# Towards Improving the Performance of BFT Consensus For Future Permissioned Blockchains

Manuel Bravo, Zsolt István
IMDEA Software Institute, Madrid
{manuel.bravo, zsolt.istvan}@imdea.org

Man-Kit Sit
City University of Hong Kong
manksit@cityu.edu.hk



***Abstract*—Permissioned Blockchains are increasingly considered in enterprise use-cases, many of which do not require geo-distribution, or even disallow it due to legislation. Examples include country-wide networks, such as Alastria, or those deployed using cloud-based platforms such as IBM Blockchain Platform. We expect these blockchains to eventually run in environments with high bandwidth and low latency modern networks, as well as, advanced programmable hardware accelerators in servers.**

**Even though there is renewed interest in BFT consensus algorithms with various proposals targeting Permissioned Blockchains, related work does not optimize for fast networks and does not incorporate hardware accelerators – we make the case that doing so will pay off in the long run. To this end, we re-implemented the seminal PBFT algorithm in a way that allows us to measure different configurations of the protocol. Through this we explore the benefits of various common optimization strategies and show that the protocol is unlikely to saturate more than 10Gbps networks without relying on specialized hardware-based offloading. We discuss two concrete ways in which the cost of consensus in Permissioned Blockchains could be reduced in high speed networking environments, namely, offloading to SmartNICs and implementing the protocol on standalone FPGAs.**

## I. Introduction

Blockchain is an emerging technology, considered increasingly often beyond the crypto-currency world for business-to-business use-cases. In contrast to public blockchains such as Bitcoin, that are open systems in which anyone can participate, in business-to-business scenarios the membership of the service is tightly controlled and this permits the use of Byzantine fault tolerant (BFT) consensus protocols at the core of the service to establish a total order of transactions, instead of the more expensive Proof-of-Work-based consensus protocols. Such systems are called *permissioned blockchains* [10], [15], [38]. In a permissioned blockchain system, often only a subset of the total number of participating nodes run the BFT protocol [8], [34] and, in general, members have more control over how and where to run the network [27].

Driven by opportunities in blockchain, there has been an increased interest in BFT consensus protocols [45], [25], [41]. Interestingly, the deployment model of permissioned blockchains can be very different from permissionless ones likes Bitcoin. While the latter is typically widely distributed, with bandwidth and latency characteristics much like that of the world wide web, in the permissioned blockchain space, there are use cases where nodes are under tighter control (e.g., those in hosted environments on Amazon Managed Blockchain [1] or IBM Blockchain Platform [4]), perhaps even geographically confined (e.g., emerging country-wide networks, such as Alastria [37] in Spain). Nodes of such deployments have access to more bandwidth, lower latency communication than what we associate today with blockchains. Given the increasing presence of programmable hardware devices in public clouds [23], [14], [30], it is likely that nodes could even rely on these for increasing performance. If successful, blockchain technology will likely replace several database solutions in the area of banking, trading and e-commerce but the performance of today's permissioned blockchains will not be satisfactory. For this reason, it is important to start investigating strategies for increasing the speed of BFT consensus using modern hardware available in the clouds and datacenters. As we show in Section II, current BFT consensus implementations are unable saturate bandwidths of 10Gbps and higher, while retaining low latency.

Our goal is to investigate how far can software get us and to what extent will it be useful, or even necessary, to use hardware accelerators in the future. We apply a experiment-driven approach to quantify the benefits of various existing optimization strategies. For this, we build a framework that integrates a streamlined variant of the seminal PBFT [18], [17] consensus protocol and can be configured at multiple levels. Our study reveals that, even after applying various optimizations, achieving 10Gbps performance in software without relying on very large batches is still unlikely – the road the 100Gbps rates will hence have to involve some forms of hardware accelerators. We can also confirm that contrary to anecdotal evidence, hardware accelerators for cryptographic operations alone will not result in significantly better performance because the biggest cost, even in modestly sized consensus groups, is that of packet parsing and hashing, that is, data-movement related operations.

To alleviate the cost of these operations, we propose two strategies for incorporating specialized hardware, namely, emerging Smart Network Interface cards (SmartNICs) and standalone FPGA nodes to provide line-rate, predictable behavior. We present micro-benchmarks motivating these strategies and discuss their main benefits and open challenges.

Overall, this work brings three contributions:

- We identify the future need for low latency and high bandwidth BFT consensus. Even though today the chal-

lenges of Permissioned Blockchains lie in determining the right governance model and integration with data management solutions, if successful, these blockchains will have to provide high throughput, low latency, and the ability to scale with faster networks and more powerful hardware.
- We provide an open-source framework[1] for experimenting with various software and hardware strategies for accelerating PBFT and similar protocols.
- Based on measurements carried out with our framework, we identify two specific hardware acceleration scenarios that will be able to saturate 10Gbps and faster networks even for small consensus groups.

## II. Motivation

In this section, we motivate the need for investigating performance-related aspects of BFT consensus protocols in environments with high network bandwidth and low latency. We first discuss three use-cases in which consensus nodes are geographically confined by design or have access to high bandwidth networking and their location can be controlled. Second, we show that state-of-the-art BFT consensus cannot efficiently take advantage of fast networks, further motivating the "de-construction" of the underlying protocol for measurement purposes.

### A. Use-cases

**Single and Multi-Cloud Hosted Blockchains.** Several cloud providers are offering hosted blockchain solutions both as Software as a Service (e.g., Azure Blockchain Service), and by simplifying the deployment of open-source, commonly used, networks such as Hyperledger Fabric (e.g., in IBM Cloud Blockchain Platform). Given that a large portion of web-facing applications already run in hosted environments, running blockchains as a "backend" in the cloud is a natural step in many scenarios. In cloud environments the blockchain nodes have access to high bandwidth networking and low latencies within regions, and it is also to be expected that with the emergence of more enterprise use-cases for permissioned ledgers, multi-cloud deployments will rely on dedicated links for higher bandwidth across clouds and reduced data movement costs, as it is already the case for CDNs. For this reason, when preparing the next generation of Blockchain-focused BFT consensus libraries, it is crucial to design with high bandwidth (10Gbps and above) and low latency networks in mind.

**Regional Replication by Design.** There are emerging country-wide permissioned blockchain networks such as Alastria [8], [37] in Spain, that set out to provide a mechanism for any company within a consortium to interact with any other one through smart contracts that are recognized under the local legislation. For this reason, these kinds of networks are run in geographically more confined environments. Furthermore, in this type of networks, only a small subset of the consortium nodes take part actively in consensus (i.e., running the core operations of the network), making them the critical performance point of the network. Thus, due to these two characteristics, we expect the consortium to optimize the environment in which consensus nodes are deployed. Hence, when this type of networks matures, we expect the group of consensus nodes to run in a environment with high bandwidth and low latency, as well as, have advanced programmable hardware accelerators (already commonly found in servers) at their disposal.

[1]Link removed for double blind submission.

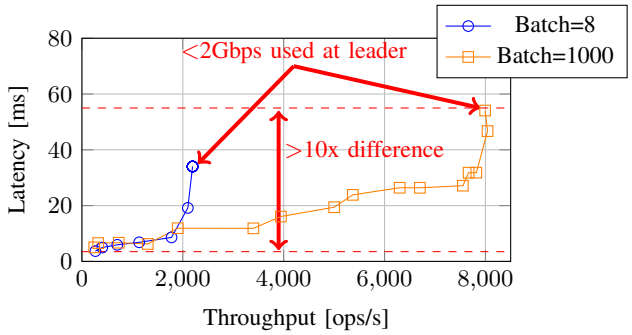


Fig. 1. State-of-the-art BFT consensus libraries fail to efficiently utilize 10Gbps LAN connections (BFT-Smart with 7 nodes, 4KB payload, no logging)

**Geo-replicated Systems with Local Optimizations.** The recent work by Gupta et al. [27], called ResilientDB, argues that, in order to build practical geo-distributed databases based on blockchain technology, it is crucial to minimize cross-region communication between nodes without reducing reliability or availability guarantees. The proposed design has a hierarchical consensus mechanism that runs several BFT consensus groups, with nodes close by, and performs a geo-replication using a step that requires only a linear number of messages in the failure-free case. Thus, given that the performance of the consensus groups would drive the overall performance of the system and that nodes within a group are close by, one can expect these groups to be deployed in a environment with high network resources.

### B. State-of-the-art BFT Consensus and Fast Networks

To experimentally show that state-of-the-art BFT consensus protocols are unable to take advantage of fast networks, we measure the performance of BFT-Smart [13], one of the most optimized BFT consensus libraries available, on such networks. We have chosen BFT-Smart for a few reasons: (i) with more that five years of development, we consider it a serious attempt of implementing a highly-performant BFT consensus, (ii) it integrates a consensus protocol similar to the seminal PBFT protocol, on which most of the modern BFT consensus protocols are based, (iii) it includes a set of optimizations and refinements aiming to enhance performance such as multi-core awareness and the use of *cheaper* cryptographic operations when possible, and (iv) it is open source and actively supported.

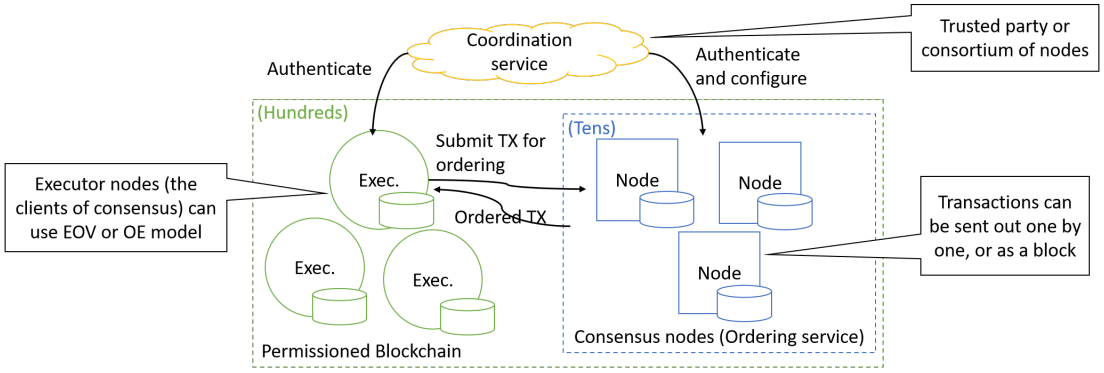


Fig. 2. In permissioned blockchains a coordination service authenticates nodes in the network and performs configuration. There are significantly less consensus nodes than regular member nodes and their churn is minimal.

We configured BFT-Smart with 7 nodes with RSA1024 signatures and MACs among the nodes, on server-grade machines connected over a 10Gbps LAN (see the Experiments section for details). We experiment with two batch sizes: 1000 request per batch (the default), and a smaller one with 8 requests per batch. Figure 1 reports the average latency vs. throughput achieved by BFT Smart when running the YCSB benchmark[2] with 4KB values. The experiments show that the leader node is far from saturating the network connection. In fact, it uses significantly less than 2Gbps-worth of bandwidth even at saturation – showing that there is a need to explore how to design BFT consensus solutions that can saturate 10Gbps bandwidth, and beyond.

Furthermore, BFT-Smart is unable to keep latencies consistently low while delivering high throughput. As Figure 1 shows, BFT-Smart exhibits a latency that is more than an order of magnitude greater than the network's response time. This is mainly because, in order to enhance throughput, BFT-Smart employs batching aggressively: it composes large batches of messages in an attempt to reduce the overhead of consensus. Figure 1 shows that when significantly reducing the batch size, the difference in throughput is stark: the throughput drops by 4x. In this work, we investigate the inherent tension between latency and throughput in BFT consensus protocols.

## III. Background

### A. Permissioned Blockchains

Public blockchains are often associated with cryptocurrencies and are characterized by the fact that nodes can join without permission. For this reason, many of these blockchains implement Proof of Work, or similar, consensus methods [42] and are designed without assumptions about the nodes. Permissioned ledgers [10], [15], [38], in contrast, rely on a trusted service or consortium to authenticate nodes when joining the blockchain, but do not assume trustworthiness of nodes. This is useful in business-to-business scenarios where the goal of the blockchain is not to offer anonymity but rather to logically centralize data and run tamper-free "smart contracts", application logic, on it. Example use-cases include ones in health care [11], supply chain management [32], etc., but also in areas such as banking and capital markets [16]. In these scenarios all actors in the system are known but they want to protect against malicious actions from the others.

As seen in Figure 2, permissioned blockchains are composed of executor nodes and consensus nodes. For generality, we consider these sets of nodes to be disjoint but, in practice, a node could implement both roles. Clients of the blockchain system, i.e., users, are external to this illustration. The coordination service shown in the image is the trusted third party or consortium of nodes (that all members of the blockchain trust) that authenticates nodes, configures the network, etc.

Permissioned ledgers usually provide one of two execution models: order-execute (OE), or execute-order-validate (EOV). The first means that smart contracts with their specific inputs are submitted first to the ordering service, that is, the consensus nodes, and then executed on all nodes of the network. The EOV model simulates the contract execution on a subset of the nodes and submits the resulting "read-write set" for ordering. The nodes receive these from the ordering service and update their state if the read-write sets do not conflict with the ledger state. Even though these two models offer different trade-offs, from the perspective of the underlying consensus logic, they are very similar. For this reason in this work we investigate BFT ordering without assuming one or the other execution model.

The question of how executor nodes "get" the ordered transactions is also orthogonal to our investigation. We make the assumption that, in general, executor nodes are interested in pulling transactions from the ordering service as soon as they are ordered (they can access at the block granularity to recover and to gossip). With this assumption it is beneficial to explore not only the throughput but also the latency improvements one could add to BFT protocols. This is relevant as there is recent work on exploring how the throughput [24], [40], [9] and latency [29] of blockchains can be improved substantially in the presence of fast networks.

[2]Default YCSB configuration from the BFT-Smart repository: updates only and a single field per entry to avoid computational overhead in the nodes

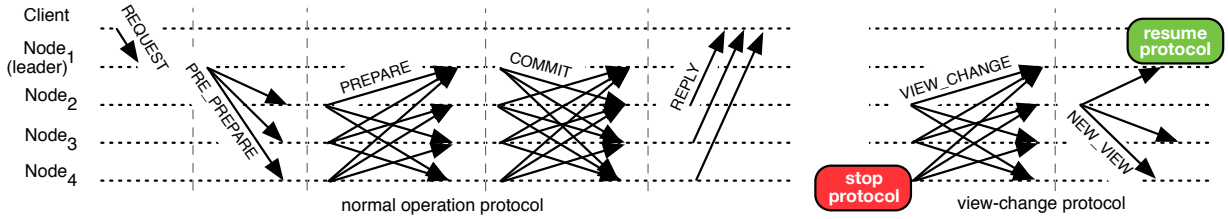


Fig. 3. PBFT communication pattern during failure-free operations and recovery.

### B. PBFT

The seminal PBFT [18] protocol is one of the most well-studied protocols and that is why we use it in our study. The protocol requires a minimum of $3f + 1$ nodes to tolerate $f$ faulty nodes. For simplicity, in this section, we consider the variant of PBFT that uses public-key signatures. We depict its communication pattern in Figure 3.

The protocol proceeds in rounds called *views*. On each view, one node is the leader and the rest are its followers. The protocol moves to the next view only if the leader is faulty or if asynchrony prevents the protocol to make progress. The process of changing view is called *view-change*. At a given view $v$, the leader sequences and proposes client's request in a `PRE_PREPARE` message to the followers. When a follower receives a `PRE_PREPARE` message, it first validates the leader's proposal by checking the authenticity of the client's request and that it does not have another client request already assigned to that sequence number. If the follower *accepts* the request, it sends a `PREPARE` message to all nodes. When a node (leader or follower) receives $2f + 1$ matching `PREPARE` messages, it considers the request as *prepared* and sends a `COMMIT` message to all nodes. Intuitively, theses first two steps of the protocol (leader's proposal and the all-to-all communication step) ensure that correct nodes agree on a total order for the requests within a view. When a node receives $2f + 1$ matching `COMMIT` messages for a client's request and all requests with a lower sequence number have been committed locally, the node considers the request as *committed* and replies to the client. This second all-to-all communication step, together with the view-change protocol, guarantees that correct replicas agree on the sequence numbers assigned to committed requests even when committing them across views. Finally, a client waits for $f+1$ matching replies before accepting the result.

When a node wants to move to the next view $v + 1$, it first stops executing the protocol and sends a `VIEW_CHANGE` message to all nodes. A node sends in its `VIEW_CHANGE` message all client requests that could have been committed. When the leader of $v + 1$ gathers $2f + 1$ of these messages, it computes the final set of potentially committed requests $\mathcal{O}$ and sends it in a `NEW_VIEW` message to its followers, together with the $2f + 1$ `VIEW_CHANGE` messages based on which $\mathcal{O}$ was computed. Upon reception of a `NEW_VIEW` message, a follower first verifies the correctness of $\mathcal{O}$. Then, it adds the new information to its local state and resumes execution.

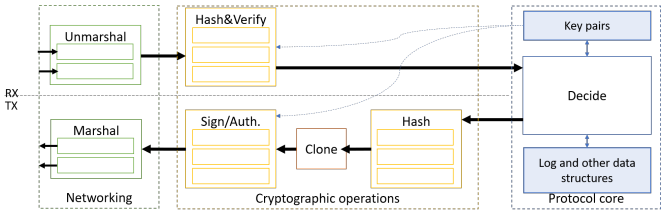


Fig. 4. The consensus logic is implemented as a software pipeline, with data parallel execution for more compute-intensive steps.

## IV. Experimental Framework

Our main goal is to study the effect of various optimizations of BFT consensus with the permissioned blockchain use-case in mind. Our framework integrates a streamlined variant of PBFT [18], [17]. We expect that the findings of this work are directly applicable to other BFT consensus protocols [26], [33], [45], [41], that mostly are optimized variants of PBFT.

One difference between our experimental framework and typical BFT consensus implementations is that we do not rely on batching by default. Our goal is to study the cost of running consensus without batching, or with very small batches, to ensure that the latency of the protocol is representative of the underlying network latency.

### A. Design

Implementing consensus protocols on multi-core CPUs leads to the question of how to exploit the available parallelism given that at their core, all protocols, including PFBT, require a serial decision making step. Many BFT and CFT consensus implementations adopted a pipelined architecture [19], [39], [13] and in this work we do the same. At its core, our framework integrates PBFT⋆, a streamlined variant of PBFT.

Pipelined architectures are also beneficial because it is easier to envision the integration of various accelerators than into monolithic ones. One drawback, however, of the pipelining approach is that performance can be bottlenecked on the slowest pipeline stage; for this reason we will consider parallelism both across pipeline stages and within pipeline stages wherever possible.

The framework is parametric to the type of cryptographic operation used for the authentication of different message types: one can choose between public-key (PK) signatures

or message authentication codes (MACs). We use in our experimental analysis the following three configurations:

1) *Off-the-shelf:* This variant makes no assumptions about the system around it and uses PK signatures on all messages.
2) *Algorithmic optimizations:* It replaces PK signatures with MACs on all inter-node messages, similar to the optimizations described in the journal version of PBFT [17].
3) *Domain-optimized for Permissioned Blockchains:* It further eliminates PK signatures on responses to clients because client requests (transactions to order) will be logically packed into blocks and it is enough to sign the blocks with a PK and responses to clients with MACs.

Incoming messages from clients are by default using private-key signatures in all variants to counter malicious clients (i.e., big-MAC attack [20]).

Furthermore, one can finely tune the batching sizes to explore the latency-throughput tradeoff when combined with cryptographic operations; and the parallelism of tasks, such as the computation or verification of cryptographic signatures.

Figure 4 shows the stages in our implementation that follow the main steps of the protocol. The only part of the behavior that requires more explanation is the sending hashing and signing: The *Hash(TX)* step hashes messages to prepare them for signing/authentication. The operation in carried out in parallel for multiple messages (round-robin). Once messages have their hashes computed, they are *cloned* to create multiple copies of them to be sent later to individual recipients. If, for instance, a `PRE_PREPARE` message has to be sent to all participants, it is hashed once in the previous step and then cloned in this step for each recipient. The *Sign/Auth.* step computes the signature/MAC to be attached to each message in parallel. This layout is advantageous because by default our implementation is set up to compute different signatures and MACs for each recipient. For protocol variants in the evaluation that only use public-key signatures, this step is merged with hashing to avoid redundant computation.

### B. Implementation

*Pipeline Execution Model.* The PBFT$\star$'s pipeline decouples the building blocks of the protocol and allows for future exploration of different acceleration opportunities (Figure 4). The stages are as follows:

1) *Unmarshal*: incoming messages are received on TCP/IP connections and unmarshaled, using one thread for each individual node and client.
2) *Hash (RX) and Verify*: each message is signed by its sender using their private-key or authenticated using a MAC. In either case, the message contents need to be hashed and this hash has to be compared to the one in the signature/auth. This operation is performed by multiple threads in a data-parallel manner using round-robin scheduling to maintain FIFO order of messages.
3) *Decision*: This is where the protocol itself is running. Depending on the internal state and the content of the incoming message, this step will produce one or multiple messages with a list of recipients each.
4) *Hash (TX)*: This step hashes messages to prepare them for signing/authentication. The operation in carried out in parallel for multiple messages (round-robin).
5) *Clone*: Once messages have their hashes computed, this step creates multiple copies of them to be sent later to individual recipients. If, for instance, a `PRE_PREPARE` message has to be sent to all participants, it is hashed once in the previous step and then cloned in this step for each recipient.
6) *Sign/Auth*: This step computes the signature/MAC to be attached to each message in parallel. This layout is advantageous because by default PBFT$\star$ is set up to compute different signatures and MACs for each recipient. For protocol variants in the evaluation that only use public-key signatures, this step is merged with hashing to avoid redundant computation.
7) *Marshal*: Signed messages with one recipient each are enqueued on threads representing individual TCP/IP sockets. These perform the serialization of the messages.

*Implementation Decisions and Optimizations.* We implemented our prototype in Golang relying heavily on goroutines for parallelism. We use the SHA256 cryptographic hash function to compute digests, RSA-2048 for signatures and AES with 256bit keys for MACs, using default Golang libraries. The messages exchanged between nodes are serialized using Protocol Buffers and follow a similar layout with a fixed set of integer fields followed by a variable length "attached data" field.

Since we do not want to restrict the applicability of our prototype, the messages coming from the clients are treated as BLOBs that are recorded in a log. They are not applied, in the traditional sense, to a state database. This is because in most blockchain systems the ordering service does not actually look at the contents of "transactions". And even if some processing of these transactions would be necessary, it can be performed off the critical path. This choice, however, introduces a question related to state compaction. While in state machine replication this can happen implicitly at specific intervals on all nodes (e.g., after each successful checkpoint), in a scenario we are looking at, compaction can only be done from the "outside" when all clients can agree. In our prototype we keep a log of 10k operations in memory that acts as a circular buffer and we have set the checkpointing frequency to 500 messages to keep most of the data structures fit in cache. In a full implementation, the log would have to be written to disk asynchronously for durability, and a suitable compaction method would have to be chosen.

Similarly, there is a decision to be had in the system whether the blockchain entries can be gossiped or not by the clients. If no (our default assumption), it is sufficient to use MACs to authenticate messages between ordering nodes and clients and for each new client to read blocks directly from the ordering service when recovering state. If yes, the nodes need to sign

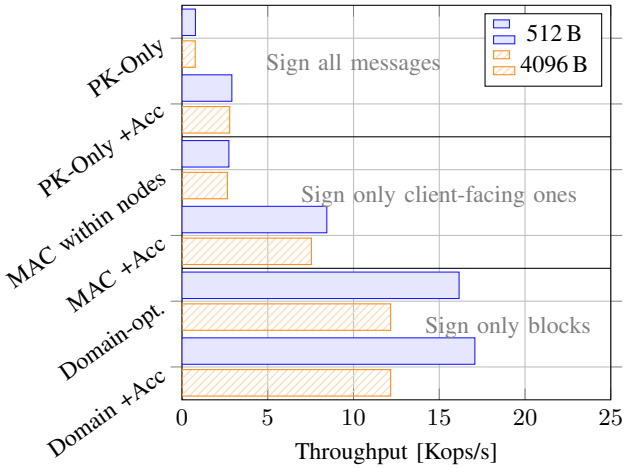


Fig. 5. We emulate different configurations with 15 nodes to quantify the expected benefits of hardware accelerators and protocol optimizations. The results show that the avoidance of PK cryptography is the most important performance factor.

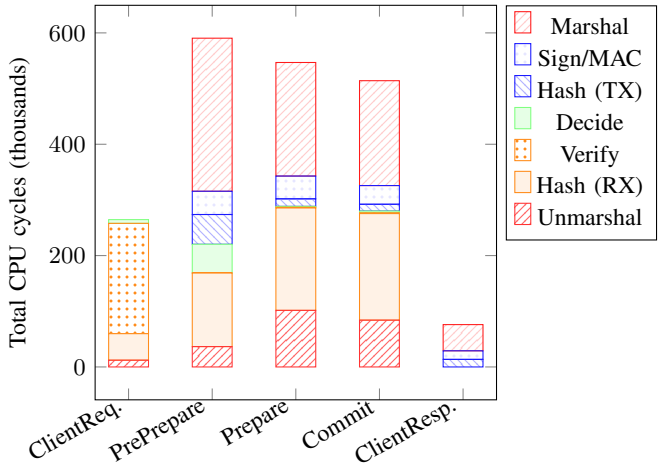


Fig. 6. The aggregate cost of each pipeline stage in a domain-optimized setup (in CPU cycles) at the leader for 15 nodes, processing 4KB values, shows that (un)marshaling and hashing before verification are the most expensive.

client responses with a PK signature. Our prototype offers both options and the signatures can be enabled with an environment variable. Incoming messages from clients are by default using private-key signatures, even if the answers are only signed with MACs, to counter malicious clients (big-MAC attack [20]).

## V. Study of Protocol Variants

All experiments are performed on a consensus group of 15 nodes, with the clients issuing either requests with 512 B or 4096 B values. We chose these two sizes close to the average size of a Bitcoin transaction [2] and more general smart contracts reported in Fabric [10]. We do not use batching, unless otherwise stated, because with new latency-focused designs for permissioned blockchains [29], [24], [9] and the increasingly fast networking in cloud environments, we believe it is important to investigate the protocol without compromising its latency. The theoretical maximum throughput, excluding TCP/IP overhead and without batching, over 10Gbps network is just above 80 kops/s for 512 B and 19 kops/s for 4096 B values. We perform our evaluation on a 10 Gbps cluster of 24 machines with 6 core Intel Xeon E-2186G CPUs. All machines run recent versions of Debian linux and Go v1.10.

### A. What is the performance of off-the-shelf protocols on fast networks?

In Figure 5 we show the throughput of "off-the-shelf" BFT, using public-key (PK) cryptography for signing all messages. The numbers are low (less than 800 ops/s) due to the high computational cost of creating signatures, even though the system uses all cores of the machine. In this scenario, any method of accelerating the crypto operations will be beneficial. In our example, using a faster module for cryptographic operations ("+Acc") leads to 4X increase in throughput.

### B. How much can be gained by using MACs instead of signatures?

The second class of BFT deployments replace PK signatures on inter-node traffic with MACs [17] that are orders of magnitude cheaper to compute than signatures. The throughput of the system increases to more than 2.6 Kops/s and is, in fact, on par with the PK-Only version with acceleration. Since the nodes still spend significant resources on signing responses to clients with a PK, adding crypto acceleration is beneficial. It brings, however, a smaller benefit than in the previous case, that is, around 2.5X vs. 4X. Even though in this experiment we do not consider batching, it is worth pointing out that the MAC-based version is an upper bound of throughput for the off-the-shelf version with batching at the leader – however batching hundreds of requests to amortize the PK signing cost on protocol messages would impact latencies significantly.

### C. How much can be gained by optimizing to the domain of permissioned blockchains?

If nodes use MACs between themselves, as well as, to answer to clients, performance increases significantly, even when those rely on acceleration, there is almost a 2X difference in throughput for small values. Using the domain-optimized version and issuing large 4096 B values, it is possible to achieve more than 60% of the theoretical maximum throughput without relying on any type of batching. For smaller, 512 B values, only 25% of the maximum is reached. It has to be noted that for completeness, signatures have to be computed periodically (e.g. at checkpoints) on the data to allow for recovery at a coarser granularity, as well as to allow clients to exchange "blocks" among themselves.

### D. What operations are costly beyond signatures?

To understand the reason why crypto acceleration provides diminishing results in the domain-optimized case, we show a breakdown of compute costs for each message type in the

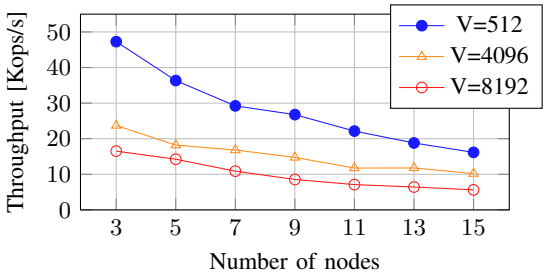


Fig. 7. Increasing the number of nodes has a predictable impact on throughput. The leader node becomes bound on its network stack for larger group sizes.

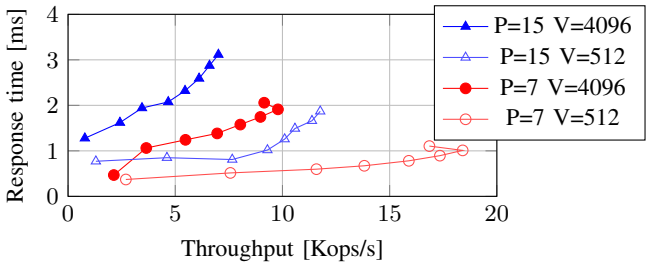


Fig. 8. Even under load the prototype delivers response times that increase predictably. For small values it is possible to keep response times under 2ms even when fully saturated.

leader of a group of 15 in Figure 6 (we ignore parallelism here and compute the aggregate time spent on each part). Thanks to the MAC optimization, the biggest cost in handling requests is the time it takes to (un)marshal them and to compute their SHA256 hash for verification. With larger consensus groups, the relative cost of these operations will increase further as there will be more messages sent between nodes. When using smaller values, the cost of hashing is reduced linearly, but the cost of (un)marshaling is not reduced significantly. Hence, for simplicity we omit other data sizes from this discussion. If we compare these costs with an implementation that signs client responses with RSA2048, this would increase the Signing cost to the order of 4.5m cycles (around 1.2 ms on our CPUs). This illustrates why avoiding its computation on client responses lead to speedup in Figure 5 (PK-Only vs. MACs).

### *E. Additional Evaluation of Our Prototype*

In this subsection we look at our prototype implementation as a domain-optimized BFT consensus service with reconfiguration and measure its performance in a cluster to show that it achieves low latency and high throughput even without hardware acceleration, making it a realistic starting point for implementing such functionality.

In Figure 7 we measure the throughput of our prototype with increasing consensus group sizes. The behavior is in line with the expectations of a leader-based protocol. The system delivers for 15 nodes more than 17 kops/s for the smallest value size and 10 kops/s respectively 5 kops/s for the 4 and 8 KB value sizes. The expectation is to scale to larger groups without issues, with a linear decrease in performance. The demonstrated throughput numbers are high enough to ensure that integration of PBFT⋆ with blockchains such as Hyperledger Fabric [10] is possible without becoming an immediate bottleneck.

With the deployment of permissioned ledgers in datacenter-like environments, it is important that the underlying BFT consensus can be performed with low latency. As shown in Figure 8, the average response time of our prototype starts from the sub-millisecond range for small values and increases slowly with load. Even close to saturation, the response time is only factor of three larger then in the unloaded case. While it is not our focus to compare to BFT-Smart since our prototype is meant as a platform for future exploration, not as a production-ready solution, it is worth pointing out that the latency stays *under* 3.5ms at all times, which is the lowest measured in Figure 1.

Overall, low and predictable latency that does not increase significantly with load is important because it ensures that the consensus nodes will not be the latency bottleneck. Even though most permissioned blockchains today do not optimize for latency at the millisecond level, as we discuss in the Related Work section, there are emerging blockchain designs, e.g. [29], [9], that could readily benefit from lower latency consensus.

## VI. Insights and Acceleration Strategies

Related works have shown that replacing PK signatures with MACs can improve performance but the improvements are seldom quantified. In this work, by measuring the difference in the same system, we reach a counter-intuitive insight: when adding crypto acceleration to the most optimized version, the performance gains are only marginal because client signatures can be verified in parallel, and block signatures can be computed on the side of normal operation. As the cost drill-down shows, for the domain-optimized case, the more significant opportunities are in acceleration of data movement and hashing. These will provide a bigger benefit than focusing solely on crypto accelerators. In the remaining we discuss two promising acceleration strategies to help reach 10Gbps line-rate performance and beyond. In preparation for their implementation, we present micro-benchmarks further motivating them and discuss their main benefits and challenges.

### *A. Offloading to SmartNICs*

Figure 6 shows that (un)marshaling and hashing costs account for a significant portion of the runtime even if we don't factor in signature verification. Today, there is an emerging offering of SmartNICs [23], [44], [22] that, in the future, could be used to offload some of these operations, e.g., serialization of messages and line-rate hashing. Furthermore, there are recent related works that use Mellanox NICs to offload TLS [35], which could be used as an equivalent of MACs.

The main question that arises when proposing the use of such SmartNICs is whether to treat them *a)* as a "stateless"

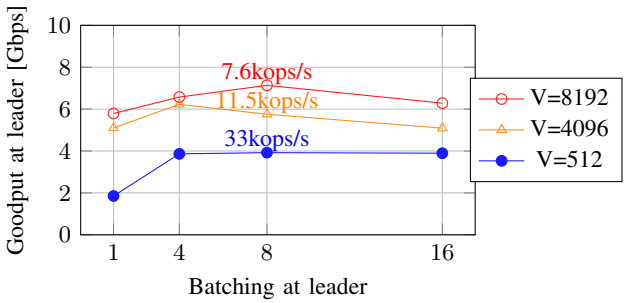


Fig. 9. Our prototype takes advantage of the 10Gbps bandwidth available at the leader, but to achieve an almost complete utilization of it some amount of batching is necessary.

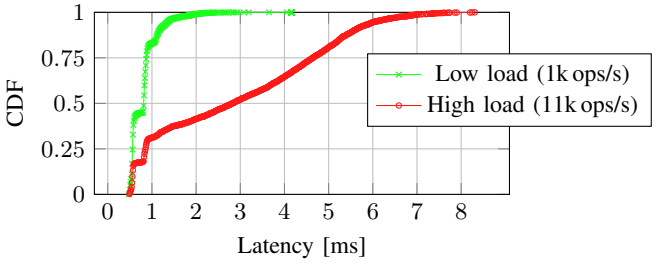


Fig. 10. Our prototype implementation highlights additional opportunities for HW: reducing the long tail of response times under load (15 nodes and 4KB payload)

accelerator that can, for instance, parse packets but does not access application state, or *b)* as a "stateful" one that can in addition also clone and send messages to different peers depending on the application state.

The first case already allows one to offload hashing and parsing, as well as, marshaling to the device, but to estimate the benefits of exposing more of the protocol's state to the NIC, we need to evaluate how efficient the current software version is when interfacing with the network. To this end, we plot in Figure 9 the useful bandwidth usage at the leader (goodput) with an increasing batching factor. Batching is implemented in the leader node by waiting for multiple messages from clients, assembling them into a vector and issuing a single `PRE_PREPARE` message for them. If there would be no TCP/IP and Ethernet overhead, it would be possible to achieve at most 10Gbps goodput in our setup, but in reality, even for very large packets the limit is lower. The results show that without batching, it reaches up to 6Gbps goodput for large values (4 and 8 KB) and around 2Gbps for small ones (512 B). Moderate batching of 4 to 8 requests can result in a better TCP stack utilization and at the leader goodput can reach more than 7Gbps for large requests and 4Gbps for small ones. This comes at the cost of higher response times, though with these batching factors, the differences remain small.

Based on this result, we foresee that for network speeds beyond 10Gbps, SmartNICs will be a sensible acceleration option. They will have to offload parts of the packet-processing operations and rely on fine-grained batching with strict latency guarantees that would be unfeasible to achieve in software.

### B. PBFT on Standalone FPGAs

Various types of hardware accelerators have been recently used to accelerate CFT consensus [28], [21], [43], [36], [30]. These solutions demonstrate latencies in the order of microseconds and are able to saturate the network regardless of the value sizes, thanks to the reduced overhead of the network stack and the low cost of data movement between network interface and the decision logic. They also bring predictable response time behavior which enables them to fulfill strict SLAs in low latency environments. If we investigate the distribution of response times and the variance at the tail of our software prototype, we see a significant increase in the high percentiles of response times, even if the median does not shift by much (Figure 10). In deployments where SLAs are important, standalone FPGA-based implementations could be beneficial because there are less factors influencing response times.

Even though hardware-based solutions provide microsecond latencies and high throughput, one challenge in this space has been the feasibility of handling not only the failure-free case but implementing reconfiguration as well. There is prior work, e.g. [28], that demonstrates that it is possible to implement reconfiguration for an atomic broadcast protocol on FPGAs and, in terms of communication patterns and metadata structures, PBFT is not significantly more complex. However, there is a significant difference between BFT and CFT algorithms in that the former requires the computation and verification of cryptographic hashes and signatures. This additional requirement, in particular RSA, make implementation challenging on FPGAs. This is because RSA (and similar ciphers) require iterative computation that is, on the one hand, resource intensive and, on the other hand, suffers from the relatively low clock rates of FPGAs. Therefore, from the three BFT variants discussed in this paper, only the domain-optimized is feasible on FPGAs because it minimizes the need for cryptographic operations, i.e., the rate of RSA ops/s.

To verify this claim, we rely on open source cores to estimate the cost of an implementation. By computing the maximum 10Gbps client-facing throughput at the leader as a function of minimum consensus group size and minimum value size, we can estimate how many resources RSA-related computations would take up on the FPGA. We use the RSA core from the Xilinx Vitis Library [7] as a representative instance and replicate it as many times as needed to match the desired throughput level. In Figure 11 we show with dashed lines how the cost of RSA computations decreases as the minimum group size increases. This is because the leader will send increasingly more intra-node messages than to/from clients.

To estimate the total cost of a complete 10Gbps BFT implementation in terms of logic resources, we synthesized a 10Gbps TCP/IP module with DRAM controller [6], SHA256 hashing cores [5], and AES [7] cores. For an estimate of the

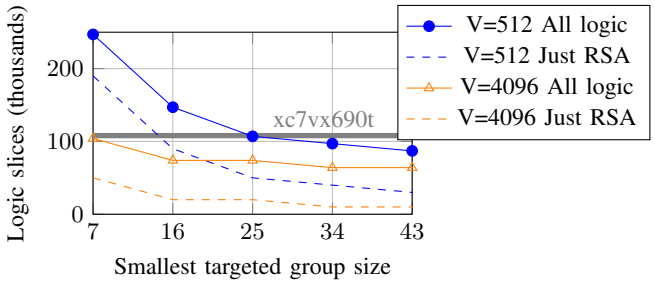


Fig. 11. Estimated resource consumption of a 10Gbps PBFT implementation on FPGAs using open-source components targeting a minimum value size and group size.

decision logic we relied on the CFT Atomic Broadcast module in Caribou [3]. The resource cost of these modules was added to the cost of RSA cores (sum shown as solid lines).

To put the logic resource numbers in perspective, we show the capacity of a mid-range FPGA[3] with the horizontal line. Overall, this resource estimation is showing an encouraging result: when using larger value sizes, or larger groups, even a mid-range FPGA could implement PBFT at 10Gpbs line-rate. Furthermore, given that there are also larger FPGAs on the market, e.g., Amazon F1 instances have around 4 times larger FPGAs, we are confident that it is realistic to implement in the future BFT on stand-alone nodes.

## VII. Related Work

### A. Protocol Variants and Optimizations

There is related work on implementing traditional BFT protocols in such a way that benefits from the multi-core parallelism of modern CPUs. A representative example is BFT-Smart [13]. The pipelined implementation demonstrates good performance on Gigabit networks. The way in which reconfiguration is performed is similar to the one we propose in PBFT⋆ but the nodes have to do more complex operations to perform state transfer. We chose not to use BFT-Smart as the framework for this study because it has implicit design choices related to, for instance, signatures, that would have made it difficult to "simulate" different BFT variants. The results of this work, nonetheless, apply to systems such as BFT-Smart.

There is an increasing interest in adopting the PBFT protocol for permissioned blockchain purposes. SBFT [26], for instance, aims to reduce communication complexity by relying on an so called "collector" node and threshold signatures, and adding a fast path to the execution (similar to Zyzzyva [33]). SBFT targets wide area networks and trades off bandwidth for more compute-intensive operations. As we shown in this work, however, in fast networking deployments there is plenty of bandwidth and relying on private-key cryptography as the default leads to sub-optimal use of the network resources.

Other recent work, such as HotStuff [45], explores how view-changes can be made cheaper. It targets permissioned blockchains that experience a high amount of failures or churn among the consensus nodes and, as a result, will require frequent view changes. The authors reduce the cost of these operations by adding an extra communication phase to each consensus round. In this work we assume business-to-business use-case of permissioned ledgers where, even though the clients of the system can be subject to churn, the core consensus nodes rarely change. In this setting optimizing for failure-free behavior is more beneficial. Nonetheless, our findings will apply to solutions such as HotStuff, as long as they are being executed in low latency environments.

Not surprisingly, the performance of PBFT and any similar protocol, including PBFT⋆, is severely limited by the leader as consensus groups grow. There is emerging work [41] that aims to solve the leader bottleneck without fundamentally changing the underlying protocol and instead relying on deterministic scheduling and data sharding that fits the permissioned ledger use-case well. Mir-BFT [41] achieves a near-linear increase in throughput with each node added to the consensus group and is competitive even when compared to ring replication in terms of bandwidth usage. The findings of this work are directly applicable to Mir, since its multi-leader approach is fully orthogonal to the actual implementation of the BFT protocol underneath.

### B. Consensus and Specialized Hardware

Various types of hardware accelerators have been used to accelerate CFT consensus algorithms [28], [21], [43], [36] and they demonstrate latencies in the tens of microseconds and are able to saturate the network regardless of the value sizes, thanks to reducing the overhead of the network stack and the data movement between network interface and the decision logic. We believe that there is an emerging opportunity in exploring how these ideas can translate to BFT consensus.

Other related work uses specialized hardware to implement trusted computing elements and through this simplify the typical three-round operation of BFT to two rounds [12], [31] and reduce the number of necessary replicas to $2f + 1$. Even thought they show promising result and are well suited to fast networks, these works introduce a different trust model for the two "parts" of the nodes.

## VIII. Conclusion

In this work we deconstructed a BFT consensus protocol with the goal of forecasting the benefits of various acceleration strategies. Our work is motivated by the emergence of permissioned blockchain use-cases that can be ran in environments with high bandwidth networking and low latencies and should be able, in the future, to take advantage of a wide range of acceleration options. Based on our study, comparing different BFT consensus variants, we concluded that the key to achieving low latency and high throughput behavior is more complex than just offloading cryptographic operations and instead will require a clever combination of improvements to multiple steps of the processing pipeline. This finding is a catalyst for research into hybrid solutions, that combine software and hardware in surprising ways.

[3]Xilinx xc7vx690t: 108k Logic Slices, 3600 DSPs and 1470 BRAMs.

## ACKNOWLEDGMENTS

This project has received funding from the European Unions Horizon 2020 research and innovation program, under the Marie Sklodowska-Curie grant agreement No. 842956, and the Spanish Research Council, through the Juan de la Cierva Formacion funding scheme.

## REFERENCES

[1] Amazon managed blockchain. https://aws.amazon.com/managed-blockchain/.

[2] Bitcoin transaction size visualization. https://bitcoinvisuals.com/chain-tx-size.

[3] Caribou: Distributed smart storage built with FPGAs. https://github.com/fpgasystems/caribou.

[4] Ibm blockchain platform. https://www.ibm.com/blockchain/platform.

[5] Open source SHA256 implementation. https://github.com/secworks/sha256.

[6] Scalable network stack supporting TCP/IP, RoCEv2, UDP/IP at 10-100Gbit/s. https://github.com/fpgasystems/fpga-network-stack.

[7] Vitis security library, xilinx. https://xilinx.github.io/Vitis_Libraries/security/index.html.

[8] Alastria network report 2019. https://alastria.io/wp-content/uploads/2019/04/2019-04-23_Alastria-Corporate-presentation_v00.08.pdf, 2019.

[9] M. J. Amiri, D. Agrawal, and A. E. Abbadi. Caper: a cross-application permissioned blockchain. *Proceedings of the VLDB Endowment*, 12(11):1385–1398, 2019.

[10] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, page 30. ACM, 2018.

[11] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman. Medrec: Using blockchain for medical data access and permission management. In *2016 2nd International Conference on Open and Big Data (OBD)*, pages 25–30. IEEE, 2016.

[12] J. Behl, T. Distler, and R. Kapitza. Hybrids on steroids: Sgx-based high performance bft. In *Proceedings of the Twelfth European Conference on Computer Systems*, pages 222–237. ACM, 2017.

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

[14] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[15] R. G. Brown, J. Carlyle, I. Grigg, and M. Hearn. Corda: an introduction. *R3 CEV, August*, 1:15, 2016.

[16] K. Buehler, D. Chiarella, H. Heidegger, M. Lemerle, A. Lal, and J. Moon. Beyond the hype: Blockchains in capital markets. Technical report, McKinsey Working Papers on Corporate & Investment Banking, 2015.

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

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

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

[20] A. Clement, M. Marchetti, E. Wong, L. Alvisi, and M. Dahlin. Bft: the time is now. In *Proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware*, page 13. ACM, 2008.

[21] H. T. Dang, P. Bressana, H. Wang, K. S. Lee, H. Weatherspoon, M. Canini, N. Zilberman, F. Pedone, and R. Soulé. P4xos: Consensus as a network service. Technical report, Research Report 2018-01. USI. http://www. inf. usi. ch/research_publication. htm, 2018.

[22] H. Eran, L. Zeno, M. Tork, G. Malka, and M. Silberstein. {NICA}: An infrastructure for inline acceleration of network applications. In *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, pages 345–362, 2019.

[23] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung, et al. Azure accelerated networking: Smartnics in the public cloud. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pages 51–66, 2018.

[24] C. Gorenflo, S. Lee, L. Golab, and S. Keshav. FastFabric: Scaling Hyperledger Fabric to 20,000 transactions per second. In *IEEE ICBC*, 2019.

[25] V. Gramoli. From blockchain consensus back to byzantine consensus. *Future Generation Computer Systems*, 2017.

[26] G. G. Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. Reiter, D.-A. Seredinschi, O. Tamir, and A. 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.

[27] S. Gupta, S. Rahnama, J. Hellings, and M. Sadoghi. Resilientdb: Global scale resilient blockchain fabric. *Proc. VLDB Endow.*, 13(6):868–883, 2020.

[28] Z. István, D. Sidler, G. Alonso, and M. 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.

[29] Z. István, A. Sorniotti, and M. Vukolić. Streamchain: Do blockchains need blocks? In *Proceedings of the 2nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, pages 1–6. ACM, 2018.

[30] X. Jin, X. Li, H. Zhang, N. Foster, J. Lee, R. Soulé, C. Kim, and I. Stoica. Netchain: Scale-free sub-rtt coordination. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pages 35–49, 2018.

[31] R. Kapitza, J. Behl, C. Cachin, T. Distler, S. Kuhnle, S. V. Mohammadi, W. Schröder-Preikschat, and K. Stengel. Cheapbft: resource-efficient byzantine fault tolerance. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 295–308. ACM, 2012.

[32] K. Korpela, J. Hallikas, and T. Dahlberg. Digital supply chain transformation toward blockchain integration. In *proceedings of the 50th Hawaii international conference on system sciences*, 2017.

[33] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: speculative byzantine fault tolerance. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 45–58. ACM, 2007.

[34] J. Kwon. Tendermint: Consensus without mining. *Draft v. 0.6, fall*, 1(11), 2014.

[35] B. Pismenny, I. Lesokhin, L. Liss, and H. Eran. Tls offload to network devices, 2016.

[36] M. Poke and T. Hoefler. Dare: High-performance state machine replication on rdma networks. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, pages 107–118. ACM, 2015.

[37] J. Ruiz. Public-permissioned blockchains as common-pool resources (alastria blockchain ecosystem), 2020.

[38] M. Russinovich, E. Ashton, C. Avanessians, M. Castro, A. Chamayou, S. Clebsch, M. Costa, C. Fournet, M. Kerner, S. Krishna, J. Maffre, T. Moscibroda, K. Nayak, O. Ohrimenko, F. Schuster, R. Schuster, A. Shamis, O. Vrousgou, and C. Wintersteiger. Ccf: A framework for building confidential verifiable replicated services. 2019.

[39] N. Santos and A. Schiper. Achieving high-throughput state machine replication in multi-core systems. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*, pages 266–275. Ieee, 2013.

[40] A. Sharma, F. M. Schuhknecht, D. Agrawal, and J. Dittrich. Blurring the lines between blockchains and database systems: the case of hyperledger fabric. In *SIGMOD'19*. ACM, 2019.

[41] C. Stathakopoulou, T. David, and M. Vukolić. Mir-bft: High-throughput bft for blockchains. *arXiv preprint arXiv:1906.05552*, 2019.

[42] M. Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In *International workshop on open problems in network security*, pages 112–125. Springer, 2015.

[43] C. Wang, J. Jiang, X. Chen, N. Yi, and H. Cui. Apus: Fast and scalable paxos on rdma. In *Proceedings of the 2017 Symposium on Cloud Computing*, pages 94–107. ACM, 2017.

[44] B. Williams, L. Aguirre Esparza, W. Poole, and S. Poole. Exploring mellanox bluefield smartnics as accelerators for heterogeneous architectures. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2019.

[45] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356. ACM, 2019.