On the Impact of Network Transport Protocols on Leader-Based Consensus Communication

Richard von Seck, Filip Rezabek, Sebastian Gallenmüller, Georg Carle
{seck,rezabek,gallenmu,carle}@net.in.tum.de
Chair of Network Architectures and Services, Technical University of Munich
Munich, Bavaria, Germany

ABSTRACT

State Machine Replication (SMR) allows implementation of fault-tolerant systems and secure critical infrastructure. The advent of cryptocurrencies has increased research toward more efficient and performant consensus and SMR systems. Still, current performance is not satisfactory for all envisioned use cases. We focus on heterogeneous SMR deployments, that cannot benefit from e.g., Remote Direct Memory Access (RDMA) in pure data center setups, but offer more predictable network conditions than a set of globally distributed virtual machines. In this space, tuning of network transport protocols allows for optimization. In this paper, we analyze secure channel and network stack interdependencies in context of leader-based consensus. We experimentally quantify the impact of four transport protocols and two secure channel implementations on a HotStuff deployment. Our results show, that delays of a single processing layer often impact all layers above, and typical optimizations such as command batching or pipelining act as amplifiers. Except for edge cases, TCP performs best but offers further optimization potential through configuration of retransmission behavior in lossy scenarios. For large loss probabilities above 2%, transport protocol configuration is not sufficient to confine significant replication performance penalties. We demonstrate that tuning of the transport protocol building block opens a novel optimization space to a class of leader-based consensus algorithm deployments.

CCS CONCEPTS

• **Computer systems organization** → **Dependable and fault-tolerant systems and networks**; • **Networks** → *Transport protocols*; *Network measurement*; *Network performance analysis.*

KEYWORDS

Fault Tolerance, Consensus, Network Protocols, Measurements

ACM Reference Format:
Richard von Seck, Filip Rezabek, Sebastian Gallenmüller, Georg Carle. 2024. On the Impact of Network Transport Protocols on Leader-Based Consensus Communication. In *The 6th ACM International Symposium on Blockchain and Secure Critical Infrastructure (BSCI:'24), July 2, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3659463.3660030

BSCI:'24, July 2, 2024, Singapore, Singapore

1 INTRODUCTION

Over the last decade, optimization efforts towards fault-tolerant systems have increased significantly. The advent of cryptocurrencies sparked new research interest in the areas of both permissionless and permissioned systems. Still, current performance properties are not satisfactory for all envisioned use cases. Systems with Byzantine Fault Tolerance (BFT) can tolerate arbitrary behavior of a certain number of nodes. This renders BFT a valuable attribute, especially for secure critical infrastructure. Using the State Machine Replication (SMR) approach [45], more robust systems can be constructed by redundantly executing operations on a set of *replicas*. For a total number of n replicas in a BFT-SMR system we distinguish between f *faulty* and $(n - f)$ *honest* nodes. A node is honest if it behaves according to protocol; faulty otherwise. To preserve correctness and operation order, request execution needs to be coordinated using (BFT) consensus. This agreement typically involves significant processing and communication overhead. Recent works focus on the optimization of complexity characteristics [21, 37, 58] scaling properties [18, 29, 34, 50], and better robustness or recovery times in case of occurring faults [5, 18, 26]. Proposals of new BFT-SMR systems or components often include implementations, measurements, or simulations [13, 18, 21, 46, 49], additional to simple specification and formalization. This increasingly practical orientation also shifts the research focus towards the underlying network and transmission [13, 18, 20, 31, 46]. Sit et al. [47] identify network processing as a relevant source of overhead for leader-based protocols in domain-optimized setups and propose leveraging hardware acceleration to support both network processing and Transport Layer Security (TLS), as a possible replacement for BFT-SMR protocol authenticators. Ailijiang et al. [3] identify message processing in the leader of Paxos setups as a bottleneck. Von Seck et al. [56] argue for the optimization of BFT-SMR building blocks, such as the underlying networking layer, due to widespread strong assumptions on available communication guarantees. Malkhi and Yin [32] confirm the potential performance gains by optimization of consensus building blocks and system considerations, unrelated to fundamental changes to the consensus protocol itself. The authors recommend a systematic study of isolated components for both performance gains and more precise system comparison.

We emphasize the importance of network stack optimization for SMR systems. For *practical* systems, consideration of network impact becomes inevitable, eventually. Either due to safety considerations [35] or simply due to scaling requirements on components [10]. Most importantly, optimization of an abstract building block bears the potential to improve all systems, using that component. Real-world deployments are typically subject to unreliable network conditions. While this uncertainty is represented through models

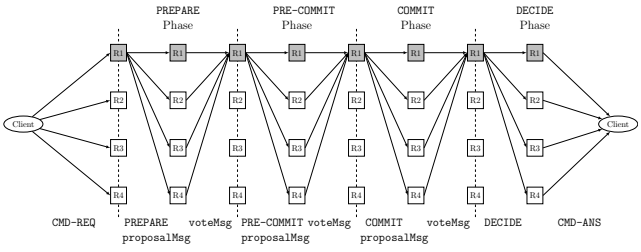


Figure 1: Basic HotStuff communication pattern from [56]

such as partial synchrony [15], performance and robustness implications of the underlying network stack are typically not considered. Hence, a thorough analysis of interrelation, secure channel dynamics, and practical evaluation are required to better understand the impact of network optimization on SMR systems in general.

In this paper, we extend the model of [56] and validate the results via measurements. We study the impact of network transport protocols on leader-based consensus communication. In this context, we analyze the theoretical and practical impact of (D)TLS secure channels. We construct a representative measurement setup and experimentally quantify the impact of transport protocol changes and configuration. We choose HotStuff, an exemplary, state-of-the-art BFT-SMR system, as a base for our work. HotStuff is subject to a significant amount of ongoing research [13, 17, 18, 21, 46, 48, 49, 59] and, as suggested by Spiegelmann et al. [48], it is claimed to be used as foundation for practical systems, such as Cypherium [12], Diem [6], and Flow [7]. The leader-based HotStuff design employs many recent optimization approaches (batching, pipelining, ...) that are present in other systems. HotStuff also shares the prevalent core networking assumptions (reliable and authenticated Point-to-Point (PTP) connections) of BFT-SMR [56], which typically serve as an abstract and not further specified interface. The performance or robustness differences presented in this paper will vary between employed consensus algorithms and deployments. However, the lockstep broadcast-vote, 1-to-n communication structure of leader-based consensus (§2) is a central component of many systems [32], either offering Crash Fault Tolerance (CFT) [27, 38] or BFT [21, 23, 58]. Optimization of a common building block can improve all depending systems. Hence, we conjecture that our core insights and their optimization potential (e.g., retransmission behavior during quorum votes) also apply to other algorithms and deployments.

Our key contributions can be summarized as follows:

- **KC1** Analysis of secure channel and network stack interdependencies in context of BFT-SMR
- **KC2** Demonstration of transport protocols as novel performance optimization space for leader-based consensus
- **KC3** Experimental performance quantification of four transport protocols and two secure channel implementations for a representative BFT-SMR system

2 HOTSTUFF

HotStuff [57, 58] is a leader-based BFT-SMR protocol, designed for partial synchrony [15]. Its general communication is visualized in Figure 1. We distinguish between two types of connections, (1) Client↔Replica and (2) Replica↔Replica. The client starts replicating a command by broadcasting its request to all replicas. Once the replicas have agreed on the request order and committed their operations, confirmations are sent to the client. After receiving at least $(f + 1)$ matching responses, the client considers a request to be successful. One decision process over a request is called a *view*.

Each view involves multiple, similar communication *phases* between replicas and the current leader in a *broadcast-vote* pattern. The leader creates a `proposalMsg` for the current phase and broadcasts it to all replicas. Replicas in turn verify the proposal and respond with a signed `voteMsg`. Note that the leader also votes for its proposals. If the incumbent leader receives $(n - f)$ valid votes, it creates a Quorum Certificate (QC), which contains the current protocol state as well as aggregated vote signature material. A valid QC serves as justification for advancing to the next phase and is included in the next `proposalMsg`. After successfully progressing through three core phases, `PREPARE`, `PRE-COMMIT`, and `COMMIT`, the leader broadcasts the final decision to the replicas, which answer the client. Leadership for a view is justified by $(n-f)$ `NEW-VIEW` messages, collected by the respective leader in the `PREPARE` phase. This allows the leader to learn the latest QC from earlier decisions. If the incumbent leader does not progress for some time, replicas will send `NEW-VIEW` messages to elect a new leader. Additionally, optimization techniques are proposed. Instead of replicating only one command per view, the current leader caches incoming requests and proposes a command batch. Only client requests and responses carry a (configurable) payload to prevent increased transmission volume. The actual consensus and QCs only reference hashes. Due to the similarity of the protocol phases, the original authors propose a pipelined HotStuff variant, *Chained HotStuff*, in which only a `GENERIC` phase exists and a single QC serves as justification for multiple, parallel views at once (Algorithm 3, [57]). Finally, the authors propose the usage of threshold signatures for QCs, allowing for signature aggregation. The available open-source implementation of HotStuff (§6), includes all of the above mechanisms, except for threshold signatures. Instead of a single aggregated signature, vectors of *secp256k1* [9] non-threshold signatures are employed.

3 SECURE CHANNELS

To implement or complement the assumption of authenticated messages between nodes, some BFT-SMR systems [34, 50, 55, 58] rely on secure channels such as TLS [41]. TLS aims to provide confidentiality, authenticity, and integrity as an abstract layer. A core requirement of TLS is an underlying *reliable* and *in-order* transport (e.g., provided by the Transmission Control Protocol (TCP)). After an initial handshake for mutual authentication as well as key and option negotiation, parties can exchange data securely. TLS involves the exchange of typed *records*. Application payloads are split up into suitably sized blocks and are protected, according to negotiated options. Assuming an underlying, ordered data stream, TLS requires strict in-order processing of incoming records. As an alternative to TLS, the Datagram Transport Layer Security (DTLS) [42] aims to provide comparable security while (1) omitting strict order and replay protection and (2) allowing operation over unreliable and datagram-based transports (e.g., User Datagram Protocol (UDP)). DTLS does not implement reliable transport for application

payloads, either. Record format and operation are similar to TLS, while DTLS adds additional fields and explicit sequence number information. A more detailed discussion is provided in §5.3.

4 RELATED WORK

Due to the plethora of research on BFT-SMR optimization in general, we restrict our scope to network stack analysis and optimization. We are aware of works that seek improvements using special (network) hardware [14, 30, 47] or primitives like Remote Direct Memory Access (RDMA) [1, 2, 44]. However, these systems introduce requirements typically only fulfilled in data center environments. Similar to [56], we aim for improvements through common network stack modification and hence consider the above works only partially related. We can roughly divide existing related work into three categories: (1) Work with general relation to network / transport layer analysis, (2) work that leverages new communication primitives without special hardware requirements, and (3) work that directly aims at the optimization of network and transport protocols.

In category (1), Ailijiang et al. [3] 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. [17] 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. [46] likewise created a theoretical model of HotStuff to predict performance. The model is simulated in OMNeT++[54] 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. [23] address the negative liveness impact of asynchrony or equivocation in leader-based BFT-SMR protocols 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. [10] work towards 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.

In category (2), a selection of works proposes a separation of transaction distribution from consensus via a *shared mempool* abstraction [13, 18, 48], addressing communication overhead in leader-based protocols. 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.

Related work in category (3) includes a theoretical analysis and performance model by Lorünser et al. [31]. The authors discuss the characteristics and applicability of TCP and UDP as transport protocols for PBFT, 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 configurable packet loss of up to 30 %. Ohba et al. [36] 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 [43]. Finally, von Seck et al. [56] argue for the optimization of building blocks in BFT-SMR systems, identifying potential improvements on the network layer. The authors conduct a theoretical analysis of transport protocol and configuration impact on BFT-SMR, using HotStuff as an exemplary system. However, neither formal nor practical validation of the claims is provided, and secure channel impact is not addressed.

Our work is an extension of the efforts of von Seck et al. 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, (ii) practically demonstrate optimization potential and (iii) experimentally quantify performance of network transport optimization on state-of-the-art BFT-SMR.

5 ANALYSIS

We now discuss performance-relevant processes and interdependencies on the network stack in an exemplary HotStuff deployment. We collect and discuss potential impacts through consensus protocol structure, network, configuration, and secure channel. Then we summarize and outline the expected system behavior.

5.1 Communication and Dependencies

The original HotStuff paper evaluates End-to-End (E2E) latency and throughput from the client's perspective. We adopt this approach. As outlined by [56], unmasked message loss between client and replicas (a violation of the common assumption of reliable transport between nodes) may lead to safety and performance reduction in the HotStuff concept implementation by the original authors. Hence, we limit the discussion of network stack changes to inter-replica links. Revisiting basic HotStuff (Figure 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 carry-over not only affects the current (batch of) commands but also the pipeline state. Since a single QC can serve in different phases simultaneously in HotStuff (§2), 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.

5.2 Network Stack

We briefly summarize key points of the analysis of [56] for general protocol and configuration space impact and add details about transport retransmission behavior. While Internet Protocol (IP) datagrams support a total size of up to 2^{16} B, the effective size of transmitted packets is determined by the Maximum Transmission Unit (MTU), which is typically smaller (1500 B). If a datagram exceeds the MTU, the IP layer fragments the data over multiple packets, incurring processing overhead. Assuming a uniform loss probability per packet, larger datagrams have a higher chance of suffering a lost fragment. The reassembled datagram is handed to the next layer for processing if all fragments are available.

Both TCP and the Stream Control Transmission Protocol (SCTP) avoid IP fragmentation by splitting payloads themselves, TCP into *segments*, SCTP into *chunks*. TCP can often benefit from hardware offloading support such as TCP Segmentation Offloading (TSO) [52]. 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. While both TCP and SCTP provide reliable transmission, they differ in features and overhead. TCP has a slightly smaller header and typically profits from TSO. While TCP supports Selective Acknowledgements (SACKs) [33], 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 Retransmission Timeout (RTO) [16] must expire, or the sender receives the third Duplicate Acknowledgment (DupAck) [4]. Operation of purely cumulative Acknowledgments (ACKs), may result in unnecessary retransmission of already received data [33]. 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 [51]; SACKs are thus generated in higher frequency. Ordered transmission is optional, and configuration of protocol behavior is easy through a powerful socket-level API.

Configuration of core transport protocol behavior is possible along at least two dimensions: (1) Delay and coalescence of writes and (2) retransmission behavior [56]. For (1), to preserve latency, corking options as well as Nagle's algorithm should be avoided - especially with Delayed Acknowledgements (DelACKs). Optimization of DelACK times may benefit edge cases with large, unidirectional streams of small payloads. For (2), Forward RTO-Recovery (F-RTO) should allow fast congestion window restoration after spurious retransmissions. Optimization of RTO was identified as a possible way to reduce latency in case of lossy links.

Table 1: Selection of Delay Elements

Layer	Delay / Blocking conditions
SMR User Layer	Receival of $\geq (f+1)$ matching `CMD-ANS` to commit
Consensus Layer	Receival of current proposal in order to vote
	Receival of enough votes to progress / propose
	Receival 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

5.3 Secure Channel

Usage of a secure channel implementation such as TLS or DTLS introduces security features but also complexity as well as channel en- / decoding overhead. We briefly discuss the properties of (D)TLS along the dimensions of protocol overhead, fragmentation, ordering, and reliability. TLS requires a three-phase handshake, initiated by the client. The typical header overhead for a single *Application Data* type record is 5 B. Record payload sizes in TLS are limited to a maximum size of $2^{14} + 1$ B. Hence, TLS records can exceed the MTU and must be fragmented accordingly. While explicit record size limitation negotiation is standardized using TLS extensions [53], this must be implemented by the calling application. TLS 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.

DTLS reuses the TLS structure, with additional fields for handshake reliability, fragmentation, and ordering control. The typical header overhead for a single *Application Data* type DTLS record is 13 B, due to additional fields for explicit sequence and epoch numbers. With a maximum record size of 2^{14} B, DTLS records can also exceed MTU and need to be fragmented. Record size limit extensions apply. Since UDP does not offer payload segmentation, fragmentation is conducted on the network layer. As DTLS does not require ordering between records, complete records of the same connection can be processed independently. If fragments are missing, the affected record cannot be processed until retransmission.

5.4 BFT-SMR Impact

We now provide an overview of introduced elements with relevant impact on latency or the potential to stall execution in Table 1. Generally, conditions from lower layers are required to process conditions of higher layers. We see, that the final E2E latency of a client request to a BFT-SMR system is influenced by many factors. To

outline interdependencies between layers, we discuss the behavior of a HotStuff setup in the face of lost packets for two scenarios.

5.4.1 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 leader-sent `proposalMsg`. On the network layer, no fragmentation occurs since our stack operates using TCP. The `proposalMsg` is already segmented to respect the MTU. 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 retransmission. Retransmission is either triggered by receipt of the third DupAck by the replica (fast-retransmit) or RTO expiry. If subsequent packet exchanges take place without (too much) loss, the leader continues sending, until it receives the third DupAck from the replica and then retransmits the missing data. If some of the subsequent packets are also lost, the sender might not receive enough DupAcks to trigger a fast retransmission and hence waits for the RTO to expire. This is costly in terms of latency since the RTO is typically a multiple of E2E consensus latency (cf. measured HotStuff latencies <40 ms [57]). 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 the 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 current 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. However, this means that at most f votes can be concurrently delayed, without adding latency. Additionally, 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 protocols [23] 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.

5.4.2 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 [51], 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 amount of SACKs is lost, sender RTO expiry is inevitable. The implications for higher levels remain the same as with TCP / TLS.

6 EXPERIMENT DESIGN & SETUP

To quantify the impact of transport protocols and configurations, we implemented multiple variants of HotStuff using TCP [16], UDP [40], and SCTP [51] as underlying transport for replica connections, respectively. While plain UDP is an example of a protocol with a low header and state processing overhead, a fair comparison between UDP and reliable protocols in lossy scenarios is hard. QUIC [28] provides reliable transmission, parallel connections between hosts, and allows for integrated TLS connection establishment in its handshake. In a Web context, these features can reduce latency and prevent Head-of-line (HOL) blocking. However, in a HotStuff context, multiple *parallel* TLS connections between two replicas or frequent TLS re-establishments are not expected. Adding more processing logic in another network layer defeats the intent of studying a simple protocol with a small overhead. Hence, we consciously omit analysis and measurement of QUIC for our experiments.

Instead, we take an approach similar to Camaioni et al. [10]. We implement a simplistic, UDP-based transport variant, which employs application-level retransmission logic in the HotStuff code. We call our variant *Robust UDP (RUDP)*. RUDP essentially mimics standard TCP retransmission in terms of smoothed RTO calculation [39]. In RUDP, the RTO values are based on implicitly measured message latency between replicas. Upper and lower bounds on RTO are chosen not to limit the current Round-trip Time (RTT) estimation and no congestion control is implemented. Successive HotStuff messages (e.g., a `proposalMsg` and `voteMsg`) serve as implicit acknowledgments for previously received messages, s.t. loss can be detected. Upon receipt of an erroneous message, an explicit NACK message is sent. Our implementation is based on the freely available HotStuff C++ codebase by the original authors[1]. A majority of the low-level network logic is implemented in the *salticidae*[2] network library. Our implementation required modification of both projects (e.g., HotStuff code for RUDP timing). For reproducible results, all measurements are automated, using the *pos* framework [19]. The target application is automatically deployed, executed, and measured on multiple *physical* machines, to allow for fine-grained control over communication links and hardware. All machines run on RAM disks, loading prebuilt images of Ubuntu Jammy on Kernel v5.15.0-72. HotStuff is built using OpenSSL v1.1.1f-1ubuntu2_amd64.deb. The machines are reset to a well-known state between measurement series for experiment isolation.

For the experiments, we use up to 17 local machines of three different specification types to introduce hardware diversity of a practical BFT-SMR setup. In a heterogeneous setup, overall system performance can be limited by the weakest hardware. In HotStuff, leader and client are the primary bottlenecks in terms of bandwidth and processing. Hence, to prevent side effects of hardware diversity, leader and client roles are always appointed to the most powerful nodes. The connection topology is shown in Figure 2. We employ four machines of Group 1 (Intel Xeon Gold 6312U, 24×2.4 GHz, 512 GB RAM), four machines of Group 2 (AMD EPYC

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

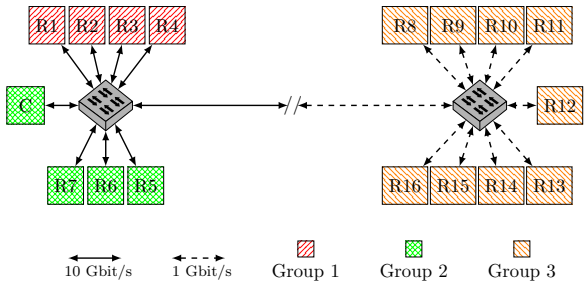


Figure 2: Testbed Hardware Topology

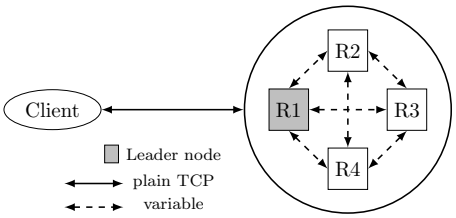


Figure 3: Logical BFT-SMR Role Setup, n=4

7543, 32×3.8 GHz, 512 GB RAM), and nine machines of Group 3 (Intel Xeon D-1518, 4×2.2 GHz, 32 GB RAM). Machines of Group 1 and 2 are connected via Intel E810-XXV, Group 3 via Intel I350 NICs. The nodes are connected via a series of switches. Client (C) and default leader (R1) are always designated to machines with a 10 Gbit/s link to prevent possible bandwidth bottlenecks. RTT between nodes is consistently below 350 µs. The default MTU is 1500 B.

7 EVALUATION

We analyze the performance of our implementations in a series of experiments. Akin to the original implementation, we employ (D)TLS secure channels only between replicas, while the Client↔Replica connections are not protected. The UDP-based implementations use DTLS, while the TCP- and SCTP-based implementations employ TLS. The logical BFT-SMR role configuration is displayed in Figure 3. In order to isolate the effects of the transport protocol choice and configuration, the measurement setup includes a single client and a single leader for replica numbers of $n = 3f+1, f \in [1,5], n \in [4,16]$ per default. Replica↔Replica links, called *variable* links, are subject to changes in transport protocol, secure channel, and configuration. Communication on Client↔Replica links is always conducted over plain TCP, to prevent safety and liveness issues with the HotStuff concept implementation in case of unmasked packet loss [56].

7.1 Measurement Setup

Due to the large parameter space, we evaluate our implementations with a focus on three categories: (1) impact of input request saturation, (2) impact of transport protocol with and w/o secure channel, and (3) impact of transport protocol under loss. For each category, we execute a series of experiments, varying a set of typical BFT-SMR parameters (e.g., batch size, payload size, replica number), as conducted in the original HotStuff preprint [57]. To best cover the parameter space and concisely visualize modification, we typically

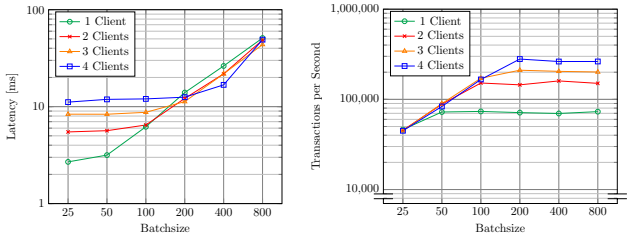


Figure 4: Saturation, Batches, TCP, noTLS, p=0 B, n=4

fix all values to a baseline default, except one variable parameter. For some experiments (§7.4) we choose parameter value combinations deviating from the baseline default, to better highlight the impact of e.g., transport protocol modification. As central metrics, we measure the E2E latency for replication of a single command (batch), analogous to the original paper. We also show the average throughput in operations per second, calculated over the whole experiment run. Each experiment is run for 60 s at maximum possible speed. In case of runs without loss, latencies are 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. For runs with loss, results are *not* filtered for outliers since loss and retransmission may create latency spikes which would disappear from visualization. For brevity, the data set is averaged into a single data point.

7.2 Input Request Saturation

The performance of a BFT-SMR system depends on the applied input load. To understand the saturation behavior of our setup, we study system performance under a varying number of client applications, running in parallel on our client machine. Bandwidth limits of the client machine were not reached for any experiment. A single HotStuff client application does not parallelize request sending. Distribution of the client applications over multiple machines is expected to yield comparable results, as long as the respective bandwidth limits are not reached and CPU power is proportionate.

Figure 4 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 with TCP, noTLS, a payload size $p = 0$ B and $n = 4$. We observe from the throughput values, that a single client suffices to saturate our setup for batches of 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 the current saturation point causes a proportional latency increase since the leader waits longer for a batch of commands to accumulate before it can propose. In Figure 5, we show average latency (top) and cumulative throughput (bottom) over all clients, in relation to command 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 system for larger payloads, and only for batches 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 required size and rate. The described trends are

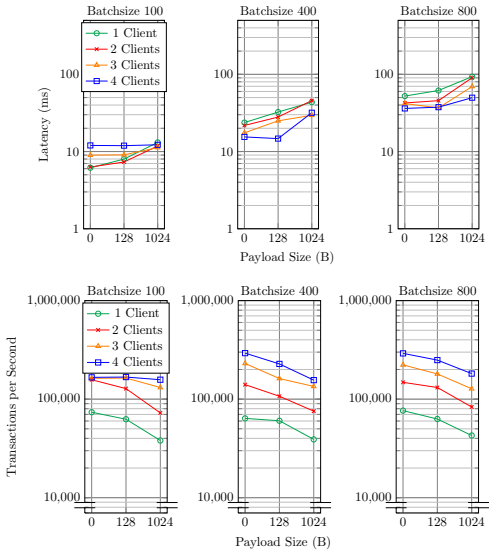
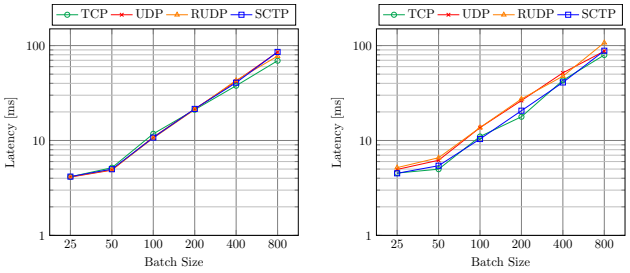


Figure 5: Saturation, Payloads, TCP, noTLS, n=4



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

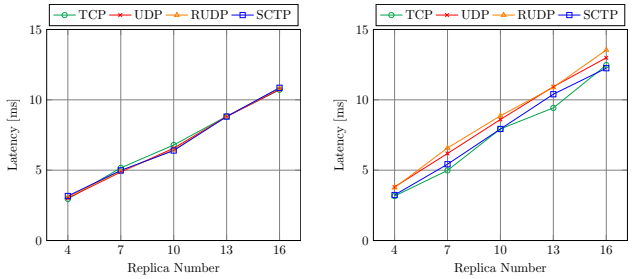
Figure 6: (D)TLS Performance, p=0 B, n=7

amplified in setups with more replicas, since the clients send and receive data from more replicas for each command (batch).

7.3 Transport Protocol and Secure Channel

Next, we investigate the impact of changes to transport protocol and secure channel in lossless operation. These results also provide a baseline for the interpretation of measurements in lossy scenarios. To better isolate performance influences, we conduct the experiments with one client. This allows us to observe system behavior with (batch size $\sim\leq 50$) and without input request saturation.

Figure 6 shows a performance comparison of all investigated protocols with and without active (D)TLS for different batch sizes. The setup is fixed to $n = 7$ replicas, and a payload size of 0 B. First, we note the increased latency gain above batches of 50, due to lacking input request saturation. From batch size 25 to 50 there is still a slight latency increase since more command hashes are included and processed in leader proposals. Without (D)TLS, the performance differences between different transport protocols in this setting are insignificant. With active secure channel, for DTLS-based setups (UDP, RUDP) we observe a consistently increased



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

Figure 7: (D)TLS Performance, p=0 B, batch size = 50

latency in comparison to the TLS-based setups (TCP, SCTP). Comparing TCP and RUDP, for batches ≤ 50, we observe between 14% to 32% increase, for larger batches between 10% up to 54%. This can be attributed to increased 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.

Figure 7 displays performance of (de-)activated (D)TLS modes 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 Figure 6. Inter-protocol differences without (D)TLS are not significant, while DTLS-based setups consistently demonstrate 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% ($n = 16$), 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 (Figure 1). Note, that the leader can progress as soon as $h(f) = (n - f) = 3f + 1 - f = 2f + 1$ correct votes arrive. The measured multiplicative latency growth for an increase in replicas ($\sim$ 1.74, 1.31, 1.29, 1.21, ...), follows the series of ratios $\frac{h(f+1)}{h(f)} = \frac{2(f+1)+1}{2f+1}, f \in [1, 2, ...] \rightarrow (\sim 1.66, 1.4, 1.28, 1.22, ...)$ with some variance for small replica numbers.

7.4 Impact of Loss

Loss is an expected variable in practical BFT-SMR systems, caused by underlying infrastructure characteristics or byzantine behavior. We study the practical performance impact of different transport protocols and RTO configuration in lossy scenarios. For this study, we assume uniformly distributed loss probabilities in the network. This models an unpredictable network with increasingly bad Quality of Service (QoS). Since the location and timing of loss are unpredictable, heuristics, or e.g., intelligent leader rotation, do not suffice to alleviate loss impact. Hence, the effect of transport protocol choice and configuration are emphasized. While this impact may be smaller for targeted loss scenarios (e.g., loss only on one or current leader link), the underlying mechanics are the same.

We apply a range of uniform loss probabilities $l = 0.0025 \cdot 2^k, k \in [0, 6]$ to all links of our setup (Figure 3). Each occurring packet loss is independent. The loss behavior is implemented using traffic

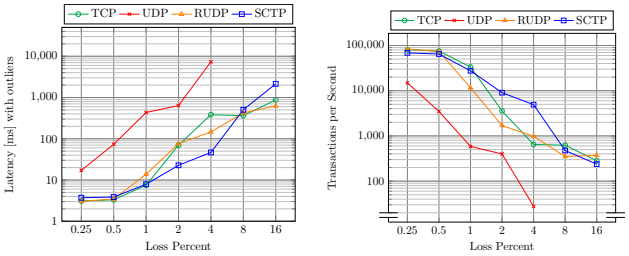


Figure 8: Loss Performance, p=0 B, n=7, batch size = 50

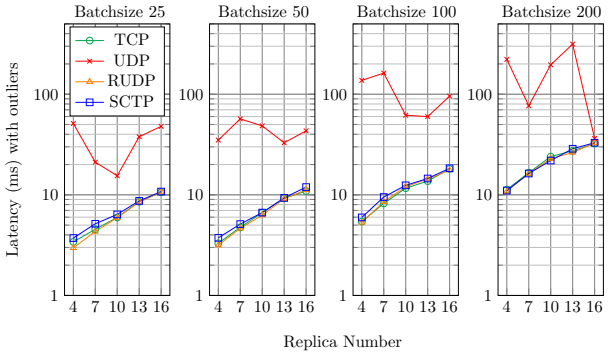


Figure 9: Replica Scaling, 0.5% loss, noTLS, p=0 B

control netem [24] of the Linux kernel, which drops a percentage of outgoing packets before they are queued [25]. For the upcoming evaluation, we recall that latency (and TPS) values are the result of processing, spread over multiple nodes, and involving more than four message roundtrips (Figure 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.

7.4.1 Transport Protocol Variation. Figure 8 shows latency (left) and throughput (right) of different transport protocols under increasing loss values. The setup is fixed to a payload size of 0 B, seven replicas, and a batch size of 50 to ensure input request saturation. For UDP, lacking retransmission logic, we observe worse results, or even not a single completed quorum at all (8%, 16%). Except for UDP, we observe roughly comparable performance for all protocols up to and including 1% loss. In this range, SCTP performs generally worse than TCP due to larger protocol overhead and larger minimum RTO (§7.4.2). RUDP generally performs worse than TCP, due to its naive retransmission implementation, whereupon loss of a single fragment the whole payload must be retransmitted.

For loss 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. 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; §5.4) and (2) inconveniently occurring losses of TCP packets and respective DupAcks, resulting in a series of high-latency leader progress stalls. Throughput values follow the latency analysis. Overall, in the "stable" loss range $\leq$ 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.

Next, we vary the number of replicas and batch size. Figure 9 shows the latency 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 two core trends in this setup. First, increasing replica numbers results in higher latencies and reduces latency differences between transport protocols. This follows directly from the larger fault threshold of the setup. More inconveniently placed packet losses need to occur at the same time at the right links to e.g., prevent at least $f+1$ votes from reaching the leader in time. For small replica numbers, protocol differences are significant, with RUDP outperforming SCTP by up to $\sim$ 20%, due to smaller protocol overhead and RTO. Second, increasing batch size reduces latency differences between transport protocols. We see the already discussed (Figure 6) general latency increase for setups without input request saturation (batch size $>$ 50), shadowing smaller latency differences. In summary, transport protocol performance differences show consistent significance with low variance in small replica setups for small loss probabilities.

Table 2: SCTP 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

7.4.2 RTO Variation. Inconveniently occurring packet loss can cause omission of ACKs or retransmission requests, resulting in an RTO (§5.4). Since RTO duration (e.g., 200 ms TCP min RTO) can exceed a regular HotStuff decision latency (e.g., ~10 ms for TCP, noTLS, $n = 16, p = 0, b = 50$) by an order of magnitude, we study the impact of RTO modification on HotStuff performance. We choose SCTP as transport protocol for these experiments, since SCTP RTO configuration values can be comfortably modified on socket level. Table 2 specifies four RTO configuration profiles, for initial (*Init*), minimum (*Min*), and maximum (*Max*) RTO values, as well as DelACK time (§5.2). All values are given in ms. *Default* describes the default SCTP settings, *TCPmock* emulates the default values of our used Linux TCP stack, and *S100* and *S50* apply a flat, *static* configuration of 100 ms and 50 ms respectively.

Figure 10 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 ensure input request saturation. We observe that in the "stable" loss region ($\leq$ 1% loss), smaller RTO values generally result in increased performance. The S50 profile consistently performs best, with e.g., up to $\sim$ 35% less latency than the default configuration for 1% loss. For loss values $>$ 1%, the variance of latency values

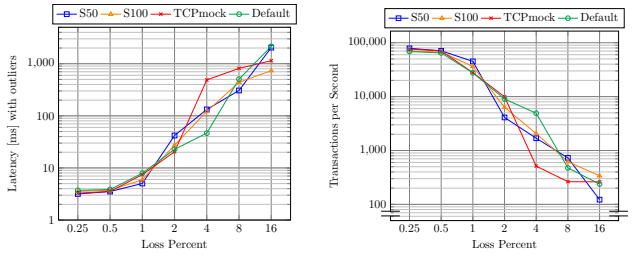


Figure 10: Loss Performance, p=0 B, n=7, batch size = 50

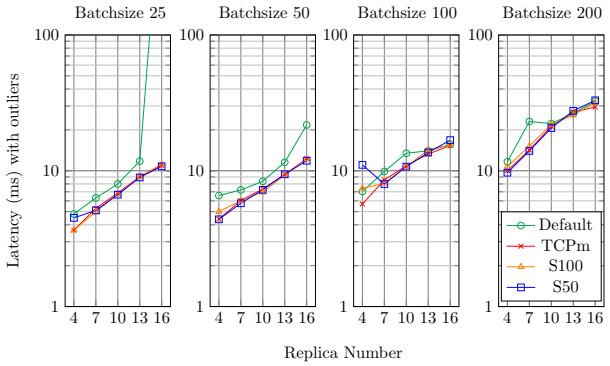
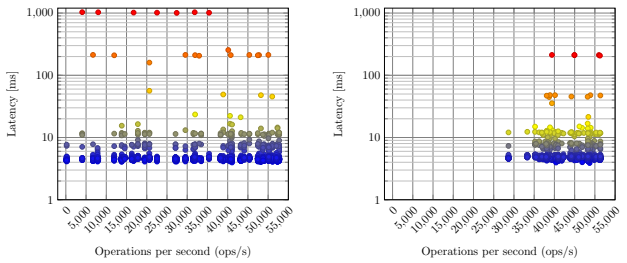


Figure 11: Replica Scaling, 1% loss, noTLS, p=0 B

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 layer 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 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.

Next, we study the effect of RTO modification over increasing replica numbers. Figure 11 shows the latency of different RTO configurations under increasing replica numbers, grouped by different batch sizes. The setup is fixed to a payload size of 0 B, and 1% loss. We observe that RTO optimization generally improves latency in saturated setups (batch size ≤ 50), even for increasing replica numbers. For non-saturated setups (batch size > 50), improvements are shadowed by the generally increasing latency values. Due to the considerable 1% loss, bigger outliers are already visible, e.g., batchsize25-Default-n16 or batchsize100-S50-n4.

Finally, we demonstrate that performance improvements of RTO modification are primarily governed by the *minimum* RTO value.



(a) Default **(b) S50**

Figure 12: Latency, 1% loss, p=0 B, n=7, batch size = 50

Figure 12 shows scatter plots of E2E latencies of client requests (Y-axis), sorted by the average performance (op/s, one-second granularity) that was achieved for the request (X-axis). Subfigure 12a holds the data for the Default RTO profile, Subfigure 12b 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 RTOs in a single client request (batch), and Λ denotes the common processing latency without loss. In Figure 12, only plateaus at $k = 1$ 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↔Replica connection, which is plain TCP (min RTO of 200 ms) for all experiments (§7). 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. In summary, RTO optimization can improve performance under loss, if the system is not limited by another bottleneck and effective RTT $\ll$ minimum RTO.

7.5 Summary

Summing up our findings, we emphasize five core insights.

1. Input request saturation affects both throughput and latency. Let client request input rate be denoted as λ_I, replica decision rate as λ_D, and batch size as b. In the optimal case, $\lambda_I \sim b \cdot \lambda_D$. Oversaturation ($\lambda_I > b \cdot \lambda_D$) introduces unnecessary latency and can be addressed by increasing b, leading to an increase in throughput. On the contrary, undersaturation ($\lambda_I < b \cdot \lambda_D$) forfeits potential throughput. In this case, expanding λ_I can directly increase throughput, reducing b can reduce latency. A "dynamic adjustment" of batch size, built on decoupled, provable dissemination of e.g., block *waves* or *bundles* [32], can (i) contribute to improved system saturation and (ii) aid latency reduction of command backlogs after asynchronous phases, by committing entire backlogs at once [22].

2. Loss values $\geq$ 2% reduced performance by approximately an order of magnitude. 1% loss already introduced considerable variance and performance loss. Hence, transport protocol configuration is not sufficient to make BFT-SMR systems suitable for high-loss environments and additional techniques (e.g., forward error correcting

codes [31]) need to be considered. In high-loss setups, high result variance shadows the effects of transport protocol modification.

3. (D)TLS usage imposes a small performance overhead, growing for larger replica numbers, of up to ∼ 16% for TCP, $n = 16$. DTLS-based setups consistently perform worse than TLS, with e.g., a difference between ∼ 5% and ∼ 25%, between TCP / TLS and UDP / DTLS for $b = 50$. Generally, we assess the usage of TLS as secure channel to be a practical solution to realize the typical BFT-SMR assumption of authenticated communication links.

4. Apart from edge cases, we assess TCP to be the best default choice among the studied protocols. TCP is well-studied and widely available hardware offloading support (e.g., TSO) increases performance in practice. We found transport protocol differences in non-lossy scenarios to be not significant in our setup. In case of loss, performance differences between protocols and configurations become smaller for larger replica numbers. Setups with small replica numbers benefit significantly from smaller protocol overhead (e.g. (R)UDP) and configuration (e.g., RTO). Setups with large numbers of connections between nodes may require other approaches to connection management to alleviate the related overhead [10]. BFT-SMR protocols with multiplexed connections between nodes may benefit from the usage of QUIC to prevent HOL blocking, prevalent in TCP streams. To defend against targeted attacks on BFT-SMR protocols, agreement-level efforts for increased robustness or proactive-recovery [8, 11, 22, 26] should be considered.

5. Modification of the minimum RTO (min_{RTO}) value of a reliable transport protocol allows for performance improvements of leader-based consensus communication, if (1) RTOs occur and $min_{RTO} \gg RTT$, and (2) performance is not governed by other, application-specific behavior (e.g., large HotStuff implementation default timeouts in face of significant packet loss). Tuning of these timeout values is expedient. In the best case $min_{RTO} = min(RTT) + \varepsilon, \varepsilon > 0$. The actual performance improvement varies and depends on e.g., target network RTT, loss probabilities, and default timeout values.

8 CONCLUSION & FUTURE WORK

We conducted an analysis of secure channel and network stack interrelation for leader-based consensus communication. We show that delays of a single processing layer often impact all above layers, and typical optimizations such as batching or pipelining act as further amplifiers. We implemented four transport protocols and two secure channel configurations for an exemplary BFT-SMR setup, based on the well-known HotStuff protocol. Through experiments, we quantify the impact in lossy and lossless scenarios. Our results show that transport and secure channel configuration impacts BFT-SMR operation, already for small replica numbers and little loss. The results indicate that it is functional to (1) adjust the command batch size according to current input request frequency, (2) consider additional robustness measures for execution in high-loss environments, (3) if possible, employ TCP as the default protocol except for edge-cases (§7.5), (4) prefer TLS over DTLS secure channels if used, and (5) adjust both transport protocol (e.g., min RTO) and consensus algorithm timeouts to match actual system timing (e.g., RTT) as close as possible. Summing up, transport protocol optimization opens a novel optimization space for leader-based consensus and SMR systems. For future work, we aim to study the optimization potential of other BFT-SMR building blocks, as well as the impact of different congestion control algorithms on performance and robustness of BFT-SMR systems for lossy and adversarial environments.

ACKNOWLEDGMENTS

This work was partially funded by the German Research Foundation (HyperNIC, DFG grant no. CA595/13-1, and EDGAR, grant approval according to Art. 91b GG with DFG grant no. INST 95/1653-1 FUGG) and by the European Union's Horizon 2020 research and innovation programme (grant agreement no. SLICES-SC 101008468). The German Federal Ministry of Education and Research (BMBF) supported our work under the projects 6G-life (16KISK002) and 6G-ANNA (16KISK107). We thank Lucas Mair and Christoph Probst for implementation support on different HotStuff transports, and Johannes Schleger for early discussions.

REFERENCES

[1] Marcos K Aguilera, Naama Ben-David, Rachid Guerraoui, Virendra Marathe, and Igor Zablotchi. 2019. The Impact of RDMA on Agreement. In *Proceedings of the 2019 ACM symposium on principles of distributed computing*. 409–418.

[2] Marcos K Aguilera, Naama Ben-David, Rachid Guerraoui, Antoine Murat, Athanasios Xygkis, and Igor Zablotchi. 2023. 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*. 862–877.

[3] Ailidani Ailijiang, Aleksey Charapko, and Murat Demirbas. 2019. Dissecting the Performance of Strongly-Consistent Replication Protocols. In *Proceedings of the 2019 International Conference on Management of Data*.

[4] M. Allman, V. Paxson, and E. Blanton. 2009. *TCP Congestion Control*. RFC 5681. IETF. http://tools.ietf.org/rfc/rfc5681.txt

[5] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. 2013. RBFT: Redundant Byzantine Fault Tolerance. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*. IEEE, 297–306.

[6] Mathieu Baudet, Avery Ching, Andrey Chursin, George Danezis, François Garillot, Zekun Li, Dahlia Malkhi, Oded Naor, Dmitri Perelman, and Alberto Sonnino. 2019. State Machine Replication in the Libra Blockchain. *The Libra Assn., Tech. Rep* (2019).

[7] Jan Bernatik. 2022. Introduction to Flow blockchain. https://jan-bernatik.medium.com/introduction-to-flow-blockchain-7532977c8af8. [Online] last accessed 2024-02-13.

[8] Alexander Binun, Thierry Coupaye, Shlomi Dolev, Mohammed Kassi-Lahlou, Marc Lacoste, Alex Palesandro, Reuven Yagel, and Leonid Yankulin. 2016. Self-stabilizing Byzantine-tolerant distributed replicated state machine. In *Stabilization, Safety, and Security of Distributed Systems: 18th International Symposium, SSS 2016, Lyon, France, November 7-10, 2016, Proceedings 18*. Springer, 36–53.

[9] Daniel RL Brown. 2010. Sec 2: Recommended Elliptic Curve Domain Parameters. *Standars for Efficient Cryptography* (2010). [Online] http://www.secg.org/sec2-v2.pdf, last accessed 2024-02-13.

[10] Martina Camaioni, Rachid Guerraoui, Matteo Monti, Pierre-Louis Roman, Manuel Vidigueira, and Gauthier Voron. 2023. Chop Chop: Byzantine Atomic Broadcast to the Network Limit. *arXiv preprint arXiv:2304.07081* (2023).

[11] Miguel Castro and Barbara Liskov. 2002. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Transactions on Computer Systems (TOCS)* 20, 4 (2002), 398–461.

[12] Cypherium. 2020. Cypherium Whitepaper 2.0. https://www.cypherium.io/whitepaper/cypherium-whitepaper-2-0/. [Online] last accessed 2024-02-13.

[13] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. 2022. Narwhal and Tusk: A DAG-based Mempool and Efficient BFT Consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*. 34–50.

[14] Huynh Tu Dang, Pietro Bressana, Han Wang, Ki Suh Lee, Noa Zilberman, Hakim Weatherspoon, Marco Canini, Fernando Pedone, and Robert Soulé. 2020. P4xos: Consensus as a Network Service. *IEEE/ACM Transactions on Networking* 28, 4 (2020), 1726–1738.

[15] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1988. Consensus in the Presence of Partial Synchrony. *Journal of the ACM (JACM)* 35, 2 (1988), 288–323.

[16] Wesley Eddy. 2022. *Transmission Control Protocol (TCP)*. RFC 9293. IETF. https://doi.org/10.17487/RFC9293

[17] Fangyu Gai, Ali Farahbakhsh, Jianyu Niu, Chen Feng, Ivan Beschastnikh, and Hao Duan. 2021. Dissecting the Performance of Chained-BFT. In *2021 IEEE*

41st International Conference on Distributed Computing Systems (ICDCS). IEEE, 595–606.

[18] Fangyu Gai, Jianyu Niu, Ivan Beschastnikh, Chen Feng, and Sheng Wang. 2022. Scaling Blockchain Consensus via a Robust Shared Mempool. *arXiv:2203.05158* (2022).

[19] Sebastian Gallenmüller, Dominik Scholz, Henning Stubbe, and Georg Carle. 2021. 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. https://doi.org/10.1145/3485983.3494841

[20] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2022. Dumbo-NG: Fast Asynchronous BFT Consensus with Throughput-Oblivious Latency. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 1187–1201.

[21] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. 2022. 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*. Springer, 296–315.

[22] Neil Giridharan, Florian Suri-Payer, Ittai Abraham, Lorenzo Alvisi, and Natacha Crooks. 2024. Motorway: Seamless high speed BFT. *arXiv preprint arXiv:2401.10369* (2024).

[23] Neil Giridharan, Florian Suri-Payer, Matthew Ding, Heidi Howard, Ittai Abraham, and Natacha Crooks. 2023. BeeGees: Stayin' Alive in Chained BFT. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing*. 233–243.

[24] Stephen Hemminger et al. 2005. Network emulation with NetEm. In *Linux conf au*, Vol. 5. 2005.

[25] Audrius Jurgelionis, Jukka-Pekka Laulajainen, Matti Hirvonen, and Alf Inge Wang. 2011. An Empirical Study of NetEm Network Emulation Functionalities. In *2011 Proceedings of 20th international conference on computer communications and networks (ICCCN)*. IEEE, 1–6.

[26] Dakai Kang, Sajjad Rahnama, Jelle Hellings, and Mohammad Sadoghi. 2023. Practical View-Change-Less Protocol through Rapid View Synchronization. *arXiv:2302.02118* (2023).

[27] Leslie Lamport. 1998. The Part-Time Parliament. *ACM Transactions on Computer Systems (TOCS)* 16, 2 (1998), 133–169.

[28] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. 2017. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *Conference proceedings – ACM special interest group on data communication*.

[29] Wenyu Li, Chenglin Feng, Lei Zhang, Hao Xu, Bin Cao, and Muhammad Ali Imran. 2020. A Scalable Multi-Layer PBFT Consensus for Blockchain. *IEEE Transactions on Parallel and Distributed Systems* 32, 5 (2020), 1146–1160.

[30] Ming Liu, Tianyi Cui, Henry Schuh, Arvind Krishnamurthy, Simon Peter, and Karan Gupta. 2019. Offloading Distributed Applications onto SmartNICs using iPipe. In *Proceedings of the ACM Special Interest Group on Data Communication*. 318–333.

[31] Thomas Lorünser, Benjamin Rainer, and Florian Wohner. 2022. Towards a Performance Model for Byzantine Fault Tolerant Services. In *CLOSER*. 178–189.

[32] Dahlia Malkhi and Maofan Yin. 2023. Lessons from HotStuff. In *Proceedings of the 5th workshop on Advanced tools, programming languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems*. 1–8.

[33] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. 1996. *TCP Selective Acknowledgment Options*. RFC 2018. IETF. http://tools.ietf.org/rfc/rfc2018.txt

[34] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The Honey Badger of BFT Protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 31–42.

[35] Joachim Neu, Srivatsan Sridhar, Lei Yang, David Tse, and Mohammad Alizadeh. 2021. Longest Chain Consensus Under Bandwidth Constraint. Cryptology ePrint Archive, Paper 2021/1545. https://eprint.iacr.org/2021/1545 https://eprint.iacr.org/2021/1545.

[36] Yoshihiro Ohba, Jing Yi Koh, Nigel Ng, and Sye Loong Keoh. 2021. Performance Evaluation of a Blockchain-based Content Distribution over Wireless Mesh Networks. In *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*. IEEE, 258–263.

[37] Afonso Oliveira, Henrique Moniz, and Rodrigo Rodrigues. 2022. Alea-BFT: Practical Asynchronous Byzantine Fault Tolerance. *arXiv:2202.02071* (2022).

[38] Diego Ongaro and John Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*. 305–319.

[39] V. Paxson, M. Allman, J. Chu, and M. Sargent. 2011. *Computing TCP's Retransmission Timer*. RFC 6298. IETF. http://tools.ietf.org/rfc/rfc6298.txt

[40] J. Postel. 1980. User Datagram Protocol. RFC 768 (INTERNET STANDARD). http://www.ietf.org/rfc/rfc768.txt

[41] E. Rescorla. 2018. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. IETF. http://tools.ietf.org/rfc/rfc8446.txt

[42] E. Rescorla, H. Tschofenig, and N. Modadugu. 2022. *The Datagram Transport Layer Security (DTLS) Protocol Version 1.3*. RFC 9147. IETF. http://tools.ietf.org/rfc/rfc9147.txt

[43] George F Riley and Thomas R Henderson. 2010. The ns-3 network simulator. In *Modeling and tools for network simulation*. Springer, 15–34.

[44] Signe Rüsch, Ines Messadi, and Rüdiger Kapitza. 2018. Towards Low-Latency Byzantine Agreement Protocols Using RDMA. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 146–151.

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

[46] Yahya Shahsavari, Kaiwen Zhang, and Chamseddine Talhi. 2022. Performance Modeling and Analysis of Hotstuff for Blockchain Consensus. In *2022 Fourth International Conference on Blockchain Computing and Applications (BCCA)*. IEEE, 135–142.

[47] Man-Kit Sit, Manuel Bravo, and Zsolt István. 2021. 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*. 55–65.

[48] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. 2022. Bullshark: DAG BFT Protocols Made Practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2705–2718.

[49] Chrysoula Stathakopoulou, Matej Pavlovic, and Marko Vukolić. 2022. State Machine Replication Scalability Made Simple. In *Proceedings of the Seventeenth European Conference on Computer Systems*. 17–33.

[50] Chrysoula Stathakopoulou, David Tudor, Matej Pavlovic, and Marko Vukolić. 2022. Mir-BFT: Scalable and Robust BFT for Decentralized Networks. *Journal of Systems Research* 2, 1 (2022).

[51] R. Stewart, M. Tüxen, and K. Nielsen. 2022. *Stream Control Transmission Protocol*. RFC 9260. IETF. http://tools.ietf.org/rfc/rfc9260.txt

[52] The kernel development community. 2023. Segmentation Offloads. *The Linux Kernel documentation* next-20230225 (2023). [Online] https://www.kernel.org/doc/html/next/networking/segmentation-offloads.html, last accessed 2024-02-13.

[53] M. Thomson. 2018. *Record Size Limit Extension for TLS*. RFC 8449. IETF. http://tools.ietf.org/rfc/rfc8449.txt

[54] András Varga and Rudolf Hornig. 2010. An overview of the OMNeT++ simulation environment. In *1st International ICST Conference on Simulation Tools and Techniques for Communications, Networks and Systems*.

[55] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, and Lau Cheuk Lung. 2009. Spin One's Wheels? Byzantine Fault Tolerance with a Spinning Primary. In *2009 28th IEEE International Symposium on Reliable Distributed Systems*. IEEE, 135–144.

[56] Richard Von Seck, Filip Rezabek, Benedikt Jaeger, Sebastian Gallenmüller, and Georg Carle. 2022. BFT-Blocks: The Case for Analyzing Networking in Byzantine Fault Tolerant Consensus. In *2022 IEEE 21st International Symposium on Network Computing and Applications (NCA)*, Vol. 21. 35–44. https://doi.org/10.1109/NCA57778.2022.10013509

[57] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. 2018. HotStuff: BFT consensus in the lens of blockchain. *arXiv preprint arXiv:1803.05069* (2018).

[58] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT Consensus with Linearity and Responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. ACM.

[59] Gengrui Zhang, Fei Pan, Yunhao Mao, Sofia Tijanic, Michael Dang'ana, Shashank Motepalli, Shiquan Zhang, and Hans-Arno Jacobsen. 2024. Reaching Consensus in the Byzantine Empire: A Comprehensive Review of BFT Consensus Algorithms. *Comput. Surveys* 56, 5 (2024), 1–41.