurement results depend on the concrete algorithms, environment, and hardware used. Our approach to limit this effect is twofold. As base for our experiments we choose a state-of-the-art BFT-SMR protocol that represents a class of systems and whose architecture is used as base for a variety of other protocols. Secondly, we set up and run all experiments in an environment optimized for reproducibility. We leverage the *pos* experiment orchestration framework [138], which allows automated experiment deployment and execution on ramdisk nodes, even for bare-metal machines. Hosts are reset to a well-known state between every experiment.

Based on our choice of representative BFT-SMR protocol, we then survey the optimization space for the target building block. We identify architectures, characteristics, and

configuration dimensions that bear optimization potential for the BFT-SMR use case. Finally, we modify and augment the chosen building blocks in our representative BFT-SMR system according to the identified dimensions. The altered artifacts are then extensively measured and compared to their baseline performance without any modifications.

6. HotStuff

As outlined in our methodology (§5.4) we require a representative BFT-SMR system from which to construct baseline and augmented artifacts for evaluation of the building block optimization impact. For this, we choose the seminal HotStuff protocol by Yin et al. [5,7]. Our description follows these primary sources if, not stated otherwise. HotStuff is subject to a significant amount of ongoing research [10–13,132,139–141] and, as suggested by Spiegelmann et al. [13], it is claimed to be used as foundation for practical systems, such as Cypherium [142], Diem [143], and Flow [144]. HotStuff employs many recent optimization approaches such as batching or pipelining, that are present in other state-of-the-art systems. It also shares the widespread BFT-SMR networking assumptions (§5.1), which typically serve as an abstract and not further specified interface. The experimentally quantified impacts presented in this thesis will vary between employed consensus algorithms and deployments. However, the lockstep broadcast-vote, 1-to-n communication structure of leader-based consensus is a central component of many systems [145], either offering CFT [43,44] or BFT [7,12,146]. Building block optimization can potentially improve all depending systems. Hence, we conjecture that our core insights and their optimization potential also apply to other algorithms and deployments.

6.1 Overview

HotStuff is a leader-based, permissioned BFT-SMR protocol, designed for operation under partial synchrony (§2.2), and envisioned for usage in large-scale blockchain scenarios with up to hundreds or even thousands of nodes, possibly in a WAN setting. In the blockchain context, *permissioned* protocols are close to classical replication architectures and assume the number and identity of all participating nodes to be known beforehand [5].

HotStuff is built around deterministic operation, where (correct) responses from a fraction of replicas are necessary and sufficient for progress. Under partial synchrony, HotStuff always guarantees safety but liveness only after GST. A central benefit is its *responsiveness* property, where an honest leader can drive progress at the speed of actual network latencies, independent of known upper bounds on e.g., message transmission delays. HotStuff guarantees *optimistic responsiveness*, where responsiveness is only given after GST.

Communication in HotStuff is built around a three-phase messaging core (§6.2). While older protocols such as PBFT [19] operate with a two-phase core, this paradigm incurs larger overhead in case of leader failure. The three-phase core increases the general messaging round complexity but simplifies and reduces the cost of leader changes. This allows

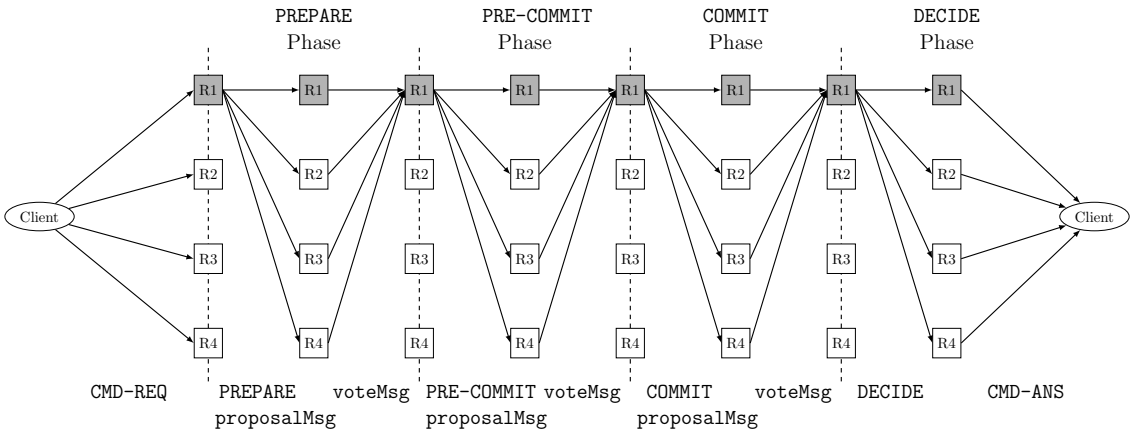


Figure 6.1: Basic HotStuff Communication Pattern

for frequent leader rotation without additional direct cost. Design of safety and liveness guarantees are logically separated. Safety predicates are specified as part of the voting rules, while liveness logic is encapsulated in a dedicated "pacemaker" module, that defines leader rotation and proposal semantics and timing.

The authors argue for the usage of *authenticator complexity*, i.e., the number of created and exchanged (message) signatures during some part of a protocol run, as primary metric for comparison with other protocols. For a reduction in this metric, the usage of threshold signatures (§15.1) is proposed, which allows combination of a sufficiently large number $t \leq n$ of valid signatures into a single one. Assuming this optimization, a correct leader incurs an authenticator complexity in $\mathcal{O}(n)$ after GST in the best case, while in the worst case of cascading leader failures the complexity is in $\mathcal{O}(n^2)$.

6.2 Protocol Details

The authors describe two HotStuff variants, a baseline variant (*Basic* HotStuff, Algorithm 2 [5]) and an optimized version that uses pipelining (*Chained* HotStuff, Algorithm 3 [5]).

Basic HotStuff

We can divide the general communication structure into two parts, namely (1) messaging between client and replicas and (2) messaging in between replicas. An overview of typical HotStuff messaging during one replication cycle is given in Fig. 6.1.

Client-replica communication effectively follows the active replication paradigm (Fig. 2.2b, §2.3.1). A client sends its request messages (`CMD-REQ`) to all available replicas. The replicas then agree on validity and order of the incoming client requests and send their answer messages (`CMD-ANS`) back to the respective client. Once the client receives $(f + 1)$ valid answers it considers the requested operation to be successful. The authors do not discuss client operation in further detail, but support of multiple clients is possible in principle, client-replica communication is also assumed to be reliable, and clients are implicitly assumed to be honest. Deviations from these assumptions and necessary logic (e.g., client request deduplication, validity checking, ...) would need to be handled accordingly.

A more detailed overview of replica-replica communication is given in Fig. 6.1. A full decision process on a single leader proposal is referred to as a *view*. A view is uniquely identified by a sequence number and begins with the responsible, incumbent leader (here: `R1`) collecting $b \in \mathbb{N}$ `CMD-REQ` messages, that are combined into a single proposal of batch size b. Each view consists of three core *phases* (`PREPARE`, `PRE-COMMIT`, `COMMIT`), sharing the

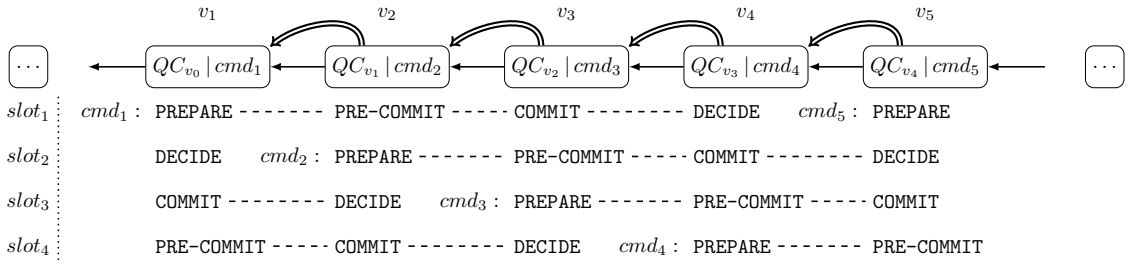


Figure 6.2: Phase Pipelining in Chained HotStuff, adapted from Fig.1 of HotStuff Paper

common communication structure of a broadcast-vote pattern, and a final announcement phase (`DECIDE`). Communication is generally limited to exchange between the current leader and all other replicas. The leader broadcasts a `proposalMsg` for the current phase and progress state to all replicas, which check the proposal validity and answer with `voteMsg` messages in case of endorsement. If the leader collects a quorum of $h = (n - f)$ valid votes on the current proposal, it creates a Quorum Certificate (QC), attesting both current protocol state as well as sufficient support by a majority of replicas. Hence, a QC contains cryptographic proof by all participating voters. This QC is then attached as justification to the proposal for the next phase.

In case the incumbent leader does not progress for "long enough", the other replicas will start an impeachment routine. The effective time threshold for triggering this routine as well as the next leader choice, depend on the concrete design and implementation of the pacemaker module. If the impeachment routine is triggered, affected replicas send a `NEW-VIEW` message to the new leader candidate, that contains the last known state and increments the current view number. Once the new leader collects h `NEW-VIEW` messages, it creates a new proposal based on this state and begins a new view as described above.

Chained HotStuff

Since the communication structure of the three core phases is effectively equivalent, the authors propose to optimize execution through view pipelining. The core idea is to decide on the progress of multiple independent proposals at once, such that a single QC serves as justification for agreement progress on multiple different command (batches) in parallel. A visualization of this concept is given in Fig. 6.2, based on Fig. 1 in the original paper [5]. We visualize four pipeline slots, here designated $slot_1, \ldots, slot_4$, each working on a dedicated command (batch). In $slot_1$, cmd_1 is proposed for completion of the `PREPARE` phase by the leader of v_1, and then finalized over views $v_2, \ldots, v_4$. However, in fault-free execution each view progresses all pipeline slots in parallel. For example, a successful decision on v_4 proves completion of the `DECIDE` phase for cmd_1, the `COMMIT` phase for cmd_2, and so forth. This decision is then certified in QC_{v_4}, justifying the proposal in v_5. Finally, due to the functional overlap between phases, chained HotStuff does not distinguish between proposal or vote types, but only implements a `GENERIC` type.

6.3 Implementation and Limitations

The authors publish a freely available, open-source, Proof-of-Concept (PoC) C++ implementation of HotStuff on Github, called `libhotstuff` [147]. This library implements the chained HotStuff variant, provides demo applications for client and replica roles, as well as convenience scripts for execution and simple visualization of benchmarking results.

In order to simplify implementation of the consensus logic, the authors moved large parts of network overlay, connection, and scheduling logic to a dedicated open-source software

library, called `salticidae` [148]. The code offers a higher-level abstraction for construction and operation of Peer-to-Peer (P2P) networking. We discuss relevant implementation details, optimization approaches, and evaluation of different network transport and secure channel architectures in Part II of this thesis.

While the authors assume and argue for usage of threshold signatures in the paper, the implementation uses simple, non-threshold secp256k1 Elliptic-Curve Digital Signature Algorithm (ECDSA) signatures on replica votes instead. As these signatures are not aggregatable, a QC always contains a list of signatures and its size grows with the number of certified votes. We discuss implications, optimization approaches and evaluation of multiple (threshold) signature schemes in detail in Part III of this thesis.

Part II

Optimization via the Underlying Network

7. Overview

The upcoming chapters provide our optimization potential analysis of the network transport building block, in the wider sense. Chapter 8 establishes necessary background content, such as a detailed introduction of relevant network transport protocols and secure channels. This includes a discussion of the rationale and details of Robust UDP (RUDP).

In Chapter 9 we select a subset of protocols for further study and provide a theoretical analysis of all candidates, in and out of operation in a BFT-SMR context.

Chapter 10 holds details about the rationale behind and the structure of our experiment design. We discuss relevant parameter variation and implementation details.

In Chapter 11 we experimentally verify performance and behavior impact through modification of the transport protocol building block in a bare-metal deployment.

Chapter 12 presents relevant related work.

We conclude Part II in Chapter 13 with a result revision and extended interpretation, discussing the outcomes in context of our research objectives.

Relation to Existing Publications

These upcoming chapters contain content from the following papers [89, 90, 149]:

- Richard von Seck, Filip Rezabek, Benedikt Jaeger, Sebastian Gallenmüller, and Georg Carle, *BFT-Blocks: The Case for Analyzing Networking in Byzantine Fault Tolerant Consensus* in 2022 IEEE 21st International Symposium on Network Computing and Applications (NCA), volume 21, pages 35–44, 2022.
- Richard von Seck, Filip Rezabek, Sebastian Gallenmüller, and Georg Carle, *On the Impact of Network Transport Protocols on Leader-Based Consensus Communication* in Proceedings of the 6th ACM International Symposium on Blockchain and Secure Critical Infrastructure, BSCI '24, page 1–11, New York, NY, USA, 2025. Association for Computing Machinery.
- Richard von Seck, Filip Rezabek, and Georg Carle, *Thresh-Hold: Assessment of Threshold Cryptography in Leader-Based Consensus* in 2024 IEEE 49th Conference on Local Computer Networks (LCN), pages 1–8. IEEE, 2024.

For improved readability, content from the papers has been rearranged, reworked, and extended. Chapter 8 is an extended version of background and analysis contents from [90], including figures used exclusively in the conference paper presentation of [89]. Chapter 9 is an extended and restructured version of analysis from both [89, 90]. Chapter 10 is an extended and restructured version of design content from [90, 149]. Chapter 11 is an extended and revised version of the evaluation from [90]. Chapter 12 is an updated and revised version of related work from both [89, 90].

The following improvements and new contributions were added:

- §8: Extended description and number of transport protocols
- §9: Added general header and payload size efficiency analysis, extended protocol operation analysis and case study for RUDP
- §10: Extended discussion of design and target scenario
- §11: Added data and evaluation of more experiments, extended already published data by additional visualization and evaluation
- §12: Updated and revised related work
- §13: Extended interpretation and discussion in context of research questions

Statement on Author's Contributions

The improvements listed above are the work of the author of this thesis. For the introduced content on protocol choice, analysis and experiment design (§9, §10) the author provided major contributions to the ideas and analysis. Initial HotStuff PoC implementations were developed by Lucas Mair (UDP) and Christoph Probst (RUDP) as part of their respective Bachelor's theses. The author contributed to conception of the thesis work, as well as design and implementation decisions during these works as a thesis advisor. The HotStuff implementation was revised, extended, and migrated by the author of this thesis. Lucas Mair first discovered that the PoC HotStuff implementation does not implement tracking of client request arrival on all replicas, potentially allowing for desynchronization in case of unreliable transport. The presented measurements, evaluation, and interpretation results are the work of the author of this thesis.

8. Background: Transport Building Block

For evaluation of the network transport building block, we study multiple protocols, distributed over several layers of a typical networking stack. In the following, we briefly introduce a selection of protocols and necessary background information.

8.1 Network Transport

To fulfill the prevalent BFT-SMR assumptions (§5.1) the network stack implementation needs to abstract the underlying network topology and mask message losses or transmission errors through retransmission. Prevention of service quality degradation through e.g., congestion and flow control may yield additional performance and robustness benefits.

We assume a network stack architecture akin to the ISO OSI model [150], as well as usage of the Internet Protocol (IP) [151] as Network Layer (Layer 3) protocol. Since retransmission and service control logic is typically implemented in the Transport Layer (Layer 4), we focus on the study of different transport protocols and their configuration space. In the following, we introduce a selection of the most prevalent and relevant transport protocols.

8.1.1 UDP

Introduced in 1980, the User Datagram Protocol (UDP) [152] provides message-based, connectionless, unreliable transport of datagrams. Besides a basic integrity check and port addressing, UDP does not provide any further complex functionality. While the lack of reliable transmission may need to be handled by an upper-level protocol, the simple and small protocol results in minimal overhead in terms of protocol header size. The absence of connection state allocation reduces latency overhead. Delay or loss of a datagram does not influence latency or handling of other datagrams on transport layer. This set of features renders UDP a suitable protocol for low latency, real-time communication.

8.1.2 TCP

With a fundamentally different design, the Transmission Control Protocol (TCP) [153] is one of the most established and widely used transport protocols in the modern Internet. TCP provides a reliable, ordered, connection-based transport, offering a *byte stream* abstraction. TCP offers a significant amount of configuration options, for its broad range of features. The protocol implements identification and retransmission of lost or corrupted segments, mechanisms to avoid overloading communication partners (Flow Control (FC)) and network infrastructure (Congestion Control (CC)), and more. While directly providing an abstraction that fulfills the BFT-SMR assumptions, operation of TCP incurs a potential overhead in terms of network stack processing and latency increase due to CC.

8.1.3 SCTP

As a connection-oriented, message-based protocol with reliable transmission of both unordered and ordered data, the Stream Control Transmission Protocol (SCTP) [154] can be interpreted as an extended, hybrid approach between UDP and TCP. SCTP provides CC, configurable reliability, and ordering with message granularity. It also features *multi-streaming* and *multi-homing*, allowing for transmission of independent data streams between logical endpoints and aggregation of multiple physical endpoints into one logical endpoint, respectively. SCTP relies centrally on Selective Acknowledgements (SACKs), improving its ability to retransmit the right segments, in case of packet loss.

8.1.4 QUIC

QUIC [155, 156] is a monolithic protocol to improve performance of HTTPS traffic and simplify both development and deployment of protocol changes. Central motivation and goals for development are twofold. Firstly, in a TCP and HTTP/2 stack, HTTP/2 recommends usage of a single TCP connection on which object transmission is multiplexed. Since TCP provides a byte stream abstraction, stalling transmission of a single object blocks progress for all other objects. This is referred to as Head-of-Line (HOL) blocking and addressed by changing underlying transport and semantics. Secondly, the authors aim for a reduction of cryptographic handshake overhead and latency in the network stack.

QUIC is implemented in userspace on top of UDP and intended to replace the functionality of a comparable TCP-based network stack, up to and including HTTP/2. Hence, QUIC functionality effectively spans layers 4-7 of the OSI model. QUIC provides a connection-oriented, reliable transport, including authentication and encryption of data, e.g., through integration of TLS version 1.3 [157]. Due to its feature set, QUIC has increased complexity in comparison to other transport protocols. The original authors attest to significantly increased CPU load on server deployments in comparison to a TCP baseline stack [156]. Recent research suggests, that UDP / QUIC / HTTP/3 stacks perform worse than TCP / TLS / HTTP/2 over fast internet connections [158].

8.1.5 RUDP

Considering the protocols introduced above, we identify either lightweight (UDP) or feature-rich protocols (TCP, SCTP, QUIC) with considerable complexity. Limited to these options, a study of a lightweight protocol that still offers reliable transport is not trivial. On the other hand, a direct comparison between message-based protocols such as between e.g., UDP and QUIC or SCTP is not fair.

In the past, there have been initiatives to address this gap for different scenarios. Bova and Krivoruchka [159] proposed a "*Reliable UDP protocol*" in an IETF internet draft, that has expired by now and was never standardized. The document specifies a UDP-based protocol for reliable, in-order data delivery. Initially aimed at implementing telephony signaling over IP, central goals of the protocol were message-based transmission, low overhead, and high performance. Effectively the document already proposed more complex logic, featuring flow control, error detection, and secure transmission.

In context of BFT-SMR, Camaioni et al. [111] propose a system architecture, involving a client request broker. This broker would be subject to significant amounts of short-lived client TCP connections. To alleviate this overhead, the authors employ a self-developed, simplistic UDP-based protocol using Acknowledgment (ACK) retransmission logic.

We acknowledge the described initiatives and benefits of a simplistic, unordered but reliable transport. For our evaluation, we propose the RUDP transport protocol abstraction, a minimal, UDP-based but reliable transport variant. Data transmission is conducted

via UDP, but the minimal retransmission logic is implemented by the calling application. Hence, akin to QUIC, RUDP effectively stretches over multiple OSI layers. Existing application layer messages serve as implicit ACKs, and upon receipt of erroneous messages, an explicit Negative Acknowledgment (NACK) is sent. The simple retransmission logic mimics TCP retransmission timer calculation, with less restrictive upper and lower value bounds. If a retransmission is triggered, RUDP always tries to retransmit the complete payload, even if only a single fragment was lost. No congestion or flow control logic is provided. We discuss RUDP in more detail in §10.3.

8.2 Secure Channels

Security, e.g., as reflected in the well-known security goals of confidentiality, integrity, and availability [160], is an essential attribute of many systems to prevent malicious influence or undesired exposure of information. In context of distributed systems, a *secure channel* provides protected communication between peers, offering (at least) confidentiality and integrity of exchanged data. Depending on the definition, secure key-exchange protocols [161], endpoint authenticity [162], and data freshness [163] are considered necessary.

In context of BFT-SMR, secure channels can be used to implement or complement message authentication [6, 7, 24, 100]. Often located in (QUIC) or on top (TLS) of the transport layer, secure channel architecture and implementation potentially influence performance and robustness. For detailed study, we select and describe two prevalent secure channel implementations, TLS and its variant Datagram Transport Layer Security (DTLS).

8.2.1 TLS

TLS [162] is a protocol to provide a secure channel between two peers. A core requirement is an underlying *reliable* and *in-order* transport (e.g., provided by TCP). Application payloads are split up into suitably sized blocks and are protected, according to negotiated options. One protected block of data is called a *record*. TLS can be structured into two protocol components, a handshake protocol that implements e.g., key exchange and option negotiation, as well as a record protocol, that implements traffic segmentation, protection, and transmission. The handshake protocol implements a three-phase handshake, initiated by the client. After successful completion, application data (records) can be transferred. The header overhead for a single *Application Data* type record is at least 5 B. Record payload sizes in TLS are limited to a maximum size of 2^{14} B. Hence, TLS records can exceed the available network Maximum Transmission Unit (MTU) and must be fragmented accordingly. While explicit record size limitation negotiation is standardized using TLS extensions [164], this must be implemented by the calling, higher-level application. TLS furthermore requires absolute ordered processing of records. A record r_i should be processed completely before record r_{i+1} is accepted. In case a fragment ϕ of r_i is lost during transmission, processing of further record data (or other complete records r_j, $j > i$) of the same connection cannot continue. Hence, ϕ must be retransmitted first to resolve the lock.

8.2.2 DTLS

There are scenarios in which a reliable transport such as TCP is unsuitable for an application, but a secure channel is still desired. DTLS [165] is a TLS variant, which allows operation over unreliable and datagram-based transports, such as UDP. Hence, DTLS neither requires nor provides reliable or ordered transport of data. DTLS reuses the TLS structure, with additional fields for rudimentary handshake reliability, fragmentation, and record numbering. The header overhead for a single *Application Data* type DTLS record is at least 13 B, due to additional fields for explicit sequence and epoch numbers. With a maximum record size of 2^{14} B, DTLS records can similarly exceed MTU and need to

be fragmented. Record size limit extensions apply. Since UDP does not offer payload segmentation, fragmentation of UDP / DTLS data records is conducted on the network layer. For handshake messages, DTLS additionally offers dedicated fragmentation logic, that allows to split a single handshake message over multiple records, that adhere to the current MTU. As DTLS does not require ordering between records, complete records of the same connection can be processed independently as long as all fragments are present. If fragments are missing, the affected record cannot be processed until retransmission.

9. Analysis

In this chapter, we analyze (1) the optimization space of transport protocols in context of BFT-SMR and (2) the impact of transport protocol change and configuration on a BFT-SMR setup. This chapter provides the base for interpretation and root-cause analysis of empirical measurements. We first define a set of suitable protocols for further study (§9.1) and discuss basic differences in terms of payload and header efficiency of the chosen protocols (§9.2). Afterward, we discuss characteristics of HotStuff communication and its bottleneck potential (§9.3). Next, we analyze the characteristics of the selected transport protocols in different operation scenarios (§9.4) and discuss their available configuration space (§9.5). Finally, we present an impact analysis of our chosen transport protocols in a BFT-SMR setting and conduct a brief case study for a HotStuff scenario (§9.6).

9.1 Protocol Selection

For concrete analysis, we choose a subset of protocols to study in detail. The subset should (1) contain the most prevalent protocols with good compatibility and hardware (offloading) support, (2) contain representatives for different ends of the protocol design complexity spectrum, and (3) allow for sufficiently simple configurability of transport behavior. From the set of introduced protocols in §8, we choose the following subset for study:

UDP With low complexity and features, UDP offers a lightweight and unreliable transport. It provides a baseline for simple protocols with low overhead.

TCP TCP is one of the most prevalent and widely supported transport protocols to date, also regarding available hardware offloading features. Providing a high-complexity, reliable, byte-stream abstraction, TCP is the default transport choice in many BFT-SMR systems (§5.1).

SCTP While less prevalent and supported than TCP, SCTP provides a message-based variant of a high-complexity, reliable transport. Differences in retransmission logic and far-reaching socket-level configuration options render SCTP an interesting comparison protocol to TCP.

RUDP With reduced complexity, RUDP offers a lightweight, but reliable transport and bridges the architectural gap between UDP and TCP. Modification of protocol logic is easily possible, due to implementation in calling application logic.

QUIC is another instance of a complex, reliable transport protocol. Related work suggests increased overhead (§8.1). In a HotStuff context, no multiplexing of independent objects or data over a single connection is conducted and HOL-blocking should not occur. Furthermore, connections between replicas are typically long-lived and frequent TLS association re-establishments are not expected. Hence, the core benefits of QUIC from a Web context do not apply. For these reasons, we omit a detailed study of QUIC and focus on RUDP instead. In a scenario where a BFT-SMR protocol conforms to these Web-typical attributes, study and application of QUIC might yield benefits.

Finally, as TLS usage has been proposed for BFT-SMR (5.1), we study both TLS and DTLS. This allows for evaluation of secure channel performance for both reliable and unreliable or unordered protocols. In the following, we always assume the operation of TLS with TCP and SCTP, and DTLS with UDP and RUDP, respectively.

9.2 Payload and Header Efficiency

Transport protocols vary in overhead, caused by different sizes of protocol *headers*. The share of header data affects both additionally required transmission bandwidth and processing. We introduce a simple model and notation to reason about the significance of this overhead. The overhead can be quantified in e.g., terms of additional instructions [166] or data to be transmitted [167, 168]. Typically, the overhead is then estimated as the additional effort in relation to the total effort invested. We focus on the payload size overhead, since metrics based on time or number of instructions may vary greatly, depending on computing platform and compiler. We assume IPv4 for brevity, if not stated otherwise.

9.2.1 Notation

We denote the size of a payload p as $|p|$ and refer to a single unit of data transmitted on a certain e.g., OSI protocol level as Protocol Data Unit (PDU). If the PDU carries a payload as a subset of its data that is interpreted by a higher-layer protocol, we refer to this payload as Service Data Unit (SDU). Hence, a protocol header is the part of a PDU that precedes its encapsulated SDU. For a header-type t, we denote the header data as $\eta(t)$ and the SDU payload, following the header, as $\rho(t)$. Depending on the payload usage function ρ, the payload size $|\rho(t)|$ can vary. For brevity, we do not model the header (size) as a variable parameter that needs to be provided as function input for every use. Consequently, the PDU of type t, consisting of header and payload is denoted as

$$P(t, \rho) := \eta(t) \circ \rho(t) \tag{9.1}$$

and its size is calculated as

$$|P(t, \rho)| := |\eta(t)| + |\rho(t)| \tag{9.2}$$

A packet with Ethernet header, IP header, and encapsulated SDU can be written as

$$\eta(\text{Ethernet}) \circ \eta(\text{IP}) \circ \rho(\text{IP}) = \eta(\text{Ethernet}) \circ P(\text{IP}, \rho)$$

Now consider a packet that consists of a sequence of headers. Assume that we know the packet consists of a successive header-type sequence $(t, t', ..., t^*)$, which we denote as $(t \to t^*)$ for brevity, and some SDU $\rho_s(t^*)$. Then, we can also express this packet as

$$\begin{aligned} P(t \to t^*, \rho_s) &= \eta(t \to t^*) \circ \rho_s(t^*) \\ &= \eta(t) \circ \eta(t') \circ ... \circ \eta(t^*) \circ \rho_s(t^*) \end{aligned} \tag{9.3}$$

Using the introduced notation, we can define the payload overhead $\psi_o(t)$ and the payload efficiency $\psi_e(t)$ of a header-type t and payload ρ as

$$\psi_o(t, \rho) := \frac{|\eta(t)|}{|P(t, \rho)|} \tag{9.4}$$

$$\psi_e(t, \rho) := \frac{|\rho(t)|}{|P(t, \rho)|} = 1 - \psi_o(t, \rho) \tag{9.5}$$

where informally $\psi_o(t)$ captures how much relative space in a message is taken up by the protocol header, given a certain SDU payload $\rho(t)$, and $\psi_e(t)$ captures the share of SDU data in the PDU. If we are not interested in the overhead or efficiency of a single header type (layer) but we want to argue about the whole packet up until header of type t^*, we can use Eq. 9.3 to calculate them for a predefined sequence of header types $(t, t', ..., t^*)$ as

$$\psi_o(t \to t^*, \rho) = \frac{|\eta(t \to t^*)|}{|P(t \to t^*, \rho)|} = \frac{\sum_{k \in (t \to t^*)} |\eta(k)|}{\left(\sum_{k \in (t \to t^*)} |\eta(k)|\right) + |\rho(t^*)|} \tag{9.6}$$

$$\psi_e(t \to t^*, \rho) = \frac{|\rho(t^*)|}{|P(t \to t^*, \rho)|} = \frac{|\rho(t^*)|}{\left(\sum_{k \in (t \to t^*)} |\eta(k)|\right) + |\rho(t^*)|} \tag{9.7}$$

In informal terms, in Eq. 9.6 and 9.7 we interpret all headers of the given types $(t \to t^*)$ as a single header $(\eta(t \to t^*))$, followed by the single target SDU payload $\rho(t^*)$.

Consider this example calculation of ψ_e and ψ_o for a raw, Layer 1, 802.3 Ethernet packet [169, p. 239ff] (including preamble, start frame delimiter, frame check sequence, and minimum interpacket gap of 12 B [169, p. 282]), without 802.1Q tag and without going into details of its encapsulated SDU. Let us assume that payload data ρ_{mtu} is always sent, considering a default MTU of 1500 B. Then

$$\psi_o(\text{Ethernet}, \rho_{mtu}) = \frac{|\eta(\text{Ethernet})|}{|P(\text{Ethernet}, \rho_{mtu})|} = \frac{38\,\text{B}}{38\,\text{B} + 1500\,\text{B}} \approx 2.47\%$$

$$\psi_e(\text{Ethernet}, \rho_{mtu}) = 1 - \psi_o(\text{Ethernet}, \rho_{mtu}) \approx 97.53\%$$

If we conduct the same calculation for the minimal possible payload ρ_{min} of 46 B, the relative payload overhead increases and the efficiency decreases:

$$\psi_o(Ethernet, \rho_{min}) = \frac{38\,\text{B}}{38\,\text{B} + 46\,\text{B}} \approx 45.24\%$$

$$\psi_e(Ethernet, \rho_{min}) = \frac{46\,\text{B}}{38\,\text{B} + 46\,\text{B}} \approx 54.76\%$$

As expected, the smaller the application payload (e.g., client request size in a BFT-SMR system), the bigger the impact of network protocol payloads becomes.

Table 9.1: Approximate Cumulative Header Sizes for Transport Protocols

	$\eta(\text{Eth} \to \text{UDP}, \rho)$	$\eta(\text{Eth} \to \text{TCP}, \rho)$	$\eta(\text{Eth} \to \text{SCTP}, \rho)$
w/o (D)TLS	66 B	78 B	86 B
w/ (D)TLS	111 B	115 B	123 B

9.2.2 Protocol Analysis

We analyze the payload efficiency of our chosen transport protocols (§9.1) with and without application of (D)TLS. Since RUDP does not add additional header data on the transport layer, we omit its analysis here and consider UDP to be representative. Let us discuss UDP. The protocol header is relatively small with a size of 8 B [152]. In minimal configuration, the IP header occupies at least 20 B, [151, p. 10ff]. Let the BFT-SMR payload be defined by ρ_u, then we can write the PDU as

$$\begin{aligned} P(\text{Eth} \to \text{UDP}, \rho_u) &= \eta(\text{Eth} \to \text{UDP}) \circ \rho_u(\text{UDP}) \\ &= \eta(\text{Eth}) \circ \eta(\text{IP}) \circ \eta(\text{UDP}) \circ \rho_u(\text{UDP}) \end{aligned}$$

With this we can write the payload efficiency as a function of an arbitrary payload ρ_u:

$$\psi_e(\text{Eth} \to \text{UDP}, \rho_u) = \frac{\rho_u}{38B + 20B + 8B + \rho_u} = \frac{\rho_u}{66B + \rho_u}$$

Overhead of (D)TLS may vary with plaintext length and Authenticated Encryption with Associated Data (AEAD) algorithm. For simplicity, we assume a static MAC size of 32 B. Assuming that the UDP header is followed by a DTLS header [165, §4] with 13 B header data and a 32 B MAC (for brevity we assume the MAC authenticator to be reflected in the header, not as trailer after the payload), we can write the PDU as

$$\begin{aligned} P(\text{Eth} \to \text{DTLS}, \rho_u) &= \eta(\text{Eth} \to \text{DTLS}) \circ \rho_u(\text{DTLS}) \\ &= \eta(\text{Eth}) \circ \eta(\text{IP}) \circ \eta(\text{UDP}) \circ \ \eta(\text{DTLS}) \circ \rho_u(\text{DTLS}) \end{aligned}$$

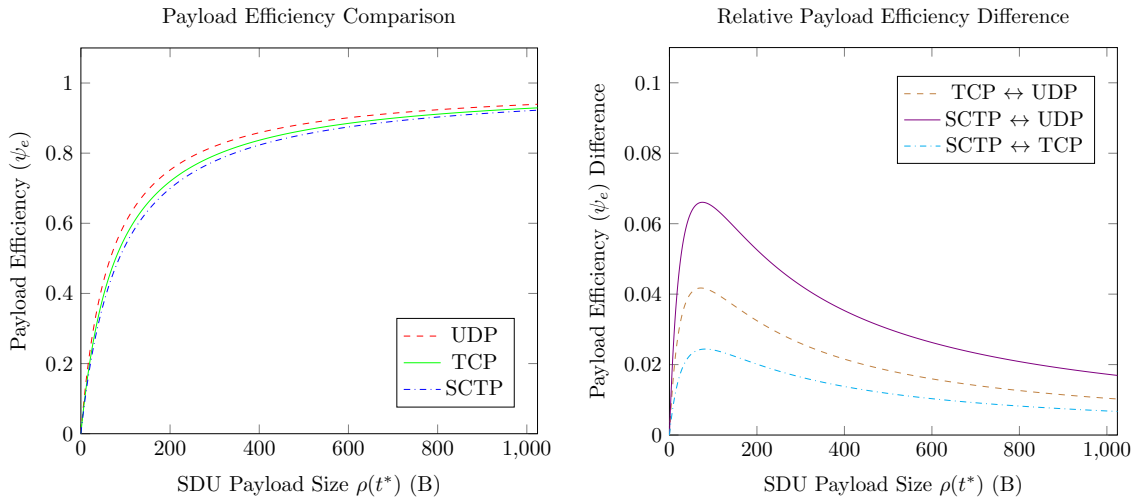
and find a respective efficiency of

$$\psi_e(\text{Eth} \to \text{DTLS}, \rho_u) = \frac{\rho_u}{38B + 20B + 8B + 45B + \rho_u} = \frac{\rho_u}{111B + \rho_u}$$

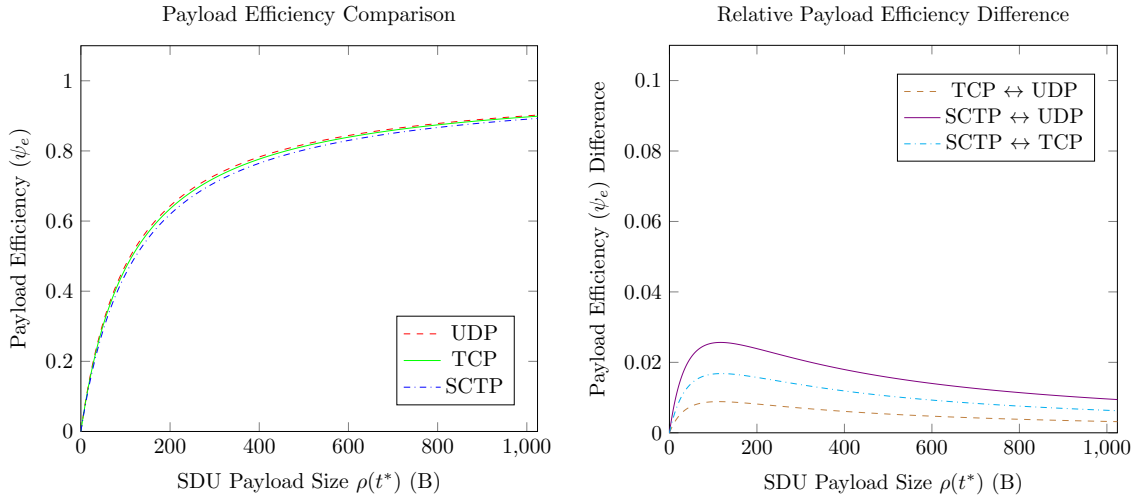
For brevity, we omit notation details for all other protocols and summarize assumptions and efficiency results in the following. The TCP protocol header occupies 20 B in minimal configuration [153, §3.1] and for TCP/TLS operation the TCP header is followed by a TLS record [162, p. 77ff], containing at least 5 B of header data and a 32 B MAC tag. We model an SCTP packet with a common header of 12 B, and a variable set of chunks of different types [170, §3.1]. We consider a minimal configuration, consisting of the common header and a single data chunk, with a data chunk header size of 16 B [170, §3.3.1].

Without defining the payload $\rho(t)$, the efficiency functions $\psi_e(t \to t^*, \rho(t))$ only vary in the header size $\eta(t \to t^*)$. Tab. 9.1 displays the calculated header sizes for all protocols with and without (D)TLS. Using this data, we can approximate the payload efficiency for representative payload sizes. Let $\rho_{min} := 20\,\text{B}$ (size of serialized HotStuff client command metadata) and $\rho_{big} := 1024\,\text{B}$ (size of biggest transmitted HotStuff client payload in the original paper measurements; [5], Fig. 5). We now provide a combined plot of the payload efficiencies (left) and relative efficiency differences (right) for all three protocols for varying payload size $\rho(t^*) \in [\rho_{min}, \rho_{big}]$ of the final SDU in Fig. 9.1. Subfigures 9.1a and 9.1b show the calculations without and with usage of (D)TLS, respectively.

Payload efficiency of all protocols increases rapidly with the payload size. Relative efficiency differences are bigger without (D)TLS but absolute efficiency differences are marginal, with, e.g., a maximum of $\sim 4\%$ difference between TCP and UDP.



(a) Without (D)TLS



(b) With (D)TLS

Figure 9.1: Payload Efficiency Comparison

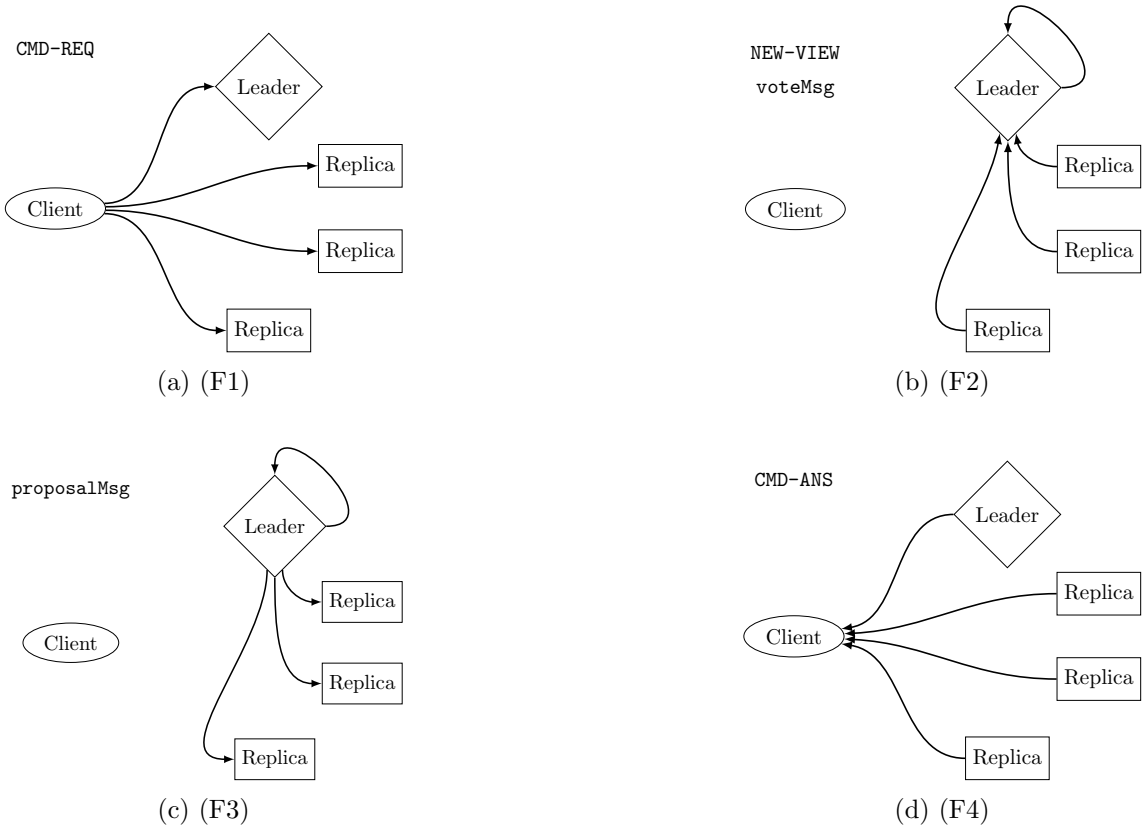
9.2.3 Summary

Protocol overhead is influenced by the amount of header data transmitted. The studied protocols vary in header size, but the resulting efficiency differences are marginal for small payloads and become insignificant for larger payloads. Hence, we do not expect a significant performance impact caused by the header overhead differences between protocols.

9.3 HotStuff Communication

To create a theoretical base for evaluation of processing and bandwidth bottlenecks, we analyze typical message flows in regular HotStuff communication. To simplify the discussion, we assume $n = 4$, a single client issuing requests and no lost command requests from client to replicas. HotStuff features two dominant communication patterns (§6), client ↔ replica and replica ↔ replica communication. Differentiating by transmission direction, we identify four flows between nodes shown in Fig. 9.2, enumerated from Flow 1 (**F1**) to Flow 4 (**F4**). We adopt the message names as defined in the original HotStuff paper.

In **F1** the client sends a command request (`CMD-REQ`) to all available replicas. **F2** describes communication from the replicas to the current leader, such as `NEW-VIEW` messages and



(a) (F1) (b) (F2)

(c) (F3) (d) (F4)

Figure 9.2: Regular Communication Flows in HotStuff

votes (`voteMsg`) for received proposals. **F3** contains new proposal messages from the current leader to other replicas. Finally, **F4** describes replicas, sending command answers (`CMD-ANS`) back to the client, after commit of the requested operation.

9.3.1 Payload Analysis

Assume that peer P has outgoing ($\rightarrow$) communication in flow ϕ. Then we denote this flow as $P_{\phi}^{\rightarrow}$ and the size of the transmitted payload as $|P_{\phi}^{\rightarrow}|$. Incoming communication is denoted by ($\leftarrow$) respectively. The payload size of a certain message m is denoted by $|m|$. Following this notation, we list the payload size of the roles of Client (C), Replica (R) and Leader (L) for the flows and message names from Fig. 9.2. Note that *leader* and *replica* are handled as separate roles in the following discussion, but in practice one physical node fulfills both roles at the same time, incurring the total overhead of both roles.

$$|C_1^{\rightarrow}| = n \cdot |\texttt{CMD-REQ}| \tag{9.8}$$

$$|R_1^{\leftarrow}| = |\texttt{CMD-REQ}| \tag{9.9}$$

In **F1**, the client sends one request to each replica, which receives exactly this request.

$$|R_2^{\rightarrow}| = |\texttt{NEW-VIEW}| \tag{9.10}$$

$$|\hat{R}_2^{\rightarrow}| = |vote(\texttt{GENERIC})| \tag{9.11}$$

$$|L_2^{\leftarrow}| = n \cdot |R_2^{\rightarrow}| = n \cdot |\texttt{NEW-VIEW}| \tag{9.12}$$

$$|\hat{L}_2^{\leftarrow}| = n \cdot |\hat{R}_2^{\rightarrow}| = n \cdot |vote(\texttt{GENERIC})| \tag{9.13}$$

In **F2**, replicas either send `NEW-VIEW` messages to the new leader (9.10), or vote for the current proposal (9.11). Since the available codebase implements the pipelined HotStuff

variant, votes and proposals are of type `GENERIC`. The incumbent leader receives at least $(n-f) \in \mathcal{O}(n)$ messages in either case. While the HotStuff code only considers the first $(n-f)$ valid votes for further cryptographic processing (e.g., creation of QCs), the incoming messages are still rudimentarily processed before this decision. We assume a scenario in which all replicas participate, i.e., fault-free execution and maximum possible load. Thus, we model the payload strain with n responses.

$$|L_3^{\rightarrow}| = n \cdot |proposal(\texttt{GENERIC})| \tag{9.14}$$
$$|R_3^{\leftarrow}| = |proposal(\texttt{GENERIC})| \tag{9.15}$$

In **F3**, the leader sends the next proposal to all replicas, which receive this message once.

$$|R_4^{\rightarrow}| = |\texttt{CMD-ANS}| \tag{9.16}$$
$$|C_4^{\leftarrow}| = n \cdot |\texttt{CMD-ANS}| \tag{9.17}$$

In **F4** the replicas answer the client request. While the client only requires $(f+1)$ matching `CMD-ANS` messages, excess messages are still received and interpreted by both networking stack and client code. We therefore model the payload strain in (9.17) with n responses.

Independent of the actual payload size, we can identify two potential bottlenecks. As outlined in previous literature (§5.2), the leader is subject to high transmission and processing volumes, scaling with the number of replicas n. However, considering a single client setup, also the client is subjected to considerable transmission and processing load, scaling with the number of replicas n. The operation of multiple parallel clients, requesting independent operations, affects both F1 and F4. Denoting the number of clients as c, we find

$$|\hat{C}_1^{\rightarrow}| = n \cdot |\texttt{CMD-REQ}| \tag{9.18}$$
$$|\hat{R}_1^{\leftarrow}| = c \cdot |\texttt{CMD-REQ}| \tag{9.19}$$
$$|\hat{R}_4^{\rightarrow}| = c \cdot |\texttt{CMD-ANS}| \tag{9.20}$$
$$|\hat{C}_4^{\leftarrow}| = n \cdot |\texttt{CMD-ANS}| \tag{9.21}$$

An increasing number of clients results in more strain on the replicas (9.19, 9.20), potentially shifting the bottleneck toward the replicas. The general strain on the client becomes more pronounced when command batching is active (§10.2). While agreement protocol messages between replicas (F2 and F3) only contain (a batch of) *hashes* of client requests to execute, client requests (F1) and responses (F4) directly contain the *full payload*. While the HotStuff codebase allows independent configuration of request/response payload size, we assume symmetric payload sizes. As `CMD-ANS` messages in F4 are sent by the n replicas at roughly the same time, the client input bandwidth is increased (§18.6).

9.3.2 Effect of message loss

If not masked by the underlying transport protocol, packet loss may impact robustness and performance of the BFT-SMR system. In the following, we analyze this impact on HotStuff flows (Fig. 9.2) and argue for the necessity of reliable communication.

F1

In F1, loss of `CMD-REQ` messages can cause instability of the protocol execution. Consensus decisions over a client request are formed via communication of *command hashes*, not the actual request payloads. In the HotStuff implementation, the requests to execute are only transmitted via `CMD-REQ` messages from client to replica. While replicas can request missing "blocks" of committed decisions from other replicas if they receive a proposal with

an unknown block hash, these blocks do not contain the request payloads to execute. Furthermore, replicas do not check if they received a `CMD-REQ` message from the client before voting on proposals. Hence, if transmission of a `CMD-REQ` fails, the replica is not able to obtain (and finally execute) the missing requests, except if the agreement fails and the requests are resent by the client. At the same time, these replicas will not send a `CMD-ANS` message to the client in F4, even if the agreement on the hashes succeeded. Thus, either replicas, missing the requests, will fall behind the state of other replicas, or the client will not receive enough $(f+1)$ responses to accept the committed decision. If the transmission of `CMD-REQ` to the leader fails, it will not create a proposal for these requests and a new leader is elected eventually. If a client waits indefinitely for some answer to its issued `CMD-REQ` messages, the whole process might get stuck for subsequent transmission failures to leaders, as the implementation limits the number of `CMD-REQ` messages in flight.

F2

Loss of a `voteMsg` does not necessarily result in a failed agreement, as long as the leader still receives $(n-f)$ valid votes. If the leader makes progress, the next successful proposal message will allow the replica to catch up. If the leader receives less than $(n-f)$ valid votes, the current view cannot continue and a new leader will be elected eventually. Leader election in HotStuff is realized by the replicas sending `NEW-VIEW` messages to the new leader. Once the new leader has collected $(n-f)$ `NEW-VIEW` correct messages, he can create a new valid proposal. Otherwise, the leader will never create a proposal for this view and execution will stall until the next `NEW-VIEW` interrupt.

F3

In case a `proposalMsg` is lost, the affected replica r cannot participate in the current voting round. Additionally, r will not transition its view state, as it is missing the QC and block from the proposal. However, the current leader and other replicas that received the proposal can indeed make progress, assuming there are enough correct replicas left to form a quorum. If the other replicas make progress, r cannot directly participate in the next view and first needs to catch up to the current block using `BLOCK-REQ` messages. Due to this delay, r could potentially lag behind, preventing progress.

F4

Loss of `CMD-ANS` messages mainly affects the client in his assumption of the replica state. If $(f+1)$ valid and matching responses are received, the BFT-SMR protocol can continue regularly. However, if the replicas accepted and executed a client command c, but the client *does not* receive enough responses, the client view and the actual state of the system diverge. This is a problem if e.g., operation o is not idempotent with respect to the replica state. The client assumes that execution of o failed and might schedule a request of o for a second time. Resulting behavior depends on the replicated application.

9.3.3 Summary

Leader replica and client are potential bottleneck nodes. Agreement on client requests is conducted on hashes, the (potentially large) request payloads are only transmitted between client and replicas. Since replicas answer to the client in parallel, ingress bandwidth spikes are expected on the client. If reliable communication is not given, lost client $\leftrightarrow$ replica messages may lead to state inconsistencies in the original PoC HotStuff implementation. Lost proposal messages have a larger performance impact than lost votes.

9.4 Transport Protocol Operation

We analyze our chosen transport protocols in terms of available features and overhead for reliable operation and CC. Implications in context of BFT-SMR operation are discussed.

9.4.1 Reliability

A core task of a transport protocol is to provide the abstraction of a reliable, point-to-point connection between peers. We focus our discussion on two situations, operation with and without packet loss. We deem analysis of dedicated attacks on transport-layer level (e.g., TCP SYN Flood [171]) out of the scope of this thesis and refer to the respective literature for performance impact and defense strategies. IP fragmentation should be avoided by choice of protocol or configuration parameters, as it bears potential for both performance reduction and security concerns [172, 173]. Either the transport protocol or the above application (layer) should ensure that transmitted payloads do not violate path MTU.

Operation without loss

We first discuss UDP. Due to its connectionless nature, no initial setup between communication partners is necessary. UDP thus avoids a small delay caused by initial handshake round trips. The average delay impact of the connection setup becomes negligible, the longer the connection persists. In scenarios with large numbers of shortlived connections or modes where requests are sent in high-throughput bursts, with considerable pause between two bursts, the impact may be larger. No additional state is saved for transmitted or received packets. If no IP fragmentation takes place, possible reordering of packets during transmission does not affect plain UDP throughput. Reordering needs to be handled by the consuming application, if necessary at all. Without omission faults, UDP thus provides small initial delay and reduced processing and state overhead during transmission.

RUDP only differs marginally from UDP, in that the upper-level application keeps timing information for potential retransmission of lost messages. Except transmission of dedicated NACK messages in case of error, no additional overhead on top of UDP is introduced.

For TCP, a three-way-handshake ($>$ 1 Round-trip Time (RTT)) is necessary before transmission of application data. TCP provides the abstraction of an ordered byte-stream communication interface. Without omission faults, logic and state keeping for retransmission of lost packets is unnecessary overhead. So is the logic for ordered delivery, in case ordering is not required. Additionally, TCP features mechanisms for CC, which may contribute to but also interfere with performant BFT-SMR protocol operation (§9.4.2). TCP automatically splits data exceeding the MTU submitted for sending, if the size of the payload, the configuration, and a suitably negotiated Maximum Segment Size (MSS) [153] allow. Consequently, without omission faults, TCP usage incurs state and processing overhead. Initial connection setup delay might be detrimental to low-latency requests, many short-lived connections, or burst transmission operation.

In case of SCTP, a four-way-handshake ($\geq$ 2 RTT) is required before the transmission of application data can proceed. Without omission faults, SCTP's retransmission logic and state are unnecessary. Ordered delivery for messages can be deactivated, providing more flexibility. Like TCP, SCTP provides CC mechanisms. SCTP additionally provides path MTU discovery and user data fragmentation [154]. Thus, without omission faults, SCTP involves more overhead than TCP.

Operation with loss

In real-world systems, especially in setups without additional Quality of Service (QoS) guarantees, packet loss can occur. To prevent such omission faults from being interpreted

as faulty replicas by the BFT-SMR protocol, reliable connections are assumed (§5.1). If not masked, they potentially trigger expensive slow-path fallback routines in optimistic protocols [8, 23, 93] or a leader-change in leader-based protocols.

In case of UDP, no reliability is provided. UDP does not track if a datagram is lost and does not take any further action. Omission fault handling responsibility is directly transferred to the higher layer. As discussed in §9.3.2 HotStuff state desynchronization is possible when using plain UDP on client $\leftrightarrow$ replica links. Safety impact on other BFT-SMR protocols may vary, although unhandled omission faults will likely reduce performance.

TCP provides explicit logic for handling retransmission of lost segments, as well as CC and FC. TCP can detect loss of segments by keeping track of sequence numbers, following the number of transmitted bytes. The receiver sends ACKs to the sender, stating the sequence number of the next expected byte. As reordering of segments during transmission can occur, not every out-of-order reception is caused by loss. To detect loss, modern TCP employs two techniques: A Retransmission Timeout (RTO) [153] and Duplicate Acknowledgments (DupAcks) [174]. The sender expects a (cumulative) ACK for each byte transmitted. If no ACK is received for a certain time, a retransmission is triggered. The actual timeout (RTO) is calculated as a bounded, smoothed, and delayed estimation of the current RTT [153]. If a retransmission is triggered because the RTO expired, the current RTO is doubled, resulting in an exponential "back off" behavior [175]. In the Linux kernel (e.g., v5.15.0-72), the default lower and upper bounds for TCP RTO are defined as

```
#define TCP_RTO_MAX     ((unsigned)(120*HZ))
#define TCP_RTO_MIN     ((unsigned)(HZ/5))
```

where `HZ` refers to the kernel timer interrupt frequency. A possible default configuration is `CONFIG_HZ=1000`, which results in a `TCP_RTO_MIN` value of 200ms. However, the delay between two successively sent segments may be much smaller than the configured `TCP_RTO_MIN` value. If a segment is lost during transmission, the receiver might receive data out of order. In this case, the receiver sends an immediate DupAck to inform the sender of the originally expected data. Since TCP masks omission faults by retransmission, the BFT-SMR protocol is affected in terms of performance. Besides the retransmission overhead, suboptimal choice of timeout values (e.g., RTO) and omission faults interpreted as network congestion can reduce the available transport protocol throughput [176].

Similar to TCP, SCTP provides reliable transmission as well as CC and FC. In order to realize reliable transmission, SCTP employs sequence numbers. However, as SCTP also allows for unordered transmission, network transmission sequence and sequence of delivery to the higher-level application are kept logically separate. In other words, SCTP accepts and caches incoming datagrams, even if the contained data chunks are out of order. Instead of exclusively cumulative ACKs as in TCP, SCTP uses SACKs, which acknowledge certain ranges of received data chunks, using explicit *Gap Ack* specifications [170, §3.3.4]. To detect loss, SCTP employs two techniques: Explicit gap specification in SACKs and an RTO. The calculation of SCTP RTO is similar to the calculation performed by TCP, including the exponential back-off behavior described before. The recommended values for SCTP RTO configuration [170, §16] are slightly differing from the TCP defaults with `RTO.Min` = 1 s and `RTO.Max` = 60 s. The basic CC algorithms used by SCTP are also based on TCP CC, as defined in RFC 5681 [174]. Differences mainly emerge for multi-homing setups. As SCTP masks omission faults by retransmission, the upper-layer BFT-SMR protocol is affected in terms of performance. While suboptimal configuration of protocol parameters (e.g., RTO) can have a negative impact, configuration is easier for SCTP due to its accessible socket option interface.

Finally, RUDP provides basic reliability, with retransmission logic implemented by the calling application. The application keeps an RTO, where RTT estimation follows the

calculation from TCP. No upper or lower bounds on the RTO are enforced on RUDP level. ACKs are implicitly defined through expected application-level responses. RUDP does not implement segmentation, hence IP fragmentation occurs if the carried SDU causes exceedance of path MTU. Also, without segmentation, retransmission logic always retransmits the whole payload, even if only a single IP fragment was lost or is erroneous. Except for inherent application-level logic requirements, no ordering is provided and neither CC nor FC are implemented. In context of BFT-SMR, RUDP provides a relatively lightweight but reliable transport. Omission faults still impact performance.

9.4.2 Congestion Control

To prevent overloading of network resources, both TCP and SCTP implement CC. Since CC in SCTP closely follows algorithms and rationale of TCP, we limit our explicit discussion to TCP here. Network congestion typically causes a reduction of throughput on the strained links. While load reduction in response to an actual congestion event is helpful (i.e., in order to prevent congestive collapse [177]), performance degradation due to negative interaction with typical BFT-SMR operation should be avoided. Detection of congestion events and response are defined by the CC algorithm in effect. Since the original specification of TCP in 1981 [178], a plethora of different CC algorithms have been proposed [179]. Approaches mainly differ in terms of congestion detection method and target network scenario. As outlined by Afanasyev et al. [179], optimization of CC revolves around balancing (1) throughput against network load and (2) loss recovery latency against unnecessary retransmissions. As concrete constraints vary with the target environment, there is no one-size-fits-all algorithm that performs best in all scenarios.

The main difference between CC in context of general BFT-SMR operation and CC in commodity use cases are assumptions on the expected behavior of participating peers. In BFT-SMR scenarios, malicious participants are explicitly part of the model. In the design process of transport protocols, basic security concerns are usually addressed (e.g., defense against sequence number attacks [180] or SYN flooding [171] in TCP) but protection against stronger adversaries is typically assumed to be handled by another layer. For example, the SCTP specification suggests employing IPSec [181] to provide extended confidentiality and integrity guarantees [154]. Hence, CC in a BFT-SMR setting is potentially subject to stronger adversarial behavior and may require additional protective measures.

9.4.3 Summary

Both TCP and SCTP avoid IP fragmentation by splitting payloads themselves, TCP into *segments*, SCTP into *chunks* before handing them to the network layer. TCP can often benefit from hardware offloading support such as TCP Segmentation Offloading (TSO) [182]. UDP does not offer comparable segmentation logic on its own. For scenarios without loss, UDP offers lower header, protocol processing, and handshake overhead in comparison to TCP and SCTP. However, this advantage quickly amortizes for longer connections and larger payloads. In lossy scenarios, plain UDP is not suited for use in a HotStuff setup due to missing reliability (§9.3). While both TCP and SCTP provide reliable transmission, they differ in features and overhead. TCP has a slightly smaller header and typically better offloading support. While TCP supports SACKs [183], information transmitted in SACK options is only advisory. That is, non-SACKed segments do not need to be retransmitted, but to trigger a retransmission either the RTO must expire, or the sender receives the third DupAck. Operation of purely cumulative ACKs, may result in unnecessary retransmission of already received data [183]. SCTP is built around the usage of SACKs and gap specifications with different semantics than TCP. SCTP specifies the generation of a SACK for each received, valid data chunk [170]; SACKs are thus generated with higher frequency. Ordered transmission is optional in SCTP.

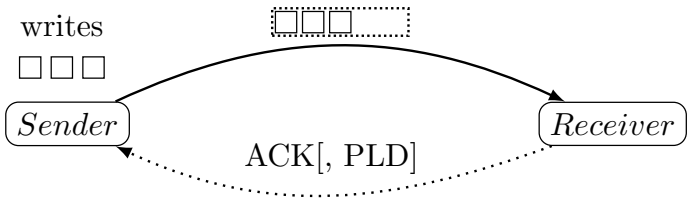


Figure 9.3: Possible Coalescence Behavior

Table 9.2: Delay and Coalescence Behavior Parameters

Option	Layer	Effect
`tcp_autocorking`	procfs	Merge socket writes in the same flow if at least one packet in Queuing Discipline (qdisc) or device transmit flow
`TCP_CORK`	Socket	Queue small submitted writes until a full frame can be sent, this option is removed or 200 ms elapsed
`TCP_NODELAY`/ `SCTP_NODELAY`	Socket	Nagle's Algorithm off, segments sent as early as possible
`TCP_QUICKACK`	Socket	Toggle Quick ACK mode; if active, ACKs are sent immediately; non-permanent, overridden by kernel
`TCP_DELACK_MIN`/ `TCP_DELACK_MAX`	Kernel	Min and max delay time if delayed ACKs are active

9.5 Protocol Configuration Space

We discuss parameters with potential impact upon execution of a BFT-SMR protocol. We limit our discussion of general transport protocol options to network stack implementations for a recent Linux Kernel (v5.15.0-72). Configuration is generally possible through three interfaces, the `proc` filesystem, socket options, and modification of kernel parameters or code. Most configuration options are provided by the TCP stack, SCTP does not offer configuration via `procfs` and the number of UDP configuration options is negligible. Relevant options can be categorized into two classes: (I) Delay and coalescence of small writes and (II) retransmission behavior. Documentation of available `procfs` and socket options is provided in the Linux Programmer's Manual pages for UDP, TCP, and SCTP [184–186].

9.5.1 Delay and Coalescence of Writes

In certain scenarios, deliberate delay or merging of many small writes to a socket can be advantageous. Consider a basic communication example between Sender and Receiver in Fig. 9.3. From a protocol efficiency perspective (§9.2) sending a large number of packets with a small payload results in increased bandwidth and processing overhead. So is frequent sending of ACKs for very small incoming writes. Frequent invocation of a (write) system call for small amounts of data may be inefficient. Possible remedies are e.g., (1) merging many small writes into a larger write and (2) adding deliberate response delay on receiver side and piggybacking ACKs to necessary payloads if possible. We list a selection of options to configure the described protocol behavior in Tab. 9.2.

If `tcp_autocorking` is enabled, small socket writes are buffered and coalesced, if at least one packet from the same flow is currently in a qdisc or device transmit flow. Thus, latency of a single segment might increase in case of many small, subsequent writes to the same target. In the context of typical HotStuff operation, this situation can only occur

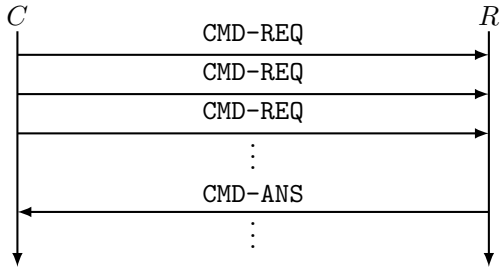


Figure 9.4: Client ↔ Replica Communication Example, Flows F1 / F4

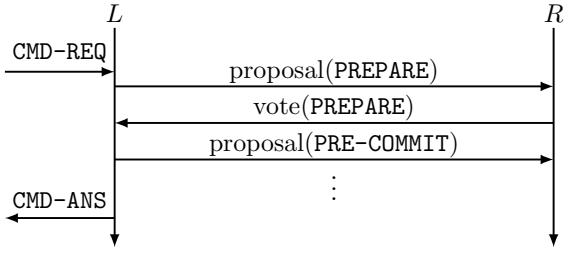


Figure 9.5: (Leader) Replica ↔ Replica Communication Example, Flows F2 / F3

in client-replica communication (**F1**, **F4**, §9.3) and if the request or response payloads are sufficiently small. In all other flows, write calls to the same target depend on previous input (e.g., the next leader / replica message). This request-response communication structure is also present in other BFT-SMR protocols. Consider the scenario from Fig. 9.4. The client tries to write multiple small `CMD-REQ` messages and the replica answers with small `CMD-ANS` messages after a successful commit. While a small payload increases relative protocol payload overhead, minimizing latency for `CMD-REQ` and `CMD-ANS` messages (or their equivalents in other protocols) should be prioritized, if the additional delay of coalescing writes exceeds the potential delay, in protocol processing and transmission time or application-level logic, without write coalescence. A number of b `CMD-REQ` requests and $(f + 1)$ `CMD-ANS` responses are required to progress on a batching leader and on client side respectively (§6). Hence, coalescence of responses might introduce additional delay, as the number of buffered messages could already exceed the required number to progress.

The `TCP_CORK` option buffers writes until a full frame can be sent, the option is removed or 200 ms have elapsed. While activation conditions are different in that manual corking is possible, usage implications are analogous to `tcp_autocorking`.

Both TCP and SCTP offer a respective `NODELAY` option, which disables Nagle's Algorithm [187]. Its write-merge behavior mainly depends on the size of the issued writes. If both data size of the planned write and window size are large enough, the full segment is sent immediately. Partial segments are only sent without buffering and merging, if there is no unacknowledged data in flight. Hence the general behavior and implications upon BFT-SMR protocols are similar to the previous options and are mainly affecting client-replica communication since the request-response pattern allows for piggybacked ACKs in replica-replica communication. Consider the scenario from Fig. 9.5. The leader receives enough `CMD-REQ` messages to form a proposal. Replication of the requested command (batch) is conducted through alternating proposal and vote messages with the replica(s). Communication is alternating, hence ACKs can be piggybacked to regular messages and the amount of unacknowledged data in flight is reduced in comparison to F1 / F4. Depending on the used signature scheme (Part III), authenticators increase message size above segment size such that Nagle's algorithm or autocorking do not apply.

Table 9.3: Retransmission Behavior Parameters

Option	Layer	Effect
`tcp_frto` / `tcp_frto_response`	procfs	Configure forward RTO discovery to prevent unnecessary retransmissions upon spurious RTO
`SCTP_RTOINFO`	Socket	Configure initial, min and max RTO
`TCP_RTO_MIN / MAX`	Kernel	Configure min and max RTO value

TCP also allows to delay and coalesce explicit, non-piggybacked ACKs under special circumstances, involving the receipt of non-full segments and mostly unidirectional communication. While this delay should not be big enough to trigger an RTO, delayed ACKs can interact badly with some configurations. For example, delayed ACKs on receiver side interact badly with active Nagle's algorithm on sender side, where the sender delays writes until an ACK is received, but the receiver delays ACKs in expectation of more incoming writes. For applications that require timely ACKs for mostly unidirectional data sent, but do not fill a segment with a single write, delayed ACKs can reduce performance. Reducing the ACK delay may counter this behavior. Delayed ACKs may affect CC behavior (and thus the performance) of the protocol. The TCP ACK delay behavior can be configured through multiple parameters. We focus our discussion on three central options here.

The socket option `TCP_QUICKACK` allows to toggle Quick ACK mode, in which ACKs are sent immediately. However, this option is not permanent and Quick ACK mode can be left and entered by the kernel, depending on TCP protocol state. Thus, manual steering of Quick ACK mode is non-trivial as it requires knowledge of internal TCP processing state, might interact badly with CC, and requires frequent re-activation. If not operating in Quick ACK mode, TCP delays ACKs to reduce protocol overhead. The effective delay is bounded by the `TCP_DELACK_MIN/MAX` parameters, which are defined to have a minimum value of 40 ms and a maximum value of 200 ms for a kernel interrupt frequency of 1000 Hz. In context of BFT-SMR protocols, delayed ACKs do not negatively influence most flows due to their message exchange structure. For practical choices of batch and payload size, no flow holds a scenario in which a party issues many writes, smaller than segment size, and requires timely explicit ACKs to continue sending. For small payloads and deactivated batching, ACK delay might need to be optimized to prevent stalling in **F1**, if the BFT-SMR client limits the number of requests in flight (e.g., as in HotStuff) too aggressively.

9.5.2 Retransmission Behavior

Reliable transport protocols typically offer a range of options to configure retransmission behavior. A selection of relevant options is given in Tab. 9.3. TCP uses a combination of DupAck and RTO to trigger retransmissions (§9.4). Due to sporadic network delay or misconfiguration, the RTO can expire even though a segment eventually arrives. This *spurious* RTO causes unnecessary retransmissions and interacts poorly with CC. To counteract this behavior, modern TCP implements Forward RTO-Recovery (F-RTO) [188]. The algorithm tries to detect spurious RTOs based on information from incoming ACKs. If detected, the typical slow start behavior is avoided and operation continues, based on old timeout values and windows. The `tcp_frto` option allows for configuration if and which F-RTO algorithm should be active. If a spurious RTO is detected, the respective response behavior can be configured through the `tcp_frto_response` option. Linux TCP implementations (e.g., Kernel v5.15.0-72) offer three possible configuration values, which halve the congestion window and slow-start threshold after one RTT (0), immediately (1) or not at all (2). Results by Sarolahti [189] suggest that the overall performance of the chosen method depends on the available link bandwidth, where more conservative behavior

(0) performs better for low-bandwidth links and aggressive restoration (2) results in more throughput on high-bandwidth links. Hence, configuration for BFT-SMR systems should include F-RTO and be conducted accordingly.

As outlined by Allman et al. [190], configuration of e.g., RTO provides a powerful tool to trade response latency against the risk of premature timeouts and thus unnecessary retransmissions. Both TCP and SCTP offer options to adjust RTO variables. For SCTP, configuration is possible on socket level using the `SCTP_RTOINFO` option. It allows configuring the initial, minimum and maximum RTO value. For TCP, dynamic configuration is not easily possible, since the `TCP_RTO_MIN / MAX` values are configured in the Linux kernel build (§9.4). In context of BFT-SMR systems, the RTO value calculation is mostly relevant in case of message loss. Consensus decision latency in e.g., HotStuff is significantly smaller than the minimum RTO in both SCTP and TCP (cf. measured latencies between 10 and 30 ms for HS3-p1024 [7]). Multiple subsequent omission failures can trigger an RTO and cause a multiple of typical consensus latency in delay. Following the observation from [190], reduction of the minimum RTO reduces latency if omission failures occur and network congestion is less likely in the target network.

9.5.3 Summary

Most relevant options are offered by TCP and SCTP. Adjustment of options is generally possible through procfs, socket options, or kernel configuration. SCTP offers more detailed configuration on the socket level than TCP. Relevant options can be grouped into two categories: (1) delay and coalescence of small writes and (2) retransmission behavior.

Category (1) contains strategies for merging and delay of sender messages, through e.g., (Auto-)Corking and application of Nagle's algorithm. These increase the latency for small, isolated writes but reduce protocol overhead. Secondly, a receiver can delay and group his responses (Delayed Acknowledgement (DelACK)) to reduce protocol overhead for long unidirectional streams. Similarly, this strategy increases response latency for sparse incoming writes or at the end of an incoming stream. For BFT-SMR application, usage of coalescence behavior should be disabled if operation is optimized for low latency. DelACK bounds should be configured to match actual network timing, if possible.

In category (2), retransmission behavior can be configured through the RTO. The choice is subject to a tradeoff between response latency and premature timeouts. If End-to-End (E2E) replication latency $\ll$ RTO and an RTO occurs, a delay in order of multiples of replication latency is possible. Secondly, F-RTO tries to counter negative impacts on CC through spurious RTOs. A configurable response allows for avoiding a potentially unnecessary slow start behavior. For BFT-SMR application, F-RTO should be activated and the RTO should be configured to match the actual system timing as closely as possible.

9.6 Protocol Impact in BFT-SMR Context

To anticipate transport protocol impact in a BFT-SMR context we discuss logic and behavior that influences final E2E replication performance, execution dependencies, and conduct a brief case study of a HotStuff packet loss scenario.

9.6.1 Delay Elements

The original HotStuff paper evaluates E2E latency and throughput from the client perspective as central metrics. We adopt this approach. As outlined in §9.3, unmasked message loss between client and replicas (a violation of the common assumption of reliable transport between nodes; §5.1) may lead to safety and performance reduction in the HotStuff concept implementation by the original authors. Hence, we limit the discussion of network

Table 9.4: Selection of HotStuff Latency Delay Elements

Layer	Delay / Blocking conditions
SMR User Layer	Receipt of $\geq (f+1)$ matching `CMD-ANS` to commit
Consensus Layer	Receipt of current proposal in order to vote
	Receipt of enough votes to progress / propose
	Receipt of enough `CMD-REQ` to form proposal
	Pipelining: Delay of single QC affects multiple batches
Application Layer	Record completely delivered for processing
	(TLS only) r_i processed, before r_{i+1} considered
Transport Layer	(TCP / SCTP only) All segments/chunks present
	(TCP / SCTP only) ACK / RTO dynamics
Network Layer	All fragments present before delivery

stack changes to inter-replica links. Revisiting the basic HotStuff communication structure (Fig. 6.1), we see that E2E latency describes the time from sending a `CMD-REQ` to the receipt of $(f+1)$ matching replica `CMD-ANS` responses at the client. Thus, we measure the time for a single decision process on a (batch of) requested operations. This process includes more than three subsequent iterations of the *broadcast-vote* pattern.

Since the execution of the next phase, p_{n+1} depends on a correct majority in the current phase p_n, delays of p_n may directly carry over to p_{n+1}. This depends on the role of the sending node. Bottleneck nodes such as client and leader amplify the impact of message corruption, delay, or loss. If a single vote is lost, the incumbent leader might be able to drive progress with other votes. Hence, the effect may be limited to the same phase. However, if a proposal message is lost, the receiver replica might lag behind multiple phases, before catching up to the current state. This carryover not only affects the current (batch of) commands but also the pipeline state. Since a single QC can serve in different phases simultaneously in Chained HotStuff (§6), delay of a single QC decision may affect $b \cdot l$ operations for batch size b and l layers of pipelining. Finally, due to the nature of exchanged messages, transmitted data volumes vary significantly between message types and roles. While vote messages are smaller and constant in size, proposal messages scale with the batch size and contain additional metadata. While the sizes of `CMD-REQ` and `CMD-ANS` messages are freely configurable in the codebase, they may grow considerably for some use cases (e.g., complex transaction logic).

We provide an overview of elements with relevant impact on latency or the potential to stall execution in Tab. 9.4, summarizing preceeding analysis from this chapter. Generally, conditions from lower layers are required to process conditions of higher layers. Hence, successful replication requires successful completion of all layers below. We see, that the final E2E latency of a client request to a BFT-SMR system (SMR User Layer) is influenced by many factors. A BFT-SMR system can tolerate at most f parallel failures before client-visible delay is caused. However, these failures can occur at any layer and hence ensuring a performant and robust networking stack is paramount. Optimization approaches such as batching and pipelining increase the potential delay impact on a larger number of requests.

9.6.2 Case Study

For a more detailed view of expected transport protocol behavior, we discuss effects of a lost packet on different transport protocol setups during a HotStuff leader transmission.

TCP/TLS

First, we consider HotStuff, using TCP/TLS connections between replicas. Now assume that we lose a single packet, that is part of a `proposalMsg`, sent from the current leader to one of the replicas. On the network layer, no fragmentation occurs since our stack operates using TCP. The `proposalMsg` is already segmented to respect path MTU and endpoints do not spend time waiting on outstanding fragments. On the transport layer, TCP ensures that all segments eventually arrive at the replica. Since TCP offers a byte-stream abstraction, a lost packet stalls data delivery to upper layers until successful retransmission. Retransmission is either triggered by receipt of the third DupAck by the replica (fast-retransmit) or RTO expiry. If subsequent exchanges take place without (too much) loss, the leader continues sending, until it receives the third DupAck and then retransmits the missing data. If some of the subsequent packets are also lost, the sender might not receive enough DupAcks and hence waits for the RTO to expire. This is costly in terms of latency since the RTO can be a multiple of E2E consensus latency (cf. measured HotStuff latencies <40 ms [5]). The described behavior can be triggered if the sender stops its transmission, and enough of the remaining DupAcks from the receiver are lost. Reasons for the sending stop include unavailability of further application data to transmit or exhaustion of either congestion or receive window of the receiver.

On application layer, TLS imposes two central restrictions: (1) All segments of a record need to be available to begin processing and (2) record r_{i+1} is only considered once processing of record r_i is finished. Hence, even if segments $\notin r_i$ or other whole records were available, they would not be processed in parallel. On the consensus layer, a delayed `proposalMsg` prevents the replica from voting on the current and future proposals, until it has caught up. Delayed votes do not delay overall consensus, as long as the leader receives $(n - f)$ votes from other replicas in time. If the progress of a single view is disturbed, this directly impacts both (1) the current *batch* of requests and (2) the complete HotStuff decision pipeline. This pipeline stalling is addressed in newer BFT-SMR protocols [146] by allowing for commits, even if the required k QCs are not contiguous. From the client perspective, the added latency of all layers below affects the E2E latency, if at least one of the first $(f + 1)$ `CMD-ANS` messages received is affected.

SCTP/TLS

Second, we consider the same scenario, but instead using SCTP/TLS replica connections. As SCTP implements payload splitting, the network layer implications remain the same. On transport layer, SCTP offers reliable transmission of data chunks and configurable ordered delivery. However, even if enabled, SCTP behaves slightly differently in the face of a lost datagram. SCTP continues to transmit planned chunks, keeps track of gaps through frequent SACKs, and retransmits missing data with three miss indications. The RTO is reset for every received SACK, which acknowledges the next ACK-outstanding data chunk [170], but not for received out-of-order SACKs with gap specifications. Retransmission is either triggered by enough incoming miss indications or expiry of the RTO. Due to the higher SACK generation frequency in SCTP, the first case is more probable. However, if a sufficient number of SACKs is lost, sender RTO expiry is inevitable. The implications for higher levels remain the same as with TCP/TLS.

RUDP/DTLS

Finally, we consider the scenario using RUDP/DTLS connections between replicas. Since RUDP does not segment data on its own, IP fragmentation can occur. Endpoints need to wait until all fragments arrive. On transport layer, RUDP provides reliable service through retransmissions. In face of a lost datagram, the leader simply continues to transmit

outstanding data. Actual retransmission is triggered by expiry of RTO or explicit NACK in case of application-level errors. Since RUDP does not segment data, upon retransmission the whole `proposalMsg` is sent again. For large payloads, the probability of a lost fragment in the retransmission payload increases, which would require another RTO expiry for another retransmission trial. No ordering of datagrams is required or enforced, except through application logic requirements. On application layer, DTLS is used. While all fragments of a record need to be available for processing, DTLS records are processed independently. Hence, loss of a single record does not necessarily stall processing of other available records. Implications of higher layers remain the same as before.

9.6.3 Summary

Replication depends on successful operation of all below (network) layers. Typically BFT-SMR can tolerate at most f parallel failures on any level. Optimization techniques such as batching or pipelining increase delay impact. Detailed discussion of loss for different transport protocols demonstrates the relation between retransmission robustness and protocol overhead. More lightweight protocols (e.g., RUDP) might perform better in a loss-free environment but are less robust than protocols with increased overhead (e.g., SCTP).

10. Experiment Design and Setup

We design experiments to verify the theoretical analysis results (§9) and quantify the performance impact of network transport building block modification. In the following, we outline the rationale behind experiment design, the impact of input request saturation on performance, and describe the implementation used for measurements.

10.1 Rationale

We elaborate on general rationale behind our experiment design, outline possible and target deployment scenarios, and describe our chosen experiment environment.

10.1.1 General Design

The central goal of our experiments is to study the impact and optimization potential of network stack modification in context of BFT-SMR, with HotStuff as a representative, state-of-the-art system. To better isolate the impact of the underlying network stack, we assume all participating replicas to be honest. Hence, potentially disruptive behavior is solely limited to the underlying network. This allows for estimation of both best-case and bad-case performance and robustness impact.

Some BFT-SMR protocols are designed for asynchrony or partial synchrony (§2.2), with e.g., interleaving phases of synchrony and asynchrony, where no bounds or assumptions on delivery or processing times exist. However, in practice, the underlying network transport protocols nevertheless define heuristically established timing constants, such as RTO bounds and default values (§9.4). Except for e.g., CC logic and related bandwidth limitation effects, a sufficiently long delay exceeding the defined timeout causes retransmission behavior comparable to an actual omission. Hence, we choose packet loss as central interference factor for our experiments. In order to emphasize the effects of network transport protocol differences and approximate "worst-case" network behavior, we choose the packet loss to be uniformly distributed between all peers. This resembles an unpredictable network with increasingly bad QoS. Here, no simple heuristic actions can be taken to alleviate the disturbance, by e.g., rotating the leader role away from a node that is consistently underperforming. Overall, these choices allow us to approximate the impact spectrum of transport protocol and configuration choices.

10.1.2 Target Scenario

We established that better knowledge about the target network behavior improves optimization potential since protocol operation constants and timeouts can be chosen closer to the actual network timing values (§9.5). Consequently, we conjecture the biggest optimization potential for network transport modification to be located in the space between (1) nearly complete predictability and (2) compelling absence of predictability. An example of the first scenario is given by a BFT-SMR system, set up purely in a data center, e.g., for high-reliability operation of a service. The setup is likely permissioned and specialized hardware and QoS guarantees allow for ultra-low latency and high-bandwidth operation. In fact, availability of specialized hardware might render usage and optimization of a commodity network stack superfluous by employing techniques like RDMA instead (§12). An example of the latter scenario is a decentralized, geo-distributed BFT-SMR deployment, operated by independent individuals, e.g., a cryptocurrency with integrated smart contract platform. The setup could be both permissioned or permissionless, but geo-distribution renders the network connections unpredictable and increased variance of latency, bandwidth, and packet loss is expected. While a commodity network stack is used here, network unpredictability reduces potential optimization impact.

Hence, our target scenario is a hybrid between the extremes described above. We consider a permissioned, small to medium-size BFT-SMR setup, where a commodity network stack is used on top of a network with medium predictability. One example of such a deployment is a reliability-focused, hierarchically organized cloud computing platform with configurable replication level in independent BFT-SMR subnets, e.g., the Internet Computer [114, 191].

10.1.3 Environment

For the choice of our experiment environment, we reconsider our research objectives (§1.2). We aim for performance improvement of *practical* BFT-SMR systems. In order to estimate real-world performances, we hence avoid simulation or emulation techniques where possible and construct an experiment setup that approximates a realistic deployment as best as possible. This allows to evaluate the impact of a full-stack deployment, that does not omit or simplify certain behavior. Since practical systems are typically more complex and unpredictable than simulations, the experiment environment should simplify root-cause analysis and reproducibility of observed results.

Conforming to the discussed goals, we deploy an implementation of the HotStuff BFT-SMR library developed for practical use on a full commodity network stack. All participants are set up and executed on bare-metal hardware hosts, without virtualization layers. The target host machines are located in a local testbed with full access to all hard- and software. For reproducibility, all experiments are conducted in an automated process, that resets machines to a well-known and configured state in between two runs, by deploying all necessary software on ramdisks only. This automation and deployment process is enabled by the *pos* framework by Gallenmüller et al. [138].

10.2 Input Request Saturation

As outlined in §5.2, attribution of bottlenecks varies across different algorithms and setups. In any case, the applied system load, or broadly "*saturation*", influences performance and allows to inspect system behavior under varying strain. Many publications of BFT-SMR systems that include measurement results consider different saturation levels, albeit the specification detail of parameters and thresholds varies. Often, combined latency-throughput graphs are used to sketch system behavior across saturation levels [5, 6, 8, 37, 99, 100, 102]. Fine-grained control over input request load saturation supports our goal of detailed analysis of network stack impact on BFT-SMR.

We assume that replicas in a BFT-SMR setup have a maximum *decision frequency* λ_D, imposed by e.g., algorithm complexity, communication overhead, and hardware limitations. Client requests arrive with an *input frequency* λ_I at the replicas. Furthermore, we assume that client requests are combined into *batches* of size $b \in \mathbb{N}$, such that each successful replication decision covers b requests at once. Based on these assumptions we can construct a simplified model of resulting transaction throughput.

Let the effective client transaction processing frequency (*system throughput*) be denoted as λ_T. Then we can approximate

$$\lambda_T \sim min(\lambda_I, b \cdot \lambda_D) \tag{10.1}$$

Now, an upper bound on λ_T is either given by insufficient client request generation capacity ($\lambda_I < b \cdot \lambda_D$), by insufficient replication speed ($\lambda_I > b \cdot \lambda_D$), and/or unsuitable batch size b. We call a system *saturated* if $\lambda_I \sim b \cdot \lambda_D$. In other words, we call a system saturated if, for a given request input frequency λ_I and batch size b, the replication process operates close to or at its maximum decision frequency λ_D. We observe, that b allows for saturation adjustment if $\lambda_I \geq \lambda_D$. In HotStuff, `CMD-REQ` are batched by the incumbent leader. It waits until b requests arrive, then forms and broadcasts a proposal. Hence, if $\lambda_I < b \cdot \lambda_D$, decreasing b may reduce latency, if $\lambda_I > b \cdot \lambda_D$, increasing b may raise throughput.

10.3 Implementation

The HotStuff codebase by the original authors (§6.3) uses TCP/TLS connections by default. To support the set of selected transport protocols from §9.1, we extend the codebase by support for SCTP, UDP, and RUDP, including (D)TLS secure channels. Most changes were conducted in the `salticidae`[1] [148] P2P networking library, which abstracts basic P2P connection management and communication. The wrapped socket calls and low-level secure channel management are exchanged or extended to fit the target transport protocol, respectively. UDP and RUDP implementations are equivalent in salticidae and only differ by the implemented retransmission logic in HotStuff logic for RUDP.

The HotStuff logic is implemented in the `libhotstuff`[2] [147] library. Except for small bugfixes of the original implementation (e.g., erroneous replica ID generation based on TLS certificate hashes instead of peer network addresses), changes to libhotstuff were mainly necessary for implementing RUDP. Retransmission logic is implemented, using RTT approximations based on regular HotStuff message exchanges. The incumbent leader keeps an RTO for each replica. After, e.g., broadcasting a proposal, the leader tracks the expired time for each replica individually. In case the RTO expires, the leader retransmits its proposal, restarts the timer, and increases the current RTT estimate (backoff behavior). If a replica disagrees with a proposal, it responds with an explicit NACK message to avoid unnecessary retransmissions. If a correct vote arrives, the leader incorporates the measured time into its RTT estimation for that connection. RTO calculation is implemented based on the original TCP Smoothed RTT (SRTT) estimation formula [178]. RTT measurements of retransmitted datagrams are ignored, following Karn's algorithm [175]. To acquire an initial RTT estimate or refresh stale values independent of any HotStuff message exchange, dedicated ping messages can be exchanged on e.g., initial replica connection setup.

We build our automated experiment scripts on existing libhotstuff code and extend where necessary. The repository already provides example applications for client and replica nodes, as well as scripts for key generation and basic latency and throughput result aggregation. The client example application measures and collects E2E replication latency

[1] https://github.com/Determinant/salticidae
[2] https://github.com/hot-stuff/libhotstuff

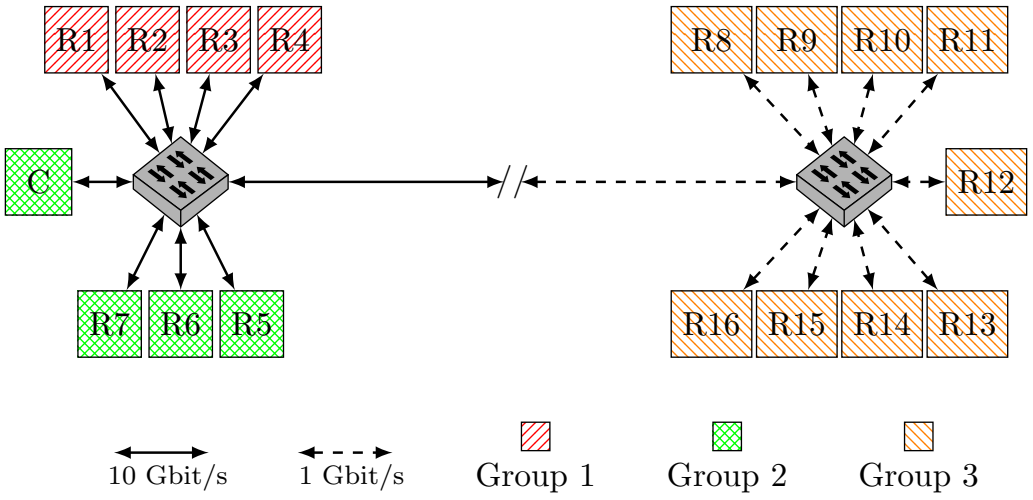


Figure 10.1: Testbed Hardware Topology

of requests. After termination, the client writes the latency values to a logfile which is interpreted by the aggregation script. All software is built and run on Ubuntu Jammy, using Linux Kernel v5.15.0-72 and OpenSSL 1.1.1f-1ubuntu2_amd64.deb.

Experiments are run on up to 17 machines that are connected via a series of switches. An overview of the topology is given in Fig. 10.1. A practical, small to medium-sized, distributed BFT-SMR setup (§10.1.2) most likely features heterogeneous hardware. Hence, we employ machines of three different specification types, here called *Groups*. In detail, we use four machines from Group 1 (Intel Xeon Gold 6312U, 24×2.4 GHz, 512 GB RAM), four machines from Group 2 (AMD EPYC 7543, 32×3.8 GHz, 512 GB RAM), and nine machines from Group 3 (Intel Xeon D-1518, 4×2.2 GHz, 32 GB RAM). Both Groups 1 and 2 are connected via Intel E810-XXV Network Interface Cards (NICs), Group 3 via Intel I350 NICs. Final system performance can potentially be limited by the weakest hardware of a heterogeneous setup. We avoid a potential negative impact of HotStuff bottleneck nodes (§9.3) by executing client and leader roles always on the most powerful hardware. These nodes are connected via 10 Gbit/s links to prevent potential bandwidth bottlenecks.

Base RTT between any two nodes is consistently below 350 µs and the default MTU for all links is 1500 B. We measured base TCP throughput between hosts using iperf3. Results are symmetric in terms of traffic direction and consistent for each link type in Fig. 10.1. Available throughput amounts to up to ∼9.41 Gbit/s within and in between Groups 1 and 2, and up to 941 Mbit/s within Group 3 and in between Group 3 and any other group.

11. Evaluation

We evaluate the impact of network transport modification in a series of experiments. The logical BFT-SMR node configuration is provided in Fig. 11.1. Experiments are executed using a single client and a single leader for replica numbers of $n = 3f+1, f \in [1, 5], n \in [4, 16]$, to better isolate their potential role influences. We refer to replica $\leftrightarrow$ replica connections as *variable* connections. Variable connections are subject to modification of transport protocol, secure channel usage, and configuration parameters. Communication on client $\leftrightarrow$ replica connections is always conducted over plain TCP to prevent potential liveness and safety issues (§9.3.2). Like in the original HotStuff codebase, these connections are also not protected by secure channels. In the following, we detail the measurement setup (§11.1) and evaluate our implementation in three categories: Impact of input request saturation (§11.2), impact of transport protocol choice with and without secure channel (§11.3), and impact of transport protocol choice and configuration under loss (§11.4).

11.1 Measurement Setup

For each of the target evaluation categories, we vary a set of parameters, including BFT-SMR parameters typically studied in literature (§12). We briefly summarize all parameters, their notation, and immediate effect in Tab. 11.1. Detailed introduction of relevant parameters is conducted in the respective subsection, if not done already. Exhaustive variation of all parameter combinations is time-consuming and not necessary for certain insights. We aim to cover a relevant part of the parameter space but keep visualization concise.

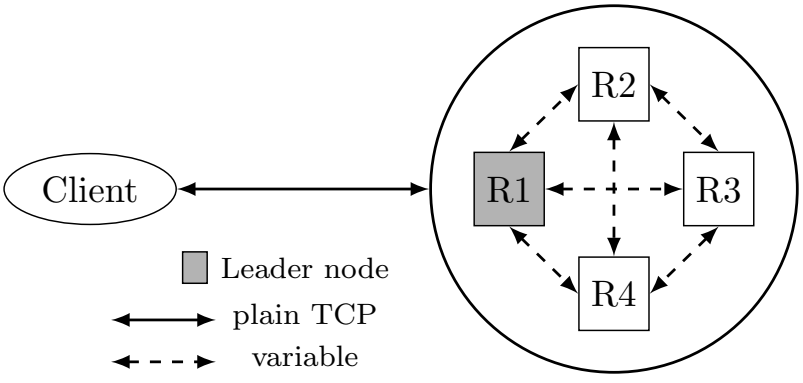


Figure 11.1: Logical BFT-SMR Role Setup, $n = 4$

Table 11.1: Overview of Variable Experiment Parameters

Parameter	Symbol	Effect
Replica Number	n	Number of participating replicas, incl. leader
Batch Size	b	Number of `CMD-REQ` considered in one proposal
Payload Size	p	Size of `CMD-REQ`/`CMD-ANS` messages
Protocol	e.g., UDP	Transport protocol used on variable connections
Secure Channel	e.g., (D)TLS	Secure channel used on variable connections, or not
Loss Probability	l	Uniform loss probability on any variable connection
RTO Profile	e.g., S100	RTO bounds profile used on variable connections

Therefore, we generally fix all but one variable parameter to a baseline default value. For certain scenarios and to emphasize edge cases or specific behavior, we deviate from this rule and choose different default values. BFT-SMR literature predominantly measures (1) E2E client replication latency and (2) transaction throughput given as Transactions per Second (TPS) in practical evaluations [5, 6, 8, 37, 99, 100, 102]. We adopt this approach.

A single experiment is run for 60 s at maximum possible replication speed, i.e., as limited by either client or replicas (§10.2). In experiments *without* loss, latency results are filtered for outliers, such that values outside $[Q_1 - 1.5 \cdot IQR, Q_3 + 1.5 \cdot IQR]$ are removed, where Q_1, Q_3, and IQR refer to the first quartile, third quartile, and interquartile range, respectively. In experiments *with* loss, outliers are *not* filtered to prevent removal of latency spikes to explain performance or robustness reduction, as well as specific impact of parameter modification. For brevity and as done in the example post-processing script of the HotStuff PoC implementation, result data is generally averaged into a single data point. Where an average is insufficient to explain the results, we provide additional details on demand.

11.2 Input Request Saturation

The measured performance of a BFT-SMR system depends on its input request saturation (§11.2). We measure System under Test (SuT) behavior for varying levels of input load to calibrate the setup for following experiments and simplify interpretation of later results. Running a varying number of client applications on our dedicated client machine in parallel allows us to adjust the input load. We expect a setup with execution of clients on different machines to yield comparable results, if available network QoS, bandwidth, and processing power are proportionate. A single client application does not parallelize request sending. Bandwidth limits were not reached for any experiment.

Fig. 11.2 shows *average* latency (left) and *cumulative* throughput (right) over all clients in relation to batch size and number of client applications, for a baseline configuration of TCP, noTLS, $p = 0\,\mathrm{B}$, and $n = 4$. We observe from the throughput values, that a single client suffices to saturate our SuT for $b = 25$ up to nearly 50. Above, an increase in batch size does not yield increased throughput. Hence, client command generation rate becomes the bottleneck. Increasing the batch size above this current saturation point causes a proportional latency increase since the leader waits longer for incoming commands to accumulate before it can create the next proposal. In Fig. 11.3, we show average latency (top) and cumulative throughput (bottom) over all clients, in relation to payload size and number of client applications, for a baseline configuration with TCP, noTLS, and $n = 4$. From throughput, we observe that only a four-client setup is able to saturate our SuT for larger payloads, up to a batch size of 100. For fewer clients and/or larger batch sizes a payload increase reduces TPS and increases latency since the clients are unable to request enough transactions of the specified size and rate. The described trends are amplified in setups with more replicas since the clients send and receive data from more replicas

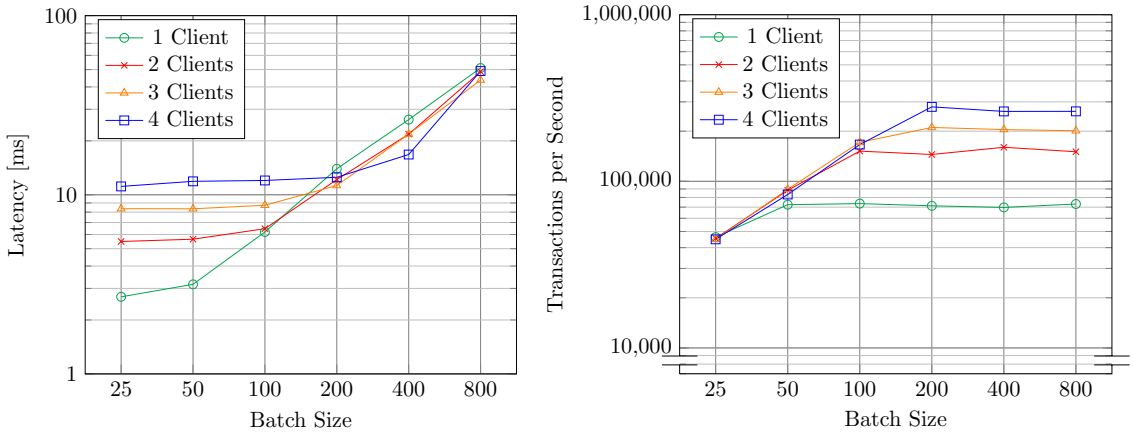


Figure 11.2: Saturation over Batch Size, TCP, noTLS, $p = 0\,\mathrm{B}, n = 4$

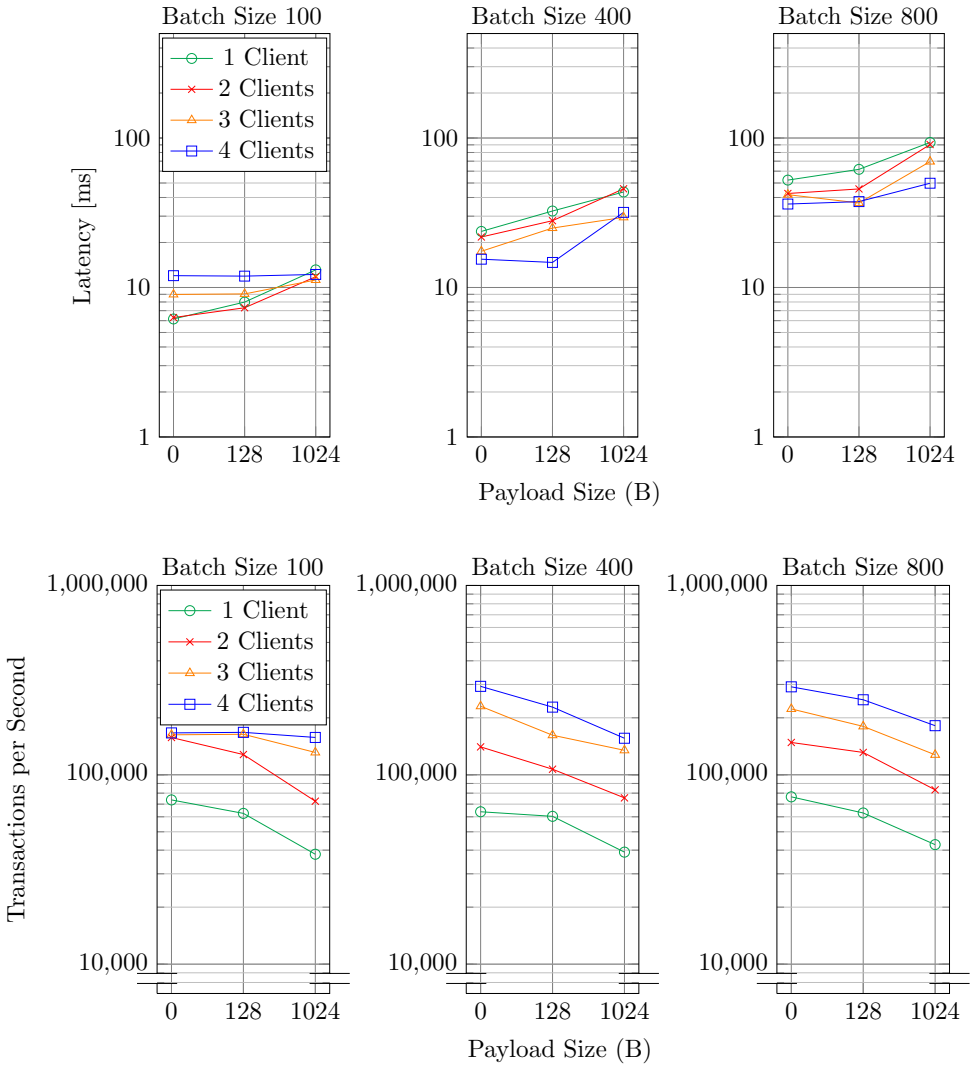
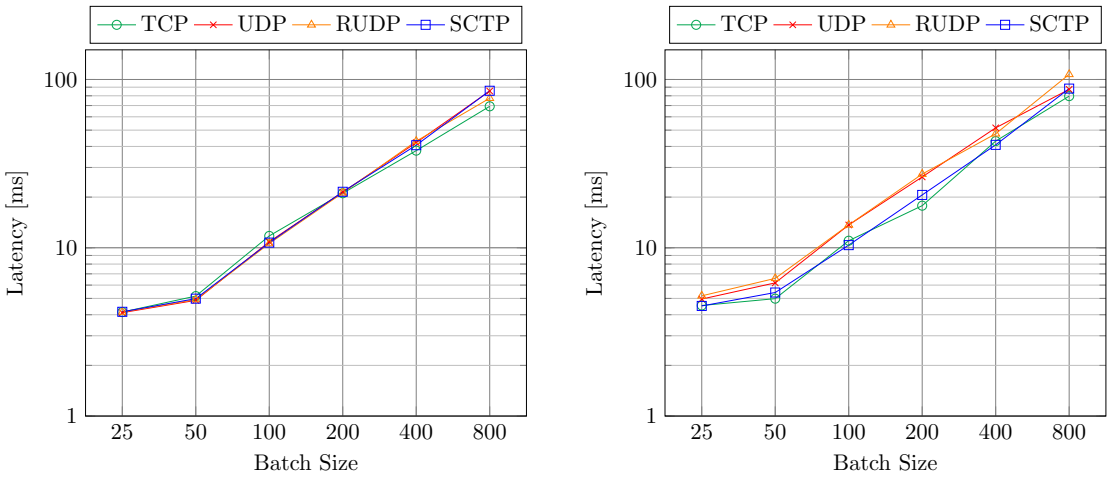


Figure 11.3: Saturation over Payload Size, TCP, noTLS, $n = 4$



(a) without (D)TLS (b) with (D)TLS

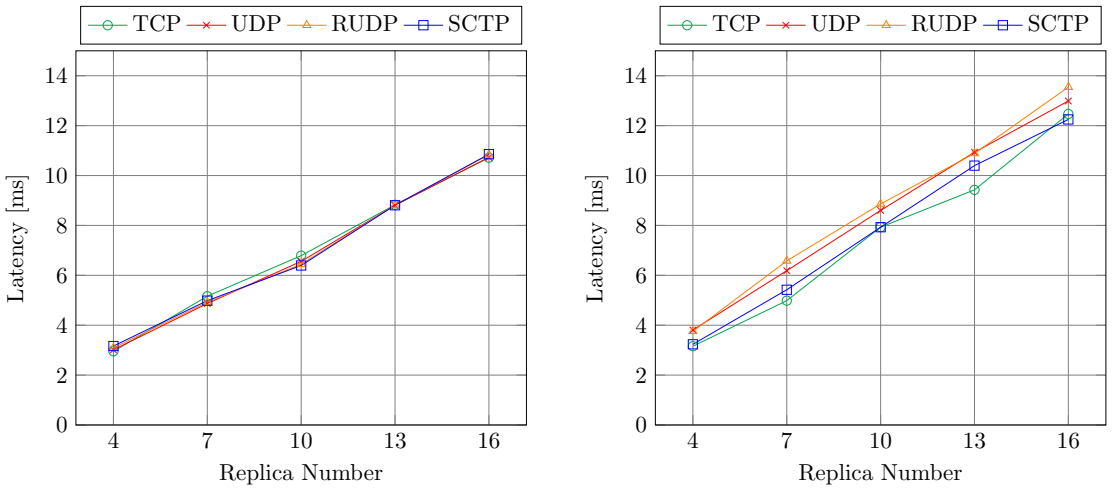
Figure 11.4: Transport Prot. Latency over Batch Size, w/(o) (D)TLS, $p = 0\,\mathrm{B}, n = 7$

for each command (batch). To prevent raw data transmission from becoming the system bottleneck, we choose a payload size suitably small for the following experiments. A more detailed discussion of payload size impact is conducted in Part III of this thesis.

11.3 Transport Protocol and Secure Channel

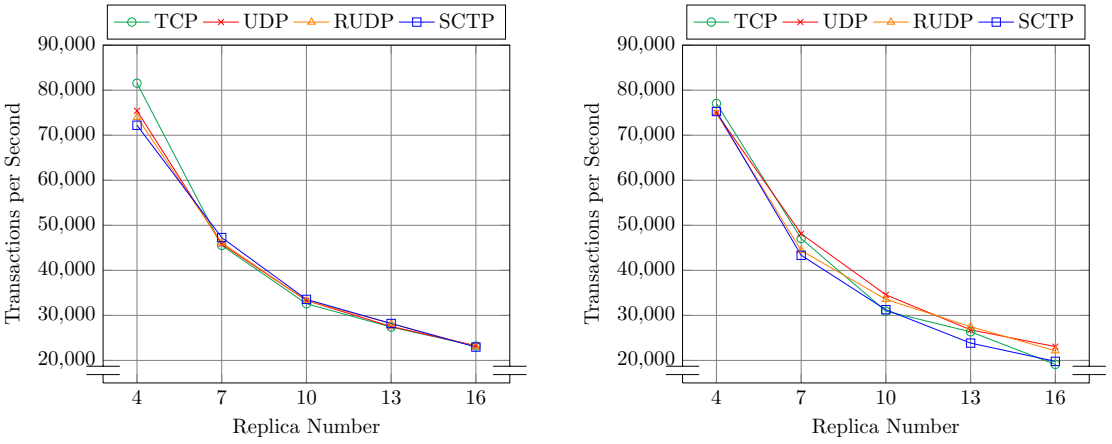
Next, we evaluate the impact of transport protocol choice and secure channel usage. These results function as a best-case operation estimate and provide a baseline for evaluation of experiments with packet loss. Fig. 11.4 shows a performance comparison of all selected transport protocols with and without active (D)TLS over different batch sizes. We fix the remaining parameters to $n = 7$ replicas, and a payload size of 0 B. We note that above a batch size of 50, the client machine becomes the bottleneck. Input request saturation is no longer given and replication latency increases significantly. Between batch sizes 25 to 50 the replicas are saturated. Latency still increases slightly since more command hashes are included in proposals and processed by the replicas. Without active (D)TLS, the latency differences between transport protocols for this setting are insignificant. Replication latency revolves around 5 ms for all protocols and $b = 50$. With active secure channel, for DTLS-based setups (UDP, RUDP) latency is consistently increased in comparison to the TLS-based setups (TCP, SCTP). Comparing TCP and RUDP, for $b \leq 50$, we observe a latency increase between 14% to 32%, for larger batches between 10% up to 54%. The increased latency can be attributed to larger DTLS protocol overhead on all communication steps for a full HotStuff replication process. Generic segmentation offloads are activated for all protocols, specific offloads are available and in effect for all protocols except SCTP.

Now let us discuss basic scaling behavior. Fig. 11.5 compares protocol performance with (de-)activated secure channel for different replica numbers. The setup is fixed to a payload size of 0 B, and a batch size of 50. The results are similar to the batch size variation in Fig. 11.4. Inter-protocol differences without (D)TLS are not significant, while DTLS-based setups consistently show larger latencies in comparison to TLS-based setups. (D)TLS usage generally incurs a small overhead. For example, activation of TLS increased TCP latency between 6% and 16% ($n = 16$), generally growing with increasing replica numbers. We attribute this to the cumulative processing overhead of more TLS connections on the bottleneck leader node. More replicas result in larger latencies in general, due to increased communication and processing overhead in between replicas, and client (Fig. 6.1). We



(a) without (D)TLS (b) with (D)TLS

Figure 11.5: Transport Prot. Latency over Replicas, w/(o) (D)TLS, $p = 0\,\mathrm{B}, b = 50$



(a) without (D)TLS (b) with (D)TLS

Figure 11.6: Transport Prot. Throughput over Replicas, w/(o) (D)TLS, $p = 0\,\mathrm{B}, b = 50$

recall that the leader can progress as soon as $h(f) = (n - f) = 3f + 1 - f = 2f + 1$ correct votes arrive. For example, the measured multiplicative latency growth for an increase in replicas using TCP without TLS ($\sim 1.74, 1.31, 1.29, 1.21, ...$), follows the expected series of ratios between the required honest nodes for two successive numbers of faulty replicas

$$\frac{h(f+1)}{h(f)} = \frac{2(f+1)+1}{2f+1}, f \in [1, 2, ...] \rightarrow (\sim 1.66, 1.4, 1.28, 1.22, ...) \tag{11.1}$$

with some variance for smaller n. In other words, the observed latency increase is dominated by the additional overhead from the minimum number of honest participants $h(f+1)$ to tolerate the increased number of faulty nodes $f + 1$.

We observe a similar behavior for the achieved throughput. Fig. 11.6 shows the corresponding throughput to the latency data in Fig. 11.5, plotted over increasing replica numbers. We fix $p = 0\,\mathrm{B}$ and $b = 50$. Without active secure channel, protocol differences are not significant. We observe variance at $n = 4$ where TCP outperforms all other protocols. With active secure channel, we observe slightly increased throughput for protocols using DTLS. Throughput scaling behavior follows the same rationale as for latency. Here, the

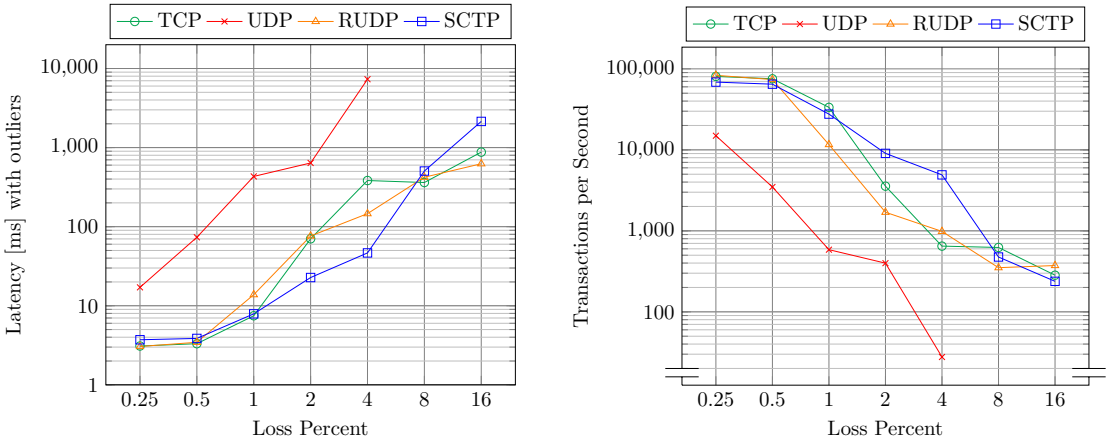


Figure 11.7: Transport Prot. Performance over Loss, noTLS, $p = 0\,\mathrm{B}, n = 7, b = 50$

measured multiplicative throughput decline for an increase in replicas using TCP without TLS ($\sim 0.55, 0.71, 0.84, 0.85, \ldots$) generally follows the inverse ratio from Eq. (11.1)

$$\frac{h(f)}{h(f+1)} = \frac{2f+1}{2(f+1)+1}, f \in [1, 2, ...] \rightarrow (\sim 0.60, 0.71, 0.78, 0.82, ...) \tag{11.2}$$

with some variance. As formally outlined, the throughput reduction for tolerating additional faults is substantial and larger for smaller replica numbers.

11.4 Impact of Loss

Packet loss is expected behavior in practical BFT-SMR systems, caused by e.g., underlying infrastructure or byzantine behavior. We study the practical performance impact of different transport protocols and RTO configurations in lossy scenarios. As outlined in §10 we assume uniformly distributed loss probabilities in the network. We apply a range of uniform loss probabilities $l = 0.0025 \cdot 2^k, k \in [0, 6]$ to all links of our setup (Fig. 11.1). Each occurring packet loss is independent. The loss behavior is implemented using traffic control netem [192], which drops a percentage of outgoing packets before they are queued [193]. For the upcoming evaluation, we recall that latency (and TPS) values are the result of processing, spread over multiple nodes, and involve more than four message roundtrips (Fig. 6.1). Hence, loss always induces variance, growing with larger loss probability. We recall, that all loss results *include* outliers to prevent discarding loss-related spikes.

11.4.1 Transport Protocol

We begin with an analysis of different transport protocols under loss. Fig. 11.7 shows latency (left) and throughput (right) of different transport protocols under increasing loss probability. We fix the parameters to a payload size of 0 B, seven replicas, and a batch size of 50 to ensure input request saturation. Since UDP is lacking retransmission logic, we observe significantly worse results, or even not a single completed quorum at all (8%, 16%). Except for UDP, we observe similar performance for all protocols up to and including 1% loss. In this range, SCTP generally performs worse than TCP due to larger protocol overhead (§9.2.2, §9.4), less hardware offloading support, and larger minimum RTO (§11.4.2). RUDP performs worse than TCP, due to its naive retransmission implementation, whereupon loss of a single IP fragment the whole payload must be retransmitted (§9.4). Hence, increasing loss probability also increases the probability that at least one IP fragment is lost and the complete payload must be retransmitted.

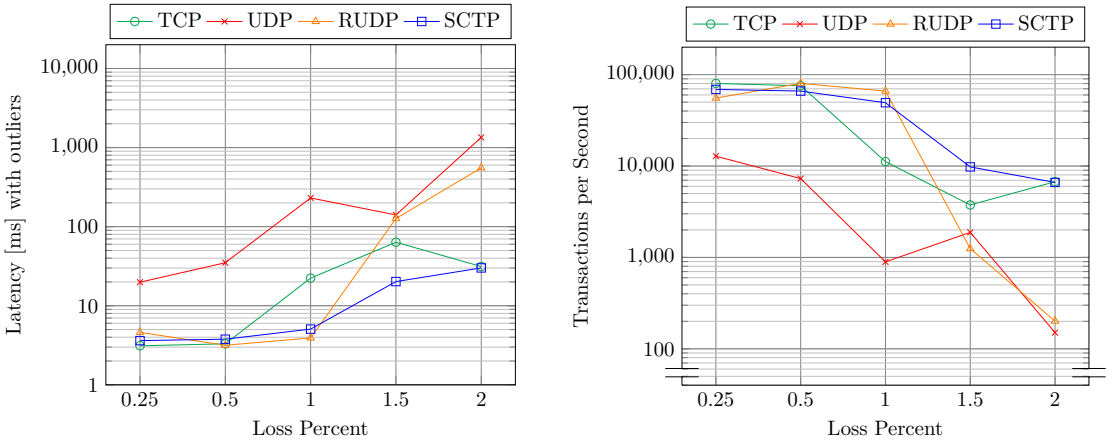


Figure 11.8: Transport Prot. Performance over Loss, noTLS, $p = 0\,\mathrm{B}, n = 4, b = 50$

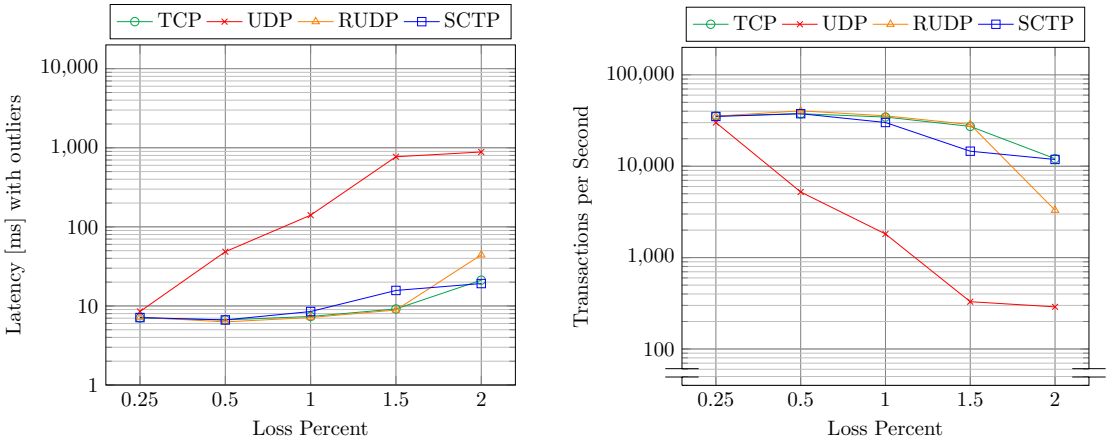


Figure 11.9: Transport Prot. Performance over Loss, noTLS, $p = 0\,\mathrm{B}, n = 10, b = 50$

For loss equal to and above 1%, result variance and the number of outliers increase fast and substantially, with the IQR of latency results spanning e.g., more than one order of magnitude for TCP with 4% and 8% loss. Hence, the average results in these loss ranges may vary upon remeasurement. For that specific measurement, we attribute the visible SCTP outperforming TCP for 2% and 4% loss to a combination of (1) TCP and SCTP retransmission differences (SCTP continues to transmit planned chunks and remembers gaps with explicit SACKs; §9.4) and (2) inconveniently occurring losses of TCP packets and respective DupAcks, resulting in a series of high-latency HotStuff leader progress stalls. Throughput values follow the latency analysis. Overall, in the more stable loss range $< 1\%$, TCP outperforms other reliable protocols (e.g., SCTP by $\sim 17\%$ for 0.25% loss). In general, the performance of the system begins to degrade substantially above 0.5% loss and is already slowed by roughly an order of magnitude for 2% loss.

An increased number of participating replicas has an additional stabilizing effect on performance and robustness under loss (§9.3.2), but also reduces the observed differences between varying transport protocols. A SuT with more replicas can tolerate more faults, including omission faults. Hence, for some fixed loss probability, in the larger system, more losses need to occur at the same time at the currently voting replicas to prevent progress, which is less likely. Consider for example Fig. 11.8 and Fig. 11.9. Here, we show the loss performance of all transport protocols for more granular loss probabilities around 1% but for $n = 4$ and $n = 10$ respectively. For $n = 4$ we observe slightly larger transport protocol differences for $l < 1\%$, but significantly increased variance. RUDP generally

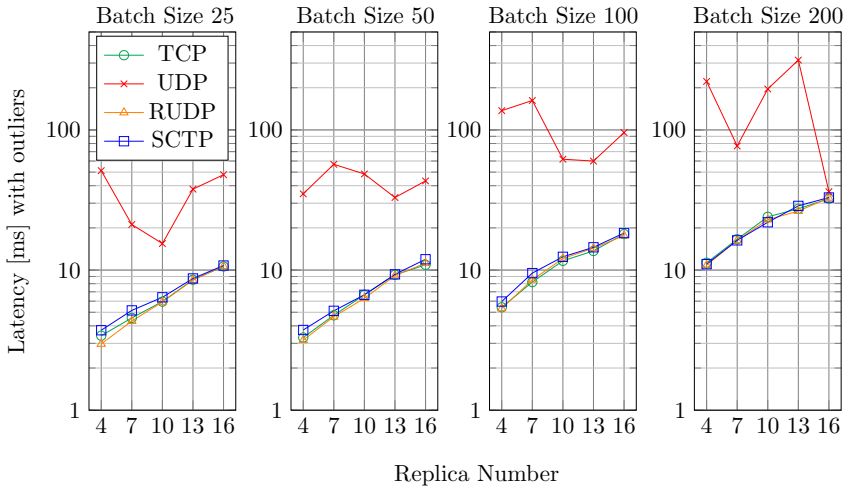


Figure 11.10: Transport Prot. Latency over Replicas, noTLS, $l = 0.5\%, p = 0\,\mathrm{B}$

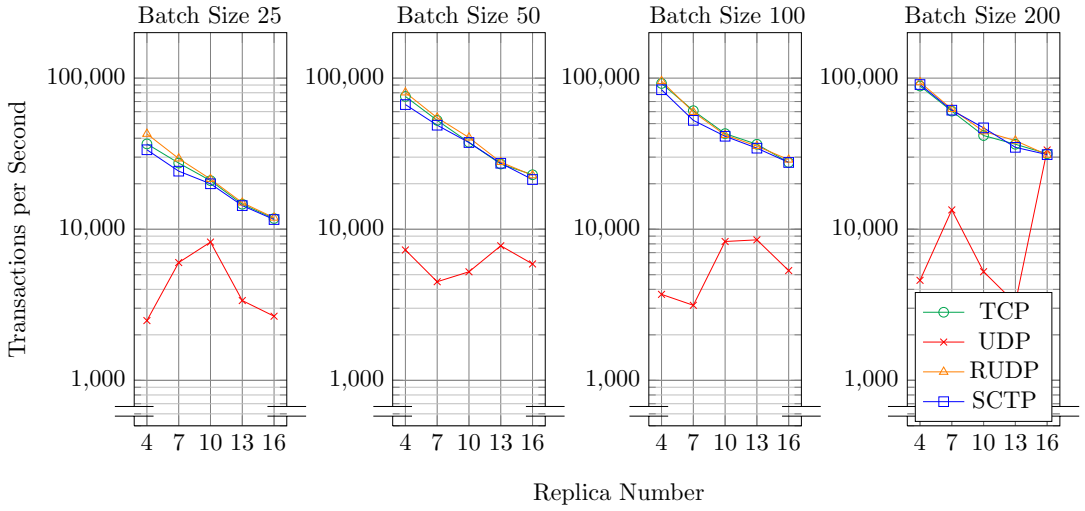


Figure 11.11: Transport Prot. Throughput over Replicas, noTLS, $l = 0.5\%, p = 0\,\mathrm{B}$

performs better for smaller loss values $l \leq 1$ due to smaller header overhead and RTO configuration closer to actual network times but shows an outlier for $l = 0.25\%$. In this concrete measurement, a cascade of packet losses at inconvenient places caused a leader progress stall and large HotStuff base timeouts ($> 10\,\mathrm{s}$) to take effect. This event skewed the average toward a higher value. For $n = 10$ the opposite is the case, both protocol differences and variance are reduced. Even the performance of the unreliable UDP variant is not significantly reduced for $l = 0.25\%$.

Next, we vary the number of replicas and batch size. Fig. 11.10 shows the latency and Fig. 11.11 shows the throughput of different transport protocols under increasing replica numbers, grouped by different batch sizes. The setup is fixed to a payload size of 0 B, and 0.5% loss. Except for the high-variance latency of the non-reliable UDP transport, we observe the two described core trends for scaling behavior in this setup again. First, increasing n result in lower performance and reduce performance differences between transport protocols. For small n RUDP outperforms SCTP and TCP by up to $\sim 20\%$ and $\sim 13\%$ in latency and by up to $\sim 27\%$ and $\sim 16\%$ in throughput, due to smaller protocol overhead and more fitting RTO. Second, increasing b reduces latency differences between transport protocols. We see the already discussed (Fig. 11.4) general latency increase for setups without input request saturation ($b \gg 50$), shadowing smaller latency differences.

Table 11.2: SCTP RTO Configuration Profiles, Values in ms

Mode	Init	Min	Max	ACKdelay
Default	3,000	1,000	60,000	200
TCPmock	200	200	120,000	200
S100	100	100	100	100
S50	50	50	50	50

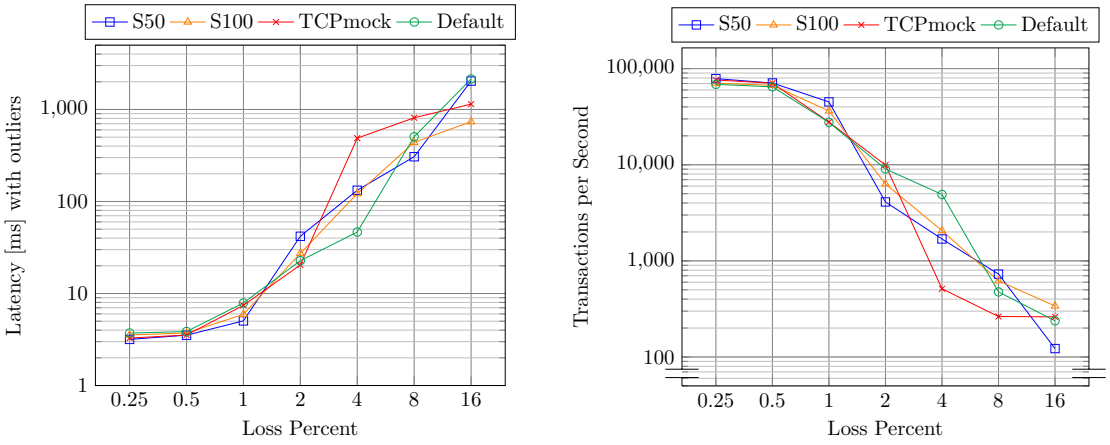


Figure 11.12: RTO Profile Performance over Loss, $p = 0\,\text{B}, n = 7, b = 50$

We furthermore note that throughput is slightly smaller for $b = 50$ than for $b = 100$. Since lost packets and retransmission reduce the effective replica decision frequency λ_T, the saturation point has shifted. Hence, the client bottlenecks only for even larger batch sizes and protocol difference reduction is stronger for larger batch sizes (e.g., $b = 200$).

11.4.2 RTO Configuration

Packet loss does not only affect regular payloads, but also ACKs or retransmission requests issued by the transport protocol. If a sufficient amount of this meta data is lost, an RTO is triggered. Both in the analysis (§9.4) and the evaluation of previous results (§11.4.1), the impact of RTOs was anticipated. RFC 6298 [175] argues that large (minimum) RTO values are necessary for conservative operation and to avoid spurious retransmissions. However, the authors also suggest that modified (i.e., smaller) values might be superior or more suited in certain scenarios. Hence, we now discuss concrete configuration values and resulting measured behavior to analyze the impact of RTO modification in context of BFT-SMR in detail. RTO behavior can be steered through configurable bounds for minimal and maximal values. We choose SCTP as transport protocol for these experiments, since SCTP RTO configuration values can be comfortably modified on socket level.

Tab. 11.2 specifies four configuration profiles, for initial (*Init*), minimum (*Min*), and maximum (*Max*) RTO values, and DelACK time (§8.1). All values are given in ms. *Default* describes the SCTP default, *TCPmock* emulates the default of our Linux TCP stack, and *S100* and *S50* apply a flat, *static* configuration of 100 ms and 50 ms respectively.

Loss Variation

We first discuss the impact of loss probability on different RTO profiles. Fig. 11.12 shows latency (left) and throughput (right) of different configuration profiles under increasing loss values. The setup is fixed to a payload size of 0 B, seven replicas, and a batch size of 50 to

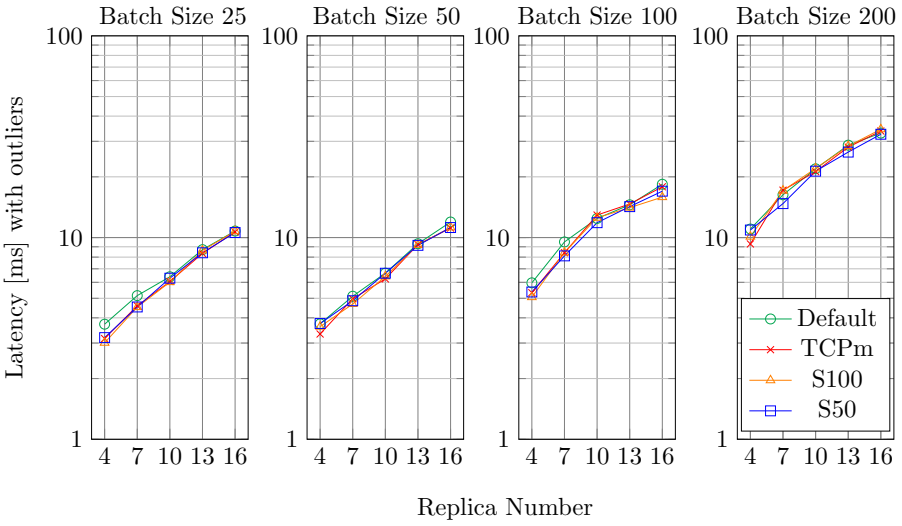


Figure 11.13: RTO Profile Latency over Replicas, noTLS, $l = 0.5\%, p = 0\,\mathrm{B}$

ensure initial input request saturation. We observe that in the more "stable" loss region ($l \leq 1\%$), smaller RTOs result in better performance. The S50 profile consistently performs best, with e.g., up to $\sim 35\%$ less latency than the default configuration for $l = 1\%$. This performance improvement can be attributed to the RTO values of the S50 profile being significantly closer to actual network RTTs than the Default profile. Hence, in case a RTO actually occurs, the scheduled waiting time is reduced. For loss values $> 1\%$, the variance of latency values increases drastically, with IQR ranges partially spanning more than one order of magnitude (e.g., S100, 4% loss). Hence, the plotted average values are subject to so much noise, that the effect of the modified RTO is no longer visible. The large latency variations are caused by occurrence (or non-occurrence) of inconveniently placed packet losses that trigger large HotStuff implementation default timeouts, skewing the average results. For example, during the measurement of S50 with 4% loss, the transport protocol failed to successfully transmit and retransmit multiple relevant HotStuff messages in time. A leader proposal was not correctly transmitted to $f + 1 = 3$ replicas, causing the replicas to start a block fetching routine to acquire the missing block from other replicas. The fetch routines were again subject to critical packet loss. Finally, this loss then triggered both a leader rotation as well as HotStuff code default timeouts (fetch timeout, leader progress timeout, ...). For S50 this occurred three times during the run, while a comparable event only took place once during the run for the Default configuration. Hence, interpretation of measurement results from high loss operation needs to be approached with care, due to the large amount of possible delay causes and variance.

BFT-SMR Parameter Variation

Second, we discuss the impact of varying batch size as well as the number of replicas for different RTO profiles. Fig. 11.13 shows the latency of different RTO configurations under increasing replica numbers, grouped into panels of different batch sizes. The setup is fixed to a payload size of 0 B, and 0.5% loss. Fig. 11.14 displays the respective throughput data for the same experiment. We observe three core trends in these two figures: The impact of (1) batch size increase, (2) replica number increase, and (3) RTO profile variation.

Let us address the batch size increase (1) first. While we established the baseline input saturation point for a single client, no loss, $p = 0, n = 4$, to be approximately $b = 50$ (§11.2), we observe that an increased batch size still yields throughput benefits up to $\sim b = 100$ in this scenario. We recall our simple throughput model (§10.2), where $\lambda_T \sim min(\lambda_I, b \cdot \lambda_D)$.

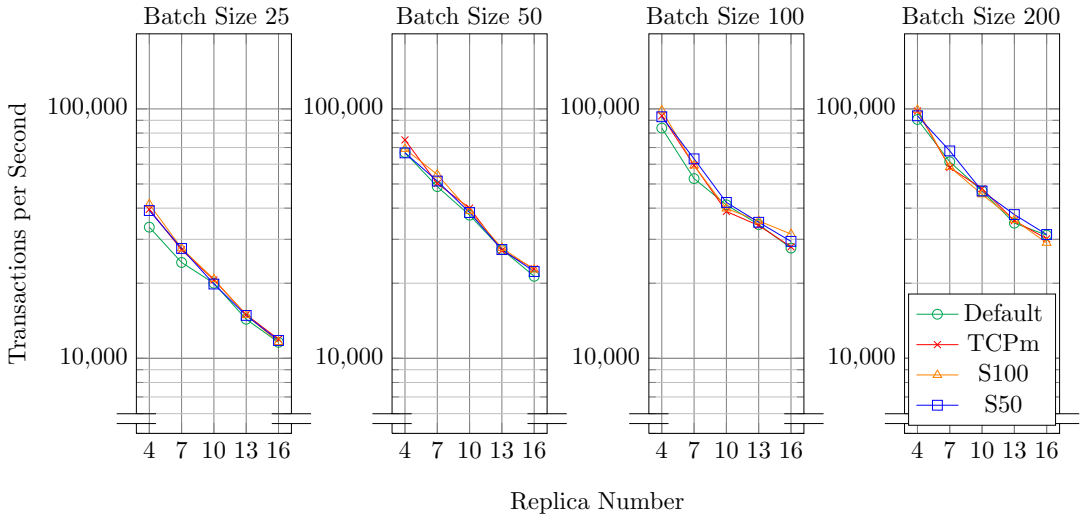


Figure 11.14: RTO Profile Throughput over Replicas, noTLS, $l = 0.5\%, p = 0\,\mathrm{B}$

Loss on inter-replica links further reduces the effective decision frequency λ_D relative to the input frequency λ_I. Hence, throughput λ_T can be increased by choosing larger batch sizes, and client load generation becomes a bottleneck only for larger batch sizes. The latter ensures that for our choice of batch size, the replica operation is still the bottleneck and we are not erroneously benchmarking client load generation.

For replica number increase (2), we observe similar effects to the scaling measurements for transport protocols under loss (e.g., Fig. 11.10, Fig. 11.11). More replicas reduce decision frequency through overhead (§11.3), shift the bottleneck toward the client and hence reduce differences between RTO profiles, and somewhat increase robustness against loss-based performance reduction since more concurrent faults can be tolerated (§11.4.1).

Finally, RTO profile variation (3) results in slightly improved performance for lower bounds, e.g., up to $\sim 15\%$ less latency for S50 in compared to the Default profile for $n = 4, b = 25$. However, the relationship between smaller RTO and better performance is not as clear for this $l = 0.5\%$ setting. While the modified profiles tend to outperform the Default profile (e.g., for $b = 100$), the differences between the modified profiles are smaller than for $l = 1\%$. A general reduction in loss probability decreases the expected number of RTOs, and therefore the potential performance improvement through modified RTO bounds.

To emphasize the observed trends, we study the same parameter variation, but fix the loss probability to $l = 1\%$ instead. Fig. 11.15 shows the latency of different RTO configurations under increasing replica numbers, grouped into panels of different batch sizes. The setup is fixed to a payload size of 0 B. Fig. 11.16 displays the respective throughput data for the same experiment. We re-identify all three core trends again.

Larger loss probability shifts the saturation points to larger values (1), a larger replica number decreases performance and potentially reduces differences between RTO profiles (2), and a tendency for increased performance for smaller RTO value configuration (3), e.g., S50 has $\sim 19\%$ less latency than the Default profile, for $p = 0, n = 7, b = 25, l = 1\%$). However, due to the larger loss probability differences emerge. First, result variance increases significantly and large outliers become visible (e.g., Default profile, $b = 25, n = 16$). While this partially obscures the small differences in between modified profiles, the general relationship between reduced latency and reduced RTO values is more pronounced for this $l = 1\%$ setup. We observe this relationship roughly until $b = 100$, while for $b = 200$ the bottleneck is shifted to the client and differences between profiles are reduced.

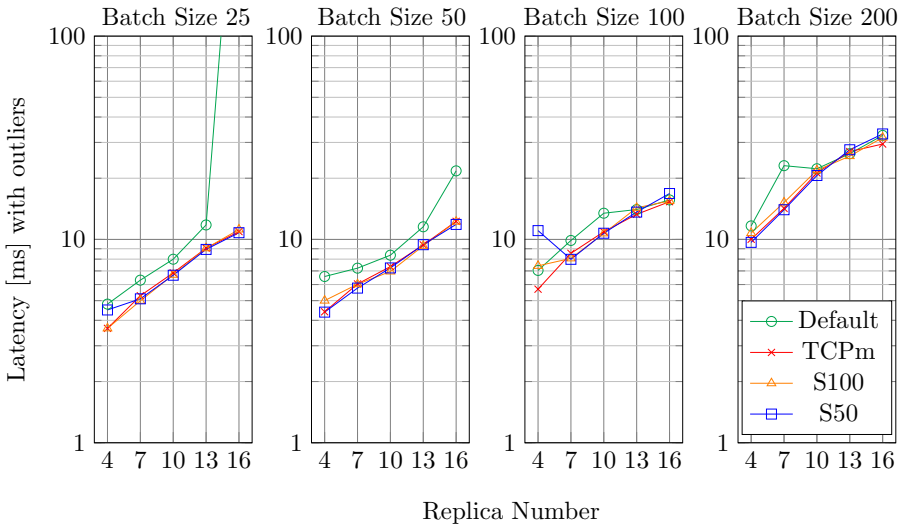


Figure 11.15: RTO Profile Latency over Replicas, noTLS, $l = 1\%, p = 0\,\mathrm{B}$

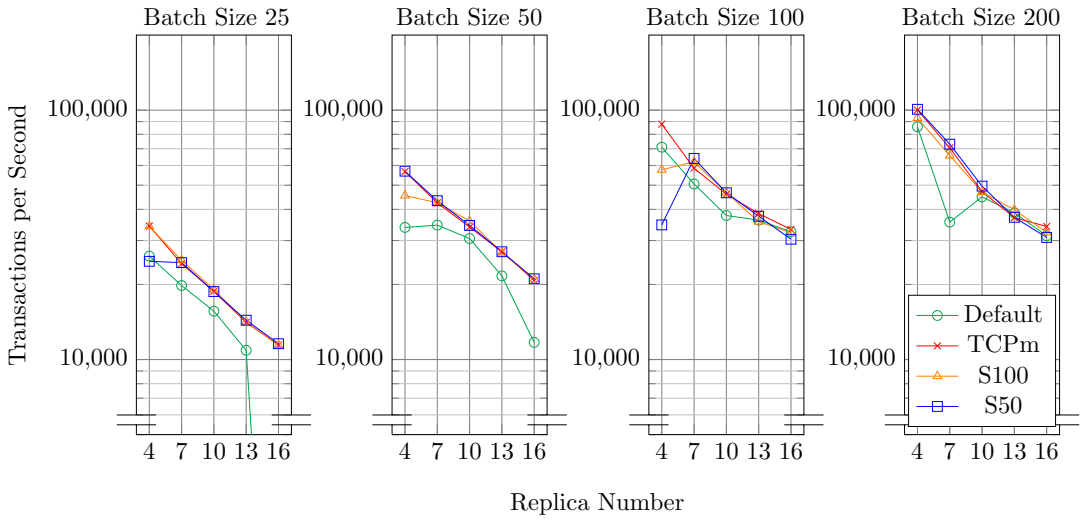
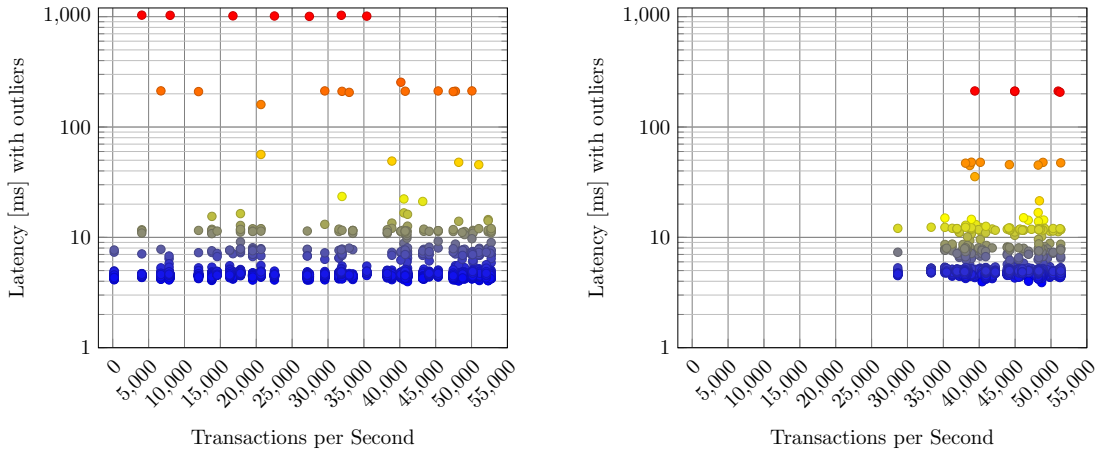


Figure 11.16: RTO Profile Throughput over Replicas, noTLS, $l = 1\%, p = 0\,\mathrm{B}$

Effect of Minimum RTO Bound

Finally, we demonstrate that the observed performance differences between RTO profiles are driven by the *minimum* RTO value bound. Fig. 11.17 shows scatter plots of E2E latencies of client requests (y-axis), sorted by the average throughput (op/s, one-second granularity) that was achieved for the request (x-axis). Subfig. 11.17a holds the data for the Default RTO profile, Subfig. 11.17b for the S50 profile. The dot color visualizes latency, where blue signifies the lowest and red the highest latency values of the dataset. The measurement was conducted with $n = 7$, 1% loss, a payload of 0 B, and a batch size of 50. We see that the Default profile results in more requests with lower op/s and clustered plateaus at latencies of $(1000 \cdot k + \Lambda)$ ms and $(200 \cdot k + \Lambda)$ ms. Here, k denotes the number of (independent) RTOs that occurred on a progress-relevant connection during processing of a single client request (batch), and Λ denotes the common processing latency without loss. In Fig. 11.17, only plateaus for $k = 1$ are visible. For profile, S50 we instead observe clusters around $(200 \cdot k + \Lambda)$ ms and $(50 \cdot k + \Lambda)$ ms respectively. The cluster positions are caused by the minimum RTO value of the Default (1000 ms) and S50 (50 ms) profiles in our setup. The 200 ms clusters originate from RTOs on the client $\leftrightarrow$ replica connection,



(a) Default (b) S50

Figure 11.17: RTO Profile Latency Histogram, $l = 1\%, p = 0\,\text{B}, n = 7, b = 50$

which is using plain TCP with a minimum RTO of 200 ms for all experiments (§11, §9.4). These visualized RTOs do not need to occur between the same replicas to delay the overall progress but can be distributed over different connections and HotStuff phases.

11.5 Summary

We now briefly summarize the results of our evaluation, grouped by evaluation section.

Input Request Saturation

E2E client replication latency and transaction throughput are central BFT-SMR benchmarking metrics (§10.2). Throughput is often subject to optimization techniques such as request batching, which affects the necessary work that replicas have to invest per single client request. To ensure that our experiments measure replica performance and are not bottlenecked by our load generator (client), we acquire the respective batch size values for which the replicas are saturated. This provides a baseline for configuration of later detailed measurements and supports the interpretation of results. Other varied parameters (Tab. 11.1) can also affect saturation and implications and effects are discussed in the respective experiment. An exemplary saturated configuration for our setup is: single client, single leader, TCP, noTLS, $p = 0, n = 4, l = 0$, resulting in a batch size of $b \sim 50$.

Performance without Loss

Differences between transport protocols are not significant in general. Usage of secure channel implementations on top of the transport protocol slightly reduces performance. Usage of DTLS is generally more costly than usage of TLS. Overall performance is centrally dominated by the number of participating replicas, where multiplicative in-/decrease between two replica numbers is governed by the ratio between the respective minimal quorum sizes $h(f) = (n - f)$. Payload size primarily affects client $\leftrightarrow$ replica connections, where larger payloads shift the bottleneck to the client.

Performance with Loss

With packet loss, we observe substantial performance degradation for larger loss values above 0.5%. Increasing the number of replicas has a stabilizing effect, due to inherently larger fault tolerance (§9.3.2). Reducing n relative to l increases result variance but also

protocol differences. Increasing n relative to l reduces protocol differences, variance, and increases overall robustness. Larger loss probabilities reduce the effective decision frequency of replicas, influencing the saturation point. Hence, in such a scenario a larger batch size (1) is necessary to shift the bottleneck toward the client and (2) may improve the effective throughput. In general, we observe high result variance for loss probabilities above 1%, such that the resulting mean values have reduced significance.

For loss $l \leq 1\%$, small differences between transport protocols are visible. As expected, UDP is not suited for lossy scenarios. TCP generally outperforms other reliable protocols (e.g., SCTP by $\sim 17\%$ for $p = 0, n = 7, b = 50, l = 0.25\%$), except for edge cases. We attribute this to superior optimization and hardware offloading support for TCP traffic. RUDP infrequently outperformed the other protocols for scenarios with small replica numbers, small payloads, and small loss probabilities (e.g., SCTP by $\sim 20\%$ and TCP by $\sim 13\%$ for $p = 0, n = 4, b = 25, l = 0.5\%$). We attribute this to reduced protocol and retransmission overhead, as well as RTO value choice closer to the actual network RTT.

Finally, adjustment of RTO to better match underlying RTT allows for performance improvements in the more stable loss region $l \leq 1\%$ in our experiments. This behavior is governed by the *minimum* RTO, which lies significantly above actual RTT in our setup.

12. Related Work

Study and optimization attempts of BFT-SMR over the last decades produced a vast amount of works and results in different dimensions (§4). For brevity, we restrict our scope to network stack analysis and optimization here. We are aware of works that seek improvements using special (network) hardware [124, 126, 133–136, 194, 195] or primitives like RDMA [123, 127, 137, 196, 197]. However, these systems introduce requirements that are typically only fulfilled in data center environments, while we aim for improvements in non-homogeneous deployments using a commodity network stack (§10.1.2). Hence we consider the above works to be only partially related.

We can roughly divide relevant related work into three categories: (1) Work with general relation to network / transport layer analysis, (2) work that considers other communication primitives without special hardware requirements, and (3) work that directly aims at the optimization of network and transport protocols.

12.1 General Relation to Network Transport and Analysis

In category (1), Ailijiang et al. [198] present analytical models of Paxos variants using queueing theory. The dedicated *Paxi* evaluation platform is later used to compare model-based simulation results against real-world measurements.

Gai et al. [139] conduct a general analysis of pipelined BFT-SMR protocols, using queueing theory. Established latency models are compared against measurements of several BFT-SMR protocols, including HotStuff, implemented in a newly developed evaluation framework. While a large set of experimental parameters is varied, no dedicated analysis of characteristics or impact of and on the transport protocol is given.

Shahsavari et al. [140] likewise created a theoretical model of HotStuff to predict performance. The model is simulated in OMNeT++ [199] and results are not compared to practical measurements. The authors consider loss as a parameter, however, possible impacts on the underlying network stack are not discussed, since the simulated packet loss is implemented as an application-level drop.

Giridharan et al. [146] address the liveness impact of asynchrony or equivocation in leader-based BFT-SMR with pipelining. They propose a new protocol with continuous leader rotation, which allows for commit, even if the k required QCs are not formed consecutively.

Camaioni et al. [111] work toward more efficient batching of client transaction payloads in BFT-SMR, applied by independent *broker* nodes before submitting the batches to replicas.

For their large-scale experiments, the authors employed a modified, reliable UDP variant for client-broker connections, to bypass the significant connection setup and state overhead of excessive amounts of short-lived TCP connections.

12.2 Other Communication Primitives without Special Hardware

PBFT [19] was originally implemented with all communication between nodes using UDP. Retransmissions were handled on agreement protocol level and no comparison to other, reliable transport protocols was conducted. In another paper, Castro and Liskov present BFS [200], a BFT NFS variant. Its implementation similarly uses UDP for communcation between nodes but does not compare the measurement results to other transport protocols.

Copeland and Zhong [201] proposed a BFT variant of the Raft [44] algorithm with CFT. The authors present an implementation of their design, using UDP-based communication. However, no measurements or comparisons to other transports were conducted.

Multiple works propose a separation of transaction distribution from consensus via a *shared mempool* abstraction [10, 11, 13], addressing communication overhead. In practice, the implementations again rely on a reliable transport such as TCP per default. The interplay between transport and memory pool implementation is not analyzed.

Neiheiser et al. [108] present an approach to reduce both network and processing strain on HotStuff leaders, using tree-based aggregation and forwarding of messages. The authors implement their solution on top of the original HotStuff codebase and attest to improved performance in measurements on physical machines. Modifications of transport protocol or configuration are not studied in more detail.

Kaklamanis et al. [202] propose usage of erasure codes to address the HotStuff leader bottleneck. Proposal blocks are encoded into chunks, which are distributed by all replicas in a decentralized fashion. The authors attest to improved throughput in experiments on a single AWS machine. Again, variation of transport protocol or options are not considered.

Another line of work aims to improve operation of CFT consensus in wireless networks, by e.g., adjusting the consensus protocol for very high packet loss scenarios in flying ad-hoc networks [203] or optimizing transmission power and bandwidth allocation [204] for reliability and reduced latency. Transport protocols or configuration are not discussed.

12.3 Direct Focus on Optimization of Network Transport

Clement et al. proposed the BFT protocol Aardvark [98], with focus on more robust and performant operation under actual byzantine behavior in the network. The authors studied the performance of a selection of BFT protocols, as well as their response to flooding with TCP and UDP traffic. Resulting performance degradation with either transport protocol is attributed to the implementation, not the general protocol design.

Chondros et al. [205] discuss the usage of UDP in PBFT and outline possible robustness and performance limitations under packet loss. The authors conclude, that in a practical setup, unreliable communication has a significant impact on performance and robustness. A direct comparison or measurements of other transport protocols is not provided.

RBFT by Aublin, Mokhtar and Quéma [99] was implemented, using both TCP and UDP transports for comparison. The original authors report comparable peak throughput but smaller latency, of up to 22% less, for the UDP-based implementation. The authors attribute this to reduced protocol operation overhead.

Kalia, Kaminsky, and Andersen [206] present a dedicated high-performance Remote Procedure Call (RPC) library (*eRPC*) for data center networks, to improve performance of

Raft CFT consensus without the need to modify core source code. The RPC library itself runs on top of an unreliable transport such as UDP. Results are compared to other solutions, based on high-performance networking technologies, such as RDMA.

Damaskinos et al. [207] present AggregaThor, a BFT distributed stochastic gradient descent system, based on TensorFlow. The authors compare a TCP-based implementation to an additional unreliable, UDP-based communication channel. In order to tolerate loss on the UDP channel, the authors provide application-level modifications, e.g., to gradient aggregation rules. The authors report a convergence performance improvement of the UDP-based AggregaThor implementation in comparison to unmodified TensorFlow over gRPC for 10% packet loss, which they attribute to the ability of AggregaThor to continue operation without the need for retransmissions. A detailed comparison of AggregaThor performance under different transport protocols is not conducted.

Ohba et al. [208] propose a content proximity distribution platform over IEEE 802.11s wireless mesh networks. The authors use Hyperledger Sawtooth Proof-of-Elapsed-Time consensus for decentralized storage of keying material, employed for access control on encrypted content. Differences between TCP and UDP were practically studied for content distribution, but not agreement. Measurements were conducted on a single machine, using a dockerized Python implementation over emulated 802.11 protocols via ns-3 [209].

Lorünser et al. [210] provide a theoretical analysis and performance model of PBFT. The authors discuss the characteristics and applicability of TCP and UDP, with and without loss. They propose optimistic usage of UDP-based communication, using forward error-correcting codes. Model verification is conducted against simulation in OMNeT++ and a Python implementation, running multiple nodes on a single machine. A discussion of interrelation with other network layers or secure channel implementation is not provided. The authors attest to significant performance differences for packet loss of up to 30 %.

12.4 Summary

We conduct a detailed analysis of transport protocol, secure channel, and configuration impact in a BFT-SMR context. Multiple protocols and channel architectures are included, case studies are centered around the representative HotStuff BFT-SMR system. We discuss both the expected isolated and BFT-SMR-integrated overhead and general behavior in best-case and lossy network scenarios. We experimentally verify the analysis results in a series of measurements, using bare-metal deployments of HotStuff for all studied protocols. Our results demonstrate existing optimization potential in the reliable network building block of BFT-SMR systems, even using a commodity network stack.

To the best of our knowledge, our work is the first to (i) analyze interdependencies of commodity network stack, secure channels, and leader-based consensus in detail and for multiple transport protocols, (ii) practically demonstrate optimization potential in this context and (iii) experimentally quantify performance of practical, commodity network transport stack optimization on state-of-the-art BFT-SMR.

13. Conclusion

In Part II of this thesis, we study the potential of network transport optimization on leader-based BFT-SMR. Ensuring equivalent base guarantees of the network building block (§5.1) allows modification of transport protocol and configuration to better fit a target networking environment. In detail, we investigate the impact of four transport protocols, two secure channels, and a variety of configuration parameters on a leader-based BFT-SMR system. For theoretical and experimental case studies, the seminal HotStuff (§6) system is used.

13.1 Result Overview

In our analysis, we establish an overview of expected behavior. Transport protocol efficiency differences based on header and data size are marginal for small and become insignificant for larger payloads. Hence, if the protocols show performance differences, they are likely caused by e.g., advanced processing logic or availability of hardware offloading.

Client and leader are bottlenecks in HotStuff, both in terms of processing and networking. Client strain scales with replica number and vice versa. Messages on client $\leftrightarrow$ replica connections are subject to BFT-SMR payload size and therefore typically larger; agreement is only conducted on payload hashes. Client $\leftrightarrow$ replica communication is especially vulnerable to losses in the HotStuff PoC implementation, while at most f lost votes can be tolerated without impeding progress, assuming no other failures occur at the same time.

While UDP provides the lowest protocol processing and state overhead, it is only suitable for lossless connections due to unreliable service. All reliable transport protocols are suitable in theory, while RUDP has the lowest state and processing overhead. TCP and SCTP both provide own segmentation logic to avoid IP fragmentation. CC improves utility in real-world, congested networks. TCP is well-studied and offers more hardware offloading capabilities in comparison to SCTP, while purely cumulative ACK operation in TCP can result in unnecessary retransmissions.

Relevant options are mainly available for TCP and SCTP. The parameter space can be categorized into options to (1) delay and coalesce writes and (2) configure retransmission behavior. Write coalescence may be disabled for improved latency, Nagle's algorithm must not be used with DelACK. Retransmission should be configured to use F-RTO and RTO bounds should match the expected network timing bounds as best as possible.

Replication in HotStuff depends on successful execution of all (network) layers below. Up to f arbitrary faults, including omission, can be tolerated in parallel before delay occurs. Batching and pipelining increase the impact of delay on a larger set of client requests.

We create a simple throughput model of a BFT-SMR SuT. Input request saturation governs the effective load distribution between replica and client nodes. Modification of the batch size parameter allows to shift this load. We use this insight and model to configure experiments such that replicas are running at maximum possible replication speed.

In our experiments, we first assert saturation points for a parameter configuration and adjust the setup accordingly. Best-case performance measurements show no significant differences between transport protocols without loss, confirming our analysis hypothesis. Usage of secure channel incurs a small latency overhead, where DTLS is more costly than TLS. Performance is centrally dominated by the number of participating replicas, followed by payload size, which mainly shifts the bottleneck toward the client.

In scenarios with packet loss, we observe a significant performance reduction for $l \leq 0.5\%$. Above 1% this coincides with considerable latency variance and above 2% loss performance is approximately reduced by one order of magnitude. Larger replica numbers entail increased inherent fault tolerance and have a stabilizing effect. Smaller replica numbers increase the differences between studied transport protocols and configurations. Loss can affect the effective replica decision frequency, shifting the saturation point and allowing for throughput re-adjustment through batch size modification. Most relevant differences between transport protocols are visible for $l \leq 1\%$.

TCP generally outperforms other protocols (e.g., SCTP by $\sim 17\%$ for $p = 0, n = 7, b = 50, l = 0.25\%$) due to superior optimization and hardware offloading support, except for edge cases. One scenario exhibits small loss, replica numbers, and payloads. Here, RUDP outperformed (e.g., SCTP by $\sim 20\%$ and TCP by $\sim 13\%$ for $p = 0, n = 4, b = 25, l = 0.5\%$) due to lower state and protocol overhead, as well as superior RTO configuration.

We observe superior performance for RTO configuration with dynamic values closer to actual network timing in slightly lossy scenarios, e.g., profile S50 has $\sim 19\%$ less latency than the Default profile, for $p = 0, n = 7, b = 25, l = 1\%$. The visible improvements are mainly governed by the minimum RTO bound.

13.2 Interpretation

A leader-based consensus architecture, as used in HotStuff, reduces overall communication complexity in the good case, but shifts the system bottleneck toward the incumbent leader. Usage of multiple parallel leaders managing independent views distributes the load, while the underlying principles and interactions remain the same. Techniques such as batching and pipelining similarly optimize the good case but extend e.g., the impact scope of delays in case of inconveniently occurring packet loss onto more client requests. Large HotStuff (default) timeouts, e.g., upon leader impeachment, can have a significant impact on resulting performance and should be configured to match target scenario expectations.

Shifting the operational load in the SuT via batch size configuration allows for precise tuning and interpretation of experiment results. An increase in specification detail of used experiment parameters in BFT-SMR systems literature would contribute to reproducibility and deeper understanding of published results.

Considering our experiment results, we make the following observations. The additional relative cost of employing secure channels for inter-replica communication is small and hence usage is functional. In our lossless scenarios, transport protocol choice does not cause significant performance differences. The main operational cost of BFT-SMR is related to complexity of communication and cryptographic processing and closely tied to the number of participating replicas. Bottleneck variations in special setups might apply (§5.2). Only manifesting under loss, advanced protocol logic, such as retransmission behavior, is

the core reason for observed transport protocol differences in terms of performance and robustness. While performance differences grow with loss probability, increasing result variance shadows the impact of different protocols or configurations. We observe tangible configuration impact for small to medium replica numbers for loss probabilities $\leq 1\%$.

We consider TCP to be the best default choice for a commodity network stack BFT-SMR setup. It is well-studied, well-tested, subject to ongoing work [153] and competition [158], and has strong hardware offloading support. In specific edge cases, choosing another protocol might be functional. In small networks with low loss probabilities and small client payloads, a reliable transport protocol with reduced state overhead can potentially yield performance improvements. In BFT-SMR setups with varying connection topologies, using e.g., data multiplexing over a single connection, usage of QUIC might provide benefits.

While the partial synchrony model provides a convenient framework for designing and proving attributes of distributed systems, implementations typically make some form of synchrony assumption by choice of e.g., default timeout values. Parameter values that do not or poorly reflect underlying network behavior can lead to overhead (e.g., spurious retransmissions) or waiting time (e.g., needlessly long timeouts). Hence, a divergence from theoretically optimal settings manifests as optimization potential. In other words, BFT-SMR systems can benefit from network transport protocol optimization, if the employed parameters deviate from the theoretical optimum settings for the respective network conditions. While this optimum might not be known or attainable in all scenarios, our measurements demonstrate multiple approaches toward optimization. The better predictable target network timing bounds or communication behavior patterns are, the more precise can one conduct adjustments of these default parameters. We observe that concrete performance gains through transport protocol choice or configuration are possible if (1) losses occur (such that differences in reliable transmission logic become relevant), (2) the overall system performance is not shadowed by another larger bottleneck, (3) loss induced variance is sufficiently small, and (4) behavior and parameter deviation from the (potentially unknown) optimum is sufficiently large for improvements to become significant.

Overall, choice and configuration of the used network transport can have a significant impact on performance and robustness of BFT-SMR. For practical systems, consideration of attributes and behavior of underlying and currently abstracted layers is advisable. Any fault in a lower infrastructure layer can potentially influence performance and robustness by absorbing a slot of the provided fault tolerance level. Hence, effectively masking all faults that are not part of the original target attacker or quality-violation model can increase performance and robustness of the final system.

13.3 Verdict

We now put the achieved insights into context of our Research Objectives (§1.2).

Q1 *How can the performance of contemporary, practical SMR be improved?*

Q1.1 *Are fundamental improvements still possible?*

Yes, if the BFT-SMR system uses a network transport protocol or configuration that deviates from an optimal response, suitable to address actual network behavior.

Q1.2 *How can safety and liveness properties be preserved?*

By compliance of the transport protocol and configuration to required functionality (e.g., reliable transmission, ordered delivery, ...), safety and liveness are preserved.

Q1.3 *Which system aspects should be investigated?*

We identify optimization of network transport protocols and configuration as suitable avenue to improve BFT-SMR performance.

Q2 *How big is the impact of the identified optimization spaces?*

Q2.1 *What are necessary preconditions?*

Potential impact of network transport protocol change and configuration on BFT-SMR are significant but depend heavily on the target network and consensus algorithm. If network characteristics prompt protocol and configuration differences to emerge as performance and robustness impact (e.g., under loss) and the current configuration is suboptimal (e.g., timeouts deviating significantly from expected real-world behavior), there is optimization potential.

Q2.2 *How big are potential performance benefits?*

Better knowledge about the target network behavior allows for more precise configuration. The magnitude of potential improvement also depends on the deviation of the old configuration from a theoretical optimum, and on the impact of other bottlenecks and e.g., loss variance.

For our target scenario of small to medium permissioned setups (§10.1.2) and limited loss probability ($\leq 1\%$), we observed up to $\sim 20\%$ better performance in comparison to the respective baseline. Optimization of transport protocol choice and configuration might not yield the largest possible improvements for a concrete BFT-SMR setup, but it is a modular approach that (1) does not require modification of a target consensus algorithm, potentially endangering safety and liveness guarantees, and (2) potentially applies to a wider range of different algorithms on top.

Part III

Optimization via the Underlying Cryptographic Primitives

14. Overview

The upcoming chapters address the optimization potential of the underlying cryptographic primitives as a building block of BFT-SMR systems. Chapter 15 provides relevant background information such as an overview of simple and advanced message authentication schemes, as well as signature scheme interactiveness.

In Chapter 16, we select a set of threshold signature schemes for further study. We then conduct an initial compatibility analysis of the schemes for usage in a BFT-SMR context. For each scheme we select an implementing software library artifact, describing the resulting interfaces as derived schemes. We analyze the complexity of derived scheme operations and describe a simplified latency model for scheme comparison.

Chapter 17 introduces experiment rationale, setup, and implementation details.

In Chapter 18, we experimentally verify performance, load, and bandwidth behavior of all derived schemes through measurements in a bare-metal deployment, and compare extrapolated scaling behavior using the model from §16.

Chapter 19 presents related work.

We conclude Part III in Chapter 20 with a revision of presented results, extended interpretation, and discussion in context of our research objectives.

Relation to Existing Publications

These upcoming chapters contain content from the following papers [90, 149]:

- Richard von Seck, Filip Rezabek, Sebastian Gallenmüller, and Georg Carle, *On the Impact of Network Transport Protocols on Leader-Based Consensus Communication* in Proceedings of the 6th ACM International Symposium on Blockchain and Secure Critical Infrastructure, BSCI '24, page 1–11, New York, NY, USA, 2025. Association for Computing Machinery.
- Richard von Seck, Filip Rezabek, and Georg Carle, *Thresh-Hold: Assessment of Threshold Cryptography in Leader-Based Consensus* in 2024 IEEE 49th Conference on Local Computer Networks (LCN), pages 1–8. IEEE, 2024.

For improved readability, content from the papers has been rearranged, reworked, and extended. Chapter 15 and Chapter 16 are extended and restructured versions of background and analysis from [149]. Chapter 17 is an extended and updated version of the design of [149]. Due to the similarity of the setup with [90], the chapter contains descriptions and a figure that was originally introduced in [90]. Chapter 18 and Chapter 19 are extended, revised, and updated versions of evaluation and related work from [149].

The following improvements and new contributions were added:

- §15: Extended introduction of authentication primitives and function
- §16: Added latency model, function interface classification, and details
- §18: Significantly extended and revised evaluation
  - Identified and fixed a bug in the microbenchmark code for Secp
  - Extension and re-evaluation of microbenchmarks and all affected results
  - Added scaling approximation evaluation, based on the model from §16.5
  - Extended replication performance evaluation on more data and in more detail
- §19: Revised and extended related work

Statement on Author's Contributions

The improvements listed above are the work of the author of this thesis. For the introduced content on analysis, modeling, and experiment design (§16, §17) the author provided major contributions for the ideas and analysis. Filip Rezabek contributed to the initial selection process of threshold signature schemes, their textual introduction in [149], and the steering process of the relevant student theses.

Initial HotStuff PoC implementations were developed by Philip Höbler (BLS), Aaron Huber (Schnorr), and Ahmet Öztürk (MP-ECDSA) as part of their respective Bachelor's theses. The author contributed to conception of the theses, as well as design and implementation decisions during these works as a thesis advisor. The HotStuff implementation was revised, extended, and migrated by the author of this thesis. The presented measurements, evaluation, and interpretation results are the work of the author of this thesis.

15. Background: Crypto Building Block

For evaluation of this building block, we study multiple threshold signature schemes. In the following, we introduce a selection of relevant primitives and background information.

15.1 Message Authentication Schemes

Authenticated messages are a widespread assumption in BFT-SMR systems (§5.1). A plethora of authentication approaches and combinations in BFT-SMR context has been proposed and studied so far (§19). Apart from this large body of work, threshold signature schemes have gained increasing interest in recent BFT-SMR evaluations and system architectures. Typically, these schemes are employed to (1) improve the authenticator complexity [5] of a protocol [7,108,111,211,212], or to (2) implement a distributed random coin for asynchronous or randomized protocols [10, 12, 13, 79, 213]. The rise of research and products in the permissionless blockchain space in turn provides usage scenarios and motivates signature scheme research [214–218]. On these grounds, the impact of threshold signature schemes on BFT-SMR is of special interest to us. In the following, we briefly introduce different categories of message authentication schemes, tools, and attributes. For the following definitions, let $\mathcal{P}$ be a group of parties $p_i \in \mathcal{P}, |\mathcal{P}| = n$.

15.1.1 Message Authentication Codes

One of the most prevalent approaches to verify authenticity and integrity of messages between communicating partners is the usage of a MAC. Typically, two target communication partners $p, q \in \mathcal{P}$ share a common secret key k, which is used to generate the respective authenticator tag $t := \text{MAC}_k(m)$ on message m. If only p, q know k and the MAC_k function is secure, then upon receipt of (m, t), p is convinced that t was created by q. Some constructions also include additional inputs of keys, nonces, or padding data [219]. Possible architectures include the usage of cryptographic hash functions (e.g., Keccak [220]) as part of an HMAC construction [221, 222], cipher-based MACs [223] or authenticator constructions such as Poly1305 [219, 224]. Usage of MAC authentication is convenient due to its relatively fast computation speed in comparison to digital signatures [19]. This is due to the availability of powerful Single Instruction Multiple Data (SIMD) primitives in modern processors, which are often leveraged by MAC and symmetric cryptography implementations. Consider for example the Intel AVX [225] SIMD instructions, and their usage to optimize Keccak in the OpenSSL TLS implementation [226]. A drawback of MAC authenticators is the inability to inherently prove authenticity of some message to a third party, that does not share the common key of the two communicating parties.

15.1.2 Digital Signatures

Digital signatures provide authenticators, addressing the aforementioned limitation of MACs. Simplified, each $p_i \in \mathcal{P}$ has a private, secret key sk_i and a public key pk_i, that is published e.g., via a Public Key Infrastructure (PKI) [227]. A signature $\sigma_i := sign_{sk}(m)$ by p_i can be verified by any $p \in \mathcal{P}$ with access to (pk_i, σ_i, m). Depending on the concrete scheme, necessary material for signature (verification) operations or the data that actually constitutes a final key may vary. While digital signatures can reduce the required number of keys to pairwise authenticate messages between $p, q \in \mathcal{P}$, cryptographic base operations are typically more expensive than MACs. There exists a wide variety of public-key cryptography algorithms to create digital signatures. We only list a brief selection of prevalent schemes used in context of cryptocurrencies and BFT-SMR systems in general.

ECDSA [228] is a variant of the Digital Signature Algorithm (DSA) using Elliptic Curve Cryptography (ECC). Integration and usage of 256-bit ECDSA signatures associated with a Koblitz curve secp256k1 [229] in the Bitcoin codebase [230] resulted in adoption and study of (secp256k1) ECDSA signatures in context of BFT-SMR systems [6, 7, 108, 212, 231, 232]. In principle, ECDSA requires a fresh, cryptographically secure, random nonce for each signature creation to prevent attacks on the scheme, e.g., nonce reuse, potentially enabling recovery of the private key by an attacker [228]. For e.g., scenarios without sufficiently available secure randomness, ECDSA can be turned into a deterministic scheme by employing a sufficiently secure pseudorandom process as specified in RFC 6979 [233].

Initially intended for use in constrained devices such as smartcards, Schnorr signatures [234] offer fast signature generation and small signature sizes. The expiry of patent protection of the Schnorr cryptosystem in February 2010 [235] sparked new research activity, with [28, 212, 236, 237] and without [218, 238–243] relation to application in a BFT-SMR context. Similar to ECDSA, Schnorr signatures require a fresh, securely random nonce for each signature to prevent attacks and potential private key leakage [238].

Edwards-curve Digital Signature Algorithm (EdDSA) signatures [244–246] are a variant of the Schnorr cryptosystem, based on (potentially twisted [247]) Edwards curves [248]. Recently, we see application of EdDSA signatures also in a BFT-SMR context (§19).

Boneh-Lynn-Shacham (BLS) signatures [249], dubbed after the initials of the authors, are a signature scheme based on elliptic curves and Weil pairings. Central focus of the work was to shorten the resulting signature size in comparison to existing schemes while keeping a comparable security level. The security of ECDSA, Schnorr, and EdDSA signatures centrally depends on the hardness of the discrete logarithm problem. The security of BLS is based on the hardness of the computational Diffie-Hellman problem, which is related but not equivalent to the discrete logarithm problem.

15.1.3 Multi-Signature Schemes

A *multi-signature scheme* [214] allows any subgroup $\Gamma \subseteq \mathcal{P}$ to generate a common signature σ on a common input message m, such that a verifier is convinced that all $q_i \in \Gamma$ participated in the process. There exist different notions of multi-signatures in literature [250], e.g., regarding the necessary set of participants ($\Gamma = \mathcal{P}$). A core property is accountability, i.e., without use of trusted third parties, a verifier can identify all participants from σ, m.

15.1.4 Threshold Signature Schemes

Threshold signature schemes [250, 251] are a variant of multi-signature schemes, with additional focus on robustness. Secrets and computation are distributed between parties to eliminate single points of failure. Some subset $\Gamma' \subseteq \mathcal{P}, t \leq |\Gamma'| \leq n$ can collaborate to create a common signature σ' on a common message m, such that a verifier that knows

the fixed public key is convinced that some $t \leq |\Gamma'| \leq n$ parties participated in the signing process. Signer identities are not (necessarily) revealed. A desirable property of threshold signature schemes is *robustness*, where $(t-1)$ byzantine parties cannot prevent the rest from creating a valid signature. We note that some schemes require $t < n, \Gamma' \subset \mathcal{P}$.

15.1.5 Aggregate Signature Schemes

Given n (possibly) distinct messages $m_1, \ldots, m_n$ and a keypair (sk_i, pk_i) for each $p_i \in \mathcal{P}$, *aggregate* signature schemes [252] allow a player set $\mathcal{P}$, to aggregate all signatures into one $\sigma = \sigma_1 \circ \ldots \circ \sigma_n$, where p_i creates $\sigma_i := sign_{sk_i}(m_i)$. A verifier can then be convinced by $(\sigma, pk_1, \ldots, pk_n, m_1, \ldots, m_n)$ that each p_i indeed signed their m_i correctly. This aggregation can be conducted by anyone and allows for compression of the resulting signature material at the cost of computational overhead.

We note that there exist inaccuracies or *function overloading* of the term *aggregate* in literature on threshold and aggregate schemes. While threshold schemes work on a *common input message*, aggregate schemes work on (possibly) *distinct input messages*. Nevertheless, the term *aggregate* is sometimes used in threshold signature schemes to refer to combination and potential compression of *Partial Signatures (PSigs)* or *signature shares* on the same input message [218, 239]. Hence, not all threshold signature schemes necessarily support aggregation of $\sigma_A \circ \sigma_B$, where $\sigma_A = sign_{sk_A}(m_A), \sigma_B = sign_{sk_B}(m_B), m_A \neq m_B$.

For clarity in context of threshold signature schemes, we hence refer to a combination (and potential compression) of a set of (partial) signatures $\sigma = \sigma_1 \circ \ldots \circ \sigma_k$, where $\sigma_i := sign_{sk_i}(m), q_i \in \Gamma'$ are created on the same message m as *agglomeration*. In case a threshold scheme additionally supports *aggregation* in the original sense, we still use *agglomerate* where appropriate for consistency to other discussed schemes.

15.1.6 Half-Aggregate Signature Schemes

Consider an aggregate signature scheme where a single, regular signature has size s. If the non-interactive signature aggregation step does not reduce the size of n signatures (on possibly distinct messages) to a constant size s but rather compresses the set to a size of $\sim \frac{n}{2}s$, it is referred to as *half-aggregate signature scheme* [241, 242].

15.2 Distributed Key Generation

During the setup of e.g., a threshold signature scheme, initial keying material needs to be distributed. This can be done by a trusted dealer or a distributed process [251]. Since elimination of single points-of-failure is a core motivation to employ threshold signature schemes in the first place, a distributed process is typically preferable. A distributed protocol that securely initializes this key material is referred to as Distributed Key Generation (DKG) [253, 254]. Effectively, the setup is not strictly limited to keying material but can also include other material, such as nonces for signature randomization [254].

15.3 Scheme Operation Interactiveness

The introduced digital signature schemes define a set of abstract operations, that are invoked by the user of a scheme. These operations may not only vary from scheme to scheme or from type to type but also in terms of invocation time and the necessary number of actively collaborating players to complete a certain operation. That is, if an operation requires communication with other players to succeed it is called *interactive*, *non-interactive* otherwise [216]. Interactive signing schemes potentially limit applicability for target use cases such as operation in context of BFT-SMR (§16.2).

Table 15.1: Selection of Abstract Signature Scheme Function Definitions in Literature

		Preprocessing		Signing			Verify	
Type	**Scheme**	**PreRound**	**PreAgg**	**PartSig**	**Agglom**	**Aggreg**	**(P)SigVeri**	**AggVeri**
Regular	[215, 249]			●			●	
Aggregate	[252]			●		●	●	●
Threshold	[215]			●			●	
SI-Threshold	[218]	●	●	●	●		●	●

To acquire an overview of existing work, we select a set of signature scheme operation definitions. This list is non-exhaustive and not necessarily representative, since other schemes or specifications might deviate. However, it offers a sample of signing function interface definitions in recent literature. We list the selection in Tab. 15.1. All scheme types define a key generation (KeyGen) function, which we omit in the table for brevity.

The interface for a regular digital signing scheme is defined over generic signing and verification functions, which are non-interactive.

The aggregate scheme by Boneh et al. [252] adds aggregation (Aggreg) and differentiates between two distinct verification functions, one for single (SigVeri) and aggregated (AggVeri) signatures and public keys respectively. These functions are non-interactive.

The threshold scheme definition by Gennaro and Goldfeder [215] specifies an interactive, signing protocol, which is not aggregate and only has a generic verification function.

Finally, Ruffing et al. [218] introduce the notion of *semi-interactive threshold signatures* (SI-Threshold), which are threshold signature schemes where the signature process is split up into two distinct steps. In the *preprocessing* or *presigning* step, players perform (potentially interactive) computations without knowing the eventual message input to sign or the set of eventually participating signers. In the *signing* step, the actual signature is computed. The authors further split up the (1) preprocessing step into preprocessing (PreRound) and presignature aggregation (PreAgg), and the (2) signing step into signature share algorithm that creates PSigs (PartSig) and signature agglomeration (Agglom) that combines PSigs. Furthermore, the authors distinguish between validation for PSigs (PSigVeri) and the agglomerated signature (AggVeri).

16. Analysis

From the large set of possible message authentication schemes to use in a BFT-SMR context, threshold signatures are of special interest to us (§15.1). The number of schemes studied in context of BFT-SMR is limited and more recent evaluations do not necessarily include practical E2E measurements (§19). We aim to leverage recent progress in threshold cryptography and study its impact as a building block (§5.1), following our general optimization methodology (§5.4) as already applied for network transports in Part II.

This chapter provides the base for interpretation of our empirical measurements. First, we identify a set of threshold signature schemes for further study (§16.1) and analyze their general suitability in a HotStuff context (§16.2). For each scheme, we choose a software library implementation and analyze pecularities and limitations of these derived schemes (§16.3), as well as the resulting complexity of relevant cryptographic operations (§16.4). Finally, we construct a simple model for performance difference approximation between a non-threshold signature baseline and a threshold signature candidate (§16.5).

16.1 Signature Scheme Selection

To enable a more detailed analysis and practical evaluation, we limit our scope to a selection of relevant threshold signature protocols. We aim to extend the set of already studied schemes with novel ones but also consider schemes that might offer additional benefits to BFT-SMR (e.g., more efficient, faster operations, ...). Finally, we choose schemes that can be practically evaluated at the time of this work, i.e., schemes for which at least a PoC implementation is available. We describe the selection in the following.

16.1.1 secp256k1 ECDSA Digital Signatures

As a default, we employ simple, non-threshold secp256k1 ECDSA digital signatures. Both the HotStuff PoC implementation [147] and evaluation [5] employ secp256k1 simple signatures, but the scheme is also used in other BFT-SMR systems (§19). The scheme provides us with a baseline, against which we compare other, threshold signature schemes.

16.1.2 BLS Threshold Signatures

BLS is one of the most widely proposed and studied schemes for threshold signatures in context of BFT-SMR (§19), due to its convenient properties (§16.2). The signature process is non-interactive, hence no preprocessing is required, and a single communication round

Table 16.1: Desired Signature Scheme Properties

1	A replica can independently create a PSig
2	PSigs can be independently verified
3	A leader can agglomerate t valid PSigs into one signature
3a	Any number $k, t \leq k \leq n$, of valid PSigs qualifies
3b	PSigs built from arbitrary group $\Gamma \subseteq \mathcal{N}, \lvert\Gamma\rvert = k$
3c	The leader can identify misbehaving signers
4	An agglomerated signature can be independently verified
5	Non-interactive signing / verification performant (relative)

suffices for signing. Using e.g., the construction from Boldyreva [251], BLS-based threshold signatures can be implemented. Boneh et al. [252] extend the functionality, additionally enabling true aggregation via bilinear maps. BLS generally incurs higher cost for signature generation and verification, which can be up to 10× slower than for ECDSA [255]. This further motivates the study of alternative threshold signature schemes.

16.1.3 FROST Schnorr Threshold Signatures

Flexible Round-Optimized Schnorr Threshold (FROST) signatures [239] are based on the Schnorr cryptosystem [234]. FROST trades scheme robustness (§15.1.4) for efficiency. That is, to successfully create a signature FROST requires a response from all $q_i \in \Gamma \subseteq \mathcal{N}$. If any $q_i \in \Gamma$ does not respond, no signature is created. ROAST [218] and FROST2 [240], which sacrifice efficiency and require two online communication rounds, address this. Similar to plain Schnorr, FROST signature creation generally consists of a nonce generation (preprocessing) phase and the actual signature computation. Nonces can be precomputed and require one communication round for multiple signature operations in a batched fashion [239]. During signature computation, only one communication round is required. Each $q_i \in \Gamma$ computes its PSig share and returns it. Afterward, an agglomerator node verifies and combines the PSigs. Hence, FROST threshold signatures can be interpreted as an instance of a semi-interactive (§15.3) scheme, where the separation of operations into a preprocessing phase can render the final signature step effectively non-interactive.

16.1.4 GG20 ECDSA Threshold Signatures

We see a significant number of ECDSA-based threshold schemes being introduced [215,217, 256,257]. We rely on *GG20* [217], which supports identifiable abort and is split up into a preprocessing phase, followed by final signature computation in one communication round. The scheme conceptually builds on *GG18* [215]. Identifiable abort offers misbehavior identification during key generation or signing, through additional Zero Knowledge Proof (ZKP) verifications. While GG20 preprocessing is more costly than FROST, it is robust and allows proving presigning correctness. Similar to FROST, GG20 can be employed as a semi-interactive scheme. Canetti et al. [216] recommend, that preprocessing data should be created for each distinct set of final signers independently, in order for security to hold. Assuming this requirement, a preselection of expected signers needs to be conducted, like in FROST. We refer to the GG20 variant of ECDSA as Threshold ECDSA (THECDSA) and to the non-threshold variant of secp256k1 signatures as *secp256k1*.

16.2 Signature Scheme Suitability

The selected signature schemes offer differing functionality but also potentially introduce new limitations on usage in a BFT-SMR context. Hence, we analyze the suitability of our scheme selection for operation in HotStuff. A desirable scheme should not e.g., obstruct

Table 16.2: Signature Scheme Property Conformity

Property	BLS	THECDSA	FROST	secp256k1
1	✓	○	○	✓
2	✓	○	✓	✓
3	✓	✓	✓	×
3a	✓	✓	○	✓
3b	✓	✓	○	✓
3c	✓	✓	✓	✓
4	✓	✓	✓	✓
5	×	○	○	✓

×: Not conform, ○: Partially conform, ✓: Conform

typical HotStuff role assumptions (§6) or regular consensus operation (§9.3). Replication assumes that on some level two distinct replicas $r, r' \in \mathcal{N}$ with the same role can provide equivalent service. If r' acts as a fallback for a faulty r, it should do so independently. A faulty replica should not be able to prevent other replicas from making progress through signature scheme constraints. This allows modular replacement of the signature scheme as a building block, without e.g., requiring adjustment of consensus protocol or timing.

We list relevant signature scheme assumptions in Tab. 16.1. Replicas should be able to sign (1) and verify (2, 4) signatures independently. A t-to-1 agglomeration of votes for QCs (3) has been frequently proposed (§15.1) to lessen authenticator complexity, reducing transmission bandwidth and scale verification cost of QCs. A leader can form a QC from k valid votes (3a). The specific signer should not matter, so long $k \geq t$ (3b). Arbitrary choice of $\Gamma \subseteq \mathcal{N}$ allows a leader to progress after the first k votes. Faulty replica identification (3c) allows for directed node maintenance. The final (non-interactive) signature or verification step should be efficient to ensure performant operation (5).

Revisiting our candidates from §16.1, we find that only a subset of the desired properties (Tab. 16.1) is present. We establish an overview of the approximate conformity in Tab. 16.2. Secp offers all properties, except for agglomeration (3). The original HotStuff PoC uses Secp vectors in QCs. Hence, QC verification time and transmission size grow proportionally to n. Base operation cost is relatively small but for large n, the inability to agglomerate may affect performance (5). BLS fulfills all properties, except for costly base operations (5). The scheme offers constructions with threshold semantics, true aggregation, and generally non-interactive operations. FROST supports PSig agglomeration and the final signature step is cheaper than BLS, but since it is non-robust it demands all expected signers Γ to participate (§16.1). Before replicas can independently create PSigs (1), presigning material is required for all possible $\Gamma \subseteq \mathcal{N}$ and valid Γ depend on the respective presigning configuration (3a/3b). Hence, if preprocessing can be executed fast enough to not limit consensus progress, FROST might be a suitable alternative to BLS. THECDSA supports PSig agglomeration and offers a cheaper final signing step than BLS. Preprocessing is required for independent PSigs creation (1). In principle, arbitrary signers $\Gamma' \subset \mathcal{N}, |\Gamma| = k, t \leq k < n$ are sufficient to create a valid signature, using the same presigning data (3a/3b) (§16.1). Verification of PSigs (2) is conducted using consistency proofs from preprocessing. Canetti et al. [216] recommend generation of distinct data for all Γ' for security. Similar to FROST, the Γ' would then be restricted to a preselection. Verification of single PSigs is only possible through collective signer effort, with varying complexities, depending on the invested preprocessing effort. Still, it allows a

leader to identify misbehaving signers. Similar to FROST, THECDSA might be a suitable alternative to BLS under the assumption that preprocessing is executed fast enough.

We conclude, that BLS is most compatible in a HotStuff context, but causes elevated base operation cost. FROST and THECDSA are potential alternatives, however, preprocessing must be executed securely, independently of agreement, and fast enough to not limit overall performance. Also, preprocessing additionally increases memory and bandwidth overhead.

16.3 Derived Schemes and Implementation Constraints

We are primarily interested in checking the suitability of different threshold signature schemes in terms of performance. Hence, we aim for best-case performance estimates, i.e., we assume no replica misbehavior and favorable network conditions. For integration into HotStuff, we require a suitable software library implementation for each signature scheme. Concrete implementations potentially introduce additional limitations, that need to be considered in analysis and evaluation. To explicitly distinguish between the theoretical capabilities of a threshold signature scheme (e.g., as defined in the publication) and the practical implementation (e.g., as provided by the library), we introduce what we call *derived schemes*. We choose a software library for each signature scheme and refer to the characteristics of this artifact as derived scheme.

16.3.1 Definition

In general, if a scheme requires an interactive presigning phase, we assume independent, successful, and timely execution of it, before the resulting material is required for signing in the consensus algorithm (§16.2). Unfortunately, the PoC libraries available at the time of development partially provide highly unoptimized presigning implementations that are effectively unfit for benchmarking. Hence, for interactive schemes, we generally limit analysis and benchmarks to the final non-interactive signing step.

For implementation of BLS, we rely on `RELIC`[1] by Aranha et al. [258], akin to SBFT [8]. Integration into the C++ HotStuff code is straightforward since RELIC is implemented in C. For FROST, we use the `frost-dalek` library[2] by Lovecruft, Komlo, and Connolly. Since it is written in Rust, integration required the implementation of a compatibility layer, wrapping data structures and function calls. Finally, THECDSA is implemented, using the `multi-party-ecdsa` repository[3] written by the ZenGo-X team [259]. Similar to FROST, the library is written in Rust and a wrapper had to be created.

We define five derived schemes (BLSc, BLSnc, THec, Shnr, Secp) for further study.

Secp

The default HotStuff signatures based on secp256k1, PSigs in a QC are never agglomerated.

BLSc and BLSnc

As shown by Tomescu et al. [260], agglomeration of BLS PSigs for a signer set $\Gamma, |\Gamma| = t$ can be realized through Lagrange interpolation [260]. This requires computation of the "Lagrange coefficients" $\mathcal{L}_i^\Gamma(0) = \prod_{j\in[t],j\neq i} \frac{0-x_j}{x_i-x_j}, |\Gamma| = t, i \in [t]$, i.e., the Lagrange base polynomial $\mathcal{L}_i^\Gamma(x) = \prod_{j\in[t],j\neq i} \frac{x-x_j}{x_i-x_j}, |\Gamma| = t, i \in [t]$ at $x = 0$. Note that the $\mathcal{L}_i^\Gamma(0)$ remain constant for a subset Γ. To reduce the overhead of $\mathcal{L}_i^\Gamma(0)$ computation on every QC agglomeration, we implement a simple cache. Once $\mathcal{L}_i^\Gamma(0)$ has been computed for Γ, is is cached for reuse. The derived scheme *BLSc* implements this caching, *BLSnc* does not.

[1] https://github.com/relic-toolkit/relic
[2] https://github.com/isislovecruft/frost-dalek
[3] https://github.com/ZenGo-X/multi-party-ecdsa

Table 16.3: Abstract Signature Function Interface for Derived Schemes

	Preprocessing		Signing			Verify	
Derived Scheme	**PreRound**	**PreAgg**	**PartSig**	**Agglom**	**Aggreg**	**(P)SigVeri**	**AggVeri**
Secp			●			●	
BLS{c,nc}			●		●	●	
THec	●		●	●			●
Shnr	◐		●	●			●

THec

To prevent THECDSA presigning from limiting HotStuff performance and to ensure we only benchmark the final signature, verification, and agglomeration operations, we assume the presigning data to be calculated and available to all replicas at all times. Hence, in our PoC we reuse e.g., the same commitment shares for each replica, to avoid dynamic computation. This is a simplifying assumption for estimating best-case performance and effectively insecure. Since dedicated presigning data for each signer set Γ is required (§16.2) we further assume $t \sim n$, i.e., all replicas always provide a valid PSig. While THECDSA offers verification of PSigs in principle (§16.2), the software library does not conveniently expose this function. Hence, we omit implementation of single PSig verification. We refer to the resulting derived scheme as *THec*. For a production-oriented implementation, an optimistic *agglomerate-then-verify* approach [212] could be used. Using identifiable abort of THECDSA, misbehavior of partial signers could be detected and kept in a blocklist.

Shnr

The integration of FROST presents similar challenges as encountered for THec. Presigning data needs to be computed for each signing round. For our PoC we follow the THec approach and (1) reuse computed presigning data and (2) fix $t \sim n$. However, the `frost-dalek` library assumes interactive execution and does not implement complete presigning isolation. Hence, these operations are executed on demand upon creation of a PSig and the resulting derived scheme *Shnr* is not effectively semi-interactive.

16.3.2 Function Interface

We summarize signature function interfaces of all derived schemes in Tab. 16.3. Secp is a regular digital signature scheme. BLS{c,nc} support signature aggregation, while we only use the threshold context of (partial) signature agglomeration. THec and Shnr involve a preprocessing phase. Since the underlying `frost-dalek` library does not implement complete presigning isolation, some operations still need to be conducted dynamically. Creation of a QC in both schemes is conducted by PSig creation and subsequent agglomeration. Both schemes only expose verification of agglomerated signatures to HotStuff.

16.4 Complexity Comparison

We compare the processing complexity of signature operations of our derived schemes (§16.3) for a threshold t, required to form a quorum in HotStuff. This calculation *does not* include presigning complexity and operations. Note that effectively $t \sim n$ for Shnr and THec. Tab. 16.4 defines core procedures in HotStuff, that involve cryptographic signature operations (① - ④). In regular HotStuff operation, PSig verification ① is conducted by the incumbent leader upon receiving a replica vote. For the default Secp implementation, verification of a QC for t votes also effectively consists of t PSig verifications. Verification of a QC ② is conducted once the respective decision block is delivered to a replica. A block

Table 16.4: Signature Scheme Operations in HotStuff

①	*Verify PartSig*:	Verify a single partial signature
②	*Verify QC*:	Verify a complete QC
③	*Create PartSig*:	Create a single partial signature on data
④	*Agglomerate QC*:	Agglomerate all partial signatures of a QC

Table 16.5: Operation Complexity of Derived Schemes

Op	BLSnc	BLSc	THec	Shnr	Secp
①	$\mathcal{O}(1)$	$\mathcal{O}(1)$	–	$\mathcal{O}(1)$	$\mathcal{O}(1)$
②	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(t)$
③	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(t)^\star$	$\mathcal{O}(t)^\dagger$	$\mathcal{O}(1)$
④	$\mathcal{O}(t^2)$	$\mathcal{O}(t^2)^\ddagger$	$\mathcal{O}(t)$	$\mathcal{O}(t^2)$	–

*Wrapper incurs small overhead depending on presigning data size in $\mathcal{O}(t)$.
†Since signers $\Gamma \subseteq \mathcal{N}$ are known in advance, each signer $r_i \in \Gamma$ only computes the group commitment and its own $\mathcal{L}_i^\Gamma(0)$ in $\mathcal{O}(t)$ respectively.
‡Only in $\mathcal{O}(t^2)$ upon first agglomeration for signers $\Gamma \subset \mathcal{N}$, before Lagrange coefficients are cached. In $\mathcal{O}(t)$ afterward for efficient caching.

data structure effectively bundles QC, references to previous blocks, client commands to replica, voting information, and more. A PSig is created ③ upon receipt of a valid proposal that a replica wishes to vote on. Finally, the received signature votes for a QC are agglomerated ④ by the incumbent leader, once the required threshold t is reached.

Tab. 16.5 lists the complexity classes of these operations in the respective derived scheme. Secp does not offer signature agglomeration (④), hence QC verification on replicas and size scales with the number of PSigs in the QC (②). Both BLSc and BLSnc shift complexity from QC verification on replicas (②) to agglomeration (④) on the incumbent leader. Calculation of Lagrange coefficients and interpolation results in $\mathcal{O}(t^2)$ for BLSnc. BLSc improves upon this through caching, once the coefficients $\mathcal{L}_i^\Gamma(0)$ for set Γ have been calculated. For THec no PSig verification is implemented (§16.3). Verification of an agglomerated QC by replicas becomes cheaper since only a single signature needs to be verified. THec benefits from full presigning isolation, i.e., in theory, the final PSig creation amounts to a constant operation (Phase 7 [217]). However, the wrapper implementation incurs a dynamic overhead, that scales with the size of the presigning data. Hence final PSig creation and agglomeration are in $\mathcal{O}(t)$ respectively. Shnr also shifts complexity from verification to agglomeration. Without full presigning isolation the `frost-dalek` implementation results in regular PSig creation in $\mathcal{O}(t)$ and agglomeration in $\mathcal{O}(t^2)$.

16.5 Scaling Analysis

We established that the chosen threshold signature schemes generally shift operation complexity from verification to agglomeration (§16.4) and that e.g., BLS has significantly higher base operation cost than our Secp baseline (§16.1.2). This raises the question if and for which parameters there is some quorum size t or scenario for which e.g., BLS can nevertheless outperform the baseline. Such scenarios might be motivated by e.g., lower bandwidth demands through reduced authenticator complexity, re-verification of existing QCs by (joining) peers or external validators, or compression of signatures to save space in a decision history that is not subject to pruning.

We construct a simplified model to roughly approximate the isolated crypto latency cost for a given scheme, depending on the respective quorum size t (§16.5.1). For comparison between schemes, we calculate the difference between latency estimations, including the number of re-verifications ρ as parameter (§16.5.2). While this model does not consider final application level BFT-SMR semantics, it allows an educated guess on the latency induced by cryptographic operations and the target parameter range of potential break-even points in terms of raw crypto operation cost. In §18 we then estimate the base latency of relevant crypto operations in our setup through microbenchmarks and subsequently insert the values into the model described here, to obtain value approximations.

16.5.1 Hot Path Crypto Latency

We focus on the isolated latency of cryptographic operations, that is *necessarily* incurred during the successful processing of a *single phase* on a ((pipelined) batch of) client requests in HotStuff. We refer to this execution path as the *hot path*. Reconsidering the basic HotStuff communication pattern (Fig. 6.1), the hot path hence encompasses all cryptographic operations in e.g., one `COMMIT` or `GENERIC` phase. One phase cycle can be described as

$$\underbrace{\text{QC verification} \rightarrow \text{PSig creation}}_{\text{Executed in parallel on replicas}} \rightarrow \underbrace{\text{PSig add} \rightarrow \text{QC Agg}}_{\text{Executed serially on leader}} \underbrace{(\rightarrow \text{leader QC verification})}_{\text{if done before propose}}$$

QC verification and PSig creation is conducted in parallel on all replicas. Hence, the effective latency for these operations is determined by the $t-$fastest replica, where t is the necessary vote/signature threshold for the leader to proceed. Adding $\geq t$ PSigs to the QC data structure upon incoming votes and finally, agglomeration of $\geq t$ PSigs are executed on the incumbent leader. These operations hence count for their full duration on the leader node. If the leader verifies the agglomerated signature of the created QC before its next proposal, another QC verification latency is incurred. We now formalize this process.

Let t be the minimum threshold required to form a QC in signature scheme s. We use symbols for the operation types of PSig signing (σ), extension of a QC data structure by a PSig (ε), agglomeration of a QC (γ), and signature verification (ν). Then the estimated crypto operation latency $oplat_s(t)$ depending on t and scheme s can be written as one of $oplat_s(t) \in \{\sigma_s(t), \varepsilon_s(t), \gamma_s(t), \nu_s(t)\}$. Note that these functions hide the underlying complexity of the respective operation, which may be of varying order (Tab. 16.5). For example, QC verification $\nu_{\text{secp}}(t)$, even though executed in parallel on all replicas, still requires verification of t independent secp256k1 ECDSA signatures on each replica.

Using this notation, we define the hot path latency of s as

$$lat_s(t) := \nu_s(t) + \sigma_s(t) + \varepsilon_s(t) + \gamma_s(t) \tag{16.1}$$

In case the incumbent leader conducts a complete QC verification before creating the next proposal, $\nu_s(t)$ needs to be counted twice; once for the latency incurred by the replicas checking the previous QC_{i-1}, and once for the leader checking the new QC_i.

16.5.2 Crypto Latency Difference

We compare two signature schemes by computing the difference between the latency cost of their operations. In detail, we are interested in how each of our selected threshold signature schemes (§16.1) performs in comparison to our Secp baseline. For this calculation, we distinguish between verification and creation operations. This allows us to consider the number of signature verifications separately, using a dedicated parameter.

Let ρ denote the number of re-verifications of a QC during our time interval of interest, e.g., $\rho = 1$ for regular operation without leader QC double checking (Eq. 16.1). Then we can write the verification latency diff between threshold scheme s and Secp as

$$v_{\text{diff}}(s, t, \rho) := \rho \left(\nu_s(t) - \nu_{\text{secp}}(t)\right) \tag{16.2}$$

Analogously, the processing time difference can be written as

$$p_{\text{diff}}(s, t) := \sigma_s(t) - \sigma_{\text{secp}}(t) + \varepsilon_s(t) - \varepsilon_{\text{secp}}(t) + \gamma_s(t) - \gamma_{\text{secp}}(t) \tag{16.3}$$

We observe that $v_{\text{diff}}(s, t, \rho) < 0$ and $p_{\text{diff}}(s, t) < 0$, if the latency of Secp operations is larger than those of s operations, for the given t, ρ; or > 0 in the opposite case.

Next, we model the total difference between s and Secp as

$$\begin{aligned} \Delta_s(t, \rho) &:= v_{\text{diff}}(s, t, \rho) + p_{\text{diff}}(s, t) && (16.4)\\ &= \rho \left(\nu_s(t) - \nu_{\text{secp}}(t)\right) && (16.5)\\ &\quad + \sigma_s(t) - \sigma_{\text{secp}}(t) + \varepsilon_s(t) - \varepsilon_{\text{secp}}(t) + \gamma_s(t) - \gamma_{\text{secp}}(t) \end{aligned}$$

Similarly, $\Delta_s(t, \rho)$ is negative if Secp is more costly for t, ρ, positive in the opposite case. This difference function can be interpreted as a 2D-surface in 3D space ($x = t, y = \rho, z = \Delta_s(t, \rho)$). If the surface intersects with the zero-plane $(t, \rho, 0)$ then this results in a function marking the estimated break-even points for some parameter combination t, ρ. In our model, these points would occur for $\Delta_s(t, \rho) = 0$. Hence, let

$$\begin{aligned} \Delta_s(t, \rho) &\overset{!}{=} 0 && (16.6)\\ \Leftrightarrow \quad \rho(\nu_s(t) - \nu_{\text{secp}}(t)) &= -p_{\text{diff}}(s, t) && (16.7)\\ \Leftrightarrow \quad h_s(t) := \rho &= -\left(\frac{p_{\text{diff}}(s, t)}{\nu_s(t) - \nu_{\text{secp}}(t)}\right) && (16.8) \end{aligned}$$

where $h_s(t)$ marks the intersection of $\Delta_s(t, \rho)$ with $(t, \rho, 0)$ for $\nu_s(t) - \nu_{\text{secp}}(t) \neq 0$.

16.6 Summary

We select a set of threshold signature schemes based on BLS, Schnorr, and ECDSA, that allow for practical evaluation at the time of this work, extending the set of studied schemes in context of BFT-SMR. We consider secp256k1 signatures as a baseline default.

For operation in HotStuff, scheme characteristics cause limitations. Overall, BLS is best suited for the context but causes large base operation cost. FROST and THECDSA are potentially cheaper alternatives, assuming that presigning can be executed sufficiently fast.

For further study and later implementation, we choose a software library for each of the selected schemes, and list resulting limitations and characteristics as derived schemes. In particular, we assume the reuse of static presigning data for our integrated BFT-SMR benchmarks and analysis, since the available libraries do not provide suitable or efficient implementations of this process. Shnr additionally does not offer full presigning isolation.

We compare the complexity of typical cryptographic operations of our derived schemes. The threshold signature schemes shift complexity from verification to agglomeration of signatures. Shnr shows elevated complexity due to incomplete presigning isolation.

For some quorum sizes t and re-verification counts ρ, usage of a threshold scheme with larger base cost but agglomeration capabilities may result in an overall reduced latency in comparison to Secp for some scenarios. We create a simple model for approximating the cost of QC-related cryptographic operations. The established formalism can be used to estimate latency differences and potential break-even points between the Secp baseline and a threshold signature scheme by inserting latency values.

17. Experiment Design and Setup

We design experiments to verify and complement our theoretical analysis (§16) and quantify the practical performance impact of the selected threshold signature schemes in context of BFT-SMR. In the following, we outline the rationale behind the experiments, and introduce the experiment design, implementation details, and final setup.

17.1 Rationale

We aim to study the impact and optimization potential of modification of signature schemes in context of BFT-SMR, using HotStuff as a representative system. We are interested in best-case performance estimates to assess basic suitability of our target schemes. Similar to our evaluation of the network transport building block (§10), we try to isolate the impact of signature scheme modification as best as possible. Furthermore, robustness characteristics of schemes require a certain level of participation (§16.3). Hence, we assume all participating replicas to be honest. All communication is conducted over reliable TCP connections without TLS, as studied in Part II. Since some of the selected schemes are (semi-)interactive, we assume successful completion and timely availability of all necessary presigning data at all times, such that HotStuff performance is not limited (§16.3).

Similar to the network transport building block, we assume a permissioned, small to medium-sized BFT-SMR setup (§10.1.2), where preemptive or parallel execution of presigning operations is potentially possible. Our approach toward the experiment environment is analogous to the network transport building block as well (§10.1.3). We evaluate the impact of a real-world full-stack HotStuff deployment on bare-metal hosts in a local testbed. As before, we employ *pos* to automate our measurements and reset all hosts to a well-defined state between experiment runs.

17.2 Experiment Design

Reasoning about potential root causes of observed BFT-SMR system behavior is not trivial due to system complexity, distribution of logic, and responsibilities. Plenty of influence factors impact the typically studied metrics, i.e., replication latency and transaction throughput [5,6,8,37,99,100,102]. To support interpretation of our results, we deconstruct the evaluation process into smaller steps, establishing insight into component behavior. A theoretical analysis (§16) provides the baseline context for result discussion.

We begin our practical evaluation with *microbenchmarks* of the derived schemes for relevant cryptographic operations in HotStuff context. These numbers allow us to verify our complexity analysis (§16.4) as well as provide baseline performance results for further scalability analysis (§18.3) and interpretation of E2E measurements (§18.5).

Reasoning about input request saturation (§10.2) allows to influence bottleneck behavior of our SuT. An upper bound on system throughput is either given by the client input rate (λ_I) or a multiple of the replica decision rate ($b \cdot \lambda_D$). Depending on the required mathematical operations, different signature schemes can exhibit significantly varying processing costs. Hence, λ_D may vary substantially. However, we can adjust the batch size b accordingly to (1) study if similar throughput can be achieved for all schemes and (2), if so, normalize throughput values for comfortable comparison of the target schemes in other metric dimensions. Note, that larger batch sizes also increase the potential impact of delays (e.g., through packet loss) to a larger set of client requests (§9.6). In any case, in every experiment configuration, we choose b such that the replica decision frequency is the initial throughput bottleneck of the system.

We then conduct integrated BFT-SMR measurement in a full HotStuff setup, comparing different signature schemes under varying parameters. Leader-based BFT-SMR and HotStuff in particular (§9.3) shift additional responsibilities, processing, and bandwidth load to the leader node. Hence, it is of interest to study the behavior of and effects on potential bottleneck nodes in more detail. Modification of the underlying signature scheme directly impacts both (1) the message size of replica $\leftrightarrow$ replica communication (e.g., signature and QC sizes) and (2) the processing power required to conduct the necessary crypto operations. Threshold signature schemes generally shift processing complexity from verification nodes to agglomerating nodes (§16.4). In HotStuff, the potential leader bottleneck is thus additionally burdened. For deeper insights, we also investigate bandwidth usage and CPU load on a selection of relevant nodes in our experiments.

17.3 Implementation

Consensus and signature logic are implemented in the `libhotstuff`[1] [147] repository. Modifications of the libhotstuff code are necessary to change the signature scheme.

17.3.1 General Structure

As described in the original HotStuff pre-print [5], HotStuff is meant to use threshold signatures, while the PoC implementation only supports regular secp256k1 ECDSA signatures. However, the original authors anticipated future changes to the codebase and designed the data structures accordingly. There exist abstract parent classes that wrap expected functionality of relevant objects. Keying material is represented in `PrivKey` and `PubKey` classes, PSigs are handled as `PartCert`, while QCs are modeled as `QuorumCert`. Concrete signature schemes then inherit from the abstract classes and implement respective specialized variants. In general, these objects behave as one would expect. A `PartCert` can be created using a private key and verified using a public key. A notable peculiarity in the default implementation is a split of collection and agglomeration of multiple PSigs in context of a QC. That is, the incumbent leader successively "adds" incoming PSigs to the QC data structure until the necessary threshold t is reached. Only then does the leader "compute" the agglomerated QC version.

The PoC implementation also includes a threaded architecture to (optionally) parallelize (1) signature verification and (2) message handling on client and replicas. For (1) libhotstuff provides the `VeriPool` abstraction, implementing a thread pool for parallel verification using a configurable number of workers. Per default, HotStuff operates with a

[1] https://github.com/hot-stuff/libhotstuff

single worker thread. We adopt this setting for our experiments. For (2), the underlying `salticidae`[2] [148] P2P networking library offers a `ConnPool` abstraction. As part of a pool, multiple threads can be dispatched to asynchronously handle read and write requests on a file descriptor. Per default, one worker is available on a replica for inter-replica communication, and up to eight workers are available on the client for communication with the replicas. As before, we adopt these default settings for our experiments.

17.3.2 Scheme Detail

The final signature scheme implementations vary in terms of architecture and features. The default secp256k1 scheme provides a simple implementation of the abstract parent classes, using the Bitcoin `secp256k1` library[3] as backend for raw signature operations. Adding a PSig to a QC results in a simple block hash sanity check and integration into the QC data structure. No agglomeration is implemented upon the QC compute function call. Verification simply checks all appended secp256k1 signatures.

Our HotStuff BLS PoC implements the abstract classes without architectural changes. Core BLS computations are handled by the `RELIC`[4] library. Similar to the Secp baseline, PSigs are added to the QC data structure after a simple sanity check. Agglomeration upon QC compute is implemented in the HotStuff codebase, optionally using Lagrange Coefficient caching. QC verification directly checks an agglomerated signature, if available.

The FROST HotStuff integration is based on `frost-dalek`[5]. Since this library is implemented in Rust, a compatibility layer for usage in HotStuff C++ code was implemented. This results in some deviations and limitations from existing data structures and functions. Each replica is assumed to work on completed presigning data for $t \sim n$ that is kept as preloaded state in the wrapper library. Hence, signing and verification functions call into and work on this data in the wrapper. Addition of PSigs to a QC effectively modifies data structures in the wrapper. Upon QC computation, the wrapper is called again and available PSigs are agglomerated. Due to incomplete presigning isolation, Lagrange Coefficients are recomputed dynamically in frost-dalek. QC verification then amounts to a direct check of a combined signature against the group key in the wrapper. Verification of PSigs is conducted as part of the agglomeration process and not exposed to HotStuff.

Finally, the THECDSA HotStuff integration is based on the `multi-party-ecdsa`[6] Rust library. Similar to FROST, a wrapper was created to enable usage in the HotStuff C++ codebase, and completed presigning data for $t \sim n$ is loaded into and then mostly kept in the wrapper library, which handles incoming calls from HotStuff code. PSig addition and QC computation are realized via explicit calls into the wrapper, in which the signature data is attached and agglomerated, respectively. Verification of an agglomerated QC signature is directly handled in the wrapper. Verification of single PSigs is conducted before agglomeration and not exposed as a function to HotStuff logic.

17.4 Experiment Setup

Our experiment setup is largely equivalent to the setup used for evaluation of the network transport building block as specified in §10.3. Employed hardware, number of machines, topology, available bandwidth, inter-node latency, software dependency versions (e.g., Linux Kernel, OpenSSL), automation via *pos*, all remain the same. Latency and throughput values are again collected via the HotStuff example client application and

[2] https://github.com/Determinant/salticidae
[3] https://github.com/bitcoin-core/secp256k1/tree/1e6f1f5ad5e7f1e3ef79313ec02023902bf8175c
[4] https://github.com/relic-toolkit/relic
[5] https://github.com/isislovecruft/frost-dalek
[6] https://github.com/ZenGo-X/multi-party-ecdsa

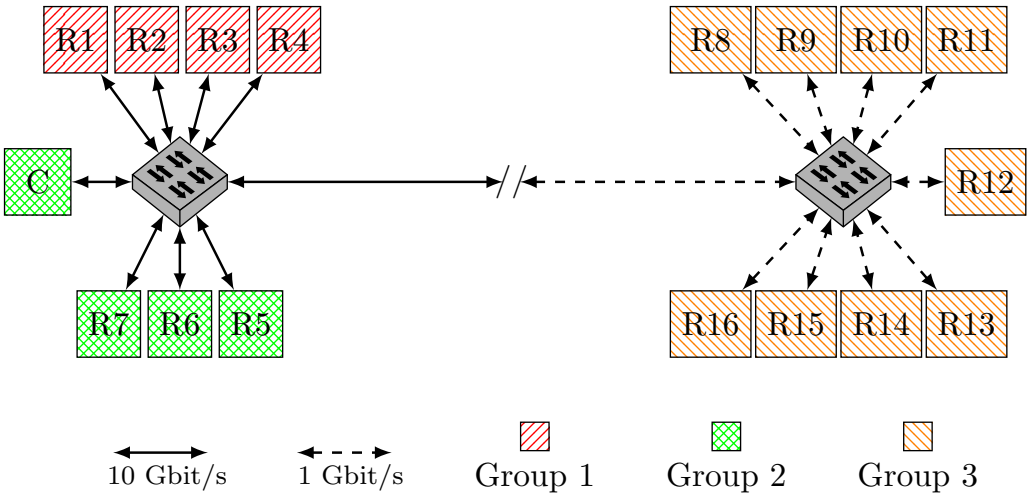


Figure 10.1: Testbed Hardware Topology

then postprocessed by our automation scripts. Reuse of this setup has the additional advantage that already collected insights into the network transport building block operation and baseline TCP performance (§11) apply to the new experiment setting. The core difference lies in the executed HotStuff versions. We are now running the replicas using the threshold signature scheme integrations. The execution of crypto operation microbenchmarks is conducted on a machine of the hardware specification from Group 2. For convenience, we include a copy of the testbed topology figure (Fig. 10.1) here.

18. Evaluation

We evaluate the impact of cryptographic signature scheme modification in a series of experiments. Experiments are executed using a single client and a single leader for replica numbers of $n = 3f + 1, f \in [1, 5], n \in [4, 16]$, to better isolate their potential role influences. Communication between all nodes is conducted over plain TCP connections without TLS. In the following, we introduce our measurement setup (§18.1) and evaluate our implementation in a series of experiments. We establish a simple performance baseline of cryptographic operations in HotStuff context via microbenchmarks (§18.2) and insert them into our simplified latency model to extrapolate potential scaling behavior (§18.3). In preparation for integrated BFT-SMR performance measurements, we then study the request saturation properties of all derived schemes (§18.4). E2E BFT-SMR performance measurements (§18.5) are then conducted over normalized saturation. Finally, we investigate bandwidth and processing load metrics to reason about underlying node behavior and potential causes for observed performance differences (§18.6).

18.1 Measurement Setup

For each of the target evaluation categories, we vary a set of parameters, including BFT-SMR parameters typically studied in literature (§12). We briefly summarize all parameters, their notation, and immediate effect in Tab. 18.1. Detailed introduction of relevant parameters is conducted in the respective subsection, if not done already.

We execute the crypto operation microbenchmarks (§18.2) on a host of hardware Group 2 (§10.3). For each operation, 1000 samples are collected and averaged into a single data point. For our integrated full-stack HotStuff measurements, we limit variation of the parameter space to combinations that provide relevant insights in our context for brevity. Hence, we generally fix all but one variable parameter to a default value. As for the

Table 18.1: Overview of Variable Experiment Parameters

Parameter	**Symbol**	**Effect**
Replica Number	n	Number of participating replicas, incl. leader
Batch Size	b	Number of `CMD-REQ` considered in one proposal
Payload Size	p	Size of `CMD-REQ`/`CMD-ANS` messages
Signature Scheme	e.g., Secp	Derived signature scheme in use for votes/QCs

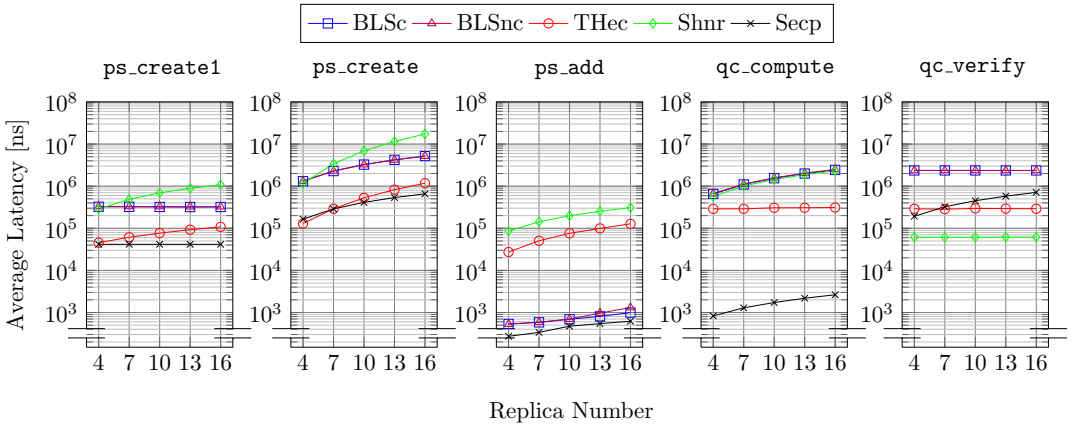


Figure 18.1: Microbenchmarks, Hardware Group 2, Replica Scaling

evaluation of the network transport building block (§11), we study E2E client replication latency and throughput as central metrics. For deeper insights into bottleneck behavior, we additionally study bandwidth usage and relative CPU load of relevant nodes.

We run a single experiment for 30 s, executing at maximum possible replication speed, i.e., as limited by either client or replicas (§10.2). If not stated otherwise, result data is filtered for outliers, discarding all values outside $[Q_1 - 1.5 \cdot IQR, Q_3 + 1.5 \cdot IQR]$, where Q_1, Q_3, and IQR refer to the first quartile, third quartile, and interquartile range, respectively. If not stated otherwise, result data is averaged into a single data point. Where an average is insufficient to explain the results, we provide additional details on demand.

18.2 Microbenchmarks

We conduct microbenchmarks of our derived schemes in order to verify our complexity analysis (§16.4), acquire approximations for use in scaling behavior estimation (§18.3), and to provide a baseline for integrated HotStuff measurements. Latency is recorded for central node operations in HotStuff. We analyze the creation of a single PSig on a proposal (`ps_create1`), the creation of n PSigs on a proposal (`ps_create`), inclusion of t PSigs from votes into a QC (`pc_add`), agglomeration of t signatures (`qc_compute`), and QC verification (`qc_verify`). All operations are called from a HotStuff code context, using the same functions and data structures from regular HotStuff operation.

Fig. 18.1 shows the average operation latency in ns, for different (t, n) where $n = 3f + 1$, $t = n - f = 2f + 1, f \in [1, 5]$. Note that we fix $t :\sim n$ for THec and Shnr (§16.3).

While `ps_create1` represents the creation of a single PSig, Shnr and THec latencies still scale with t. This is due to dynamic wrapper overhead related to the size of presigning data (THec) and incomplete presigning isolation (Shnr) (§16.4). Secp and BLS incur constant overhead, where BLS is almost an order of magnitude more expensive than Secp.

For `ps_create` n PSig signatures are created instead of one. The measured values follow the aforementioned complexities, additionally scaling with n. Hence, creating a larger number of signatures n naturally results in increased signature time. Both BLS schemes perform equally since no Lagrange coefficients are involved in this step. In general, BLS incurs a significant static cost for the isolated signing operation.

In `ps_add`, BLS and Secp simply move signature data between data structures. Both Shnr and THec incur overhead by data passing and (de-)serialization between the wrapper library and HotStuff. Additionally, the Shnr wrapper loads agglomerator presigning data for the first add operation per signer set, resulting in a one-shot constant latency overhead.

For `qc_compute` Secp does not implement any action in the overloaded function. Aggregation in THec scales linearly but significantly better, due to full presigning isolation (§16.4) and PSig data availability in the wrapper. The Shnr wrapper also has all PSigs but not the complete presigning data available. Hence, agglomeration requires dynamic calculation of binding factors, commitments, and Lagrange coefficients. This overhead results in similar performance and scaling behavior as the BLS schemes. The difference between BLSc and BLSnc is insignificant for the studied signer set sizes and amounts to a maximum of $\sim 3\%$.

For `qc_verify` a variation of t has insignificant effects on validation time for the studied threshold signature schemes, since a QC verification amounts to the check of a single agglomerated signature. Latency is mostly dominated by scheme operation cost, static execution costs of HotStuff and wrapper environments. For Secp, where sequential verification of $\mathcal{O}(t)$ PSigs is conducted, we observe a linearly scaling latency. Secp is outperformed by Shnr verification for $t \geq 4$, by THec verification for $t \geq 7$, and bound to be outperformed by BLS eventually. We model potential break-even points in more detail in §18.3. Shnr generally shows the best performance, THec involves more costly wrapper overhead, and the BLS schemes have the highest base operation cost. We find that in terms of isolated verification operation cost, the Secp baseline is quickly outperformed by its competitors for larger QC sizes, even though Shnr and THec operate under additional constraints ($t \sim n$).

18.3 Scaling Extrapolation

In §16.5 we constructed a simplified latency model. The model takes the minimally necessary threshold t of PSigs for QC creation and the number of re-verification operations ρ as input parameters. Using the model, we can extrapolate latency scaling of HotStuff QC crypto operations of different signature schemes. The resulting difference enables an educated guess on the rough scaling behavior of the target schemes and possible existence of break-even points. For brevity, we only consider BLSc in this estimation.

18.3.1 Limitations

We note that our model is subject to significant limitations. The model only accounts for the latency of raw crypto operations (in the HotStuff codebase) and does not account for latencies or dependencies of e.g., higher-level logic or (BFT-SMR) context. The model also does not consider the latency incurred by presigning for schemes in which a presigning phase is necessary for final signing. Since presigning can incur non-negligible processing and communication complexity [217, 218, 239] and hence growing overhead with increasing player numbers, a suitable estimation of presigning costs is necessary for a more comprehensive projection. Unfortunately, the signature scheme libraries available at the time of development are severely limited (§16.3). If (complete) presigning is implemented, it is done in a non-optimized manner, effectively unsuitable for obtaining realistic benchmarks. We consider a more detailed study of practical presigning performance and scaling as future work. Notwithstanding these limitations, the model projections may potentially assist in understanding measurement results observed in BFT-SMR integrated measurements.

18.3.2 Application

We use latency values obtained from the microbenchmarks (§18.2, Fig. 18.1) to estimate scheme operation base latencies. That is, for each signature scheme s we acquire estimates of the respective $oplat_s(t) \in \{\sigma_s(t), \varepsilon_s(t), \gamma_s(t), \nu_s(t)\}$ functions. Considering our reasoning behind hot path latency (§16.5.1) we assume that QC verification (`qc_verify` $\rightarrow \nu_s(t)$) and PSig creation latencies are parallelized on replicas (`pc_create1` $\rightarrow \sigma_s(t)$), while extension of a QC data structure with $\geq t$ PSigs (`ps_add` $\rightarrow \varepsilon_s(t)$) and agglomeration of $\geq t$ PSigs into a QC (`qc_compute` $\rightarrow \gamma_s(t)$) are conducted serially on the leader. For calculation of

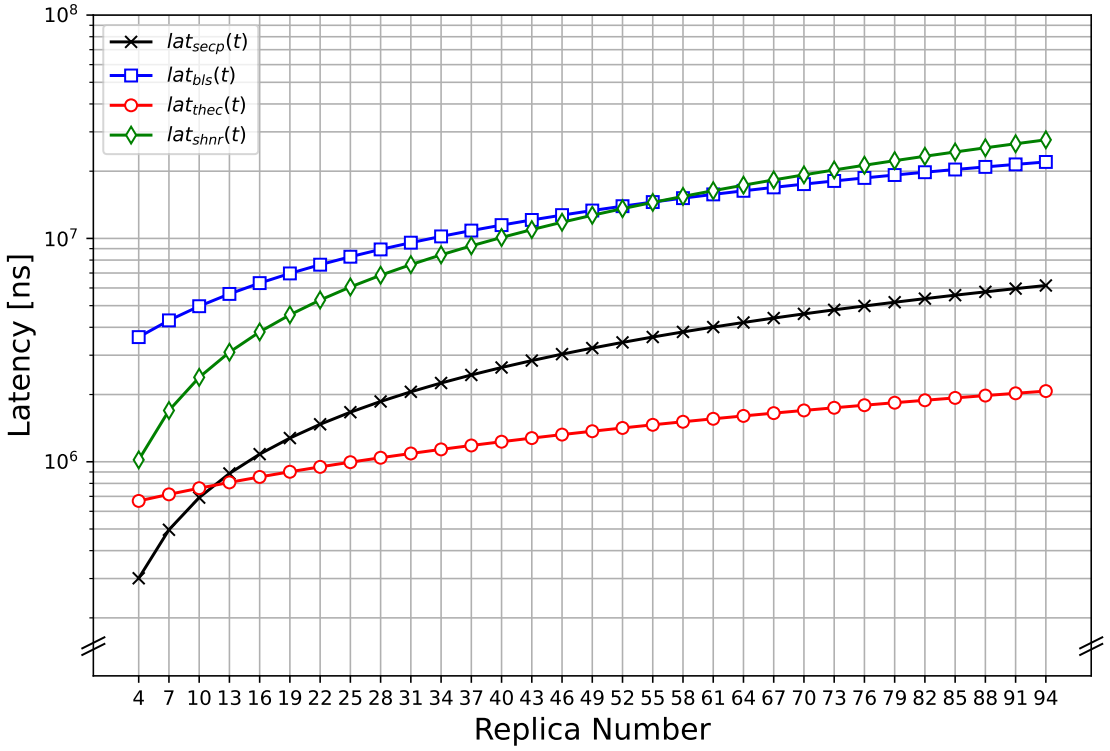


Figure 18.2: Extrapolated $lat_s(t)$ for all Derived Signature Schemes s

all $oplat_s(t)$, we consider the derived scheme base complexities (§16.4) and fit a polynomial function of the correct order through the microbenchmark measurement samples, or simply return the average of all samples in case the function is in $\mathcal{O}(1)$.

18.3.3 Hot Path Crypto Latency

In §16.5.1, we defined the hot path crypto latency $lat_s(t)$ of signature scheme s (Eq. 16.1) as the sum of all crypto operation latencies that are incurred during the successful processing of a single phase in HotStuff. Hence, $lat_s(t)$ offers an approximate lower bound on the phase latency of a deployment. By evaluating $lat_s(t)$ for a range of quorum thresholds t, we can extrapolate the expected minimum latency for larger replica numbers.

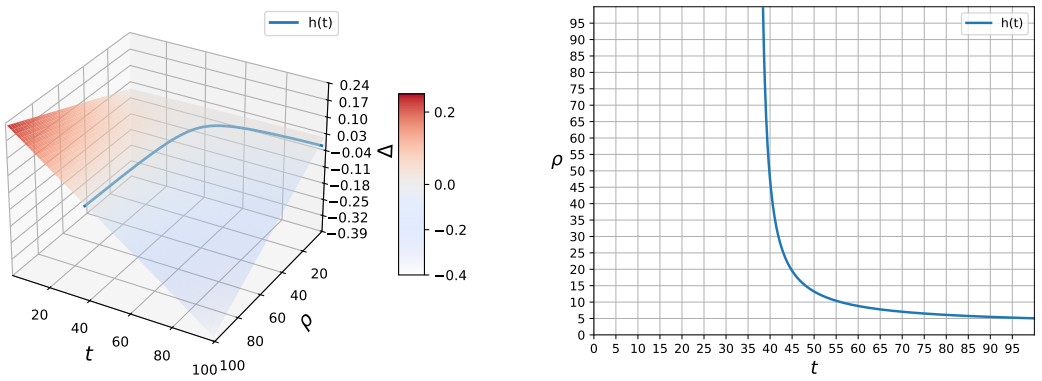
We plot $lat_s(t)$ for all schemes in Fig. 18.2. The x-axis denotes the total replica number $n, n = 3f + 1, f \in [1, 32]$, for which each derived scheme s infers its respective minimum threshold t (§16.3). On the y-axis, the extrapolated hot path latency is plotted in ns.

The Secp baseline demonstrates the expected lowest base cost (Fig. 18.1) but $lat_{\text{secp}}(t)$ grows quickly for larger n, due to verification complexity in $\mathcal{O}(t)$ (Tab. 16.5).

BLS has the highest initial $lat_{\text{bls}}(t)$ due to large base operation cost, except for adding PSigs to a QC. While verification cost is expensive and constant, aggregation is expensive and polynomial. This combination keeps BLS one of the most expensive options.

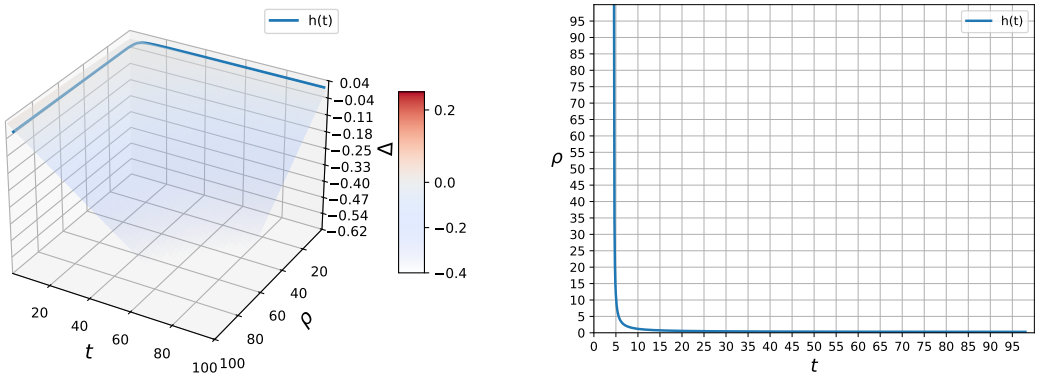
While the theoretical base cost of Shnr is lower than BLS, its effective cost is significant due to missing presigning isolation. Shnr mainly excels in low verification cost. The resulting $lat_{\text{shnr}}(t)$ is initially smaller than lat_{bls} but grows faster due to higher complexity.

THec signature cost is comparable to Secp, but scales worse due to wrapper overhead. PSig adding and agglomeration base costs are significant but agglomeration scales well. THec QC verification is constant and quickly outperforms Secp. This results in a higher initial lat_{thec} than lat_{secp}, but improved scaling behavior. Overall, the extrapolation projects THec to outperform Secp for $n > 10$ and BLS to outperform Shnr for $n > 55$.



(a) $\Delta_{\text{bls}}(t, \rho)$, $h_{\text{bls}}(t)$ (b) $h_{\text{bls}}(t)$

Figure 18.3: Approximated Crypto Latency Difference between Secp and BLS



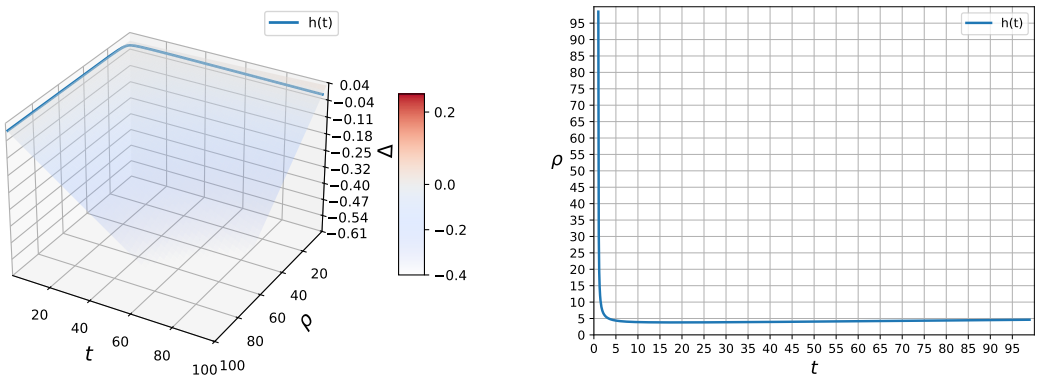
(a) $\Delta_{\text{thec}}(t, \rho)$, $h_{\text{thec}}(t)$ (b) $h_{\text{thec}}(t)$

Figure 18.4: Approximated Crypto Latency Difference between Secp and THec

18.3.4 Crypto Latency Difference

Besides the basic hot path latency, we can also consider the combined impact of quorum threshold t and re-verification count ρ. To better visualize projected break-even points between a target threshold signature scheme s and Secp, we calculate the total latency difference $\Delta_s(t, \rho)$ (Eq. 16.5). The resulting intersection with the zero-plane $h_s(t)$ marks the estimated break-even points for the parameter combination t, ρ. We visualize $\Delta_s(t, \rho)$ (subfigures (a)) and $h_s(t)$ (subfigures (b)) for BLS (Fig. 18.3), THec (Fig. 18.4), and Shnr (Fig. 18.5). In subfigures (a), the resulting $\Delta_s(t, \rho)$ value is additionally color-coded, where a red area marks a parameter combination t, ρ for which threshold scheme s is projected to be *more* expensive than Secp, and a blue area marks the opposite. $h_s(t)$ marks the intersection for $\Delta_s(t, \rho) = 0$ in subfigures (a), and is plotted into the 2D-plane for convenience and details in subfigures (b). For all three threshold schemes, we observe significant latency differences in the respective parameter space.

For BLS, the area of projected break-even points consistently starts only at higher t. An increase of ρ effectively amplifies the existing verification cost difference between the target scheme and Secp. Assuming an increased ρ, then for low t the BLS verification is more expensive, for higher t Secp verification is. For $\rho = 1$, as plotted in $lat_{\text{bls}}(t)$, no break-even point is projected since the aggregation cost of BLS grows fast enough to diminish the verification benefit for large t. Larger ρ emphasize the BLS verification benefits for large t, but projected break-even points occur only for large base values of one of the parameters. Other works that study BLS in context of BFT-SMR partially identify break-even points,



(a) $\Delta_{\text{shnr}}(t, \rho)$, $h_{\text{shnr}}(t)$ (b) $h_{\text{shnr}}(t)$

Figure 18.5: Approximated Crypto Latency Difference between Secp and Shnr

but attribute them to e.g., bandwidth efficiency in an E2E measurement instead of raw cryptographic cost [108]. Cheng et al. [212] similarly assess BLS threshold signatures to perform generally worse than Secp without clear break-even point in a one-shot consistent broadcast microbenchmark. Finally, Li et al. [211,261] do not attribute significant benefits to an EdDSA-based HotStuff implementation over a BLS Multi-Signatures (BLS-MS)-based HotStuff implementation in integrated E2E measurements. Hence, we do not expect a break-even between BLS and Secp for hot path latency except for special scenarios.

For THec break-even points are projected to be reached quickly, with increased improvements for growing t, ρ. For $\rho = 1$ a break-even is projected for $n \in [10, 13]$, as already anticipated by $lat_{\text{sec}}(t), lat_{\text{thec}}(t)$ (Fig. 18.2). Increasing ρ reduces the necessary t for projected break-even in the range of $t \in [5, 10]$.

Break-even points for Shnr are projected to not be reached for $\rho <\sim 5$, as anticipated (Fig. 18.2). Due to incomplete presigning isolation the latencies of most Shnr operations grow quicker than Secp. For $\rho \geq\sim 5$ projected break-even points are quickly reached since the Shnr verification improvement over Secp is significant and quickly amplified.

18.4 Request Saturation

We now evaluate the saturation behavior (§10.2) for all implemented HotStuff variations. To obtain a batch size impact baseline, we limit the saturation experiments to $n = 4$ and payload size $p = 0\,\text{B}$ since increasing n and p can affect both λ_I and λ_D. We vary λ_I by running multiple HotStuff client applications on our dedicated client machine. Bandwidth limits were not reached for any experiment on the client. A single client does not parallelize request sending. Distribution of clients to multiple machines is expected to yield comparable results if CPU power is comparable and no bandwidth limits are reached.

18.4.1 Saturation Points

Fig. 18.6 shows *average* latency (left) and *cumulative* throughput (right) over all client applications in relation to batch size, for Secp. We identify input request saturation points P_s around batch sizes $b = \{50, 100, 200, 200\}$ for client numbers $c = \{1, 2, 3, 4\}$ respectively. For $b < P_s$, λ_D is the bottleneck, resulting in a throughput increase proportional to b and a small increase in latency. For $b \geq P_s$, throughput remains generally constant, but now latency increases proportional to b. More client applications result in larger λ_I, causing relatively higher latency below their respective P_s since the replica processing speed is the bottleneck. Around their respective P_s, however, a larger overall throughput is reached.

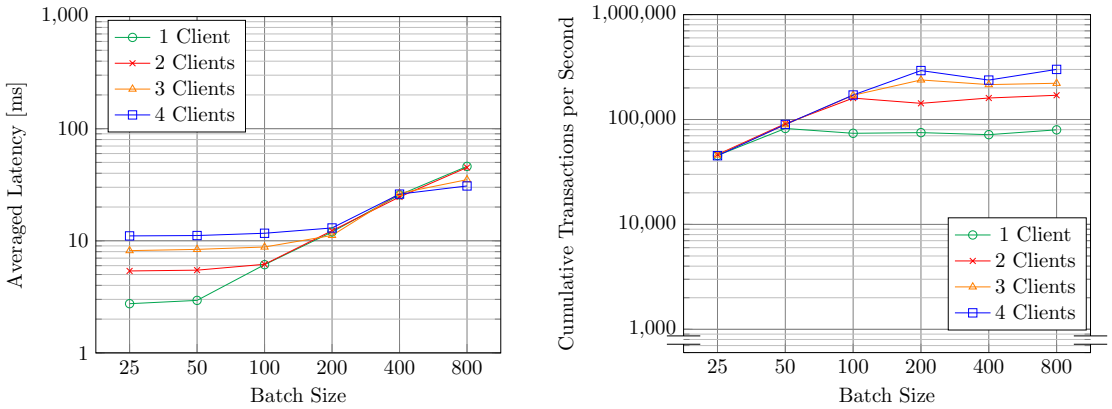


Figure 18.6: Saturation over Batch Size, Secp, $p = 0\,\mathrm{B}, n = 4$

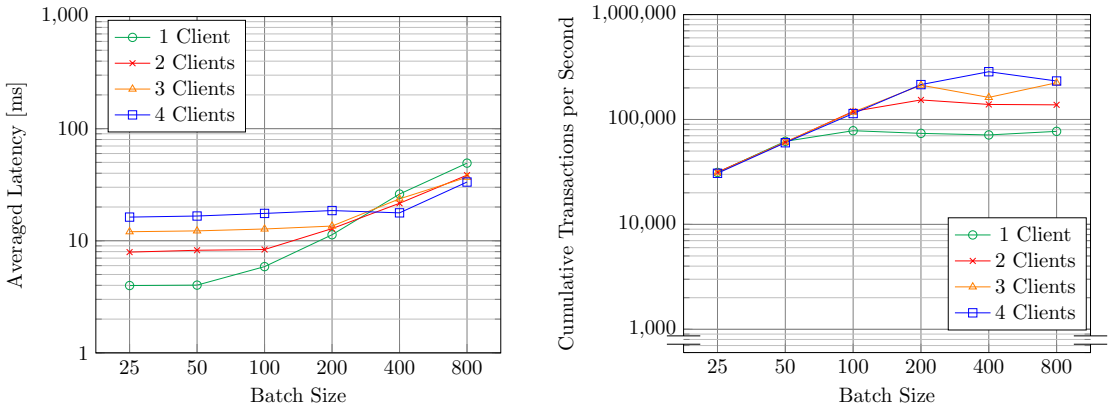


Figure 18.7: Saturation over Batch Size, THec, $p = 0\,\mathrm{B}, n = 4$

We show the average latency and cumulative throughput over all client applications for the remaining protocols in Fig. 18.7 (THec), Fig. 18.8 (Shnr), Fig. 18.9 (BLSnc), and Fig. 18.10 (BLSc). We observe that the general form of saturation behavior stays the same, albeit the value of the respective P_s varies significantly depending on the target signature scheme. While saturation for THec and Shnr occurs around $b \sim 50$ and $b \sim 200$ for a single client, respectively, the BLS-based schemes only saturate for for $b \sim 800$. We do not observe significant differences between BLSc and BLSnc.

We summarize the approximate P_s of our setup for a given number of client applications c for all derived schemes in Tab. 18.2. The table is sorted by ascending P_s. Immediately, we observe the relationship between the signature base operations cost (§18.2) and P_s, e.g., Secp has much smaller P_s than the BLS-based schemes. Costly signature operations

Table 18.2: Approximate Batch Size Saturation Points (P_s)

Protocol	$c = 1$	$c = 2$	$c = 3$	$c = 4$
Secp	50	100	200	200
THec	50	100	200	200
Shnr	200	200	400	800
BLSc	800	1600	3200	6400
BLSnc	800	1600	3200	6400

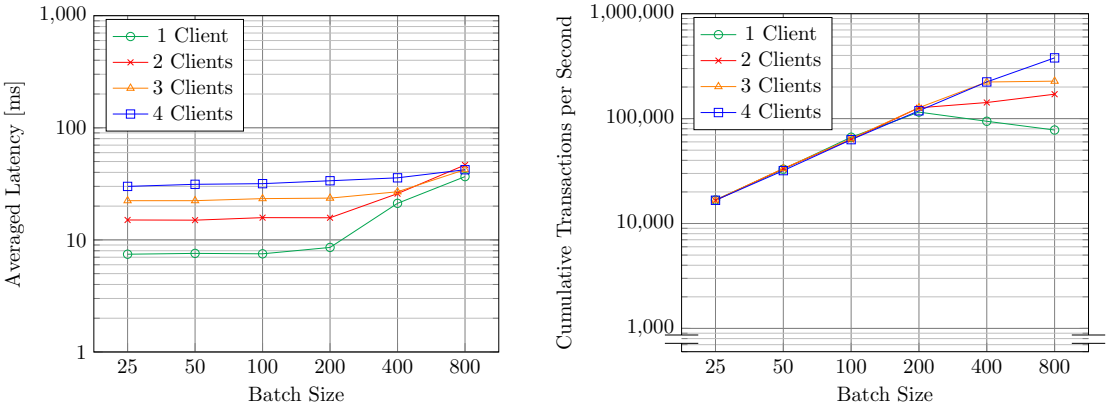


Figure 18.8: Saturation over Batch Size, Shnr, $p = 0\,\mathrm{B}, n = 4$

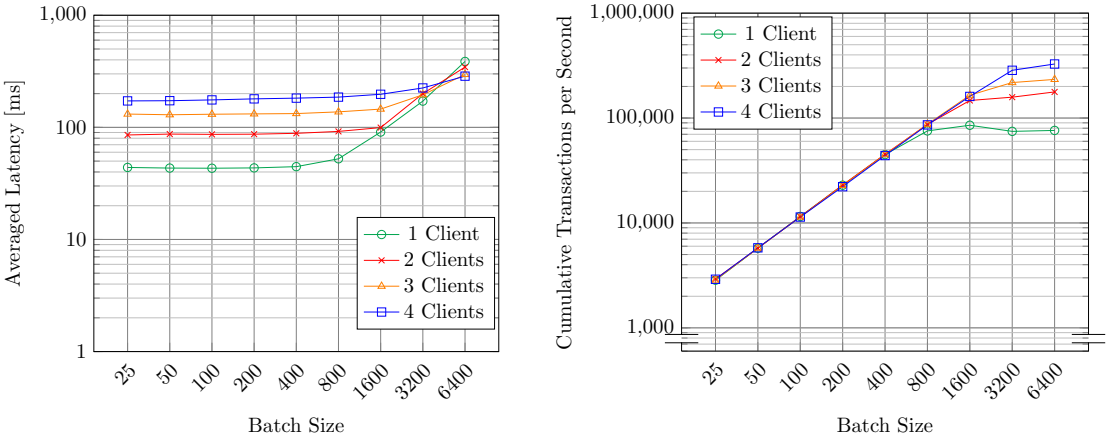


Figure 18.9: Saturation over Batch Size, BLSnc, $p = 0\,\mathrm{B}, n = 4$

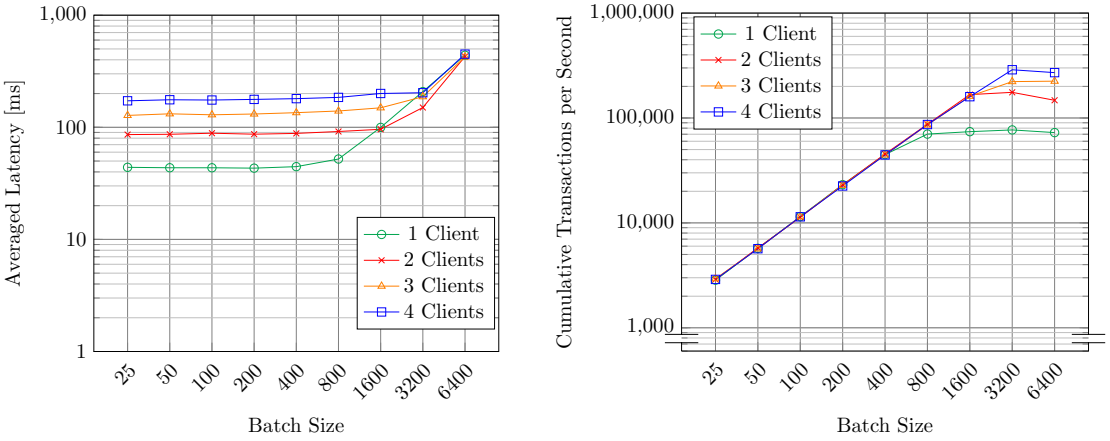


Figure 18.10: Saturation over Batch Size, BLSc, $p = 0\,\mathrm{B}, n = 4$

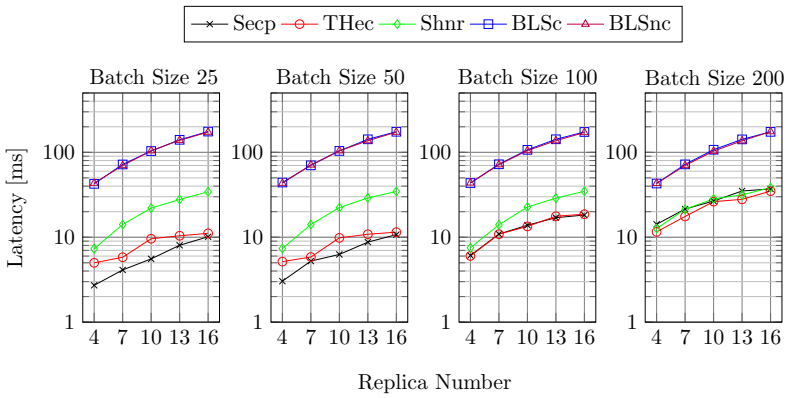


Figure 18.11: HotStuff Latency over Replicas, Batch Sortation, $p = 0\,\mathrm{B}$

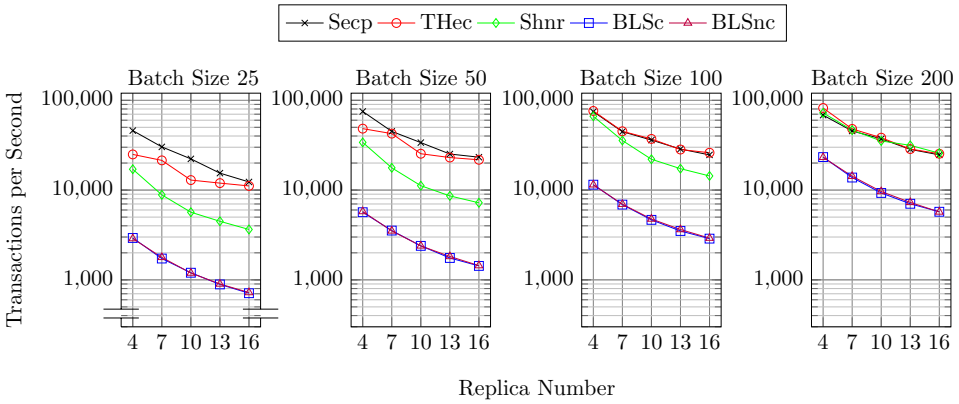


Figure 18.12: HotStuff Throughput over Replicas, Batch Sortation, $p = 0\,\mathrm{B}$

increase the HotStuff decision latency on a batch of b requests, reducing the effective λ_D. To achieve saturation ($\lambda_I \sim b{\cdot}\lambda_D$) with constant λ_I, a larger batch size $b \sim P_s$ is necessary.

18.4.2 Normalization Approach

We can observe the immediate effect of different batch sizes on the performance of integrated HotStuff E2E measurements. For emphasis, we show some exemplary results to discuss batch size impact without discussing scheme-specific causes for the observed values. We conduct this more detailed evaluation in §18.5. In Fig. 18.11 we plot the E2E client replication latency over replica number, grouped by batch size, with $p = 0\,\mathrm{B}$ for all derived schemes. Fig. 18.12 shows the transaction throughput data from the same measurement. Let us first consider $b = 25$. The batch size is sufficiently small to ensure that the replicas are the bottleneck for all derived schemes, hence, we observe significant differences in latency and throughput. While we aim to keep the SuT bottleneck at replica side by keeping b sufficiently small, from our saturation analysis (§10.2, §18.4) we know that there exists potential for throughput increase for schemes with larger P_s (e.g., BLSc).

However, if we consistently increase the batch size for all signature schemes then the observed performance for some schemes will be bottlenecked by the client and not the replicas anymore. For this, consider the performance for $b = 200$. The effective throughput of the BLS-based schemes increased by close to an order of magnitude, but all other schemes converged to similar latency and throughput values, as limited by the client.

For a fair comparison between our derived schemes, we would like to ensure (1) that the SuT performance is bottlenecked by the replicas while (2) the respective possible throughput is maximized. In this scenario, we can study all schemes operating close to

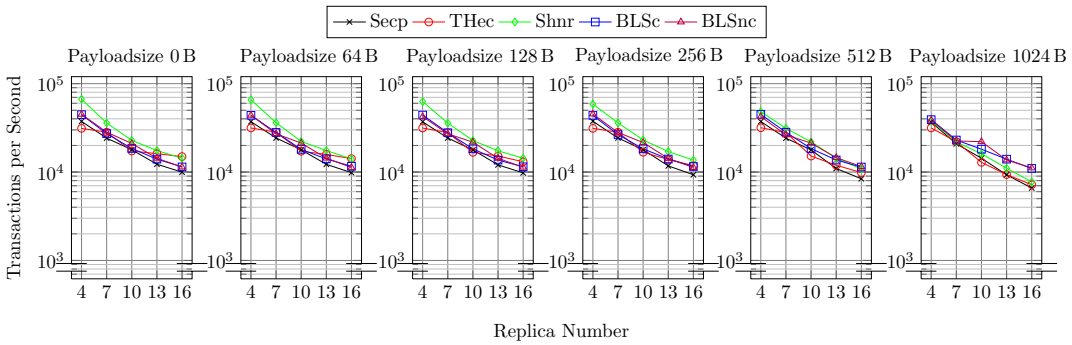


Figure 18.13: HotStuff Throughput over Replicas, Saturation Normalized

their respective "optimal" throughput, while reducing load generation artifacts. We achieve this by choosing a suitable batch size for each derived scheme based on its P_s.

18.5 Normalized Saturation Performance

By choosing b around a respective, approximate saturation point P_s for a derived scheme, we normalize the performance characteristics of the measurement data along the lines of system throughput (§10.2). This representation simplifies identification of differences in other metrics and system bottleneck shifts around the respective P_s. For the following figures, we normalize the following datasets around $b \sim \frac{P_s}{2}$ of a derived scheme for $c = 1$ (Tab. 18.2). Hence, data for e.g., Secp is provided for $b := \frac{50}{2} = 25$. By deliberately choosing $b < P_s$, we ensure that $\lambda_T \sim min(\lambda_I, b \cdot \lambda_D) = b \cdot \lambda_D$, so the replicas run at maximum decision frequency. This way we can observe shifts in λ_I, λ_D, and saturation, that are not initially related to b or client limitations, but rather to p and n.

18.5.1 Sortation over Replica Number

Fig. 18.13 shows subpanels of throughput over replica number n for all derived schemes. Panels are grouped by payload size p. All results are normalized by saturation. Throughput for all derived schemes is generally similar and follows a common trend of decrease for larger n. Larger p further reduces throughput differences between schemes for smaller n and absolute throughput for larger n. Approximation errors of P_s, choice of $b < P_s$, and variance contribute to the observed throughput differences between schemes. Larger p shifts processing and bandwidth strain toward the client (§18.6), limiting λ_I. This reduces throughput as well as inter-scheme differences if $\lambda_I < b \cdot \lambda_D$. As already argued and observed for the transport protocol building block (§11.3, Eq. 11.2) the observed throughput reduction for growing replica numbers is also related to the quorum size, caused by elevated processing and communication overhead in-between replicas and client. Usually (e.g., using Secp), a HotStuff leader can progress as soon as $h(f) = (n - f) = 2f + 1$ correct votes arrive. The measured multiplicative throughput decline for increasing n for Secp, $p = 0, (\sim 0.65, 0.74, 0.69, 0.81, \ldots)$, follows the series of ratios

$$\frac{h(f)}{h(f+1)} = \frac{2f+1}{2(f+1)+1}, f \in [1, 2, \ldots] \rightarrow (\sim 0.6, 0.71, 0.78, 0.81, \ldots) \tag{18.1}$$

with some variance. Shnr looks slightly different since the leader always requires all signatures. Hence, $h_s(f) = n = 3f + 1$. Consequently, the measured multiplicative throughput decline for increasing n, for Shnr, $p = 0, (\sim 0.54, 0.64, 0.77, 0.82, \ldots)$ follows

$$\frac{h_s(f)}{h_s(f+1)} = \frac{3f+1}{3(f+1)+1}, f \in [1, 2, \ldots] \rightarrow (\sim 0.57, 0.7, 0.77, 0.81, \ldots) \tag{18.2}$$

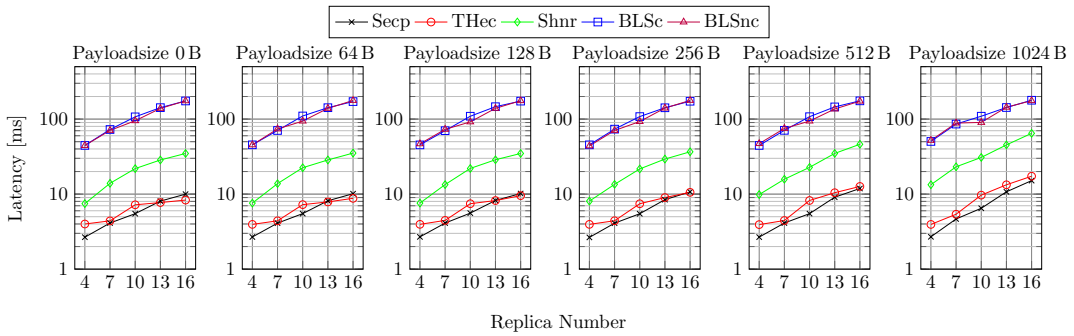


Figure 18.14: HotStuff Latency over Replicas, Saturation normalized

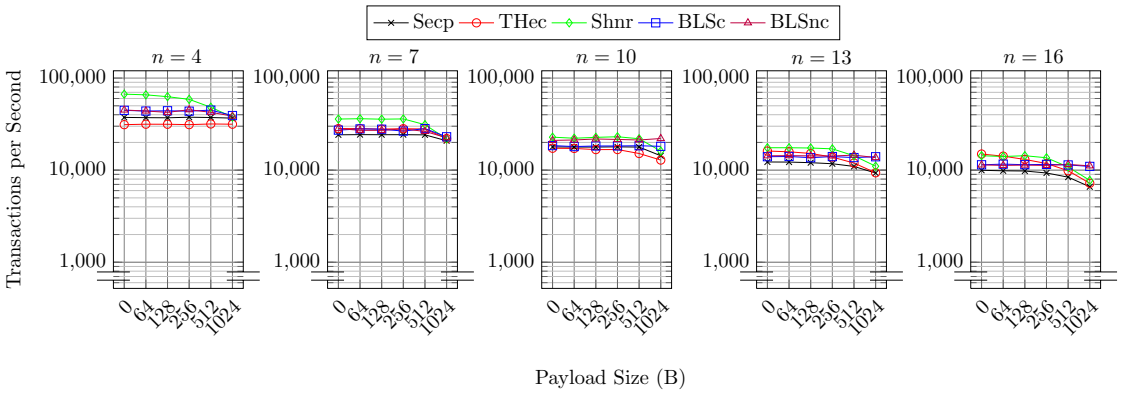


Figure 18.15: HotStuff Throughput over Payload Size, Saturation Normalized

Fig. 18.14 shows the same dataset as Fig. 18.13, but with latency on the y-axis instead. Latency follows a similar growth, with a strong increase for smaller n. The curve is related to the ratio of active quorum participants, as already discussed for Fig. 18.13. However, absolute latency differs significantly, up to an order of magnitude (Secp, BLSc). BLS shows the largest, Secp the smallest values. This is due to diverging crypto base operations cost (§18.2). Additionally, the BLS-based schemes display less throughput reduction for large p and n, since their performance is dominated by leader CPU load (§18.6). We observe a small jump in the latency of THec for $n := 7 \to 10$. Since $t \sim n$ for THec, votes from nearly all replicas need to be collected by the leader to progress. For $n > 7$ we employ additional, less powerful hosts from Hardware Group 3 (§17), which take longer to respond with a vote and hence increase the decision latency slightly. For larger p, the behavior is less significant due to a bottleneck shift toward the client.

18.5.2 Sortation over Payload Size

Fig. 18.15 and Fig. 18.16 showcase the same dataset as Fig. 18.13 and Fig. 18.14, but with an adjusted sortation to better visualize the impact of payload size variation. As argued before, throughput values are roughly but not perfectly normalized due to measurement variance, as well as saturation point choice and approximation errors. We observe the discussed reduction in throughput (Fig. 18.15) for larger replica numbers due to increased complexity in client communication and consensus. For increasing payload values the figures highlight the payload size value for which the bottleneck is shifted toward the client (e.g., $n = 13, 256\,\mathrm{B}$, for all schemes except BLS). The BLS-based schemes incur significantly increased strain on the leader node, delaying the bottleneck toward the client, even for higher p. The respective replication latency (Fig. 18.16) similarly follows the described trends. A bottleneck shift toward the client for larger n, p is also visible in the

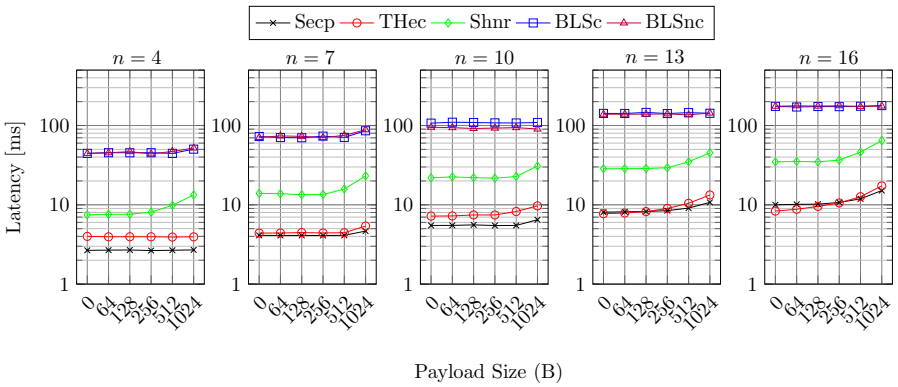


Figure 18.16: HotStuff Latency over Payload Size, Saturation Normalized

latency. The derived scheme latency differences are necessarily analogous to the already discussed representation of the measurement data over replicas (Fig. 18.14).

18.6 Bandwidth and Processing Load

To reason about load distribution and bottlenecks of the normalized performance results we analyze bandwidth and CPU load. These metrics were collected during the previously presented experiment runs and are part of the same dataset as the results from §18.5. Hence, the presented values are also normalized around $b \sim \frac{P_s}{2}$ of the respective derived scheme. We study (1) the bandwidth *peak* ingress (RX) and egress (TX) values to reason about the maximum required link capacity, and (2) the relative CPU time (*%CPU*) of our target process (group), e.g., the `hotstuff{-app,-client}` application, averaged over the experiment. %CPU can be interpreted as the relative share of total time a CPU spent processing, for a target process, as exposed by `top`. In case of multithreaded processes values larger than 100% are shown. These metrics only provide a high-level view of system behavior and are subject to influence by many different factors. Hence, the metrics are not as precise as data from e.g., Linux *perf*, as visualized in *FlameGraphs* [262]. However, they allow to identify rough trends that can be investigated further upon demand.

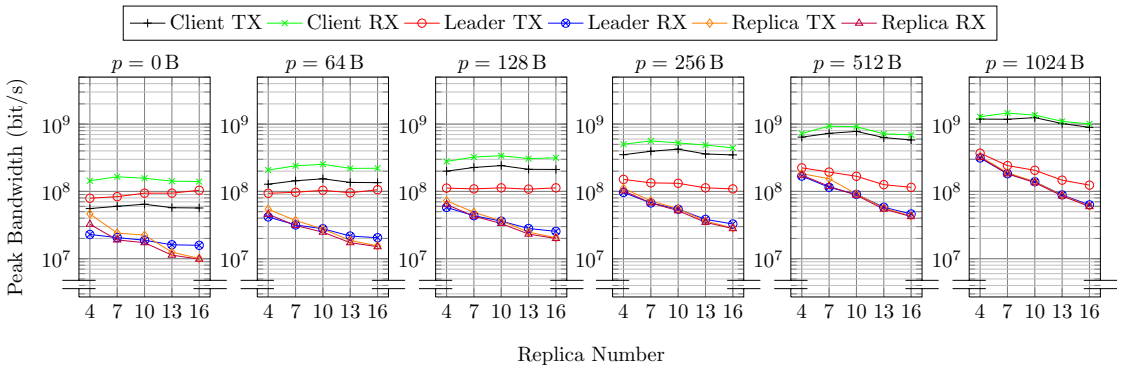
For brevity, we only visualize client, leader, and one non-leader replica from Hardware Group 1 (Fig. 10.1). Individual replicas from Group 3 were not CPU or bandwidth limited for any of our measured metrics. The chosen nodes are connected via 10 GbE $= 10^{10}$ bit/s NICs (§17). The HotStuff PoC uses own wrapper implementations of *libUV* event contexts in *salticidae*. Thread pools can be created for asynchronous message sending, receiving, and signature verification. We run HotStuff using the predefined thread number default settings (§17.3.1). We visualize the same measurement data in two different representations, for increasing n per panel (§18.6.1) and increasing p per panel (§18.6.2).

18.6.1 Sortation over Replica Number

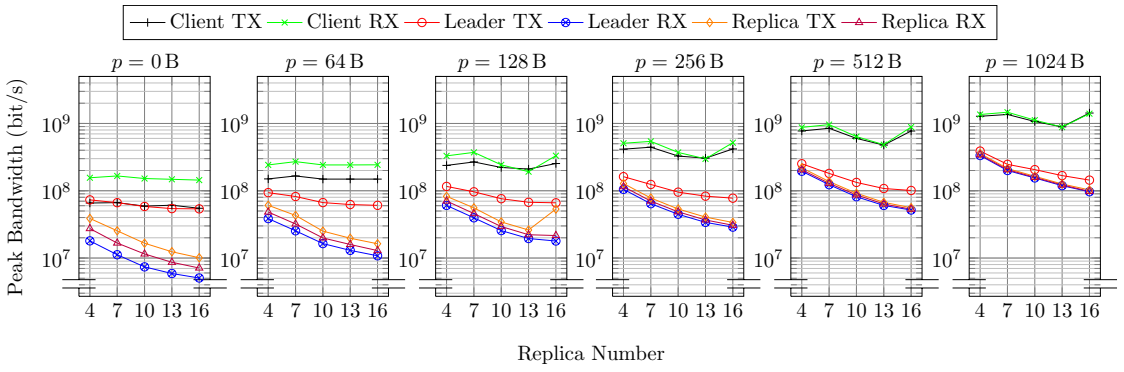
We begin our evaluation with a comparison of the bandwidth peaks between all derived schemes, followed by a comparison of CPU load behavior.

Bandwidth Peaks

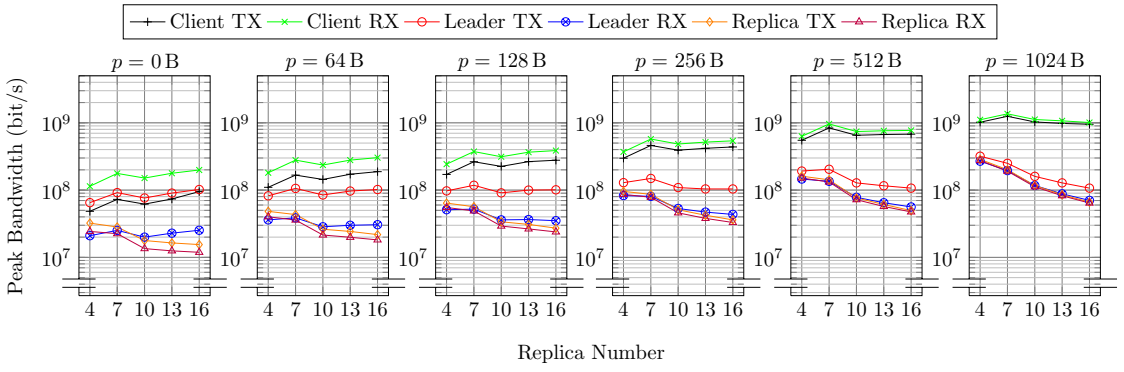
The related figures show subpanels of peak bandwidth in relation to replica number n for client, leader, and one replica. Panels are grouped by payload size p. All results are normalized by saturation. We present results for Secp (Fig. 18.17a), BLSc (Fig. 18.17b), THec (Fig. 18.17c), and Shnr (Fig. 18.17d). Due to insignificant differences between BLSc and BLSnc for our studied parameter ranges, we omit BLSnc for brevity.



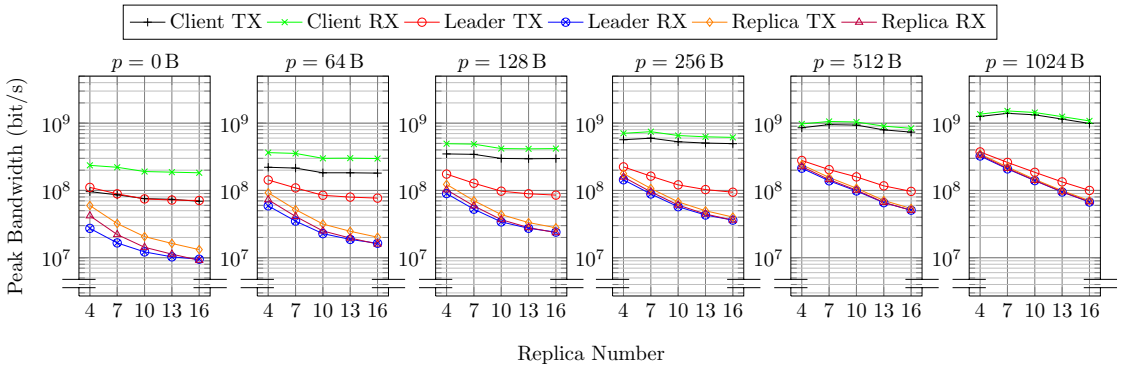
(a) Secp



(b) BLSc



(c) THec



(d) Shnr

Figure 18.17: HotStuff Bandwidth Peaks over Replicas, Saturation Normalized

For the Secp baseline, we make a couple of core observations that partially apply to the other schemes. First, client RX shows the highest bandwidth peaks. This is due to replicas answering the client with `CMD-REQ` messages in parallel (§9.3.1, Eq. 9.17). We recall, that only `CMD-REQ` and `CMD-ANS` messages carry the configurable payload size p. The high client RX is followed by leader TX for $p = 0\,\mathrm{B}$, where the impact of the leader sending out QCs/`proposalMsg` to all replicas (Eq. 9.14) shows. For larger $p \geq 64\,\mathrm{B}$, client TX overtakes leader TX again due to the increasing size of `CMD-REQ` messages (Eq. 9.8). In terms of scaling, we observe that client RX/TX does not vary significantly for increasing n. We recall that b is chosen such that replica decision frequency λ_D is the initial bottleneck. An increase in n prompts the client to communicate with more replicas but λ_D should not grow. This trend is visible as a slight increase in client RX/TX until $n = 10$. For $n = 13, t = n - f = 9$ the setup already requires a quorum containing at least two machines from Hardware Group 3 (§17.4) with weaker hardware. This results in a slight reduction in performance and bandwidth peaks visible on client (and leader for $p \leq 128\,\mathrm{B}$). Leader TX slightly increases for growing n and small p, since increasingly larger Secp-vector QCs are communicated to replicas. Larger p generally increase client and replica peaks but also shift the system bottleneck toward the client for sufficiently large n, p. This is reflected in peak decrease for replicas and leader for growing n and overall lower system throughput.

BLSc has similar core trends to Secp with some deviations. Since BLSc operations are more costly, b is larger, and λ_D is smaller than for Secp. Lower λ_D means a lower communication frequency in between replicas in general. The leader in BLSc is also subject to significant cryptographic processing strain, growing with n. These effects and small, agglomerated BLSc QCs result in decreased performance, reducing leader and client peaks.

The behavior of THec is close to Secp in general. For THec the effects of a bottleneck shift toward the client are only pronounced for larger n, p. We observe a small reduction in client RX/TX and leader RX peaks for $n = 10$, where for $t \sim n$ weaker machines from Hardware Group 3 are part of the necessary quorum. Leader TX/RX grows faster than Secp for larger n, since with $t \sim n$ the leader needs to communicate with more replicas.

Finally, Shnr bandwidth peaks are closer to BLSc shape. However, Shnr exerts larger initial leader and replica peaks, due to smaller b and hence higher λ_D. A client bottleneck shift is visible quickly, even for smaller n, p, likely due to inaccuracies during P_s estimation.

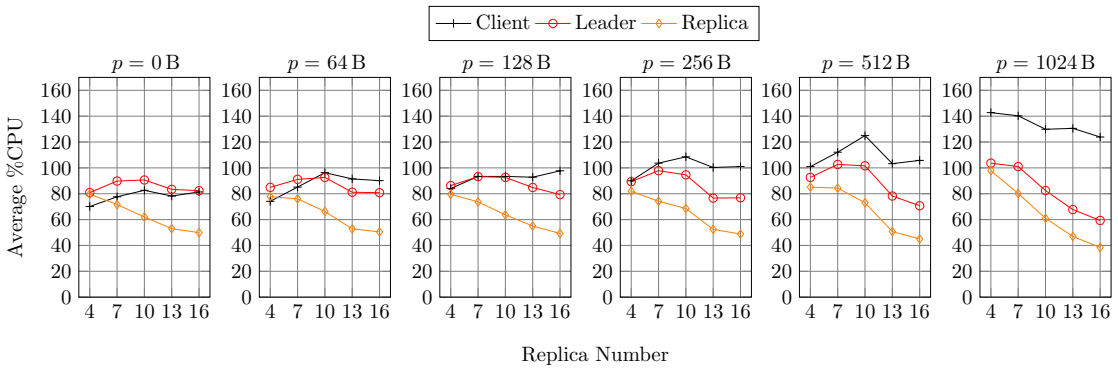
Average %CPU

We visualize the same dataset as Fig. 18.17, with %CPU on the y-axis. We present results for Secp (Fig. 18.18a), BLSc (Fig. 18.18b), THec (Fig. 18.18c), and Shnr (Fig. 18.18d).

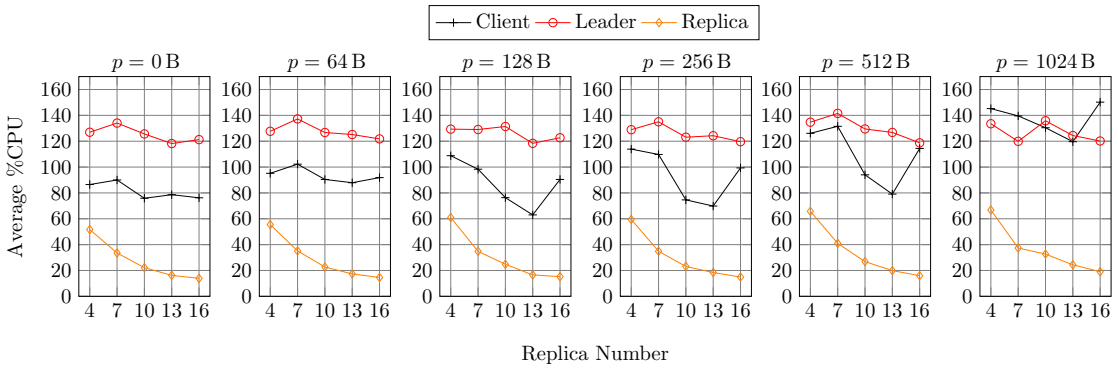
We begin with core observations about our Secp baseline. The leader initially displays the highest load and is only overtaken by the client for large p. This load is due to the general bottleneck position of the leader. We observe considerable load on the regular replica due to QC verification cost scaling with n (§16.4). For $n \geq 13, t = n - f \geq 9$, we observe a small reduction in load due to the weaker machines joining the minimum necessary quorum. We observe the bottleneck shift toward the client node for large n, p. Client load is elevated while load of leader and replicas decreases for growing n.

BLSc load differs significantly from Secp. The leader is subject to significant processing load for all parameter choices of n, p due to increased cryptographic cost of BLSc. Simple replica load is reduced in comparison to Secp, since only a single signature needs to be verified in a QC and λ_D is lower. A bottleneck shift toward the client occurs for large p.

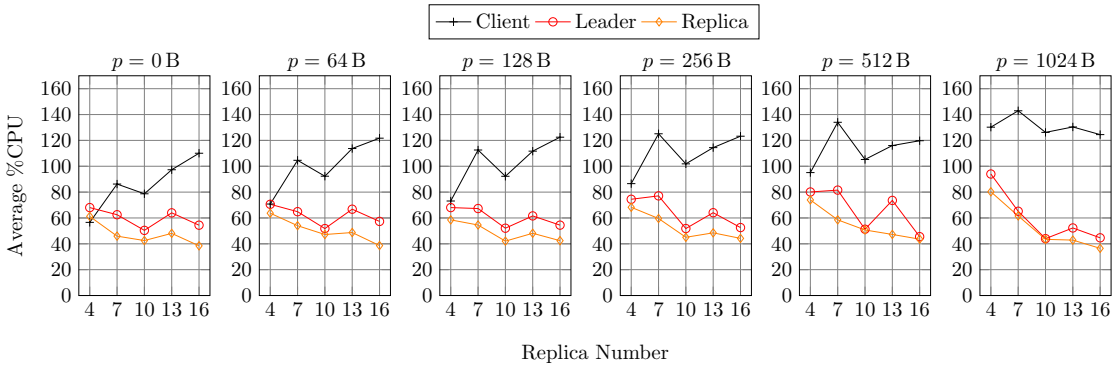
For THec we also observe a gradual bottleneck shift toward the client for large p, with generally reduced load on both leader and replicas in comparison to Secp. QC verification



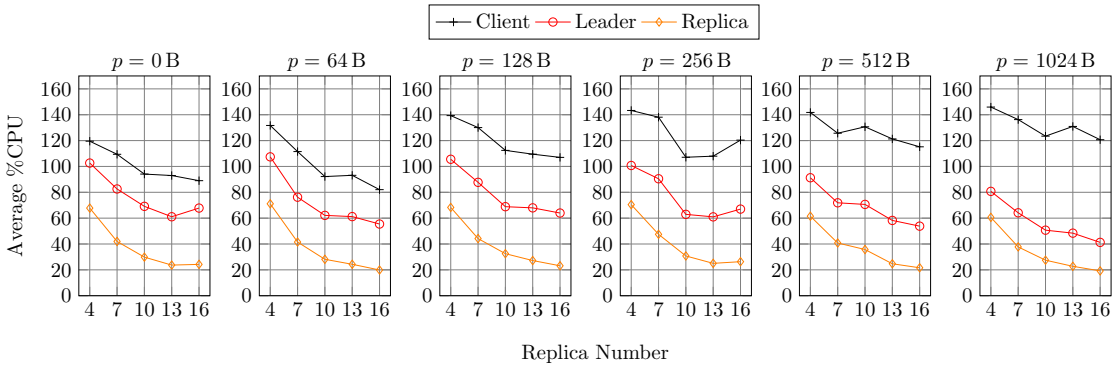
(a) Secp



(b) BLSc



(c) THec



(d) Shnr

Figure 18.18: HotStuff Avg. %CPU over Replicas, Saturation Normalized

is constant in THec and agglomeration on the leader is less complex than for the other derived schemes due to presigning isolation (§16.4). We also observe a visible reduction in load for $n = 10$ where machines from Hardware Group 3 join the $t \sim n$ quorum.

For Shnr, the client always displays the highest load for the parameter space. Both leader and replica load are pronounced due to missing presigning isolation and $t \sim n$. In general, the bottleneck shift to the client occurs for smaller n, p than for the other derived schemes.

18.6.2 Sortation over Payload Size

We re-visualize the data from Fig. 18.17 and 18.18 for detailed analysis. Analogous to the replica sortation, we begin with a comparison of the bandwidth peaks between all derived schemes, followed by a comparison of CPU load behavior.

Bandwidth Peaks

The related figures show subpanels of peak bandwidth in relation to payload size p for client, leader, and one replica. Panels are grouped by replica number n. All results are normalized by saturation. We present results for Secp (Fig. 18.19a), BLSc (Fig. 18.19b), THec (Fig. 18.19c), and Shnr (Fig. 18.19d).

For Secp, we now see the relation between an increase in p and bandwidth peaks conveniently visualized. The general scaling trends develop (necessarily) as already described in §18.6.1, e.g., client RX shows the biggest peaks and we observe a bottleneck shift toward the client for larger n, p. We observe client RX/TX over 10^9 bit/s = 1 Gbit/s and leader TX over $3{\cdot}10^8$ bit/s = 300 Mbit/s for large $p = 1024$ B and $n = 4$. For lower $p = 64$ B client RX bandwidth peaks already surpass 200 Mbit/s, and leader TX is close to 100 Mbit/s.

In comparison to Secp, BLSc shows up to $\sim 50\%$ reduced leader peak TX for large n and small p. This is due to lower λ_D for BLSc ($b_{\text{Secp}} \ll b_{\text{BLSc}}$) and smaller QC size due to agglomerated signatures. Some outlier artifacts (e.g., replica TX, $n = 16$, $p = 128$ B), are not cleaned as the metric tracks the maximal observed bandwidth value.

THec and Secp do not differ significantly in bandwidth peaks, scaling behavior is as discussed before. Shnr bandwidth is similar to BLSc, except for reduced leader peaks.

In general, we observe that client and leader are subject to significant bandwidth peaks, even for smaller n, p, rendering them potential bandwidth bottlenecks.

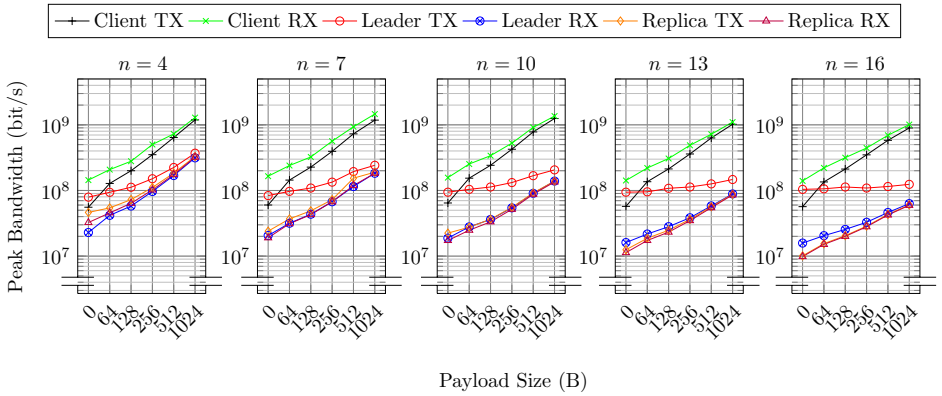
Average %CPU

Related figures show the data of Fig. 18.19, with %CPU on the y-axis. We present results for Secp (Fig. 18.20a), BLSc (Fig. 18.20b), THec (Fig. 18.20c), and Shnr (Fig. 18.20d).

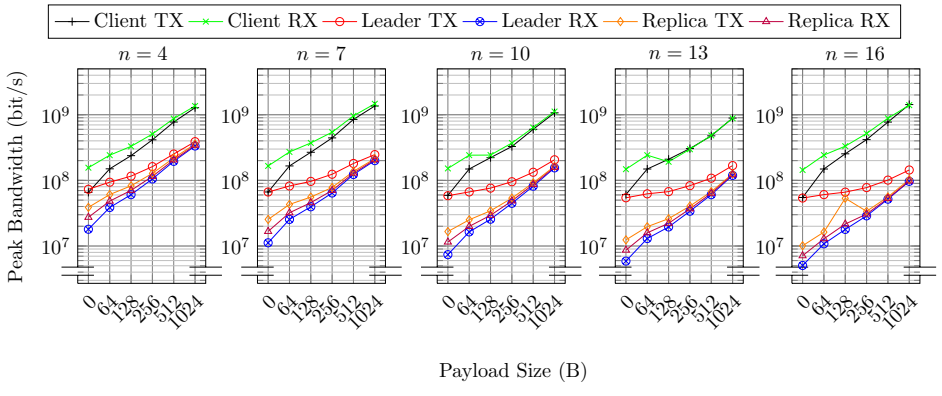
For Secp we observe the significant increase of client load up to $\sim 140\%$ for larger p, due to the processing and sending of `CMD-REQ`/`CMD-ANS` messages. Leader and replica load are similarly affected by larger p, until the system bottleneck shifts toward the client.

BLSc displays significantly increased leader CPU load up to $\sim 140\%$ over the whole p parameter space. An increase in p does not significantly affect leader load, which is dominated by increased cryptographic operation cost. A brief bottleneck shift toward the client is still visible for large n, p. Replica load is reduced, since $\lambda_D^{BLSc} < \lambda_D^{Secp}$ and the leader is subject to significant strain, although raw verification crypto cost of an agglomerated BLSc QC is more expensive for our range of studied n (§18.2).

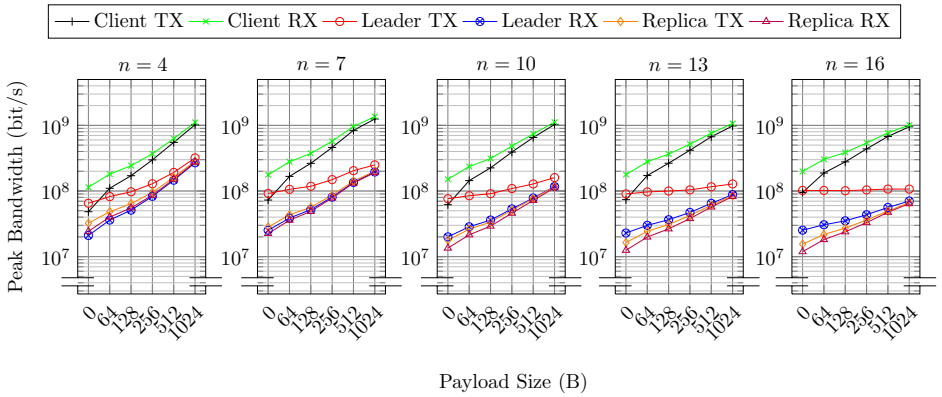
For THec we observe the client bottleneck tendency for larger n, p. Leader and replica load are less pronounced as for Secp, also for increasing p. The discussed load reduction for $n = 10$ is also well visible again.



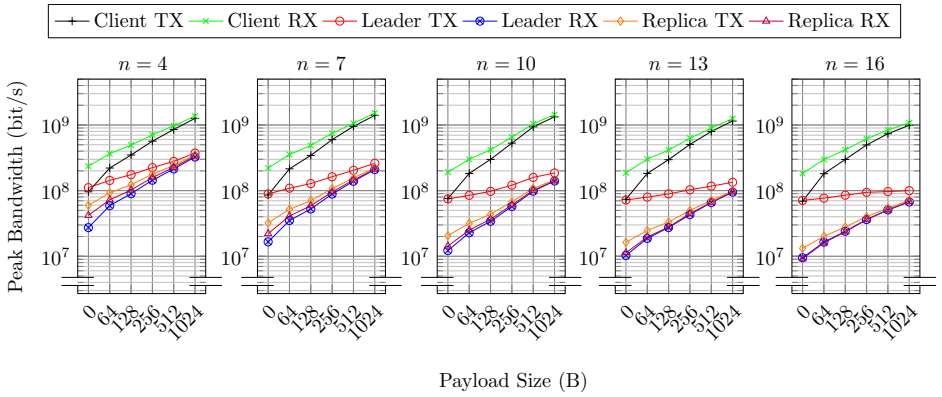
(a) Secp



(b) BLSc

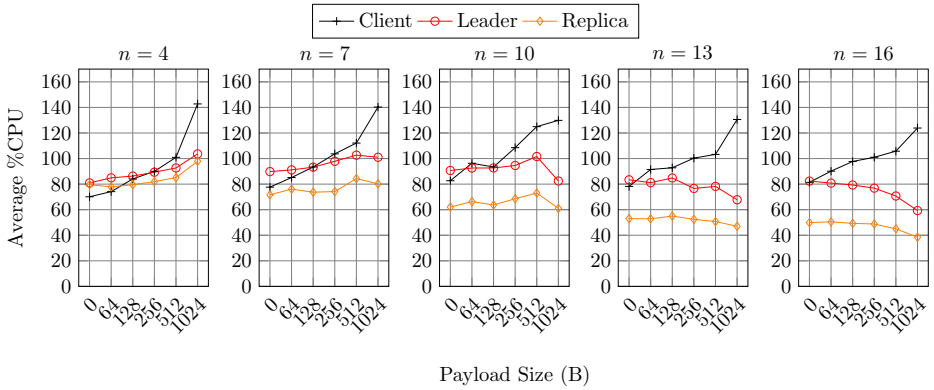


(c) THec

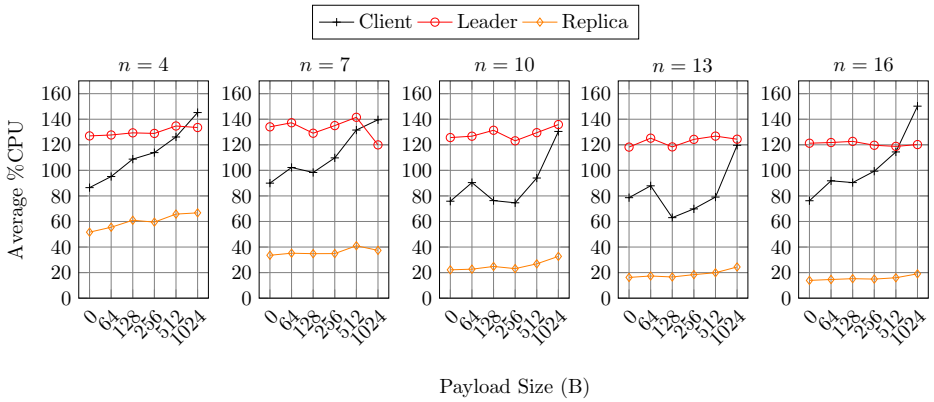


(d) Shnr

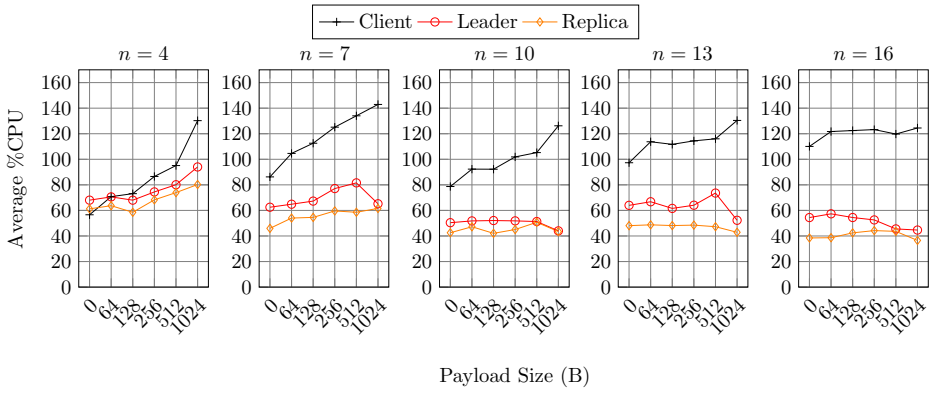
Figure 18.19: HotStuff Bandwidth Peaks over Replicas, Saturation Normalized



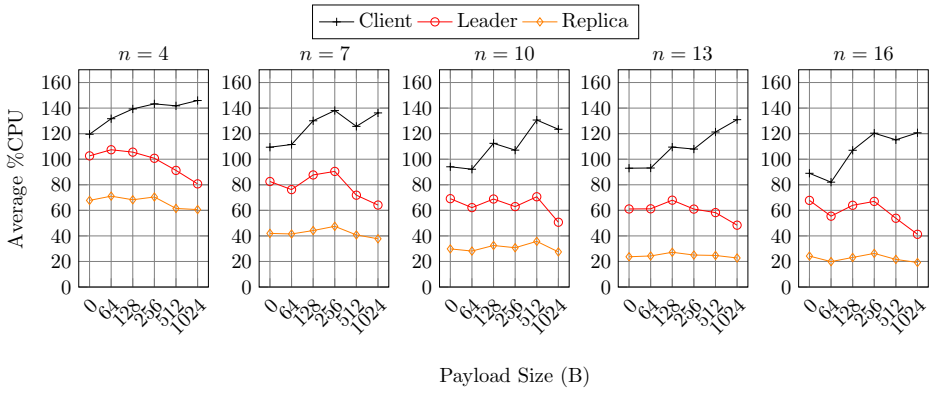
(a) Secp



(b) BLSc



(c) THec



(d) Shnr

Figure 18.20: HotStuff Avg. %CPU over Replicas, Saturation Normalized

Finally, Shnr shows an immediate shift of the bottleneck toward the client already at $n = 4$. Larger p further reduces performance and load. For smaller n the leader load is pronounced relative to replicas and in comparison to Secp, due to incomplete presigning and high complexity of cryptographic operations.

18.7 Summary

We now briefly summarize the results of our evaluation, grouped by evaluation section.

Microbenchmarks

We conducted microbenchmarks for relevant cryptographic base operations in HotStuff. The BLS-based schemes and Shnr show the largest latencies, BLS due to high base operation cost and Shnr due to incomplete presigning isolation. THec is less costly than the other threshold signature schemes and closer to Secp. Secp is generally the most performant scheme in terms of base operation latency, except for the rising QC verification cost, scaling with t. If we only consider verification cost, Secp is outperformed by THec and Shnr for small replica numbers and bound to be outperformed by BLS for larger replica numbers. The constant verification cost of the studied threshold schemes holds potential for break-even points, assuming large QCs or additional re-verification load.

Scaling Extrapolation

We constructed a simplified latency model to roughly approximate raw crypto operation cost in a HotStuff context (§16.5). We insert operation latency approximations from our microbenchmarks to extrapolate the scaling behavior of our derived schemes beyond the replica numbers for which we conduct E2E replication benchmarks. We study and visualize the (I) hot path crypto latency, which is the sum of relevant crypto latencies during execution of a single HotStuff phase, and the (II) crypto latency difference between a target derived scheme s and Secp, which is effectively the difference between hot path latencies with an additional parameter ρ that models re-verification count. For (I), our model projects BLSc with the highest base latency. Secp is projected to have the lowest cost until $n \in [10, 13]$, from where THec is projected to outperform Secp. Shnr is initially projected to be cheaper than BLSc but then outgrows it for $n \in [55, 58]$ due to incomplete presigning isolation. For (II), potential break-even points for $\rho = 1$ are only projected to happen for THec, as anticipated by (I). BLSc and Shnr signature handling costs grow faster than the expected savings by more efficient verification. Break-even points for larger ρ are projected for e.g., $(\rho \approx 20, t \approx 45)$ for BLSc, and $(\rho \approx 5, t \approx 4)$ for Shnr.

Request Saturation

Before conducting integrated E2E benchmarks, we study the saturation behavior of all derived schemes. Secp and THec show the lowest saturation point batch sizes, followed by Shnr and BLS. Central cause are differences in latency of the derived scheme cryptographic base operations. If batch size b is chosen such that replicas are the SuT bottleneck but b is significantly lower than the saturation point, the measured replication latency differences capture scheme cost differences but throughput potential is forfeited in the comparison. Instead, we choose b based on the respective scheme saturation point, roughly normalizing BFT-SMR performance by throughput while preserving latency differences.

Normalized Saturation Performance

We conduct integrated BFT-SMR performance benchmarks, normalized by saturation batch size. Due to this normalization, throughput values and trends are similar between the schemes. Visible performance reduction in throughput is centrally related to the ratio between necessary quorum members for two replica network numbers n, n'. In terms of latency, Secp performs best until it is marginally outperformed by THec for low payload sizes and higher $n > 13$. We note that this rough replica count was anticipated by our scaling extrapolation. This suggests that cryptographic cost remains a central latency cause not only in related work (§19) but also in our setup. BLS displays the highest latency, increased by roughly an order of magnitude in comparison to the Secp baseline. Shnr shows the second highest latency, followed by THec and finally Secp. In terms of scaling behavior, we observe three core points. For increasing p and n the system bottleneck is gradually shifted toward the client, reducing performance. This effect is less pronounced for BLS-based schemes, as the leader node is subject to significant processing strain. Similar to throughput, latency scaling behavior is centrally influenced by the ratio of necessary quorum sizes. Finally, we partially observe a performance reduction when weaker machines join the minimal necessary quorum, e.g., for THec, $n = 10, t \sim n$.

Bandwidth and Processing Load

Client and leader are potential bandwidth bottleneck nodes across all studied schemes, with bandwidth peaks of up to 1 Gbit/s for the client and up to 300 Mbit/s for the leader node in some scenarios. Processing load varies significantly between studied schemes and node roles. All schemes show a tendency of a bottleneck shift toward the client node for larger p. The impact of quorum size assumption $t \sim n$ is visible for THec and Shnr as soon as weaker machines from Hardware Group 3 join the quorum. BLSc shows significant processing strain on the leader node, due to costly crypto base operations. Shnr also displays an increased leader cost until the bottleneck is shifted toward the client. Secp and THec are similar, with THec showing reduced leader and replica load in direct comparison.

19. Related Work

Optimization approaches to BFT-SMR systems are plentiful (§4). Bandwidth, data distribution, and (cryptographic) processing are frequently identified as limitation. Recently proposed remedies include (i) decoupling of dissemination and agreement processes [10,13,102,111,212,263,264], (ii) changes to (communication) topology [108,212,263], and (iii) usage of coding techniques [113, 212, 265].

Here, we restrict our focus to strategies concerned with the modification or study of authentication concepts in a BFT-SMR context, as a potential building block to optimize. Even under this restriction, there exists a significant amount of work. We address the general usage and study of authentication concept in BFT-SMR in §19.1.

As motivated by the progress in, frequently proposed usage of, and our focus on threshold cryptography (§15.1), we consider these more general approaches to be only partially related. We discuss more closely related works, that are concerned with the practical evaluation of threshold signature schemes in E2E BFT-SMR setups in §19.2.

19.1 General Authentication Concepts in SMR

In pursuit of optimizing SMR systems, different authentication concepts have been proposed and evaluated in a plethora of research. Some works explicitly consider the impact and structure of the employed primitives, while others simply assume existence of the primitive as an abstract interface. Tab. 19.1 provides a non-exhaustive list of SMR systems that suggest usage of, or implement certain authentication concepts as introduced in §15.1. This includes simple digital signatures (*Digital Sig*), MACs, the assumption of (abstract) authenticated communication channels between nodes (*Auth Channel*; as implemented by e.g., TLS), and advanced or more complex signature constructions such as threshold/multi-/aggregated signatures (*TH/MS/AG Sig*).

In order to address the relatively significant processing cost of digital signatures, a branch of research proposes the usage of (vectors of) MACs as replacement, where possible. Since MACs do not inherently prove message authenticity to a third party, digital signatures remain in use for some messages (e.g., view-change) while others are replaced [19]. To address missing non-repudiation of MAC authenticators, additional communication stages are considered [25]. Since cryptographic processing cost is considered to be a significant potential bottleneck in BFT-SMR (§5.2), the number of received [7] or processed [32] authenticators (authenticator complexity) during protocol execution (e.g., to reach one consensus decision) is used as another metric to compare algorithms.

Table 19.1: Suggested or Implemented Authentication Concepts (Non-Exhaustive)

Handle	Ref	Digital Sig	MACs	Auth Channel	TH/MS/AG Sig
Streamlet	[34]	●			
PBFT	[19]	●	●		
Zyzzyva	[23]	●	●		
Aardvark	[98]	●	●		
Upright	[25]	●	●		
Abstract/AZyzzyva	[16]	●	●		
RBFT	[99]	●	●		
BFT-SmaRt	[27]	●	●		
ResilientDB	[33]	●	●		
Spinning	[24]		●	●	
FaB	[22]	●		●	
{Th,V,Z}elma	[29]	●		●	
Sync HotStuff	[31]	●		●	
Kuznetsov et al.	[38]	●		●	
Mir-BFT	[100]	●		●	
Prime	[26]	●			●
Chop Chop	[111]	●			●
HotStuff	[7]	●		●	●
JUMBO	[212]	●		●	●
HoneyBadgerBFT	[6]			●	●
SBFT	[8]			●	●
Naor-Keidar	[36]			●	●
FnF-BFT	[32]			●	●
BigBFT	[37]				●
Kauri	[108]				●
Li et al.	[211]				●

Other works consider an abstract secure channel medium between nodes to implement or complement message authentication. Finally, progress and availability of advanced signature schemes prompted new designs and improvements of BFT-SMR systems. In general, we observe a wide variety of authentication-related optimization approaches in the field, with an increased interest and study of advanced signature schemes over the last years. We acknowledge the importance of this direction and hence focus on the evaluation of threshold signature schemes in this work.

19.2 Study of Advanced Signature Concepts

The usage of threshold signature schemes in context of BFT-SMR has been widely proposed [7,8,12,32,97,101,102,108,146,211–213] but only a few works study one or more threshold schemes in a comparable environment in detail. We consider these works to be most closely related and describe them in more detail.

Neiheiser et al. [108] propose *Kauri*, a tree-based communication abstraction, inspired by HotStuff. Votes are dynamically aggregated, as they are forwarded to the leader. Kauri uses BLS for votes and QCs. Both theoretical analysis and performance measurements of Kauri and HotStuff were conducted for Secp and BLS-MS. Experiments were conducted for medium to large setups ($60 \leq n \leq 400$), running on 20 physical machines, partially saturating the hardware. The authors attest to dominant best-case performance of Kauri, but identify HotStuff-BLS outperforming HotStuff-Secp for bandwidth-limited scenarios. Microbenchmarks are not provided, no further threshold schemes were studied.

Berger et al. [266] focus on performance study of large-scale BFT-SMR systems. The authors use network simulation to predict performance, mimicking experiment scenarios from existing literature, and then validate their predictions via measurements, set up in AWS data centers. Evaluated protocols include Kauri and HotStuff BLS-MS. The authors confirm HotStuff-BLS outperforming HotStuff-Secp for bandwidth-limited settings. No further threshold schemes are studied, and no scheme microbenchmarks are provided.

Li et al. [211] benchmark EdDSA and BLS-MS in context of wide-area network BFT-SMR. Group element precomputations are presented as an optimization technique for BLS-MS verification. The schemes are implemented on top of Jolteon [12], a 2-chain HotStuff variant, and experiments are conducted in a geo-distributed AWS setup. The authors suggest that BLS-MS is better suited for small deployments than EdDSA.

Cheng et al. [212] identify interactiveness of threshold and multi-signature schemes as a hindrance to adaption in BFT-SMR, and inefficient bandwidth utilization as performance limitation. An analysis of the relative share of signature operations in communicated payload and CPU time in Dumbo-NG [231] motivates their signature scheme study. The authors propose (1) a signature-free agreement protocol (*FIN-NG*), and (2) a signature-based, improved version of Dumbo-NG, called *JUMBO*. Four different signature schemes are discussed; plain ECDSA, non-interactive, Half-Aggregated Schnorr Signatures (HA-Schnorrs) [241, 242], BLS-MS [214, 249], and BLS threshold signatures. The authors evaluate scheme base operation microbenchmarks and CPU time in the critical path of a one-shot Consistent Broadcast (CBC). The CBC experiment considers signing, PSig verification, agglomeration, and QC verification. The authors furthermore propose an optimistic verification (*batch-verify*) strategy, where PSigs are agglomerated first and the result is verified. In the presented results, BLS-MS and threshold BLS without batch-verify scale worst, followed by batch-verify threshold BLS, plain ECDSA, and the HA-Schnorr variants. The batch-verify BLS-MS variants scale best, but are outperformed by plain ECDSA and HA-Schnorr for $n \lesssim 30$. Microbenchmarks demonstrate BLS verification to be an order of magnitude slower than other schemes. BLS-MS agglomeration is significantly cheaper than BLS threshold agglomeration and BLS QC verification outperforms the other schemes for $t = n - f, n \gtrsim 35$. The JUMBO-BLS performance is then experimentally quantified in E2E measurements and compared against Dumbo-NG and FIN(-NG). AWS deployments are used to emulate different scenarios for $64 \leq n \leq 256$.

19.3 Summary

Overall, the number of practical studies on threshold signature impact on BFT-SMR is limited and often restricted to a few or just one scheme. Some works that conduct integrated BFT-SMR measurements study the signature schemes as part of systems with different consensus algorithms or architectures [111, 212]. This complicates identification and evaluation of the specific impact of a signature scheme. Other works focus on multi-signatures [28, 108, 211, 212, 267] or study half-aggregatable schemes [212].

Our main focus lies on *threshold* signature schemes. To the best of our knowledge, our work is the first to study and directly compare multiple *threshold* signature schemes in a comparable E2E BFT-SMR context in detail. We integrate the target schemes into a seminal BFT-SMR protocol and experimentally quantify consensus performance, crypto scheme base operation performance, and load distribution, using extended metric collection on key nodes. We execute experiments in exclusively non-virtual hardware deployments while varying a range of typical BFT-SMR parameters. All measurements are conducted in the same, fixed environment, increasing comparability of results. We extend the set of practically studied threshold signature schemes in BFT-SMR in number and detail.

20. Conclusion

In Part III of this thesis, we study the potential of threshold signature schemes in context of leader-based BFT-SMR. We compare the impact of three threshold signature schemes to a baseline of simple secp256k1 digital signatures. For theoretical and experimental case studies, the HotStuff (§6) system is used.

20.1 Result Overview

We select a set of threshold schemes based on BLS, Schnorr, and ECDSA signatures to compare against a secp256k1 baseline. Scheme characteristics, such as necessarily interactive steps, cause limitations for usage in a HotStuff context. BLS is the most compatible scheme due to its non-interactiveness but has elevated cryptographic base cost. Semi-interactive schemes based on Schnorr and ECDSA hold potential performance benefits, if presigning can be executed completely, independently, and sufficiently fast to not limit BFT-SMR performance. We choose a software library for each scheme and implement a PoC into HotStuff, referring to resulting artifacts as derived schemes. The derived schemes introduce functional (e.g., incomplete presigning isolation for Shnr) and performance (wrapper library overhead) limitations. A complexity study of the derived threshold schemes demonstrates a general shift from verification to aggregation complexity and additional overhead from wrappers. We create a simplified model of raw cryptographic operation latencies of typical operations in HotStuff. This allows extrapolation of hot path latencies and potential latency differences, depending on quorum size and re-verification count.

In our evaluation we first conduct microbenchmarks of typical cryptographic operations in HotStuff. BLS and Shnr show the highest latencies due to large base cost and incomplete presigning isolation. THec performs better and closer to the Secp baseline. Secp shows small latencies, apart from increasing verification cost, potentially allowing for break-even with other schemes. We approximate possible break-even points by scaling extrapolation, inserting latency approximations from the microbenchmarks into the latency model. For simple hot path latency, Secp is projected to perform with the lowest latency until $n \in [10, 13]$, from which it is projected to be outperformed by THec. Signature operations in BLS and Shnr are costly, and simple verification benefits are projected to not break even. When considering re-verification counts $\rho > 1$, potential break-even points are projected for larger QC sizes and ρ, including for BLS ($\rho \approx 20, t \approx 45$) and Shnr ($\rho \approx 5, t \approx 4$).

We study the request saturation for all derived schemes. Different cryptographic processing costs result in significantly varying replica decision frequencies. We normalized scheme

throughput by suitable choice of batch size, while ensuring the replica decision frequency remains the bottleneck for the experiment start.

We then study HotStuff replication performance with normalized saturation. This results in similar throughput values over all derived schemes, with remaining differences in latency. In terms of latency, Secp performs best until $n \in [10, 13]$ after which it is slightly outperformed by THec for lower payload sizes. We note that this result was roughly projected by the latency model, emphasizing the relevance of cryptographic cost anticipated in related work (§5.2). BLS shows the highest latency, followed by Shnr. For larger payloads, also in combination with larger replica numbers, the system bottleneck shifts toward the client, reducing performance. Independent of this effect, performance change between replica numbers is centrally related to the ratio of respective necessary quorum sizes. The implemented quorum requirements for THec and Shnr ($t \sim n$) manifest as slight performance reduction upon the join of weaker hardware. Study of bandwidth and processing load data shows that client and leader are potential bandwidth bottleneck nodes across all studied schemes. Even for our smaller deployments, peaks of up to 1 Gbit/s and 300 Mbit/s are respectively reached in some scenarios. Node processing load varies significantly between schemes. BLS shows large processing strain on the leader due to cryptographic operation cost. A bottleneck shift toward the client is likewise visible for larger payloads.

20.2 Interpretation

Selection and implementation of authentication concepts have a significant impact on performance, robustness, and scaling behavior of the parent BFT-SMR system. For the integration of threshold signature schemes, interactiveness is a core limitation to compatibility. Semi-interactive signature schemes, with the capability to offload interactive computations into a preprocessing phase may be candidates with improvement potential. However, this requires fully isolated, independent, and sufficiently fast execution of presigning. The overall viability of this approach can be determined by future benchmarks of sufficiently optimized presigning implementations. Besides the additional computational cost, our studied schemes required additional optimistic assumptions on node participation ($t \sim n$). Multi-signature approaches used in related work (§19) are likewise concerned with management or selection of a suitable signer set. The widely-studied BLS scheme is the most compatible out of our studied schemes while incurring increased base cost. Nevertheless, we deem study of alternative schemes to be worthwhile. Usage of threshold (or aggregate) signature schemes can shift complexity from verification to agglomeration. Whether this behavior and other effects such as potential signature size reduction are beneficial for overall performance depends on the target scenario.

For the integrated BFT-SMR measurements an adjusted batch size resulted in all schemes performing with comparable throughput in our setup. Hence, if latency constraints are relaxed but high throughput and e.g., PSig agglomeration properties are required, batch size configuration might enable usage of a scheme with increased cryptographic base cost. Potential tradeoffs between throughput and robustness apply (§9.6).

Our measurements confirm the elevated costs of BLS. Shnr performance is additionally hindered by incomplete presigning isolation. THec performs considerably well, even under its wrapper overhead. Secp latency was slightly outperformed by THec for some scenarios, rendering it an interesting candidate for further study, if the assumption of sufficiently performant presigning holds. While the usage of threshold signatures has been widely proposed in literature (§19.2), we find that increased cost of cryptographic base operations outweighs the potential scaling benefits for small to medium HotStuff deployments. For small to medium networks and simple scenarios without e.g., special bandwidth or reverification demands, we consider simple digital signatures the best default solution.

In related work, potential performance benefits of advanced signature schemes over simple digital signatures in the same base system were attributed to bandwidth-limited scenarios or were reported for medium to large network sizes [108, 266]. Furthermore, related work often focuses on *multi-signatures* instead of threshold signatures [28, 108, 211, 212, 267].

Bandwidth and CPU load on bottleneck nodes across all signature schemes emphasize the importance of structural optimization approaches, that relieve bottleneck nodes by e.g., decoupling transaction dissemination from agreement (§19). Hence, even for smaller deployments sufficient provisioning is essential to prevent performance limitations.

20.3 Verdict

We now put the achieved insights into context of our Research Objectives (§1.2).

Q1 *How can the performance of contemporary, practical SMR be improved?*

Q1.1 *Are fundamental improvements still possible?*

If the BFT-SMR system is executed in a scenario that imposes specific restrictions, such as bandwidth limitations or significantly increased verification load, usage of advanced signature schemes may potentially yield performance benefits.

Q1.2 *How can safety and liveness properties be preserved?*

By ensuring compliance and compatibility of the chosen signatures to the requirements and assumptions of the BFT-SMR system, safety and liveness are preserved.

Q1.3 *Which system aspects should be investigated?*

Usage of threshold signatures in context of BFT-SMR is widely proposed in literature. Replacement of simple digital signatures by advanced signature schemes may yield benefits in specific scenarios or specific algorithms.

Q2 *How big is the impact of the identified optimization spaces?*

Q2.1 *What are necessary preconditions?*

The impact of signature scheme replacement heavily depends on the scenario in which a BFT-SMR system is run. If a semi-interactive signature scheme is used, presigning must be executed independently and sufficiently fast to not limit final BFT-SMR operation. A scenario, in which additional signature re-verifications are required, may emphasize the superior verification complexity of threshold or aggregate signature schemes, leading to a potential break-even with a digital signature baseline.

If additional latency and processing cost are not a primary concern, using a suitable batch size may yield throughput, comparable to the baseline, while still offering the features of a threshold signature scheme. Related work reports benefits for multi-signature schemes in bandwidth-limited scenarios. Adjusting a BFT-SMR algorithm to exploit the impact of e.g., signature agglomeration may increase benefits, but contradicts our goal to limit modification to building blocks only.

Q2.2 *How big are potential performance benefits?*

For our measurements with small to medium node numbers, we observe that a simple replacement of classical digital signatures by an advanced signature scheme does *not* yield BFT-SMR performance benefits in itself. Usage of e.g., BLS heavily increased processing load on the leader node, which already operates in a bottleneck position.

Our presented cases, where a derived scheme came close to or outperformed our baseline (Secp vs. THec), are subject to strong assumptions on speed and independence of the presigning. Should these assumptions hold true for future implementations, we consider the studied or comparable schemes to be promising candidates for further evaluation that may yield direct benefits for sufficiently large quorum or QC sizes.

We conclude that exchange of the underlying cryptographic primitive by advanced signature schemes does not necessarily provide general performance benefits. Execution in specific scenarios may be improved by choice of a suitable signature scheme.

Part IV

Closure

21. Research Objectives

> *"If you find that you're spending almost all your time on theory, start turning some attention to practical things; it will improve your theories. If you find that you're spending almost all your time on practice, start turning some attention to theoretical things; it will improve your practice."*
>
> Donald E. Knuth, Keynote address for the 11th World Computer Congress (1989) [268]

Knuth [268] asserts that balance between theoretical and practical approaches to a problem is beneficial. Due to the abundance of theoretical work on BFT-SMR systems, we pursue a practical strategy with focus on study and modification of necessary components.

21.1 Building Block Optimization as a Method

In Part I, we motivate the optimization of building blocks as a method to improve BFT-SMR systems. A benefit of building block optimization is its potential to apply to a range of different BFT-SMR systems. However, in practice, each potential building block improvement is subject to its own limitations and assumptions. Some limitations are still imposed by the algorithm structure on top, e.g., a leader-based consensus algorithm has a certain communication and bottleneck structure (§9.3, §9.6). Other limitations are imposed by the building block itself and the environment in which it is executed, e.g., the effectiveness of a modification to a network transport protocol depends on the behavior and structure of the network on which it runs (§9.4, §13). Hence, the effective improvement potential of the building block is not completely independent from the systems on top.

Nevertheless, we consider the possible impact range of building block optimization to be a desirable property, even if it is subject to (other) limitations. If the replacement or reconfiguration of the target building block component adheres to the original assumptions and features, previous guarantees on the system can be preserved. Furthermore, studying the performance and characteristics of "component candidates" for a certain building block in a comparable fashion may offer additional insights. The space of BFT-SMR systems and optimization approaches is complex (§4.1, §5.2). For a detailed study of a certain building block component or parameter, it is helpful to reduce the potential influence by other accompanying changes. We increase the precision of our results by conducting measurements in a comparable environment, using the same consensus algorithm, parameters, hardware, and default settings. We are convinced that an emphasis on comparable and reproducible measurements is beneficial for BFT-SMR research in general.

The usefulness of the method should be judged by its success or failure in dedicated component areas and scenarios. Our obtained results are ambivalent. We encounter both performance gain and decline in our experiments (§11, §18). In this thesis, we focus on network transport protocols and threshold signature schemes in particular. Overall, there is a significant parameter and building block component space left to investigate (§5.3).

21.2 Answers to the Research Questions

In §1.2 we introduce our central research objectives and ask research questions. In the following, we subsume our insights to provide answers to the questions.

Q1 *How can the performance of contemporary, practical SMR be improved?*

There exists a plethora of optimization approaches for BFT-SMR (§4.1). Besides conceptual modification to consensus algorithms or scenarios (e.g., separation of transaction dissemination and agreement [10,11,13]), performance of modern BFT-SMR systems partially reaches a level where underlying components reach their limits in practice (e.g., overhead of short-lived TCP transactions in large-scale BFT-SMR deployments [111]).

Q1.1 *Are fundamental improvements still possible?*

As discussed in §21.1, building block optimization potentially offers improvements on a range of different BFT-SMR systems. If a suboptimal building block component is replaced with a superior candidate, improvement of upper-layer systems may be possible, even though the system on top already operates at e.g., the optimal complexities for the problem it aims to solve. In practice, depending on the upper-layer algorithm, execution scenario, and building block, limitations apply. Hence, a specific change might not (positively) affect all systems on top.

Q1.2 *How can safety and liveness properties be preserved?*

By ensuring that chosen implementation and architecture of the target building block conforms to the assumptions on its interface, upper-layer applications and logic preserve their original behavior.

Q1.3 *Which system aspects should be investigated?*

There are multiple possible strategies for selection of target building blocks. In this thesis we select our targets based on the prevalence of model assumptions listed in related work (§5.1, §5.3). Here we assume that improvement of a building block, that implements a frequent assumption, will maximize impact. Other selection strategies could be based on bottlenecks reported in related work (§5.2), or anticipation of future requirements (e.g., scaling to large networks with the requirement of more efficient signature verification (§15.1.5)).

In the following, we discuss **Q2** for all of our chosen building blocks.

21.2.1 Building Block: Network Transport

We study the network transport building block in the wider sense, including evaluation of secure channel protocols (Part II).

Q2 *How big is the impact of the identified optimization spaces?*

Modification of the network transport building block can have a significant impact. Concrete optimization potential depends on the target environment and behavior (§13).

Q2.1 *What are necessary preconditions?*

Differences between candidate network protocols and configurations emerge through an appropriate trigger in the execution environment or usage pattern, e.g., network loss that results in varying retransmission logic and parameter use to become active. Optimization potential can manifest as e.g., (1) inherent shortcoming of the used configuration or transport protocol in relation to current network behavior (e.g., suboptimal choice of timeout values), or (2) concrete BFT-SMR logic or semantics that can be addressed by a change in transport protocol or configuration (such as connection overhead or potential HOL blocking in multiplexed communication).

In our HotStuff case study, we observed improvement potential in category (1), i.e., adjustment of protocol and configuration for operation in lossy scenarios (§11.4).

Q2.2 *How big are potential performance benefits?*

Greater knowledge about the behavior and true bounds of the target network allows for more precise adjustment of protocol and configuration. For our scenario of a small to medium-sized ($n \leq 16$) permissioned BFT-SMR setup and limited packet loss probabilities ($\leq 1\%$), we observed up to $\sim 20\%$ improved performance in comparison to the respective baseline configuration.

21.2.2 Building Block: Cryptographic Primitives

For the building block of cryptographic primitives, we focus on evaluation of threshold signature schemes. We consider the study of other authentication mechanisms or protocols (§15.1) to be interesting future work.

Q2 *How big is the impact of the identified optimization spaces?*

Modification of the underlying cryptographic primitives can have a significant performance impact, although we did not observe direct optimization potential in our studied environment. Usage of threshold signature schemes can impose additional limitations on performance and applicability in a BFT-SMR scenario.

Q2.1 *What are necessary preconditions?*

Necessary preconditions heavily depend on the target scenario. We observe a shift from verification to aggregation and signing complexity, that affects signature scheme suitability in a BFT-SMR context. The studied, semi-interactive schemes require independent and sufficiently fast execution of their presigning phase, in order to not limit BFT-SMR performance. Scenarios which consider additional re-verification cost may emphasize the impact of the reduced verification complexity of threshold or aggregate signature schemes. In such a scenario, this emphasis may lead to a break-even point with the baseline of regular digital signatures.

If replication latency is not a primary concern, threshold scheme throughput with performance, comparable to the baseline, may be achieved by using optimization techniques such as request batching. Significant bandwidth peaks on bottleneck nodes (§18.6) and related work studies (§19) suggest that bandwidth-limited scenarios may be a promising environment for usage of threshold or aggregate signature schemes.

Q2.2 *How big are potential performance benefits?*

In our studied environment and for the chosen set of signature schemes, a replacement of digital signatures by threshold signatures did not yield any inherent performance benefits. We observe a significantly increased leader load for BLS, due to the complexity shift from verification to agglomeration, further increasing the severity of the existing leader bottleneck in HotStuff. The cases, where a semi-interactive signature scheme came close to or outperformed the baseline, are subject to strong assumptions. If these assumptions hold true, we consider further study of these semi-interactive schemes to be a promising approach for larger quorum sizes and re-verification counts. Hence, the target scenario must be fitting for potential benefits to manifest.

22. Comparison to the State of the Art

We examine the distinction of our contribution from related work. We briefly discuss our method of building block optimization (§22.1) and provide a comparison to the state-of-the-art for the building block of network transport protocols (§22.2) and underlying cryptographic primitives (§22.3). A more detailed discussion of related work can be found in the respective related work chapters (§12, §19).

22.1 Building Block Optimization

As outlined in §4.3, replacement of system components with newer or superior versions is an established optimization approach. We consider building block optimization not to be a fundamentally new paradigm but a helpful method to organize and conduct BFT-SMR research. Optimization of a building block potentially allows the improvement to apply to a range of systems on top, even if these systems already operate at the optimum solution complexity for their problem class.

We consider two additional benefits of this approach. Firstly, by requiring the same interface during exchange of building block components, systems on top retain guarantees that are based on these interface assumptions. Secondly, fixing system and experiment parameters, except for the target building block components to study, renders experiments more comparable. We are not aware of existing work that considers building block optimization for BFT-SMR as explicitly and for the reasons we do.

22.2 Building Block: Network Transport

We summarize the detailed discussion of related work for the network transport protocol building block (§12). Tab. 22.1 provides an overview of the most closely related work to our contribution in Part II. We compare candidates along a selection of metrics. We consider the target SMR algorithm and component (*SMR Target*), the set of studied transport protocols (*Transport*), if the work investigates secure channel semantics and performance (*SecChan*), if the work discusses transport protocol behavior in detail (*TransDetail*), and if integrated BFT-SMR measurements were conducted for the chosen parameters (*Meas*).

Table 22.1: Transport Building Block: Overview of Most Closely Related Work

SMR Target	Ref	Transport	SecChan	TransDetail	Meas
AggregaThor	[207]	UDP	✗	∼	✓
HL Sawtooth Content Dist.	[208]	UDP, TCP	✗	✓	❍
Chop Chop Broker Conn.	[111]	RelUDP	✗	∼	✓
Aardvark	[98]	UDP, TCP	✗	✗	✓
RBFT	[99]	UDP, TCP	✗	∼	✓
Raft	[206]	UDP/eRPC	✗	✗	✓
PBFT	[205]	UDP	✗	∼	✓
PBFT	[210]	UDP, TCP	✗	✓	❍
HotStuff	Our Work	UDP, RUDP, TCP, SCTP	✓	✓	✓

✓: Available, ✗: Not Available, ∼: Partially Available, ❍: Simulation

We observe a series of works that study the impact of transport protocols but target an adjacent problem or non-consensus logic. Damaskinos et al. [207] study a BFT distributed stochastic gradient descent system, Ohba et al. [208] study transport protocol impact on *content distribution* in a Hyperledger Sawtooth context, and Camaioni et al. [111] target *broker node* connections for transaction dissemination in byzantine atomic broadcast.

All of the considered works conduct some kind of experimental verification, partially via integrated measurements, partially only via simulations. Behavior and effect of transport protocol logic is typically not discussed in detail, or in a scope limited to acute challenges. Behavior and performance of secure channels are not considered in any of the works. Studied transport protocols are typically limited to TLS and UDP, with the notable exception of Camaioni et al. [111], who employ a reliable, UDP-based transport to fit their needs.

Our work extends the set of studied transport protocols in context of BFT-SMR in number and detail. We analyze both transport protocol and secure channel interrelation with a representative BFT-SMR system and evaluate the performance of the base protocols and different configurations, in a comparable fashion using a bare-metal, full-stack deployment.

22.3 Building Block: Cryptographic Primitives

We summarize the detailed discussion of related work for the underlying cryptographic primitive building block (§19). Tab. 22.1 provides an overview of the most closely related work to our contribution in Part III. Due to our focus on threshold signature schemes, we adjust the metrics along which to compare candidates. We list the target SMR system in which the schemes were studied (*SMR Target*), the studied signature schemes (*BLS, Schnorr, EdDSA, ECDSA*), and if additional metrics (e.g., bandwidth and CPU load) were collected and evaluated. For each of the schemes, we additionally consider the respective scheme type (e.g., multisignatures, threshold signatures, ...) as introduced in §15.1.

All of the considered works conducted some form of integrated BFT-SMR measurements. The original HotStuff publication [5, 7] proposes usage of threshold signatures, but implementation and measurement results were only conducted for simple secp256k1 digital signatures. A group of works studies HotStuff [108, 111, 266] or its variants [108, 211, 266]. While usage of threshold signatures has been widely proposed for usage in context of BFT-SMR (§19.2), we observe that many works effectively study multi-signature schemes instead. Reasons include avoidance of complex DKG [211] and performance benefits [212].

Table 22.2: Crypto Building Block: Overview of Most Closely Related Work

SMR Target	Ref	BLS	Schnorr	EdDSA	ECDSA	Add. Metrics
HotStuff	[7]				○	
Kauri	[108]	●			○	◈
HotStuff / Kauri	[266]	●			○	
Jolteon	[211]	●		○		
Chop Chop	[111]	●		○	○	◈
JUMBO	[212]	●◑†	○†◐†		○	◈ ✹
HotStuff	Our Work	◑	◑		○◑	◈ ✹

†: Microbenchmarks, ○: Simple, ◐: Half-Aggr., ◑: Threshold, ●: Multi-Sig, ◈: Bandwidth, ✹: CPU

Furthermore, works mostly focus on BLS due to its convenient properties (§16.2). A notable exception is the work of Cheng et al. [212], which studies a variety of schemes. However, the authors evaluate most of the signature schemes only as part of a microbenchmark setup, measuring CPU time in the critical path of a one-shot CBC. Besides typical BFT-SMR metrics such as replication latency and throughput, some recent works [108,111,212] also collect and evaluate additional metrics based on CPU and bandwidth.

Our work extends the set of studied threshold signature schemes in context of BFT-SMR. We study complexity, scalability, and performance of a set of threshold signature schemes, including two semi-interactive schemes. We conduct our experimental evaluation in a comparable fashion on a bare-metal, full-stack BFT-SMR deployment, collecting additional metrics for bottleneck impact analysis.

23. Conclusion

We use the method of building block optimization to explore improvement potential of practical BFT-SMR systems in a generalized way, that can leave existing algorithm guarantees intact but still potentially apply to a range of systems. While the space of BFT-SMR research is rich and complex, common assumptions emerge. We choose target building blocks that relate to these assumptions and study the underlying network stack and underlying cryptographic primitives in detail. As representative system, we select the seminal, leader-based HotStuff BFT-SMR protocol for detailed analysis, implementation, and practical experiments.

Our evaluation demonstrates potential for both improvement (e.g., more fitting adjustment of network transport to unstable networks) and additional overhead (e.g., larger cryptographic base cost for threshold signature schemes) in the studied scenarios. We contribute to the state-of-the-art of BFT-SMR optimization by extending the set and detail of studied approaches in the chosen areas. Still, there is ample opportunity for future work, including further study of both the discussed and other potential building blocks.

Acknowledgments:

This thesis would not have been possible without the support of many people. I will not be able to name all of them, but I would like to highlight some.

First of all I would like to thank Prof. Dr.-Ing. Georg Carle for his supervision and giving me the opportunity to join the Chair for Network Architectures and Services. Thank you for your time, commitment, steering, as well as the honest and grounded feedback.

I would like to thank my mentor, Dr. Marcel von Maltitz, for his guidance, encouragement, valuable discussions, and reflections on this process. Your insights were immensely helpful to me. Also, I would like to thank Dr. Sebastian Gallenmüller, a good soul, for both his infrastructure and general support at any day and any time. I will dearly miss our late night discussions at the office.

I am thankful to my co-authors and colleagues for their direct and indirect support. I would like to thank Filip Rezabek for collaboration and feedback. Thank you for your unshakeable optimism and drive, that I find highly motivating. I also would like to thank Kilian Glas and Lion Steger for their teaching support, interesting discussions and rigorous challenging of ideas. Thank you, to all my colleagues, for the smooth operation of daily business, that freed time and energy for research.

Thank you, to my students, that I was allowed to supervise and who supported our research as part of their thesis work. I hope you learned as much as I did.

Finally, I would like to thank Dr. Carolin Schweimer for spurring me to pursue this path.

List of Acronyms

ACK Acknowledgment. 36, 37, 48–52, 55, 69, 79

ACS Asynchronous Common Subset. 13

AEAD Authenticated Encryption with Associated Data. 42

BFT Byzantine Fault Tolerance. i, ii, v, vii, 3, 6, 9, 11–13, 17–27, 33, 35–37, 39–42, 45–62, 66, 69, 70, 73, 75–77, 79–82, 85, 87, 88, 90–92, 97–100, 103, 105, 107, 121–125, 127–130, 133–135, 137–139, 141

BLS Boneh-Lynn-Shacham. 88, 91–96, 98, 101, 104–109, 111, 113, 121, 122, 124, 125, 127, 128, 130, 136, 138, 139

BLS-MS BLS Multi-Signatures. 108, 124, 125

CBC Consistent Broadcast. 125, 139

CC Congestion Control. 35, 36, 47–49, 52, 53, 57, 79

CFT Crash Fault Tolerance. 6, 9, 27, 76, 77

DelACK Delayed Acknowledgement. 53, 69, 79

DKG Distributed Key Generation. 89, 138

DLT Distributed Ledger Technology. 20

DSA Digital Signature Algorithm. 88

DTLS Datagram Transport Layer Security. 37, 38, 40, 55, 56, 65, 73, 80

DupAck Duplicate Acknowledgment. 48, 49, 52, 55, 67

E2E End-to-End. 53–55, 59, 62, 72, 73, 91, 100, 103, 104, 108, 111, 121, 123, 125

ECC Elliptic Curve Cryptography. 88

ECDSA Elliptic-Curve Digital Signature Algorithm. 30, 88, 91, 92, 97, 98, 100, 125, 127, 138

EdDSA Edwards-curve Digital Signature Algorithm. 88, 108, 125, 138

F-RTO Forward RTO-Recovery. 52, 53, 79

FC Flow Control. 35, 48, 49

FPGA Field-Programmable Gate Array. 25

FROST Flexible Round-Optimized Schnorr Threshold. 92–95, 98, 101

GST Global Stabiliziation Time. 8, 27, 28

HA-Schnorr Half-Aggregated Schnorr Signature. 125

HOL Head-of-Line. 36, 40, 135

IP Internet Protocol. 35, 36, 49, 55, 66, 79

MAC Message Authentication Code. 21, 42, 87, 88, 123

MSS Maximum Segment Size. 47

MTU Maximum Transmission Unit. 37, 38, 41, 47, 49, 55, 60

NACK Negative Acknowledgment. 37, 47, 56, 59

NIC Network Interface Card. 60, 114

P2P Peer-to-Peer. 30, 59, 101

PDU Protocol Data Unit. 40–42

PKI Public Key Infrastructure. 88

PoC Proof-of-Concept. 29, 34, 46, 62, 79, 86, 91, 93–95, 100, 101, 114, 127

PSig Partial Signature. 89, 90, 92–97, 100, 101, 104–106, 125, 128

QC Quorum Certificate. 29, 30, 45, 46, 54, 55, 75, 93–98, 100, 101, 103–106, 116, 118, 121, 124, 125, 127, 130

qdisc Queuing Discipline. 50

QoS Quality of Service. 47, 57, 58, 62

RDMA Remote Direct Memory Access. 25, 58, 75

RPC Remote Procedure Call. 76, 77

RTO Retransmission Timeout. 48, 49, 52, 53, 55–57, 59, 62, 66, 68–74, 79, 80

RTT Round-trip Time. 47, 48, 52, 59, 60, 70, 74

RUDP Robust UDP. 33, 34, 36, 37, 39, 40, 42, 47–49, 55, 56, 59, 64, 66, 67, 74, 79, 80

SACK Selective Acknowledgement. 36, 48, 49, 55, 67

SCTP Stream Control Transmission Protocol. 36, 39, 40, 42, 47–50, 53, 55, 56, 59, 64, 66–69, 74, 79, 80

SDU Service Data Unit. 40–42, 49

SIMD Single Instruction Multiple Data. 87

SMR State Machine Replication. i, ii, v, vii, 1–3, 9, 11, 13, 17–27, 33, 35–37, 39–42, 45–62, 66, 69, 70, 73, 75, 77, 79–82, 85, 87, 88, 90–92, 97–100, 103, 105, 107, 121–125, 127–130, 133–135, 137–139, 141

SRTT Smoothed RTT. 59

SuT System under Test. 62, 67, 80, 100, 111, 121

TCP Transmission Control Protocol. 35–37, 39, 40, 42, 47–53, 55, 59–62, 64–69, 73, 74, 76, 77, 79–81, 99, 102, 103, 134

TEE Trusted Execution Environment. 6, 20

THECDSA Threshold ECDSA. 92–95, 98, 101

TLS Transport Layer Security. 24, 36, 37, 40, 42, 55, 59, 62, 64–66, 73, 80, 87, 99, 103, 123, 138

TO-Broadcast Total Order Broadcast. 11, 13

TPS Transactions per Second. 62, 66

TSO TCP Segmentation Offloading. 49

UDP User Datagram Protocol. 35–40, 42, 47–50, 59, 62, 64, 66, 68, 74, 76, 77, 79, 138

WAN Wide Area Network. 24, 27

ZKP Zero Knowledge Proof. 92

Literature

[1] Leslie Lamport. Subject: distribution. https://lamport.azurewebsites.net/pubs/distributed-system.txt, May 1987. Accessed: 2024-08-12.

[2] Richard D Schlichting and Fred B Schneider. Fail-Stop Processors: An Approach to Designing Fault-Tolerant Computing Systems. *ACM Transactions on Computer Systems (TOCS)*, 1(3):222–238, 1983.

[3] Fred B Schneider. Implementing Fault-Tolerant Services Using the State Machine Approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.

[4] Satoshi Nakamoto et al. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. [Online] https://bitcoin.org/bitcoin.pdf, last accessed: 08.01.2025.

[5] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. HotStuff: BFT Consensus in the Lens of Blockchain. *arXiv:1803.05069*.

[6] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The Honey Badger of BFT Protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 31–42, 2016.

[7] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. HotStuff: BFT Consensus with Linearity and Responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. ACM, 2019.

[8] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. SBFT: A Scalable and Decentralized Trust Infrastructure. In *2019 49th Annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pages 568–580. IEEE, 2019.

[9] Chrysoula Stathakopoulou, Tudor David, Matej Pavlovic, and Marko Vukolić. Mir-BFT: High-Throughput Robust BFT for Decentralized Networks. *arXiv preprint arXiv:1906.05552*, 2019.

[10] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and Tusk: A DAG-based Mempool and Efficient BFT Consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 34–50, 2022.

[11] Fangyu Gai, Jianyu Niu, Ivan Beschastnikh, Chen Feng, and Sheng Wang. Scaling Blockchain Consensus via a Robust Shared Mempool. *arXiv:2203.05158*, 2022.

[12] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and Ditto: Network-Adaptive Efficient Consensus with Asynchronous Fallback. In *Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers*, pages 296–315. Springer, 2022.

[13] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: DAG BFT Protocols Made Practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2705–2718, 2022.

[14] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of Distributed Consensus with One Faulty Process. Technical report, Massachusetts Inst of Tech Cambridge lab for Computer Science, 1982.

[15] Michael J Fischer. The Consensus Problem in Unreliable Distributed Systems (A Brief Survey). In *International conference on fundamentals of computation theory*, pages 127–140. Springer, 1983.

[16] Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The Next 700 BFT Protocols. In *Proceedings of the 5th European conference on Computer systems*, pages 363–376, 2010.

[17] Leslie Lamport. Using Time instead of Timeout for Fault-Tolerant Distributed Systems. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 6(2):254–280, 1984.

[18] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the Presence of Partial Synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.

[19] Miguel Castro, Barbara Liskov, et al. Practical Byzantine Fault Tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[20] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography. *Journal of Cryptology*, 18(3):219–246, 2005.

[21] HariGovind V Ramasamy and Christian Cachin. Parsimonious Asynchronous Byzantine-Fault-Tolerant Atomic Broadcast. In *International Conference on Principles of Distributed Systems*, pages 88–102. Springer, 2005.

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

[23] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: Speculative Byzantine Fault Tolerance. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 45–58, 2007.

[24] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, and Lau Cheuk Lung. Spin One's Wheels? Byzantine Fault Tolerance with a Spinning Primary. In *2009 28th IEEE International Symposium on Reliable Distributed Systems*, pages 135–144. IEEE, 2009.

[25] Allen Clement, Manos Kapritsos, Sangmin Lee, Yang Wang, Lorenzo Alvisi, Mike Dahlin, and Taylor Riche. UpRight Cluster Services. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 277–290, 2009.

[26] Yair Amir, Brian Coan, Jonathan Kirsch, and John Lane. Prime: Byzantine Replication Under Attack. *IEEE transactions on dependable and secure computing*, 8(4):564–577, 2010.

[27] Alysson Bessani, João Sousa, and Eduardo EP 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.

[28] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In *25th {usenix} security symposium ({usenix} security 16)*, pages 279–296, 2016.

[29] Ittai Abraham, Guy Gueta, Dahlia Malkhi, and Jean-Philippe Martin. Revisiting Fast Practical Byzantine Fault Tolerance: Thelma, Velma, and Zelma. *arXiv:1801.10022*, 2018.

[30] Mohammad M Jalalzai, Jianyu Niu, Chen Feng, and Fangyu Gai. Fast-Hotstuff: A Fast and Resilient HotStuff Protocol. *arXiv:2010.11454*, 2020.

[31] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync Hotstuff: Simple and Practical Synchronous State Machine Replication. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 106–118. IEEE, 2020.

[32] Zeta Avarikioti, Lioba Heimbach, Roland Schmid, and Roger Wattenhofer. FnF-BFT: Exploring Performance Limits of BFT Protocols. *arXiv:2009.02235*, 2020.

[33] Suyash Gupta, Sajjad Rahnama, and Mohammad Sadoghi. Permissioned Blockchain Through the Looking Glass: Architectural and Implementation Lessons Learned. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 754–764. IEEE, 2020.

[34] Benjamin Y Chan and Elaine Shi. Streamlet: Textbook Streamlined Blockchains. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 1–11, 2020.

[35] Jianyu Niu and Chen Feng. Leaderless Byzantine Fault Tolerant Consensus. *arXiv:2012.01636*, 2020.

[36] Oded Naor and Idit Keidar. Expected Linear Round Synchronization: The Missing Link for Linear Byzantine SMR. In *34th International Symposium on Distributed Computing*, 2020.

[37] Salem Alqahtani and Murat Demirbas. BigBFT: A Multileader Byzantine Fault Tolerance Protocol for High Throughput. In *2021 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, pages 1–10. IEEE, 2021.

[38] Petr Kuznetsov, Andrei Tonkikh, and Yan X Zhang. Revisiting Optimal Resilience of Fast Byzantine Consensus. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 343–353, 2021.

[39] Gabriel Bracha and Sam Toueg. Asynchronous Consensus and Broadcast Protocols. *Journal of the ACM (JACM)*, 32(4):824–840, 1985.

[40] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. On the Minimal Synchronism Needed for Distributed Consensus. *Journal of the ACM (JACM)*, 34(1):77–97, 1987.

[41] Fred B Schneider. Byzantine Generals in Action: Implementing Fail-Stop Processors. *ACM Transactions on Computer Systems (TOCS)*, 2(2):145–154, 1984.

[42] Tushar Deepak Chandra and Sam Toueg. Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM (JACM)*, 43(2):225–267, 1996.

[43] Leslie Lamport. The Part-Time Parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2):133–169, 1998.

[44] Diego Ongaro and John Ousterhout. In Search of an Understandable Consensus Algorithm. In *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, pages 305–319, 2014.

[45] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[46] Paulo Esteves Veríssimo, Nuno Ferreira Neves, and Miguel Pupo Correia. Intrusion-Tolerant Architectures: Concepts and Design. In *Architecting dependable systems*, pages 3–36. Springer, 2007.

[47] Alysson Neves Bessani, Miguel Correia, and Paulo Verissimo. Intrusion Tolerance: The "killer app" for BFT Protocols (?). In *BFTW3: Why? When? Where? Workshop on the theory and practice of Byzantine fault tolerance*, 2009.

[48] Xavier Défago, André Schiper, and Péter Urbán. Total Order Broadcast and Multicast Algorithms: Taxonomy and Survey. *ACM Computing Surveys (CSUR)*, 36(4):372–421, 2004.

[49] Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. Attested Append-Only Memory: Making Adversaries Stick to their Word. *ACM SIGOPS Operating Systems Review*, 41(6):189–204, 2007.

[50] Johannes Behl, Tobias Distler, and Rüdiger Kapitza. Hybrids on Steroids: SGX-Based High Performance BFT. In *Proceedings of the Twelfth European Conference on Computer Systems*, pages 222–237. ACM, 2017.

[51] Mingyuan Gao, Hung Dang, Ee-Chien Chang, and Jialin Li. Mixed Fault Tolerance Protocols with Trusted Execution Environment. *arXiv:2208.01946*, 2022.

[52] Ines Messadi, Markus Horst Becker, Kai Bleeke, Leander Jehl, Sonia Ben Mokhtar, and Rüdiger Kapitza. SplitBFT: Improving Byzantine Fault Tolerance Safety using Trusted Compartments. In *Proceedings of the 23rd ACM/IFIP International Middleware Conference*, pages 56–68, 2022.

[53] Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. In *Concurrency: the Works of Leslie Lamport*, pages 179–196. 2019.

[54] Brian M Oki and Barbara H Liskov. Viewstamped Replication: A New Primary Copy Method to Support Highly-Available Distributed Systems. In *Proceedings of the seventh annual ACM Symposium on Principles of distributed computing*, pages 8–17, 1988.

[55] Rachid Guerraoui and André Schiper. Software-Based Replication for Fault Tolerance. *Computer*, 30(4):68–74, 1997.

[56] Pascal Felber and André Schiper. Optimistic Active Replication. In *Proceedings 21st International Conference on Distributed Computing Systems*, pages 333–341. IEEE, 2001.

[57] Leslie Lamport. The Implementation of Reliable Distributed Multiprocess Systems. *Computer Networks (1976)*, 2(2):95–114, 1978.

[58] David K Gifford. Weighted Voting for Replicated Data. In *Proceedings of the seventh ACM symposium on Operating systems principles*, pages 150–162, 1979.

[59] Vassos Hadzilacos and Sam Toueg. A Modular Approach to Fault-Tolerant Broadcasts and Related Problems. Technical report, Cornell University, 1994.

[60] George F Coulouris, Jean Dollimore, Tim Kindberg, and Gordon Blair. *Distributed systems: Concepts and Design*. Pearson Education, fifth edition, 2011.

[61] Fred B Schneider. Synchronization in Distributed Programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(2):125–148, 1982.

[62] Achour Mostefaoui and Michel Raynal. Leader-Based Consensus. *Parallel Processing Letters*, 11(01):95–107, 2001.

[63] Miguel Correia, Nuno Ferreira Neves, and Paulo Veríssimo. From Consensus to Atomic Broadcast: Time-Free Byzantine-Resistant Protocols Without Signatures. *The Computer Journal*, 49(1):82–96, 2006.

[64] Miguel Correia, Giuliana Santos Veronese, Nuno Ferreira Neves, and Paulo Verissimo. Byzantine consensus in asynchronous message-passing systems: a survey. *International Journal of Critical Computer-Based Systems*, 2(2):141–161, 2011.

[65] Marcos K Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. Stable leader election. In *Distributed Computing: 15th International Conference, DISC 2001 Lisbon, Portugal, October 3–5, 2001 Proceedings 15*, pages 108–122. Springer, 2001.

[66] Mikel Larrea, Antonio Fernández, and Sergio Arévalo. Eventually consistent failure detectors. In *Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, pages 326–327, 2001.

[67] Cynthia Dwork, Joseph Y Halpern, and Orli Waarts. Performing Work Efficiently in the Presence of Faults. In *Proceedings of the eleventh annual ACM symposium on Principles of distributed computing*, pages 91–102, 1992.

[68] Michael Ben-Or and Nathan Linial. *Collective Coin Flipping*. Leibniz Center for Research in Computer Science, Department of Computer . . . , 1987.

[69] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable Leader Election. In *SODA*, volume 6, pages 990–999, 2006.

[70] Murali Krishna Ramanathan, Ronaldo A Ferreira, Suresh Jagannathan, Ananth Grama, and Wojciech Szpankowski. Randomized Leader Election. *Distributed Computing*, 19:403–418, 2007.

[71] Petra Berenbrink, Dominik Kaaser, Peter Kling, and Lena Otterbach. Simple and Efficient Leader Election. In *1st Symposium on Simplicity in Algorithms (SOSA 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[72] Dan Boneh, Saba Eskandarian, Lucjan Hanzlik, and Nicola Greco. Single Secret Leader Election. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 12–24, 2020.

[73] Ittai Abraham, Danny Dolev, and Joseph Y Halpern. Distributed Protocols for Leader Election: A Game-Theoretic Perspective. *ACM Transactions on Economics and Computation (TEAC)*, 7(1):1–26, 2019.

[74] Danny Dolev, Nancy A Lynch, Shlomit S Pinter, Eugene W Stark, and William E Weihl. Reaching Approximate Agreement in the Presence of Faults. *Journal of the ACM (JACM)*, 33(3):499–516, 1986.

[75] Soma Chaudhuri. More Choices Allow More Faults: Set Consensus Problems in Totally Asynchronous Systems. *Information and Computation*, 105(1):132–158, 1993.

[76] Gabriel Bracha. Asynchronous Byzantine Agreement Protocols. *Information and Computation*, 75(2):130–143, 1987.

[77] Michael Ben-Or and Ran El-Yaniv. Resilient-Optimal Interactive Consistency in Constant Time. *Distributed Computing*, 16(4):249–262, 2003.

[78] Bowen Alpern and Fred B Schneider. Defining Liveness. *Information processing letters*, 21(4):181–185, 1985.

[79] Sisi Duan, Xin Wang, and Haibin Zhang. FIN: Practical Signature-Free Asynchronous Common Subset in Constant Time. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 815–829, 2023.

[80] Michael O Rabin. Randomized Byzantine Generals. In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*, pages 403–409. IEEE, 1983.

[81] Hagit Attiya, Amotz Bar-Noy, Danny Dolev, Daphne Koller, David Peleg, and Radiger Reischuk. Achievable Cases in an Asynchronous Environment. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 337–346. IEEE, 1987.

[82] Michael F Bridgland and Ronald J Watro. Fault-Tolerant Decision Making in Totally Asynchronous Distributed Systems. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 52–63, 1987.

[83] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching Agreement in the Presence of Faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

[84] Tadeu Freitas, João Soares, Manuel E Correia, and Rolando Martins. Deterministic or probabilistic? - A survey on Byzantine fault tolerant state machine replication. *Computers & Security*, 129:103200, 2023.

[85] Hagit Attiya and Jennifer L Welch. Bounds on Worst-Case Responsiveness for Agreement Algorithms. In *27th International Conference on Principles of Distributed Systems (OPODIS 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2024.

[86] Leslie Lamport. Lower bounds for asynchronous consensus. *Distributed Computing*, 19(2):104–125, 2006.

[87] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

[88] Martin Kleppmann. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems.* " O'Reilly Media, Inc.", 2017.

[89] Richard von Seck, Filip Rezabek, Benedikt Jaeger, Sebastian Gallenmüller, and Georg Carle. BFT-Blocks: The Case for Analyzing Networking in Byzantine Fault Tolerant Consensus. In *2022 IEEE 21st International Symposium on Network Computing and Applications (NCA)*, volume 21, pages 35–44, 2022.

[90] Richard von Seck, Filip Rezabek, Sebastian Gallenmüller, and Georg Carle. On the Impact of Network Transport Protocols on Leader-Based Consensus Communication. In *Proceedings of the 6th ACM International Symposium on Blockchain and Secure Critical Infrastructure*, BSCI '24, page 1–11, New York, NY, USA, 2025. Association for Computing Machinery.

[91] Idit Keidar and Sergio Rajsbaum. On The Cost Of Fault-Tolerant Consensus When There Are No Faults – A Tutorial. In *Latin-American Symposium on Dependable Computing*, pages 366–368. Springer, 2003.

[92] Michael Abd-El-Malek, Gregory R Ganger, Garth R Goodson, Michael K Reiter, and Jay J Wylie. Fault-Scalable Byzantine Fault-Tolerant Services. *ACM SIGOPS Operating Systems Review*, 39(5):59–74, 2005.

[93] Klaus Kursawe and Victor Shoup. Optimistic Asynchronous Atomic Broadcast. In *International Colloquium on Automata, Languages, and Programming*, pages 204–215. Springer, 2005.

[94] Ittai Abraham, Guy Gueta, Dahlia Malkhi, Lorenzo Alvisi, Rama Kotla, and Jean-Philippe Martin. Revisiting Fast Practical Byzantine Fault Tolerance. *arXiv:1712.01367*, 2017.

[95] Rafael Pass and Elaine Shi. Thunderella: Blockchains With Optimistic Instant Confirmation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–33. Springer, 2018.

[96] Roland Schmid and Roger Wattenhofer. A Permit-Based Optimistic Byzantine Ledger. *arXiv:1906.10368*, 2019.

[97] Xiaohai Dai, Bolin Zhang, Hai Jin, and Ling Ren. ParBFT: Faster Asynchronous BFT Consensus with a Parallel Optimistic Path. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 504–518, 2023.

[98] Allen Clement, Edmund L Wong, Lorenzo Alvisi, Michael Dahlin, and Mirco Marchetti. Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults. In *NSDI*, volume 9, pages 153–168, 2009.

[99] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. RBFT: Redundant Byzantine Fault Tolerance. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*, pages 297–306. IEEE, 2013.

[100] Chrysoula Stathakopoulou, David Tudor, Matej Pavlovic, and Marko Vukolić. Mir-BFT: Scalable and Robust BFT for Decentralized Networks. *Journal of Systems Research*, 2(1), 2022.

[101] Eleftherios Kokoris-Kogias. Robust And Scalable Consensus for Sharded Distributed Ledgers. *Cryptology ePrint Archive*, 2019.

[102] Neil Giridharan, Florian Suri-Payer, Ittai Abraham, Lorenzo Alvisi, and Natacha Crooks. Motorway: Seamless high speed BFT. *arXiv preprint arXiv:2401.10369*, 2024.

[103] Leslie Lamport. Brief Announcement: Leaderless Byzantine Paxos. In *International Symposium on Distributed Computing*, pages 141–142. Springer, 2011.

[104] Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. Scalable and Probabilistic Leaderless BFT Consensus through Metastability. *arXiv:1906.08936*, 2019.

[105] Karolos Antoniadis, Julien Benhaim, Antoine Desjardins, Elias Poroma, Vincent Gramoli, Rachid Guerraoui, Gauthier Voron, and Igor Zablotchi. Leaderless Consensus. *Journal of Parallel and Distributed Computing*, 2023.

[106] Yair Amir, Claudiu Danilov, Jonathan Kirsch, John Lane, Danny Dolev, Cristina Nita-Rotaru, Josh Olsen, and David Zage. Scaling Byzantine Fault-Tolerant Replication to Wide Area Networks. In *International Conference on Dependable Systems and Networks (DSN'06)*, pages 105–114. IEEE, 2006.

[107] David Mazieres. The Stellar Consensus Protocol: A Federated Model for Internet-Level Consensus. *Stellar Development Foundation*, 32, 2015.

[108] Ray Neiheiser, Miguel Matos, and Luís Rodrigues. Kauri: Scalable BFT Consensus with Pipelined Tree-Based Dissemination and Aggregation. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 35–48, 2021.

[109] Suyash Gupta, Sajjad Rahnama, Shubham Pandey, Natacha Crooks, and Mohammad Sadoghi. Dissecting BFT Consensus: In Trusted Components we Trust! In *Proceedings of the Eighteenth European Conference on Computer Systems*, pages 521–539, 2023.

[110] Atul Singh, Tathagata Das, Petros Maniatis, Peter Druschel, and Timothy Roscoe. BFT Protocols Under Fire. In *NSDI*, volume 8, pages 189–204, 2008.

[111] Martina Camaioni, Rachid Guerraoui, Matteo Monti, Pierre-Louis Roman, Manuel Vidigueira, and Gauthier Voron. Chop Chop: Byzantine Atomic Broadcast to the Network Limit. *arXiv preprint arXiv:2304.07081*, 2023.

[112] Christian Cachin et al. Architecture of the Hyperledger Blockchain Fabric. In *Workshop on distributed cryptocurrencies and consensus ledgers*, volume 310, pages 1–4. Chicago, IL, 2016.

[113] Jan Camenisch, Manu Drijvers, Timo Hanke, Yvonne-Anne Pignolet, Victor Shoup, and Dominic Williams. Internet Computer Consensus. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 81–91, 2022.

[114] Internet Computer Association. Architecture of the Internet Computer. [Online] https://internetcomputer.org/how-it-works/architecture-of-the-internet-computer/, last accessed: 27.09.2024, 2024.

[115] Lars Hupel. Interoperability aspects of central bank digital currency across ecosystems and borders. *Journal of Payments Strategy & Systems*, 17(4):422–432, 2023.

[116] Christian Straubert and Eric Sucky. How Useful Is a Distributed Ledger For Tracking and Tracing in Supply Chains? A Systems Thinking Approach. *Logistics*, 5(4):75, 2021.

[117] Idit Keidar and Danny Dolev. Efficient Message Ordering in Dynamic Networks. In *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, pages 68–76, 1996.

[118] Joachim Neu, Srivatsan Sridhar, Lei Yang, David Tse, and Mohammad Alizadeh. Longest Chain Consensus Under Bandwidth Constraint. Cryptology ePrint Archive, Paper 2021/1545, 2021. https://eprint.iacr.org/2021/1545.

[119] Sandro Coretti, Aggelos Kiayias, Cristopher Moore, and Alexander Russell. The Generals' Scuttlebutt: Byzantine-Resilient Gossip Protocols. *Cryptology ePrint Archive*, 2022.

[120] Iulian Moraru, David G Andersen, and Michael Kaminsky. There Is More Consensus in Egalitarian Parliaments. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 358–372, 2013.

[121] Huynh Tu Dang. *Consensus Protocols Exploiting Network Programmability*. PhD thesis, Università della Svizzera italiana, 2019.

[122] Manuel Bravo, Zsolt István, and Man-Kit Sit. Towards Improving the Performance of BFT Consensus For Future Permissioned Blockchains. *arXiv:2007.12637*, 2020.

[123] Marcos K Aguilera, Naama Ben-David, Rachid Guerraoui, Virendra J Marathe, Athanasios Xygkis, and Igor Zablotchi. Microsecond Consensus for Microsecond Applications. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, pages 599–616, 2020.

[124] Marios Kogias and Edouard Bugnion. HovercRaft: Achieving Scalability and Fault-tolerance for microsecond-scale Datacenter Services. In *Proceedings of the Fifteenth European Conference on Computer Systems*, pages 1–17, 2020.

[125] Nibesh Shrestha, Ittai Abraham, Ling Ren, and Kartik Nayak. On the Optimality of Optimistic Responsiveness. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 839–857, 2020.

[126] Huynh Tu Dang, Pietro Bressana, Han Wang, Ki Suh Lee, Noa Zilberman, Hakim Weatherspoon, Marco Canini, Fernando Pedone, and Robert Soulé. P4xos: Consensus as a Network Service. *IEEE/ACM Transactions on Networking*, 28(4):1726–1738, 2020.

[127] Cheng Wang, Jianyu Jiang, Xusheng Chen, Ning Yi, and Heming Cui. APUS: Fast and Scalable PAXOS on RDMA. In *Proceedings of the 2017 Symposium on Cloud Computing*, pages 94–107. ACM, 2017.

[128] Michael K Reiter. Secure Agreement Protocols: Reliable and Atomic Group Multicast in Rampart. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 68–80, 1994.

[129] Dahlia Malkhi and Michael Reiter. A High-Throughput Secure Reliable Multicast Protocol. *Journal of Computer Security*, 5(2):113–127, 1997.

[130] Salem Alqahtani and Murat Demirbas. Bottlenecks in Blockchain Consensus Protocols. In *2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS)*, pages 1–8. IEEE, 2021.

[131] Wenyu Li, Chenglin Feng, Lei Zhang, Hao Xu, Bin Cao, and Muhammad Ali Imran. A Scalable Multi-Layer PBFT Consensus for Blockchain. *IEEE Transactions on Parallel and Distributed Systems*, 32(5):1146–1160, 2020.

[132] Chrysoula Stathakopoulou, Matej Pavlovic, and Marko Vukolić. State Machine Replication Scalability Made Simple. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 17–33, 2022.

[133] Jialin Li, Ellis Michael, Naveen Kr Sharma, Adriana Szekeres, and Dan RK Ports. Just Say {NO} to Paxos Overhead: Replacing Consensus with Network Ordering. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 467–483, 2016.

[134] Xin Jin, Xiaozhou Li, Haoyu Zhang, Nate Foster, Jeongkeun Lee, Robert Soulé, Changhoon Kim, and Ion Stoica. NetChain: Scale-Free Sub-RTT Coordination. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 35–49, 2018.

[135] Ming Liu, Tianyi Cui, Henry Schuh, Arvind Krishnamurthy, Simon Peter, and Karan Gupta. Offloading Distributed Applications onto SmartNICs using iPipe. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 318–333. 2019.

[136] Zsolt István, David Sidler, Gustavo Alonso, and Marko Vukolic. Consensus in a Box: Inexpensive Coordination in Hardware. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, pages 425–438, 2016.

[137] Signe Rüsch, Ines Messadi, and Rüdiger Kapitza. Towards Low-Latency Byzantine Agreement Protocols Using RDMA. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 146–151. IEEE, 2018.

[138] Sebastian Gallenmüller, Dominik Scholz, Henning Stubbe, and Georg Carle. The pos Framework: A Methodology and Toolchain for Reproducible Network Experiments. In *CoNEXT '21: The 17th International Conference on emerging Networking EXperiments and Technologies, Munich, Germany, December, 2021*. ACM, 2021.

[139] Fangyu Gai, Ali Farahbakhsh, Jianyu Niu, Chen Feng, Ivan Beschastnikh, and Hao Duan. Dissecting the Performance of Chained-BFT. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 595–606. IEEE, 2021.

[140] Yahya Shahsavari, Kaiwen Zhang, and Chamseddine Talhi. Performance Modeling and Analysis of Hotstuff for Blockchain Consensus. In *2022 Fourth International Conference on Blockchain Computing and Applications (BCCA)*, pages 135–142. IEEE, 2022.

[141] Gengrui Zhang, Fei Pan, Yunhao Mao, Sofia Tijanic, Michael Dang'ana, Shashank Motepalli, Shiquan Zhang, and Hans-Arno Jacobsen. Reaching Consensus in the Byzantine Empire: A Comprehensive Review of BFT Consensus Algorithms. *ACM Computing Surveys*, 56(5):1–41, 2024.

[142] Cypherium. Cypherium whitepaper 2.0. https://www.cypherium.io/whitepaper/cypherium-whitepaper-2-0/. [Online] Accessed 2023-03-14.

[143] Mathieu Baudet, Avery Ching, Andrey Chursin, George Danezis, François Garillot, Zekun Li, Dahlia Malkhi, Oded Naor, Dmitri Perelman, and Alberto Sonnino. State Machine Replication in the Libra Blockchain. *The Libra Assn., Tech. Rep*, 2019.

[144] Jan Bernatik. Introduction to Flow Blockchain. https://jan-bernatik.medium.com/introduction-to-flow-blockchain-7532977c8af8. [Online] last accessed 2023-03-14.

[145] Dahlia Malkhi and Maofan Yin. Lessons from HotStuff. In *Proceedings of the 5th workshop on Advanced tools, programming languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems*, pages 1–8, 2023.

[146] Neil Giridharan, Florian Suri-Payer, Matthew Ding, Heidi Howard, Ittai Abraham, and Natacha Crooks. BeeGees: Stayin' Alive in Chained BFT. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing*, pages 233–243, 2023.

[147] Maofan Yin and Dahlia Malkhi. libhotstuff - General-Purpose BFT State Machine Replication Library with Modularity and Simplicity. https://github.com/hot-stuff/libhotstuff, 2019. [Online], last-accessed: 04.09.2024.

[148] Maofan Yin, Ray Neiheiser, and Aaaron Buchwald. Salticidae: Minimal C++ Asynchronous Network Library. https://github.com/Determinant/salticidae, 2018. [Online], last-accessed: 10.09.2024.

[149] Richard von Seck, Filip Rezabek, and Georg Carle. Thresh-Hold: Assessment of Threshold Cryptography in Leader-Based Consensus. In *2024 IEEE 49th Conference on Local Computer Networks (LCN)*, pages 1–8. IEEE, 2024.

[150] I Standardization. Iso/iec 7498-1: 1994 information technology–open systems interconnection–basic reference model: The basic model. *International Standard ISOIEC*, 74981:59, 1996.

[151] J. Postel. Internet Protocol. RFC 791, IETF, September 1981.

[152] J. Postel. User Datagram Protocol. RFC 768 (INTERNET STANDARD), August 1980.

[153] Wesley Eddy. Transmission Control Protocol (TCP). RFC 9293, IETF, August 2022.

[154] R. Stewart. Stream Control Transmission Protocol. RFC 4960 (Proposed Standard), September 2007. Updated by RFCs 6096, 6335.

[155] J. Iyengar and M. Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, IETF, May 2021.

[156] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *Conference proceedings – ACM special interest group on data communication*, 2017.

[157] M. Thomson and S. Turner. Using TLS to Secure QUIC. RFC 9001, IETF, May 2021.

[158] Xumiao Zhang, Shuowei Jin, Yi He, Ahmad Hassan, Z Morley Mao, Feng Qian, and Zhi-Li Zhang. QUIC is not Quick Enough over Fast Internet. In *Proceedings of the ACM on Web Conference 2024*, pages 2713–2722, 2024.

[159] Tom Bova and Ted Krivoruchka. RELIABLE UDP PROTOCOL. Internet-Draft draft-ietf-sigtran-reliable-udp-00, Internet Engineering Task Force, February 1999. Work in Progress.

[160] Spyridon Samonas and David Coss. The CIA Strikes Back: Redefining Confidentiality, Integrity and Availability in Security. *Journal of Information System Security*, 10(3), 2014.

[161] Ran Canetti and Hugo Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. In *Advances in Cryptology—EUROCRYPT 2002: International Conference on the Theory and Applications of Cryptographic Techniques Amsterdam, The Netherlands, April 28–May 2, 2002 Proceedings 21*, pages 337–351. Springer, 2002.

[162] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, IETF, August 2018.

[163] Yacine Gasmi, Ahmad-Reza Sadeghi, Patrick Stewin, Martin Unger, and N Asokan. Beyond Secure Channels. In *Proceedings of the 2007 ACM workshop on Scalable trusted computing*, pages 30–40, 2007.

[164] M. Thomson. Record Size Limit Extension for TLS. RFC 8449, IETF, August 2018.

[165] E. Rescorla, H. Tschofenig, and N. Modadugu. The Datagram Transport Layer Security (DTLS) Protocol Version 1.3. RFC 9147, IETF, April 2022.

[166] David D Clark, Van Jacobson, John Romkey, and Howard Salwen. An Analysis of TCP Processing Overhead. *IEEE Communications magazine*, 27(6):23–29, 1989.

[167] John David Cavanaugh. Protocol Overhead In IP/ATM Networks. *Minnesota Super, computer Center, Inc*, 1994.

[168] Vasil Sarafov. Comparison of IoT Data Protocol Overhead. In *Proceedings of the Seminars of Future Internet (FI) and Innovative Internet Technologies and Mobile Communication (IITM)*, volume 720, 2018.

[169] IEEE Standard for Ethernet. *IEEE Std 802.3-2022 (Revision of IEEE Std 802.3-2018)*, pages 1–7025, 2022.

[170] R. Stewart, M. Tüxen, and K. Nielsen. Stream Control Transmission Protocol. RFC 9260, IETF, June 2022.

[171] W. Eddy. TCP SYN Flooding Attacks and Common Mitigations. RFC 4987, IETF, August 2007.

[172] Christopher A Kent and Jeffrey C Mogul. Fragmentation Considered Harmful. *ACM SIGCOMM Computer Communication Review*, 25(1):75–87, 1995.

[173] Jeffrey C Mogul and Christopher A Kantarjiev. Retrospective on Fragmentation Considered Harmful. *ACM SIGCOMM Computer Communication Review*, 49(5):41–43, 2019.

[174] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. RFC 5681, IETF, September 2009.

[175] V. Paxson, M. Allman, J. Chu, and M. Sargent. Computing TCP's Retransmission Timer. RFC 6298, IETF, June 2011.

[176] Partho Pratim Mishra, Dheeraj Sanghi, and Satish K Tripathi. TCP Flow Control in Lossy Networks: Analysis and Enhancement. In *NETWORKS*, pages 181–192. Citeseer, 1992.

[177] Mario Gerla and Leonard Kleinrock. Flow Control: A Comparative Survey. *IEEE Transactions on Communications*, 28(4):553–574, 1980.

[178] J. Postel. Transmission Control Protocol. RFC 793 (INTERNET STANDARD), September 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[179] Alexander Afanasyev, Neil Tilley, Peter Reiher, and Leonard Kleinrock. Host-to-Host Congestion Control for TCP. *IEEE Communications surveys & tutorials*, 12(3):304–342, 2010.

[180] F. Gont and S. Bellovin. Defending against Sequence Number Attacks. RFC 6528, IETF, February 2012.

[181] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301, IETF, December 2005.

[182] The kernel development community. Segmentation Offloads. *The Linux Kernel documentation*, next-20230225, 2023. [Online] https://www.kernel.org/doc/html/next/networking/segmentation-offloads.html, last accessed: 26.02.2023.

[183] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. RFC 2018, IETF, October 1996.

[184] *UDP(7), Linux Programmer's Manual*, 5.13 edition, 03 2021.

[185] *TCP(7), Linux Programmer's Manual*, 5.13 edition, 03 2021.

[186] Sridhar Samudrala. *SCTP(7), Linux Programmer's Manual*, 10 2005.

[187] J. Nagle. Congestion Control in IP/TCP Internetworks. RFC 896, IETF, January 1984.

[188] P. Sarolahti, M. Kojo, K. Yamamoto, and M. Hata. Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP. RFC 5682, IETF, September 2009.

[189] Pasi Sarolahti. Congestion Control on Spurious TCP Retransmission Timeouts. In *GLOBECOM'03. IEEE Global Telecommunications Conference (IEEE Cat. No. 03CH37489)*, volume 2, pages 682–686. IEEE, 2003.

[190] Mark Allman and Vern Paxson. On Estimating End-to-End Network Path Properties. *ACM SIGCOMM Computer Communication Review*, 29(4):263–274, 1999.

[191] Massimo Albarello, Jakub Sliwinski, Yann Vonlanthen, and Roger Wattenhofer. Fast internet computer consensus. *arXiv preprint arXiv:2312.05869*, 2023.

[192] Stephen Hemminger et al. Network emulation with NetEm. In *Linux conf au*, volume 5, page 2005, 2005.

[193] Audrius Jurgelionis, Jukka-Pekka Laulajainen, Matti Hirvonen, and Alf Inge Wang. An Empirical Study of Netem Network Emulation Functionalities. In *2011 Proceedings of 20th international conference on computer communications and networks (ICCCN)*, pages 1–6. IEEE, 2011.

[194] Man-Kit Sit, Manuel Bravo, and Zsolt István. An Experimental Framework for Improving the Performance of BFT Consensus For Future Permissioned Blockchains. In *Proceedings of the 15th ACM International Conference on Distributed and Event-based Systems*, pages 55–65, 2021.

[195] Guangda Sun, Xin Zhe Khooi, Yunfan Li, Mingliang Jiang, and Jialin Li. The Case for Accelerating BFT Protocols Using In-Network Ordering. *arXiv:2210.12955*, 2022.

[196] Marcos K Aguilera, Naama Ben-David, Rachid Guerraoui, Virendra Marathe, and Igor Zablotchi. The Impact of RDMA on Agreement. *arXiv:1905.12143*.

[197] Marcos K Aguilera, Naama Ben-David, Rachid Guerraoui, Antoine Murat, Athanasios Xygkis, and Igor Zablotchi. uBFT: Microsecond-Scale BFT using Disaggregated Memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 862–877, 2023.

[198] Ailidani Ailijiang, Aleksey Charapko, and Murat Demirbas. Dissecting the Performance of Strongly-Consistent Replication Protocols. In *Proceedings of the 2019 International Conference on Management of Data*, 2019.

[199] András Varga and Rudolf Hornig. An Overview of the OMNeT++ Simulation Environment. In *1st International ICST Conference on Simulation Tools and Techniques for Communications, Networks and Systems*, 2010.

[200] Miguel Castro and Barbara Liskov. Byzantine Fault Tolerance Can Be Fast. In *2001 International Conference on Dependable Systems and Networks*, pages 513–518. IEEE, 2001.

[201] Christopher Copeland and Hongxia Zhong. Tangaroa: a Byzantine Fault Tolerant Raft, 2016.

[202] Ioannis Kaklamanis, Lei Yang, and Mohammad Alizadeh. Poster: Coded Broadcast for Scalable Leader-Based BFT Consensus. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 3375–3377, 2022.

[203] Zihao Wang, Hang Wang, Zhuowen Li, Xinghua Li, Yinbin Miao, Yanbing Ren, Yunwei Wang, Zhe Ren, and Robert H Deng. Robust Permissioned Blockchain Consensus for Unstable Communication in FANET. *IEEE/ACM Transactions on Networking*, 2023.

[204] Dachao Yu, Yao Sun, Yuetai Li, Lei Zhang, and Muhammad Imran. Communication Resource Allocation of Raft in Wireless Network. *IEEE Sensors Journal*, 2023.

[205] Nikos Chondros, Konstantinos Kokordelis, and Mema Roussopoulos. On the Practicality of 'Practical' Byzantine Fault Tolerance. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 436–455. Springer, 2012.

[206] Anuj Kalia, Michael Kaminsky, and David Andersen. Datacenter {RPCs} can be General and Fast. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 1–16, 2019.

[207] Georgios Damaskinos, El Mahdi El Mhamdi, Rachid Guerraoui, Arsany Hany Abdelmessih Guirguis, and Sébastien Louis Alexandre Rouault. Aggregathor: Byzantine Machine Learning via Robust Gradient Aggregation. In *The Conference on Systems and Machine Learning (SysML), 2019*, number CONF, 2019.

[208] Yoshihiro Ohba, Jing Yi Koh, Nigel Ng, and Sye Loong Keoh. Performance Evaluation of a Blockchain-based Content Distribution over Wireless Mesh Networks. In *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*, pages 258–263. IEEE, 2021.

[209] George F Riley and Thomas R Henderson. The ns-3 network simulator. In *Modeling and tools for network simulation*, pages 15–34. Springer, 2010.

[210] Thomas Lorünser, Benjamin Rainer, and Florian Wohner. Towards a Performance Model for Byzantine Fault Tolerant Services. In *CLOSER*, pages 178–189, 2022.

[211] Zhuolun Li, Alberto Sonnino, and Philipp Jovanovic. Performance of EdDSA and BLS Signatures in Committee-Based Consensus. In *Proceedings of the 5th workshop on Advanced tools, programming languages, and Platforms for Implementing and Evaluating algorithms for Distributed systems*, pages 1–5, 2023.

[212] Hao Cheng, Yuan Lu, Zhenliang Lu, Qiang Tang, Yuxuan Zhang, and Zhenfeng Zhang. JUMBO: Fully Asynchronous BFT Consensus Made Truly Scalable. *arXiv preprint arXiv:2403.11238*, 2024.

[213] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically Optimal Validated Asynchronous Byzantine Agreement. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 337–346, 2019.

[214] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact Multi-Signatures for Smaller Blockchains. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 435–464. Springer, 2018.

[215] R. Gennaro and S. Goldfeder. Fast Multiparty Threshold ECDSA with Fast Trustless Setup. pages 1179–1194, 2018.

[216] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC Non-Interactive, Proactive, Threshold ECDSA With Identifiable Aborts. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1769–1787, 2020.

[217] R. Gennaro and S. Goldfeder. One Round Threshold ECDSA with Identifiable Abort. 2020.

[218] Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. ROAST: Robust Asynchronous Schnorr Threshold Signatures. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 2551–2564, New York, NY, USA, 2022. Association for Computing Machinery.

[219] Daniel J Bernstein. The Poly1305-AES message-authentication code. In *International workshop on fast software encryption*, pages 32–49. Springer, 2005.

[220] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 313–314. Springer, 2013.

[221] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, IETF, February 1997.

[222] James M Turner. The Keyed-Hash Message Authentication Code (HMAC). *Federal Information Processing Standards Publication*, 198(1):1–13, 2008.

[223] JH. Song, R. Poovendran, J. Lee, and T. Iwata. The AES-CMAC Algorithm. RFC 4493, IETF, June 2006.

[224] Y. Nir and A. Langley. ChaCha20 and Poly1305 for IETF Protocols. RFC 8439, IETF, June 2018.

[225] Reinders, James R. Intel AVX-512 Instructions. 2017. [Online] https://www.intel.com/content/www/us/en/developer/articles/technical/intel-avx-512-instructions.html, last accessed: 12.11.2021.

[226] Andy Polyakov. Keccak-1600 for AVX2. https://github.com/openssl/openssl/blob/33388b44b67145af2181b1e9528c381c8ea0d1b6/crypto/sha/asm/keccak1600-avx2.pl, 2017. [Online], last-accessed: 28.10.2024.

[227] Ueli Maurer. Modelling a Public-Key Infrastructure. In *Computer Security—ESORICS 96: 4th European Symposium on Research in Computer Security Rome, Italy, September 25–27, 1996 Proceedings 4*, pages 325–350. Springer, 1996.

[228] Don Johnson, Alfred Menezes, and Scott Vanstone. The Elliptic Curve Digital Signature Algorithm (ECDSA). *International journal of information security*, 1:36–63, 2001.

[229] Daniel RL Brown. Sec 2: Recommended Elliptic Curve Domain Parameters. *Standards for Efficient Cryptography*, 2010. [Online] http://www.secg.org/sec2-v2.pdf, last-accessed: 31.03.2022.

[230] Optimized C library for EC operations on curve secp256k1. https://github.com/bitcoin-core/secp256k1, 2024. [Online], last-accessed: 28.10.2024.

[231] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo-NG: Fast Asynchronous BFT Consensus with Throughput-Oblivious Latency. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1187–1201, 2022.

[232] Bingyong Guo, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Speeding Dumbo: Pushing Asynchronous BFT Closer to Practice. *Cryptology ePrint Archive*, 2022.

[233] T. Pornin. Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA). RFC 6979, IETF, August 2013.

[234] C. P. Schnorr. Efficient Identification and Signatures for Smart Cards. In *Conference on the Theory and Application of Cryptology*, pages 239–252. Springer, 1989.

[235] Claus P Schnorr. Method for identifying subscribers and for generating and verifying electronic signatures in a data exchange system, February 19 1991. US Patent 4,995,082.

[236] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities" honest or bust" with decentralized witness cosigning. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 526–545. Ieee, 2016.

[237] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. In *2018 IEEE symposium on security and privacy (SP)*, pages 583–598. IEEE, 2018.

[238] Marc Beunardeau, Aisling Connolly, Houda Ferradi, Rémi Géraud, David Naccache, and Damien Vergnaud. Reusing Nonces in Schnorr Signatures: (and Keeping It Secure...). In *Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part I 22*, pages 224–241. Springer, 2017.

[239] C. Komlo and I. Goldberg. FROST: Flexible Round-Optimized Schnorr Threshold Signatures. 2020.

[240] Elizabeth Crites, Chelsea Komlo, and Mary Maller. How to Prove Schnorr Assuming Schnorr: Security of Multi- and Threshold Signatures. *Cryptology ePrint Archive*, 2021.

[241] Konstantinos Chalkias, François Garillot, Yashvanth Kondi, and Valeria Nikolaenko. Non-interactive half-aggregation of EdDSA and variants of Schnorr signatures. In *Cryptographers' Track at the RSA Conference*, pages 577–608. Springer, 2021.

[242] Yanbo Chen and Yunlei Zhao. Half-Aggregation of Schnorr Signatures with Tight Reductions. In *European Symposium on Research in Computer Security*, pages 385–404. Springer, 2022.

[243] Renas Bacho, Julian Loss, Gilad Stern, and Benedikt Wagner. HARTS: High-Threshold, Adaptively Secure, and Robust Threshold Schnorr Signatures. *Cryptology ePrint Archive*, 2024.

[244] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of cryptographic engineering*, 2(2):77–89, 2012.

[245] Daniel J Bernstein, Simon Josefsson, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. EdDSA for more curves. 2015.

[246] S. Josefsson and I. Liusvaara. Edwards-Curve Digital Signature Algorithm (EdDSA). RFC 8032, IETF, January 2017.

[247] Daniel J Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. Twisted Edwards Curves. In *Progress in Cryptology–AFRICACRYPT 2008: First International Conference on Cryptology in Africa, Casablanca, Morocco, June 11-14, 2008. Proceedings 1*, pages 389–405. Springer, 2008.

[248] Harold Edwards. A Normal Form for Elliptic Curves. *Bulletin of the American mathematical society*, 44(3):393–422, 2007.

[249] Dan Boneh, Ben Lynn, and Hovav Shacham. Short Signatures from the Weil Pairing. In *International conference on the theory and application of cryptology and information security*, pages 514–532. Springer, 2001.

[250] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-Subgroup Multisignatures. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pages 245–254, 2001.

[251] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the Gap-Diffie-Hellman-group signature scheme. In *International Workshop on Public Key Cryptography*, pages 31–46. Springer, 2002.

[252] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *Advances in Cryptology—EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4–8, 2003 Proceedings 22*, pages 416–432. Springer, 2003.

[253] Torben Pryds Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Annual international cryptology conference*, pages 129–140. Springer, 1991.

[254] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. In *Advances in Cryptology—EUROCRYPT'99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18*, pages 295–310. Springer, 1999.

[255] Dan Boneh, Sergey Gorbunov, Riad S. Wahby, Hoeteck Wee, Christopher A. Wood, and Zhenfei Zhang. BLS Signatures. Internet-Draft draft-irtf-cfrg-bls-signature-05, Internet Engineering Task Force, June 2022. Work in Progress.

[256] Michaella Pettit. Efficient Threshold-Optimal ECDSA. *Cryptology ePrint Archive*, 2021.

[257] Jens Groth and Victor Shoup. Design and Analysis of a Distributed ECDSA Signing Service. *Cryptology ePrint Archive*, 2022.

[258] D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby, and K. Liao. RELIC is an Efficient Library for Cryptography. https://github.com/relic-toolkit/relic.

[259] ZenGo-X Team. Multi-Party ECDSA - Rust Implementation of {t,n}-threshold ECDSA. https://github.com/ZenGo-X/multi-party-ecdsa, 2020. [Online], last-accessed: 04.09.2024.

[260] Alin Tomescu, Robert Chen, Yiming Zheng, Ittai Abraham, Benny Pinkas, Guy Golan Gueta, and Srinivas Devadas. Towards Scalable Threshold Cryptosystems. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 877–893. IEEE, 2020.

[261] Zhuolun Li, Alberto Sonnino, and Philipp Jovanovic. Performance of EdDSA and BLS Signatures in Committee-Based Consensus. *arXiv:2302.00418*, 2023.

[262] Brendan Gregg. The Flame Graph. *Communications of the ACM*, 59(6):48–57, 2016.

[263] Zhenxing Hu, Shengjie Guan, Wenbo Xu, Zhen Xiao, Jie Shi, Pengze Li, Qiuyu Ding, Hui Ding, and Chao Zeng. A Data Flow Framework with High Throughput and Low Latency for Permissioned Blockchains. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*, pages 1–12. IEEE, 2023.

[264] Alexander Spiegelman, Balaji Aurn, Rati Gelashvili, and Zekun Li. Shoal: Improving DAG-BFT Latency and Robustness. *arXiv preprint arXiv:2306.03058*, 2023.

[265] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo: Faster Asynchronous BFT Protocols. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 803–818, 2020.

[266] Christian Berger, Sadok Ben Toumia, and Hans P Reiser. Scalable Performance Evaluation of Byzantine Fault-Tolerant Systems Using Network Simulation. In *2023 IEEE 28th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 180–190. IEEE, 2023.

[267] Chenyu Huang, Zeyu Wang, Huangxun Chen, Qiwei Hu, Qian Zhang, Wei Wang, and Xia Guan. RepChain: A Reputation-Based Secure, Fast, and High Incentive Blockchain System via Sharding. *IEEE Internet of Things Journal*, 8(6):4291–4304, 2020.

[268] Donald E. Knuth. Theory and Practice, 1991.

ISBN 978-3-937201-84-9
DOI 10.2313/NET-2025-11-2
ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)