Bidl: A High-throughput, Low-latency Permissioned Blockchain Framework for Datacenter Networks

Ji Qi[†1], Xusheng Chen[†1], Yunpeng Jiang[†], Jianyu Jiang[†], Tianxiang Shen[†], Shixiong Zhao[†], Sen Wang[§], Gong Zhang[§], Li Chen[§], Man Ho Au[†], Heming Cui[†*]
[†]The University of Hong Kong [§]Huawei Technologies Co., Ltd.

Abstract

A permissioned blockchain framework typically runs an efficient Byzantine consensus protocol and is attractive to deploy fast trading applications among a large number of mutually untrusted participants (e.g., companies). Unfortunately, all existing permissioned blockchain frameworks adopt sequential workflows for invoking the consensus protocol and executing applications' transactions, making the performance of these applications much lower than deploying them in traditional systems (e.g., in-datacenter stock exchange).

We propose Bidl, the first permissioned blockchain framework highly optimized for datacenter networks. We leverage the network ordering in such networks to create a *shepherded parallel workflow*, which carries a sequencer to parallelize the consensus protocol and transaction execution speculatively. However, the presence of malicious participants (e.g., a malicious sequencer) can easily perturb the parallel workflow to greatly degrade Bidl's performance. To achieve stable high performance, Bidl efficiently shepherds all participants by detecting their misbehaviors, and performs denylist-based view changes to replace or deny malicious participants. Compared with three fast permissioned blockchain frameworks, Bidl's parallel workflow reduces applications' latency by up to 72.7% and improves their throughput by up to 4.3× in the presence of malicious participants. Bidl is suitable to be integrated with traditional stock exchange systems. Bidl's code is released on github.com/hku-systems/bidl.

CCS Concepts: • **Security and privacy** → *Distributed systems security*; • **Software and its engineering** → **Consistency**; • **Networks** → **Data center networks**.

Keywords: permissioned blockchains, byzantine fault tolerance, high-performance blockchain workflows

* Heming Cui is the corresponding author.
[1] Ji Qi and Xusheng Chen contribute equally.

SOSP '21, October 26–29, 2021, Virtual Event, Germany

ACM ISBN 978-1-4503-8709-5/21/10.
https://doi.org/10.1145/3477132.3483574

1 Introduction

Cross-enterprise trading applications (e.g., stock exchange [2, 4, 5, 7, 8]) facilitate instant recording and clearing of transactions among mutually untrusted participants (e.g., clients and merchants). These trading applications are often deployed in one datacenter or multiple datacenters connected with dedicated network cables [46–48, 79, 97] to process a large number of transactions with real-time commit latency [57, 105]. For instance, the Hong Kong stock exchange [4] runs at about 50k transactions per second (txns/s), and the commit latency is about tens of milliseconds [38, 40, 76].

The prosperity of blockchains attracts industry and academia to develop various permissioned blockchain frameworks (typically, Diem [16], Hyperledger Fabric [31], and Quorum [26]) for trading applications. Unlike permissionless blockchains (e.g., Bitcoin [77]), a *permissioned* blockchain is maintained by *identified* member nodes (for short, nodes) and runs a fast Byzantine-fault-tolerant (BFT) consensus protocol (e.g., BFT-SMaRt [39]) among a subset of nodes (can be hundreds [24, 50, 99]) to commit transactions. A permissioned blockchain is especially suitable for trading applications because it can achieve much higher performance and energy efficiency [31] than a permissionless blockchain.

More and more permissioned blockchains are deployed within a datacenter or datacenters connected with dedicated cables [65, 67, 73]. For instance, a Singapore company develops a stock trading system based on the most notable permissioned blockchain Hyperledger Fabric (HLF) [31] within a datacenter [86]. Cloud providers such as Amazon [11], IBM [12], and Microsoft [18] also provide permissioned blockchains as cloud services in their datacenters.

Unfortunately, even deployed in a fast datacenter network, existing permissioned blockchains' performance is much lower than traditional trading systems [4]. For instance, Quorum [26, 37] and Diem [16, 104] achieve a throughput of 2k txns/s with 200ms latency in a datacenter. We ran HLF [31] with its mainstream BFT protocol BFT-SMaRt [39] (four consensus nodes) in our cluster with 40Gbps network; HLF achieved a throughput of 9.3k txns/s with 100ms latency (§6).

We attribute this low performance to the sequential workflow of typical permissioned blockchains [16, 26, 31]. Blockchain workflows are divided into two categories. The first category is the *execute→consensus→validate* workflow proposed by HLF [31] to support non-deterministic transactions. Nodes first concurrently *execute* received transac-

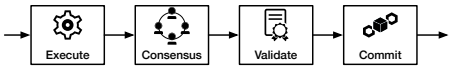


Figure 1. HLF's sequential workflow.

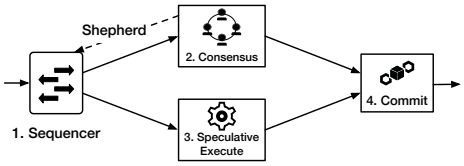


Figure 2. Bidl's new parallel workflow.

tions; then agree on the order of execution results through a BFT *consensus* protocol; after the consensus is achieved, nodes *validate* the result, commit valid ones and abort invalid ones. This consensus-on-result mechanism enables HLF to support non-deterministic languages (e.g., C++ with multi-threading), but it may cause a high abort rate on contending transactions (i.e., transactions modifying the same key concurrently). Diverse real-world blockchain applications [65, 81, 85] can contain over 40% contending transactions (e.g., supply chains [65]).

The second category is the *consensus→execute* workflow taken by diverse blockchain systems (e.g., Diem [16] and Quorum [26]). Nodes first use a BFT *consensus* protocol to order transaction contents, then individually *execute* transactions according to this order with zero abort rate. To make the individually executed transactions achieve consistent output, all transactions must be rewritten with a deterministic single-threaded language [28], which can be tedious and inefficient.

Overall, we believe the sequential nature of existing workflows is the root cause of making their end-to-end performance often a small fraction of traditional trading applications (e.g., 50k txns/s [4]).

Inspired by existing ultra-fast consensus protocols developed for a datacenter network [44, 69, 70, 79], our key observation is that the network ordering has great potential to improve the performance of existing permissioned blockchains by moving a part of the workflow (i.e., transaction ordering and distribution) to the routing layer. Specifically, we can run a dedicated node as the *sequencer*, which adds sequence numbers to all transactions and routes all transactions to all nodes by routing-aware multicast [15]. If the sequencer is never faulty (i.e., always sending the same sequence of transactions to all nodes), all nodes can receive almost all transactions in the same order (the consensus is hardly needed) and can commit the transactions on their own.

However, all these notable systems for datacenter networks [44, 69, 70, 79] target only the crash fault tolerance (CFT) model [29, 66] and cannot fit many permissioned blockchain applications, which require the BFT model. For instance, a malicious sequencer can send different transactions to different nodes [42, 92]. Consequently, different nodes may receive inconsistent transactions and need to invoke an extra BFT consensus to agree on their received transactions, making the system even slower than the sequential workflows.

In this paper, we propose Bidl[1], the first high performance permissioned blockchain framework that explores the network ordering in datacenter networks and complies with blockchains' BFT model. We present a new *shepherded parallel workflow* as shown in Figure 2: nodes invoke a BFT consensus protocol to agree on the order of transactions; in parallel, nodes speculatively execute the transactions from the sequencer. Then, nodes safely commit the execution results if they are consistent with the consensus results, and re-execute the transactions if they are inconsistent.

A key performance challenge of Bidl's parallel workflow is on ensuring correct speculative executions in the presence of malicious nodes. Although there exist BFT consensus protocols [30, 33, 43] for detecting malicious nodes and recovering performance, a smart malicious node in Bidl can still break Bidl's parallel workflow by broadcasting carefully crafted transactions to other nodes, making other nodes speculate on the crafted transactions, and causing dramatic performance penalty due to transaction re-executions.

To tackle this challenge, Bidl's workflow carries a *denylist-based view change protocol* to shepherd participants in a permissioned blockchain, which replaces a malicious sequencer and adds malicious nodes to the denylist. This protocol can greatly reduce the performance penalty caused by the transaction re-executions, making Bidl's workflow run at high performance and comply with the BFT model.

This parallel workflow enables Bidl to address non-determinism in transactions efficiently. After executing transactions in a block, nodes commit a transaction only if they produce consistent outputs, which ensures their local states never diverge. Moreover, in Bidl's execution phase, nodes execute contending transactions in the sequence number order, which can eliminate the aborts caused by concurrent executions of contending transactions. Overall, Bidl inherits the best of the two categories of existing sequential workflows and offsets their deficiencies.

We implemented Bidl on HLF's codebase [3]. Bidl's modular architecture enables existing BFT protocols to be deployed in Bidl with moderate engineering efforts (§4.2). We compared Bidl with HLF [31] FastFabric (FF) [51] and StreamChain [65] (two optimized frameworks designed for CFT deployment and based on HLF) in the same network, and enabled network ordering (e.g., consensus-on-hash [32, 42]) for all frameworks. We ran four popular BFT consensus protocols, including BFT-SMaRt [39], SBFT [50], HotStuff [100], and Zyzzyva [63] in Bidl, and evaluated them with typical trading workloads. Evaluation shows that:

- In Figure 3, Bidl achieved 60.2% lower latency and 3.3× higher throughput on average than HLF and FF.

[1]Bidl stands for Blockchain-powered In-Datacenter Ledger.

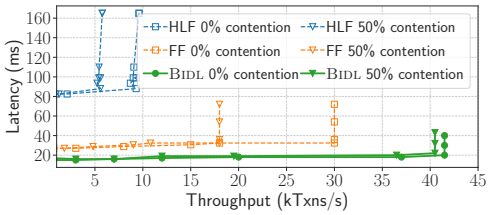


Figure 3. Performance on four consensus nodes without any malicious node (which favors FF's CFT deployment), and 50 transaction execution nodes. FF ran its built-in Raft [51]; HLF and Bidl ran BFT-SMaRt [87]. Contention ratio is the ratio of contending transactions among all transactions.

- Bidl is robust to contended workloads. In Figure 3, when the contention ratio increased from 0% to 50% according to real-world blockchain applications [65] and recent evaluation [81, 85], Bidl's throughput (40.1k txns/s) outperformed FF by 2.2× with zero abort rate, while FF's abort rate was 37.7% (analyzed in §6.3);
- Bidl maintained high throughput and low latency in the presence of various malicious nodes, greatly outperforming existing efficient blockchain frameworks (§6.2).

Our main contribution is Bidl, the first high-performance permissioned blockchain framework designed for datacenter networks. We exploit the network ordering to design a new high-performance and BFT-compatible parallel workflow. Bidl makes the first effective attempt to tackle both non-determinism in transactions and contending transaction aborts. Bidl has the potential to be deployed in industries (e.g., Amazon [11] and Nasdaq [19]).

In the rest of the paper: §2 discusses Bidl's related work; §3 gives an overview of Bidl's parallel workflow; §4 shows Bidl's protocol; §5 shows Bidl's correctness and performance analysis; §6 shows our evaluation, and §7 concludes.

2 Related Work

2.1 Permissioned Blockchains

Permissioned blockchains are often maintained by a large number of mutually untrusted organizations [50, 89]. For these applications, running a BFT protocol on many consensus nodes in order to tolerate many malicious nodes is essential. For instance, HLF is deployed with dozens of consensus nodes [89], and Cosmos runs 125 consensus nodes [24]. Bidl can support many consensus nodes.

In this paper, we present a permissioned instead of a permissionless blockchain for two reasons. Firstly, a permissionless blockchain usually incentivizes nodes to follow its protocol using cryptocurrencies, which require nodes to contribute high computation resources (e.g., PoW [77]) or wealth (e.g., PoS [49, 95]). However, in our target applications (e.g., a trading system), cryptocurrency is usually unavailable, and high power consumption is unacceptable. Secondly, our goal is to build a high-performance blockchain. Permissioned blockchain can use consensus protocols with explicit membership, such as PBFT [42] and HotStuff [99] (used in Diem [16]), which generally provide higher performance than the consensus protocols of permissionless blockchains.

The concept of *parallel execution* has been widely exploited by existing replication systems and inherited by blockchain systems. Eve [60] lets replicas first optimistically execute a batch of requests in parallel, and falls back to sequential execution if the replicas' states diverge. HTBFT [64] leverages a conflict detector to execute non-conflicting ordered transactions in parallel. Saraph et al. [84] proposes to speculatively execute transactions in parallel within each block of the Ethereum blockchain. These speculation techniques require an order to refer to and can be integrated into Bidl's execution phase because transactions have order hints provided by the sequencer.

Many research efforts have been put on improving blockchain's performance by sharding [62, 74, 94, 102]. Omniledger [62] and RapidChain [102] partition blockchain nodes into several shards, where each node belongs to one shard and processes a subset of transactions. With sharding, the blockchain nodes are dispersed into individual shards, thus greatly lowers the attack bar on a specific shard. To tackle attacks towards an individual shard, Ostraka [74] and Monoxide [94] allow a blockchain node to participate in multiple shards, which makes attacking towards a single shard as difficult as attacking the entire network. Blockchain sharding techniques are orthogonal with Bidl and can be integrated into Bidl to further improve Bidl's performance.

2.2 Systems Leveraging Datacenter Networks

A datacenter network is highly predictable. As shown in [54, 79, 90], due to the structured topology of a datacenter network, the latencies between nodes in a datacenter generally follow the triangle inequality property: $l_{i\to j} + l_{j\to k} \geq l_{i\to k}$, where $l_{i\to j}$ is the network latency from node n_i to n_j.

The triangle inequality is also available for datacenters connected with dedicated network cables, which are being actively deployed by major cloud providers [21, 22, 25]. For instance, a vast majority of inter-datacenter links (e.g., intra-continental) in AWS can satisfy this property [21, 23]. Bidl is designed to deploy in datacenters connected with such links.

We consider triangle-inequality difficult to break for two reasons. Firstly, for any edge of a triangle, the naturally robust redundant network paths in one datacenter or datacenters connected with dedicated network cables [75, 98] can practically prevent packet delays on some routing paths affecting the fast delivery along this edge via other paths. Secondly, notable tools [80] can detect network delay anomalies on datacenter networks precisely and efficiently.

Many influential systems [36, 44, 59, 69–71, 79] leverage the datacenter networks to achieve fast crash-fault-tolerance (CFT). Bidl differs from these notable systems in three aspects. Firstly, Bidl handles the Byzantine failures (i.e., malicious sequencer and malicious nodes). In Bidl's BFT model,

a malicious sequencer may send inconsistent transactions with the same sequence numbers to different nodes, and any node can disguise itself as the sequencer. Therefore, we design a denylist-based view change protocol to maintain stable high performance in the presence of malicious participants. Secondly, existing network-ordering systems assume a deterministic state machine, while Bidl handles non-determinism and ensures consistency of execution results.

Thirdly, Bidl's parallel workflow is tailored for blockchains, where transactions can invoke smart contracts: a piece of executable stateful code stored on the blockchain [31, 41]. The execution time of permissioned blockchains' smart contracts is often several milliseconds [31, 91], comparable to the latency of BFT consensus protocols. Therefore, parallelizing consensus and execution greatly reduces the Bidl workflow's end-to-end latency. In contrast, these CFT systems are mainly designed for fast-response server applications (e.g., key-value store) where the time cost of processing a key-value request is often tens of microseconds, much shorter than the time cost of consensus.

3 Overview

3.1 System Model

Bidl's participants consist of member nodes (for short, nodes) and clients. Only clients can generate and sign transactions. Nodes are grouped into *organizations* (e.g., banks and merchants); each organization runs a few consensus nodes (a process conducting BFT consensus on committing transactions) and normal nodes (a process conducting transaction execution). Consensus nodes and normal nodes can run on the same server. We denote the set of all consensus nodes as *CN*, all normal nodes as *NN*, and all clients as *CL*.

Bidl is permissioned: all nodes and clients are explicitly identified and managed using the standard membership mechanism same as HLF. Each node or client has a unique and explicit secret/public key pair. We use $\langle M \rangle_{\sigma_i}$ to denote a message M signed with node n_i's secret key; or $\langle T \rangle_{\sigma_c}$ to denote a transaction T signed with client c's secret key.

Threat model. Bidl adopts the BFT model, where malicious participants can behave arbitrarily. At most $f = \lfloor \frac{|CN|-1}{3} \rfloor$ consensus nodes can be malicious. Same as typical permissioned blockchains (e.g., HLF [31]), a node trusts all nodes in the same organization; a node does not trust any node in another organization. All clients can be malicious. All malicious nodes and clients can collude, and we call them the adversary (denoted as $\mathcal{A}$). We make standard assumptions on cryptographic primitives, including collision-resistant hashing, message authentication codes (MAC), and signatures.

Bidl must support *non-determinism* (e.g., caused by data races): given the same state and input, correct nodes may produce different execution results on the same transaction.

Bidl's guarantees. Bidl's *safety* guarantee ensures that all correct nodes commit the same blocks of totally ordered transactions and the same execution result for each transaction (to tackle non-determinism). Bidl's *liveness* ensures that if a correct client submits a transaction, it will eventually be committed [31]. Moreover, Bidl can effectively achieve *high performance* by maintaining its parallel workflow.

Network requirements. For *safety*, Bidl needs only an asynchronous network, where network packets can be dropped, delayed, and reordered. For *liveness*, Bidl requires a partial synchrony network [87]: there exists a global stabilization time (GST), after which all messages between correct nodes are delivered within a known maximum delay.

For *high performance*, Bidl requires all nodes to be deployed in one datacenter or datacenters connected with dedicated cables, and they should support IP multicast and satisfy the triangle inequality property (§2.2). This property is only for Bidl to ensure a low rate of suspecting a correct client as malicious for Bidl's denylist protocol (§4.6); Bidl does not need this property for safety or liveness. Clients can be deployed within or outside the datacenter network(s).

Bidl's deployment setting is increasingly pervasive due to two trends. Firstly, for high performance, many permissioned blockchain-powered security-critical applications are deployed on cloud datacenters (e.g., supply chain [65], financial trading [7, 86], and medical [17]). BFT is essential for these applications, because some participating organizations on clouds can be malicious (e.g., obtaining economic benefits and creating forged transactions [31]). Secondly, major cloud providers are deploying both blockchain services [11, 12] and high-performance dedicated inter-datacenter networks (e.g., AWS [21], Azure [22], and Nasdaq's millimeter-wave network [19]). Overall, we found the organizations (e.g., consensus nodes) of permissioned blockchains are usually deployed on high-performance cloud datacenters, and developing BFT systems on high-performance networks is an important topic [8, 63, 65, 82]. Nevertheless, Bidl ensures safety in asynchronous networks and retains good performance in tough scenarios such as malicious participants (§6.2), high contending transaction rate (§6.3), and packet loss (§6.4).

3.2 Bidl's Workflow Overview

Figure 4 shows Bidl's workflow with five phases.

• **Phase 1: submit.** Typically, a BFT consensus protocol has a leader to order client transactions. Clients submit signed transactions to the leader of consensus nodes via a TLS-enabled link. The leader drops the connection if the client sends malformed transactions (e.g., with invalid signatures) to avoid the client mounting DoS attacks on the leader.

• **Phase 2: multicast.** We let Bidl's BFT leader act as the sequencer by running a sequencing thread, and we call the sequencer and leader interchangeable. The leader assigns received transactions with consecutive sequence numbers and multicasts them to all consensus and normal nodes. For performance, the leader does not sign on the multicast messages (§4.1).

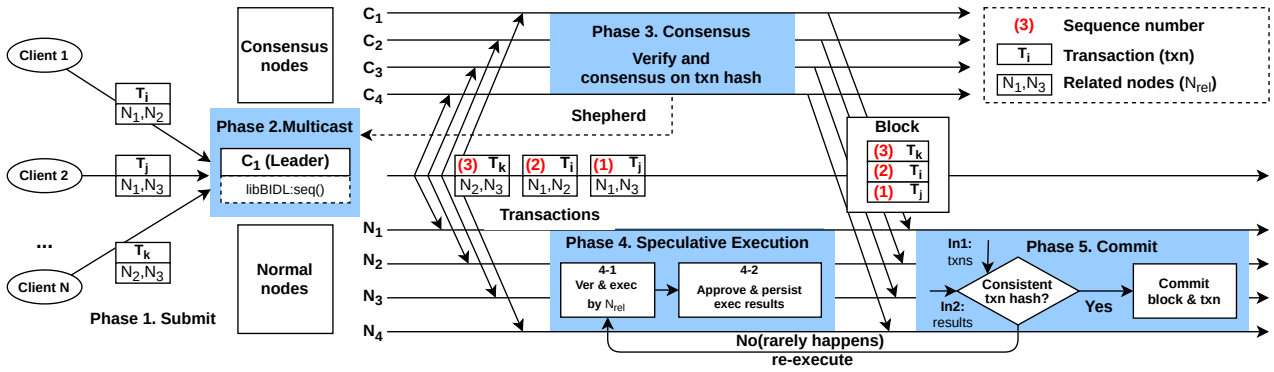


Figure 4. Detailed workflow of Bidl.

• **Phase 3: consensus.** The consensus nodes run an instance of a BFT protocol (e.g., BFT-SMaRt [39]) to *agree* on a sequence of hashes (see evaluation setup in §6) for transactions multicasted from the leader. Same as HLF [31], Bidl treats the BFT protocol as a blackbox (§4.2) so that Bidl can inherit the mature safety and performance optimizations from existing BFT protocols.

• **Phase 4: speculative execution. Phase 4** runs in parallel with **Phase 3**. In **Phase 4**, normal nodes speculatively execute client transactions according to sequence numbers (§4.3). Different from the HLF's *execute-order* workflow, in which all contending transactions (defined as transactions concurrently access the same key) are executed in parallel, Bidl speculatively executes contending transactions sequentially in the sequence number order. By doing so, Bidl eliminates transaction aborts caused by transaction dependencies and improves the system's performance on contended workloads (§6.3). To ensure state consistency in the presence of non-deterministic transactions, we adopt a multi-write [45] protocol (§4.4) to make nodes follow identical execution results, and abort a transaction if nodes produce inconsistent results for the transaction; this protocol's latency is often masked by the BFT consensus (**Phase 3**).

In the presence of an adversary, the sequence of speculatively executed transactions may differ from the sequence of transactions agreed by **Phase 3**. Bidl's **Phase 5** falls back to the sequential workflow by letting normal nodes follow the agreed transactions and re-execute them, ensuring both safety and liveness (§5.1). Moreover, Bidl's denylist protocol (§4.6) is designed to detect such an adversary and to retain Bidl's high performance of parallel workflow.

• **Phase 5: commit.** Bidl's normal nodes commit a transaction only after receiving matching agreed transactions in **Phase 3** and persisted execution results in **Phase 4**, ensuring safety, reasonable liveness, and reasonable high performance even with non-deterministic transactions (proved in §5.1).

Overall, the highlight of Bidl's workflow stems from achieving safety and high performance in malicious environments, and we illustrate in two aspects. Firstly, Bidl's new workflow achieves (1) safety in the presence of non-deterministic transactions as in the *execute-consensus* workflow (e.g., HLF [31]), and (2) zero abort rate in the presence of contented workloads, a key advantage in the *consensus-execute* workflow (e.g., Quorum [26]). In HLF, transaction aborts are caused by both the contented workload and non-determinism; in Bidl, transaction aborts are only caused by non-determinism (§6.3). Meanwhile, Bidl parallelizes the consensus and execution phases to greatly reduce the latency compared to both workflows. Secondly, different from existing BFT protocols [30, 43] which tackle the misbehaviors of consensus nodes in the consensus phase, Bidl shepherds the entire workflow to tackle the misbehaviors of all participants.

API	Role	Description
	Sequencing API	
❶ `invoke(Txn)`	CL[1]	Submit a client transaction to Bidl's leader node (§4.1).
❷ `seq(Txn)`	CN[2]	Add sequence number to a transaction T and relay T to all Bidl nodes (§4.1).
❸ `newTxn(Txn)`	CN,NN[3]	Process new transactions. Return *true* if the transaction is not discarded (§4.1).
❹ `checkProp(Propose)`	CN	Requests lost transactions proposed by n_L. Returns *true* if all transactions are received (§4.2).
	Shepherding API	
❺ `shepherdInit()`	CN	Initialize a thread to shepherd Bidl's parallel workflow (§4.6).

[1] **CL**: CLient [2] **CN**: Consensus Node [3] **NN**: Normal Node

Table 1. The API provided by libBidl.

4 Protocol Description

Bidl moves through a succession of views with consecutive view numbers. Each view v in Bidl has one consensus node being the leader (n_L). The state of each view is cached by Bidl clients and nodes.

libBidl API. Bidl provides a library called libBidl, which provides basic API (Table 1) that facilitates different consensus protocols to be deployed in Bidl. The sequencing API consists of functions for clients to submit transactions and

for consensus nodes to order client transactions. We will discuss how to integrate a consensus protocol into Bidl framework using libBidl in §4.2. The `shepherdInit()` API enables consensus nodes to shepherd Bidl's parallel workflow by monitoring the performance of all phases (§4.5) and running Bidl's denylist protocol (§4.6).

4.1 Transaction Submission

Figure 4 shows Bidl's parallel workflow with five phases.

Phase 1: Clients submit transactions to the leader n_L. A client c issues a transaction $T : \langle \text{Txn}, \mathcal{T}, O, v, pk \rangle_{\sigma_c}$ by the API call `invoke(`T`)`, where $\mathcal{T}$ is the transaction payload, O indicates the related organizations or nodes of this transaction, v is the view number (§3.1), pk is the client's public key. Specifically, related nodes of a transaction are defined as all normal nodes in related organizations of the transaction in HLF; related nodes should execute the transaction. The client calls `invoke(`T`)` to fetch the current view number v from Bidl nodes, and then sends T to the corresponding n_L of view v.

Phase 2: n_L multicasts transactions to all Bidl nodes. n_L adds a sequence number s to each incoming transaction and broadcasts the transaction to all nodes.

Although digital signatures on sequence numbers can partially mitigate the transaction re-executions caused by the crafted transactions from malicious nodes (§1), Bidl eliminates digital signatures on sequence numbers for two reasons. Firstly, signing each sequence number and verifying them on all nodes is computationally expensive [42, 88]. Our Intel E5 CPU can only verify less than 10k signatures for each core, and the signature verification needs to be conducted on every consensus and normal node. If Bidl lets the sequencer sign on sequence numbers, Bidl needs 5× more computation resources only for verifying the signatures on every node. Furthermore, even with the signatures, Bidl's sequencer shepherding protocol is also essential for reducing the transaction re-executions in Bidl's parallel workflow (§4.5).

Secondly, and more importantly, using signatures opens the door for the adversary $\mathcal{A}$ to mount resource exhaustion attacks [43] on Bidl by broadcasting malformed sequenced transactions with invalid signatures. Since a node can know whether a received message is correctly signed only after verifying the signatures; if the signature is incorrect, it is infeasible to identify which node launched the attack. If $\mathcal{A}$ disguises itself as the leader, although it cannot create valid signatures, it can easily exhaust the computing resources of all nodes by letting them repetitively verify signatures.

One may think of using the hybrid MAC-signature mechanism [43] to address the exhaustion DoS attacks. Bidl does use this mechanism for client transactions, but this mechanism is not suitable for signing ordered transactions from Bidl's sequencer. The key reason is that the hybrid signature must have a full vector of MAC entries for *all* nodes instead of only for nodes in related organizations. Otherwise, a node having a gap in its sequence number space will be unable to tell if the gap results from an irrelevant or lost transaction. More importantly, assuming that notifying a sequence number is not related to a normal node does not require a MAC for this normal node. Then, any malicious node can notify certain nodes that no sequence number is related (through crafted transactions), greatly reducing Bidl's performance.

Therefore, generating hybrid signatures for sequence numbers will add significant computational overhead to the critical path of Bidl workflow (**Phase-2**) because the time for generating one hybrid signature grows linearly with the number of nodes. Note that when using this hybrid signature for client signatures (in both Bidl and Aardvark [43]) spreads the computation over all clients, but signing on sequence numbers centralizes the computation to the one sequencer's critical path. Consequently, we eliminate signatures of sequence numbers and use the denylist design.

Due to the above issues and the performance reason, Bidl eliminates the signatures on sequence numbers and develops a denylist protocol in §4.6.

Transaction verification. When a consensus or normal node n_i receives a transaction T: $\langle \langle \text{Txn}, \mathcal{T}, O, v, pk \rangle_{\sigma_c}, s \rangle$, n_i stores T and verifies T by three steps.

1. **Sequence check.** If the sequence number s lies in the current block, go to step (2); if n_i has already received a transaction with the same sequence number s, or s belongs to a previous block, n_i discards T; if s belongs to a future block, n_i caches T for further processing.
2. **Replay check.** If n_i never received a transaction with the same hash, go to step (3); otherwise, n_i discards T.
3. **Signature check.** As the signature check is computationally expensive, n_i only checks the client signature of T after T has passed all previous verification steps.

4.2 Consensus

Phase 3: The consensus protocol *agrees* on a sequence of transactions. Bidl provides a set of generic API to support general BFT protocols, and we have integrated four influential BFT protocols, including BFT-SMaRt [39], SBFT [50], HotStuff [99], and Zyzzyva [63], into Bidl. In this section, we show that a developer can implement Bidl's consensus phase based on an existing BFT protocol (**Algo** 1) without modifying the BFT protocol's most complex phase: the agreement phase.

Each consensus node collects transactions from the leader node n_L. After receiving enough valid transactions for a block or a timeout elapsed, the n_L runs a BFT consensus to reach agreement on the SHA-256 hashes of transaction's payloads by sending a $\langle \text{Propose}, v, b, \mathcal{H}, n_L \rangle_{\sigma_L}$ message to all consensus nodes (**Algo** 1, **line** 9), where v is the current view, b is the block ID, $\mathcal{H}$ is the list of transaction hashes.

On receiving the Propose message, a consensus node n_i checks transactions' hashes in $\mathcal{H}$ and asks the leader to retransmit missed transactions with the `checkProp()` API

Algo 1: BIDL workflow at consensus node n_i.

```
1  Initialization
2      H ← ∅; shepherdInit();                          ▷ Invoke API 5
   // Phase 2: Sequencing
3  Upon reception of
   T : ⟨TXN, T, O, v, pk⟩_σc ∧ newTxn(T) do            ▷ Invoke API 3
4      if isLeader(n_i) = true then
5          s ← seq(T);                                 ▷ Invoke API 2
6          H ← H ∪ {⟨s, hash(T)⟩};
7      else send T to n_L;
   // Phase 3: Start consensus
8  Upon (|H| > blkSize ∨ timeout) ∧ isLeader(n_i) do
9      start agreement of ⟨PROPOSE, v, b, H, n_L⟩_σL;
10 Upon reception of M : ⟨PROPOSE, v, b, H, n_L⟩_σL do
11     if checkProp(M)=true then                       ▷ Invoke API 4
12         start agreement of M;
   // Phase 3: Finish consensus
13 Upon agreed ⟨b, H, cert⟩ do                         ▷ agreed(T)=true
14     B : ⟨BLK, b, txns, cert⟩ ← assemble(H, cert);
15     send B to ∀n_j ∈ NN;
   // Phase 4-2: Persist
16 Upon reception of R : ⟨RESULT, r⃗⟩_σj from n_j ∈ NN do
17     if approved(R) ∧ match(H, R) ∧ localStore(R) then
18         send P : ⟨PERSIST, s, h, r⃗, i⟩_σi to ∀n_j ∈ NN;
```

Algo 2: BIDL workflow at normal node n_i.

```
   // Phase 4: Speculative execution
1  Upon reception of
   T : ⟨⟨TXN, T, O, v, pk⟩_σc, s⟩ ∧ newTxn(T) do ▷ Invk. API 3
2      execute(T);
3  Function execute(T) do
4      r ← verExec(tx);
5      r⃗ ← approve(r) ;                    ▷ approved(T, r⃗)=true
6      persist(T, r⃗);
7  Function persist(T, r⃗) do
8      if valid(r⃗) then send ⟨RESULT, r⃗⟩_σi to ∀n ∈ CN ;
   // Phase 5: Commit
9  Upon reception of B : ⟨Blk, b, txns, cert⟩ ∧ valid(B) do
10     if checkHash(B)=true then
11         commitBlock(B);
12     else                                  ▷ Re-execute (rarely happens)
13         commitBlock(B);
14         foreach T ∈ B do execute(T) ;
15 Upon reception of P : ⟨PERSIST, s, h, r⃗, j⟩_σj ∧ valid(P) do
16     tmp ← tmp ∪ P;
17     if |tmp| ≥ 2f + 1 then                ▷ persisted(T, r⃗)=true
18         commitTxn(r⃗);
```

(**Algo 1, line 11**). After receiving all transactions in the PROPOSE message, n_i starts the agreement on the transaction's hashes. Once BIDL reaches agreement on the hashes, the transactions corresponding to the hashes are *agreed* (**Algo 1, line 13**). Each consensus node assembles transactions into a *block* according to $\mathcal{H}$ and delivers the block to normal nodes.

4.3 Transaction Execution and Commit

Phase 4-1. Normal nodes speculatively execute transactions. As n_L has ordered the transactions, the normal nodes speculatively execute related transactions according to the sequence numbers without waiting for the consensus results, then invoke the protocol in **Phase 4-2** to handle non-deterministic transactions (§4.4).

Due to the sequence numbers (a hint) given by the leader, the normal nodes speculatively execute transactions on the results of all contending transactions with smaller sequence numbers in the same block, which eliminates the aborts of execution results caused by concurrently executing contending transactions (in HLF). The generated execution results of related transactions will be committed or aborted in **Phase 5**.

For now, each normal node in BIDL executes only transactions related to this node's organization sequentially in the hinted order. This implementation is fair to our baseline systems in our evaluation: for all evaluated systems, each transaction was only executed on the same number of normal nodes (endorsers). As discussed in §2, BIDL can adopt existing parallel execution techniques [64, 84] within the execution phase to further improve BIDL's performance.

Phase 5. Normal nodes commit valid transactions and blocks. Upon receiving a block from **Phase 3** (e.g., contains $2f+1$ valid signatures from different consensus nodes), a normal node n_i checks whether its locally executed transactions are consistent with the transactions in the block by comparing the hashes. If all transactions are consistent, n_i commits the execution results of related transactions to its local state (**Algo 2**, **line 18**) and adds the block to the ledger (**Algo 2**, **line 13**). Otherwise, n_i re-executes all related transactions in the block; in this situation, the parallel workflow falls back to the sequential workflow. This happens if some participants are malicious, handled by the denylist protocol (§4.6).

4.4 Handling Non-deterministic Transactions

Same as HLF [31], a non-deterministic transaction in BIDL may generate inconsistent execution results on different nodes. A transaction's smart contract may be multi-threading and has data races, so BIDL treats all transactions as potentially non-deterministic. Note that, in BIDL, data races occur only within each transaction instead of among transactions due to the sequential execution on nodes (§4.3).

To ensure consistency in the presence of non-deterministic transactions, it is essential for BIDL to produce an *identical* and *retrievable* execution result for each transaction. By *identical*, we mean that BIDL must produce one identical result for each transaction. By *retrievable*, we mean that once BIDL has produced a result, the result is able to be retrieved (read) by all correct nodes. BIDL designs a simple two-step multi-write protocol [45] to guarantee correct normal nodes will produce and follow one identical result for each related transaction.

Phase 4-2. Approve and persist execution results. BIDL normal nodes first execute a related transaction T and generate a result (i.e., all modified keys and values). Since normal nodes in an organization trust each other (§3.1), these nodes select one delegate, and the delegate signs on one result.

For each transaction T, we call the first organization in its related organization list as T's corresponding organization o_c. The delegate of o_c then collects the signed results from T's related organizations to produce a vector $\vec{r}$ of these results. We say $\vec{r}$ is *approved*, if $\vec{r}$ contains the signed results (can be batch-signed) from all related organizations. Note that the results in $\vec{r}$ may be inconsistent due to non-determinism or malicious nodes.

After obtaining the result vector $\vec{r}$, Bidl runs a simple *persist* protocol to ensure $\vec{r}$ is both *identical* and *retrievable*. The delegate of o_c *persists* $\vec{r}$ by running a multi-write protocol adopted from [45], which sends $\vec{r}$ to all consensus nodes. Each consensus node broadcasts a Persist message to all normal nodes if $\vec{r}$ matches the transaction hashes proposed by the leader (**Algo** 1, **line** 18), and each consensus node only calls `localStore()` to persist one $\vec{r}$ for each transaction. Upon receiving the Persist messages for $\vec{r}$ from $2f + 1$ consensus nodes, each normal node regards $\vec{r}$ has been successfully *persisted* (**Algo** 2, **line** 17). Then, each normal node commits $\vec{r}$ to its local state only if all results in $\vec{r}$ are consistent (**Algo** 2, **line** 18); otherwise, normal nodes abort the transaction T and $\vec{r}$.

If malicious organizations produce two different approved result vectors ($\vec{r}$ and $\vec{r'}$) and send $\vec{r}$ and $\vec{r'}$ to consensus nodes, then at most one of the two results can be successfully persisted (§5.1). Therefore, a malicious organization's misbehaviors can at most affect the liveness of its own related transactions: no execution results are successfully persisted for its own transactions. The *safety* of Bidl is not affected (§3.1).

More importantly, such misbehavior can be easily detected in a permissioned blockchain by the administrator. Note that, different from Bidl's denylist protocol that *suspects* malicious participants with high probability, these signatures in $\vec{r}$ and $\vec{r'}$ are conclusive evidence for accusing these participating organizations trying to break the permissioned blockchain.

4.5 Shepherding the Leader

A malicious leader can degrade Bidl's performance by sending inconsistent transactions to nodes, dropping specific clients' transactions, or creating gaps in sequence numbers.

Same as existing work [42], Bidl uses view changes to address malicious leaders. Bidl's view change protocol is the same as PBFT's [42] with only two differences. Firstly, different from PBFT where the leadership is rotated among consensus nodes in a round-robin way, in Bidl, the leadership rotation is unpredictable (more details in §4.6). Secondly, the view change messages piggyback message fields for maintaining Bidl's denylists.

A Bidl view change is triggered by the consensus nodes in three cases. Firstly, to ensure high performance, consensus nodes initiate a view change upon detecting a non-trivial re-execution rate in **Phase 4** (by default, $> 1\%$) or a notable throughput degradation in **Phase 3** (by default, $< 90\%$ of the peak throughput in previous $|CN|$ views, same as existing work [32, 43]). A consensus node can know that a transaction has to be re-executed by detecting a mismatched $\vec{r}$ in §4.4, which implies the speculated transaction on a normal node mismatches the one agreed by the consensus nodes. In addition to letting normal nodes speculate wrong transactions, a malicious leader can also try to break Bidl's parallel workflow by consistently delaying sending transactions to normal nodes until the consensus is achieved. Thanks to Bidl's persist protocol (§4.4), this attack can be easily detected by consensus nodes due to the increasing delay for receiving persist requests (**Algo** 1, **line** 16).

Under fluctuating workload or network churns, Bidl may invoke extra view changes, but we regard it justifiable because a view change in recent BFT protocols (e.g., HotStuff [99]) is efficient, and recent work [32, 43] points out that, compared to running at low performance, it is worthwhile to try view changes to regain good performance.

Secondly, same as PBFT [42], to ensure liveness, if a client fails to receive a response for a transaction after a timeout, the client sends the transaction to all consensus nodes. If the transaction is not committed to the blockchain in the following blocks, the consensus nodes initiate a view change.

Thirdly, a correct leader proactively invokes a view change (§4.6) on detecting suspected misbehaviors; a malicious leader can behave arbitrarily. The correct leader invokes a view change to proactively elect a random leader. This is essential for Bidl's denylist protocol as illustrated by the dilemma in §4.6.

4.6 Bidl's Denylist Protocol

Since Bidl eliminates signatures on sequence numbers (§4.1), the adversary $\mathcal{A}$ can pretend to be the leader, broadcast crafted transactions (i.e., real transactions with fake sequence numbers), and cause conflicts in other nodes' sequence number spaces. These conflicts may lead to transaction retransmissions in **Phase 3** and cause the speculative execution to fail in **Phase 4**, degrading Bidl's performance.

The *denylist* mechanism is widely used in BFT systems (e.g., Aardvark [43], PeerReview [53], and CATS [101]) to detect malicious participants. However, these protocols are not suitable for Bidl's scenario for two reasons. Firstly, Bidl eliminates sequence number signatures on ordered transactions (§4.1) and conducts IP multicast, but these protocols require message senders (i.e., sequencers for detecting malicious sequencers) to sign every multicast message and wait for a signed reply. Secondly, these protocols can only monitor identified participants in the system, but in Bidl, any non-member node in the datacenter can broadcast forged messages to Bidl nodes.

Definition 4.1 (Conflicting transactions)**.** If the s^{th} transaction that node n_i received in **Phase 2** is different from the s^{th} transaction proposed in **Phase 3**, n_i regards these two transactions as conflicting.

Since Bidl cannot forbid an adversary $\mathcal{A}$ (defined in §3.1) to broadcast messages, Bidl carries a new denylist protocol to detect malicious clients who sign and provide crafted transactions for $\mathcal{A}$ to cause conflicts. Detecting the malicious clients is already effective for Bidl to maintain high performance due to two reasons. Firstly, $\mathcal{A}$ cannot create a large number of clients in a permissioned blockchain. Secondly, in a Bidl view with a correct leader, $\mathcal{A}$ can only use transactions signed by malicious clients to cause conflicts; if a malicious leader is causing conflicts, it will be replaced (§4.5). Specifically, since correct clients submit transactions to the correct leader in a TLS-enabled link (§4.1), if $\mathcal{A}$ wants to use a correct client's transaction T to cause conflicts, $\mathcal{A}$ must re-broadcast T with a different sequence number *after* receiving T from the correct leader. However, in Bidl's triangle-inequal network model (§3.1), most nodes should have already received T from the leader first, and just ignore the re-broadcasted T from $\mathcal{A}$. Therefore, if $\mathcal{A}$ uses a client c_m's signed transactions to cause conflicts, c_m must be within $\mathcal{A}$.

However, Bidl faces a dilemma between a low false-negative rate (high probability for detecting malicious clients) and a low false-positive rate (low probability for accusing correct clients). Specifically, it is hard to distinguish whether one conflict is caused by the leader assigning two different transactions with the same sequence number, or by $\mathcal{A}$ broadcasting the crafted transactions.

Bidl tackles this dilemma by monitoring conflicting transactions across *views with different leaders*. Our idea is that, if we can ensure a low false-positive rate for views with correct leaders, and transactions signed by a client c_m conflict with other transactions across $f+1$ views with different leaders (at least one of them must be a correct leader in BFT), then there is a high probability that c_m is malicious.

A subtle scenario is that a smart $\mathcal{A}$ can escape from this mechanism by using transactions crafted by a specific group of clients in only views of the same leader. For instance, $\mathcal{A}$ use transactions from a malicious client c_m only in views led by consensus node n_1, so c_m will not present in $f+1$ views with different leaders. To address this scenario, Bidl prevents $\mathcal{A}$ precisely controlling to cause conflicts in which views, via two mechanisms.

Firstly, Bidl lets a correct leader proactively invoke a view change on detecting conflicts. For a given sequence number, any node accepts only the first received transaction and will discard subsequent ones (§4.1). To successfully cause conflicts, $\mathcal{A}$ must broadcast transactions of any sequence numbers faster than the leader does. Therefore, once the correct leader proactively changes its view, $\mathcal{A}$ inevitably causes conflicts in the next view.

The first mechanism is necessary but insufficient. If Bidl rotates leadership in a round-robin way as in typical BFT protocols, the next leader after n_1 is deterministic (name it as n_2), so $\mathcal{A}$ can always cause conflicts in only n_1 and n_2's views. If $f > 2$, $\mathcal{A}$ can still escape.

Secondly, Bidl rotates leaders in an unpredictable and random way. Bidl divides views into *epochs*, each containing $3f+1$ views, and each consensus node is the leader of one view in each epoch. The second mechanism is to prevent a malicious leader from always passing its leadership to another malicious node. When a view change is invoked, the new leader is randomly selected from those consensus nodes not having been leaders in this epoch, using the hash of the last committed block as the random seed. Using the two mechanisms, even if $\mathcal{A}$ tries to cause conflicts using transactions signed c_m in only views led by n_1 (the previous subtle example), $\mathcal{A}$ will inevitably cause conflicts in views with different leaders succeeding n_1's views, which will accumulatively make c_m cause detectable conflicts within $f+1$ views of different leaders. Because c_m must be within $\mathcal{A}$, if $\mathcal{A}$ repeatedly uses c_m's signed transactions, Bidl will deny c_m.

Based on the two mechanisms, Bidl's detailed denylist protocol consists of three steps. Step (1): if node n_i detects two conflicting transactions, n_i suspects its locally received transaction in **Phase 2** as malicious and adds the client c_m of this transaction into a local suspect list $\mathcal{S}$. Step (2): n_i continuously monitors the misbehaviors of clients in $\mathcal{S}$. If n_i suspects c_m (in $\mathcal{S}$) as malicious across $f+1$ views with different leaders, n_i regards c_m as malicious. Step (3), during the next view change, c_m is carried in n_i' view change message. If c_m is regarded as malicious by $f+1$ consensus nodes, all non-faulty consensus nodes add c_m into the denylist $\mathcal{D}$.

For a client c_m in the denylist $\mathcal{D}$, normal nodes will not speculate transactions signed by c_m in **Phase 4**. However, if these transactions are agreed by the consensus nodes, normal nodes will still re-execute them. This design ensures that, although Bidl may have a non-zero false-positive rate (usually low, see §5.1), Bidl will not break the liveness or fairness for these clients. Overall, Bidl's denylist-based view-change protocol is only for ensuring Bidl's high performance without affecting liveness or safety. Even if sometimes the triangle inequality does not hold in Bidl's network (e.g., some switches misbehave), it will at most affect Bidl's performance, false-positive rate, and false-negative rate.

With more consensus nodes, Bidl must monitor more views (i.e., with a longer time window) to detect malicious clients. A longer time window may slow down the reaction of Bidl's denylist, but it does not affect Bidl's performance in the long run for three reasons. Firstly, in a permissioned blockchain, the list of authenticated clients is explicit and usually changes infrequently, so the adversary has only a limited number of malicious clients and cannot affect Bidl's parallel workflow once these malicious clients get denied. Secondly, Bidl makes the rejoining time of all denied clients much longer than the detection time window, greatly reducing the frequency of malicious behaviors. Thirdly, within the time window (during which the adversary is conducting attacks), Bidl can still achieve decent performance because such attacks merely make Bidl's speculative execution fail

and fall back to the slower sequential workflow (same as some evaluated baseline systems).

5 Correctness and Performance Analysis

5.1 Proof Sketch of Safety

Lemma 5.1 (consistency of agreed transactions)**.** For two transactions T and T' *agreed* (defined in §4.2) with the same sequence number s at correct normal nodes, $T = T'$.

Proof. This lemma directly inherits from the safety guarantee of the BFT protocol [39, 42, 50, 63, 99] running in **Phase 3** of Bidl. The BFT protocol ensures that there is only one unique hash value h committed with sequence number s. Therefore, we have $hash(T) = h = hash(T')$ and thus $T = T'$.

Lemma 5.2 (consistency of persisted execution results)**.** For two execution results $\vec{r}$ and $\vec{r'}$ *persisted* (defined in §4.4) with the same sequence number s at correct normal nodes, $\vec{r} = \vec{r'}$.

Proof. We prove this by contradiction. Suppose $\vec{r} \neq \vec{r'}$. There is a set of consensus nodes $S_1 \subseteq CN$ having persisted $\vec{r}$ and another $S_2 \subseteq CN$ having persisted $\vec{r'}$. According to Bidl's persist protocol (§4.4), each set has at least $2f + 1$ consensus nodes, with at least $f + 1$ intersections. This cannot happen in Bidl because a correct consensus node only persists one execution result for each sequence number (§4.4), and there are at most f faulty consensus nodes.

Proof of Bidl's safety. By Lemmas 5.1 and 5.2, all non-faulty nodes see the same sequences of agreed transactions and persisted execution results. Therefore, all non-faulty normal nodes consistently determine whether to commit each transaction and its execution result, which ensures safety. Note that Bidl's denylist protocol does not affect safety because adding a client c_m to the denylist will only make normal nodes not speculate c_m's transactions (§4.6); if these transactions are agreed by the consensus nodes, normal nodes will re-execute the transactions (§4.6).

5.2 Effectiveness of Bidl's Denylist Protocol

Low false-negative rate. Bidl can effectively detect any malicious client c_m used by $\mathcal{A}$ to constantly cause conflicts and add c_m to the denylist. If $\mathcal{A}$ uses c_m to cause non-negligible conflicts in some views, c_m will be suspected in this view. However, to add c_m to the denylist, c_m must be suspected by $f + 1$ views with different leaders (§4.6). To escape from the denylist, the best strategy for $\mathcal{A}$ is to cause conflicts only in views with specific leader(s). However, $\mathcal{A}$ cannot ensure this because it will inevitably cause conflicts in subsequent views with unpredictably selected leaders due to Bidl's proactive view change and randomized leader rotation mechanisms (§4.6). Therefore, $\mathcal{A}$ had better use another c'_m; since $\mathcal{A}$ cannot arbitrarily increase its colluded client list in a permissioned blockchain, Bidl often has a low false-negative rate.

Low false-positive rate. The probability that $\mathcal{A}$ manages to add a correct client c to Bidl's denylist $\mathcal{D}$ is low in Bidl's deployment model (§3). Recall that $\mathcal{A}$ can add c to Bidl's suspect list $\mathcal{S}$ only when the triangle inequality property is violated, then entries in $\mathcal{S}$ will be merged by nodes into $\mathcal{D}$ (§4.6). The triangle inequality property is violated when $l_\triangle : l_{i\to j} + l_{j\to k} - l_{i\to k} \leq 0$ (§2.2). If the triangle inequality is violated, $\mathcal{A}$ can re-broadcast c's transactions received in **Phase 2** from the leader to cause conflicts on c's transactions, then Bidl nodes will add conflicting transactions' clients to $\mathcal{D}$. If the triangle inequality is not violated, nodes should receive c's transaction from the leader before the transactions are re-broadcasted by $\mathcal{A}$, so $\mathcal{A}$'s re-broadcasted transaction will be ignored (§4.1). We then illustrate Bidl's low false-positive rate in two steps.

Firstly, the success rate that $\mathcal{A}$ manages to add a correct client c into the suspect list $\mathcal{S}$ in a view with a correct leader is negligible (the success rate is 1 if the leader is with $\mathcal{A}$). Bidl adds only conflicting transactions' clients to $\mathcal{S}$ (§4.6). As illustrated in §2.2, the triangle inequality is difficult to break due to the redundant paths in dedicated datacenter networks. Therefore, the probability that $\mathcal{A}$ exploits an arbitrary delayed path and makes $l_\triangle : l_{i\to j} + l_{j\to k} - l_{i\to k} \leq 0$ is overwhelmingly small (analyzed in existing work [54, 68, 80, 90]), making the probability that $\mathcal{A}$ causes conflicts on c's transactions overwhelmingly small.

Secondly, to add the correct c to the denylist, c must be suspected in at least $f+1$ views with different leaders, where at least one of the leaders is correct. With the two steps, Bidl can achieve a low false-positive rate in practice.

5.3 Liveness and High Performance

Liveness. In Bidl, committing a transaction T requires: (1) the consensus nodes agree on T, and (2) T's related organizations approve and persist T's execution result (§4.4). T's related organizations have the ability to withhold the persistence of T, but they have no motivation to block their own transactions. Therefore, it is sufficient to prove Bidl's liveness for achieving BFT consensus on a client transaction. Note that Bidl's denylist protocol does *not* affect Bidl's liveness, because the punishment for transactions whose clients are in the denylist is only not to speculate these transactions (§4.6). These transactions will be executed only if they are ordered by the consensus nodes. After all, the denylist is only designed for maintaining the parallel workflow for high performance, but will not affect Bidl's liveness or safety.

Bidl's liveness guarantee inherits from typical BFT protocols (e.g., PBFT [42]). Consider the synchronous period in the partial synchrony model (§3.1); if a correct client c cannot see its transaction get committed after a timeout, c sends this transaction to all consensus nodes, which will relay the transaction to the leader (§4.1). If the transaction can still not be committed, consensus nodes will initiate a view change (§4.5). Since there exists at least $2f + 1$ non-faulty consensus nodes, the transaction will eventually (after at most f view

changes) be committed when a non-faulty consensus node becomes the leader (§4.5).

High performance. Bidl's high performance (i.e., maintaining the parallel workflow) is illustrated from two aspects. Firstly, for views with a malicious leader, consensus nodes will invoke view changes on detecting a non-negligible re-execution rate or throughput degradation (§4.5) to replace the leader. Secondly, the malicious clients used by the adversary $\mathcal{A}$ (e.g., a malicious node) to constantly cause conflicts in normal nodes' sequence number spaces will be added to Bidl's denylist thanks to Bidl's low false-negative rate (§5.3). Since $\mathcal{A}$ cannot create arbitrary clients in a permissioned blockchain, Bidl can effectively ensure high performance in Bidl's network model (§3.1).

6 Evaluation

We implemented a software-based sequencer on a conventional server using Intel's Data Plane Development Kit [6]. Prior work [70] shows that such a software-based sequencer already achieves almost line rate in adding sequence numbers to transactions and multicasting them. For 1KB transactions, our sequencer adds about 20μs to the transfer delay.

We ran all experiments (including all clients and nodes) in a cluster with 20 Dell R430 servers, each equipped with an Intel 2.60GHz E5-2690 V3 CPU, 64GB memory, and 40Gbps NIC. The RTT among any two servers was about 0.2ms.

Baseline. We compared Bidl with three state-of-the-art blockchain frameworks: HLF [31], StreamChain [65], and FastFabric [51]. HLF is the most popular permissioned blockchain framework widely used by various cloud providers [11, 12], companies [17], and governments [8].

FastFabric [51] and StreamChain [65] are high-performance permissioned blockchain frameworks based on HLF. FastFabric re-architects the codebase of HLF and provides highly efficient optimizations (e.g., designating a single node as the orderer for sending transaction hashes to consensus nodes and assembling blocks) in a CFT scenario. FastFabric and StreamChain have a built-in Raft [78]. FastFabric's performance subsumed HLF in fault-free (i.e., no malicious participant) scenarios, so we focus on comparing Bidl with FastFabric instead of HLF except for the experiments with malicious participants (§6.2). StreamChain eliminates batching transactions into blocks and processes transactions in a stream fashion.

To achieve high performance in Bidl's consensus phase (**Phase 3**), Bidl performs consensus on only transaction hashes (i.e., 32 bytes for SHA-256) rather than the entire transaction payloads (typically, 1k bytes). This optimization is also adopted by existing BFT protocols [32, 42, 63] and permissioned blockchains [51]. For a fair comparison, we enabled this optimization for all Bidl, HLF, FF, and StreamChain frameworks in our evaluation (§6).

We integrated Bidl with four BFT protocols: BFT-SMaRt [39], Zyzzyva [63], HotStuff [100], and SBFT [50]. We implemented the batch optimization [63] of Zyzzyva and selected a non-leader node to collect consensus nodes' responses to the clients and to send the *commit* messages for each block. SBFT contains $c + 1$ collectors to create threshold signatures, and by default $c = 1$. The default transaction size was 1 KB (compressed with gzip) and the default block size was 500 transactions, typical settings in HLF [31, 88].

Workloads and metrics. We evaluated all blockchain frameworks with the popular SmallBank [13, 20] workload used in various studies [13, 85]. SmallBank has diverse types of transactions for bank operations. At the beginning of an experiment, SmallBank *creates* a random number of accounts for each organization and initializes each account with the same balance, and then random transactions are generated to *transfer* money among accounts of different organizations. The *create* transaction is related to one organization, and the *transfer* transaction is related to two organizations (§4.3).

We built a distributed benchmark based on Tape, an efficient benchmark tool for HLF [27]. Our benchmark spawned 100 clients on multiple servers, and collects the average latency and total throughput across all clients. For *latency*, we measured the client-perceived latency (also called end-to-end latency) from when a client submits a transaction to when the client is notified its transaction is committed.

Settings. We have two evaluation settings. In evaluation setting A, we ran four consensus nodes (f=1) and 50 normal nodes to favor FF and StreamChain's deployment. In evaluation setting B, we ran four to 97 organizations, and each organization has one consensus node and one normal node. Setting B is only used for evaluating Bidl's scalability (§6.1). We ran each node in a single docker container with a dedicated core. We focus on these questions:

§6.1: How efficient is Bidl's parallel workflow?
§6.2: How robust is Bidl to malicious participants?
§6.3: How robust is Bidl's performance to non-deterministic and high-contention workloads?
§6.4: How is Bidl's performance over multiple datacenters?

6.1 End-to-end Performance

We first evaluated the workflow performance of Bidl, FastFaric, and StreamChain in the fault-free case. As shown in Figure 5, StreamChain achieved the lowest latency by trading off its peak throughput because StreamChain processes transactions in a streaming fashion.

Bidl outperformed FastFabric in both throughput and latency. Bidl's throughput was higher than FastFabric because in Bidl, normal nodes execute related transactions according to their sequence numbers (**Phase 4**). Therefore, Bidl does not need to conduct the heavy *sequential* MVCC contention check in FastFabric's *consensus-execute* workflow. Specifically, in FastFabric's validate phase (**P3** in Table 2), the MVCC check took about 48.7% of the latency. We also found the MVCC check processed only 32.3K transactions per second,

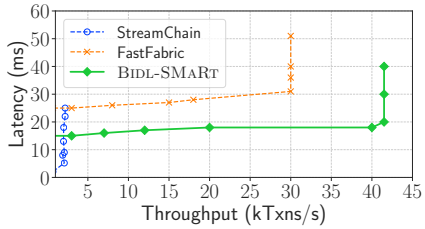


Figure 5. Throughput vs. latency on four consensus nodes without any malicious node (which favors FastFabric and StreamChain).

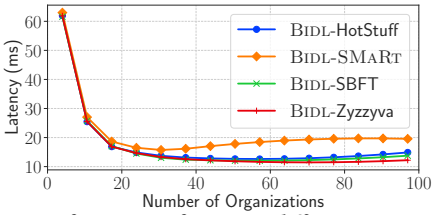


Figure 6. Performance of BIDL on different BFT protocols.

due to unmarshalling the read-write keys from the transactions and sequentially checking the keys with database queries. Moreover, BIDL avoids transaction aborts caused by executing contending transactions in different orders among nodes in baseline systems (Figure 8). In sum, BIDL achieves better throughput than FastFabric. BIDL's lower latency was primarily due to BIDL's parallelization of the execution and consensus phases.

Scalability. We collected the latency of BIDL with different numbers of organizations, as shown in Figure 6. Each organization has one consensus node and one normal node in order to evaluate BIDL scalability. With the number of organizations increased, BIDL's latency on four BFT decreased quickly and increased gently. To understand BIDL's low latency, we recorded the time taken for each step of the workflows in FastFabric and BIDL, as shown in Table 2 and Table 3. To ease comparison, we modified FastFabric to make it run BFT-SMaRt without spawning any malicious node. In these two tables, the number of consensus nodes and normal nodes setting (each consensus node ran with only three normal nodes due to our limited number of servers) is the same as in Figure 6. Note that this setting and the latency results are different from those in Figure 3 and Figure 5.

In BIDL, the consensus and execution phases are parallel, while in FastFabric, the endorse (same as execution) phase and consensus are sequential. When the number of organizations increased, BIDL's latency first decreased then increased. This is because when the number of organizations was small, BIDL workflow's performance was dominated by the latency of transaction execution: normal nodes of each organization need to verify and execute more transactions with fewer organizations. When the number of organizations increased towards 30, the number of transactions processed by each organization decreased, leading to lower latency. When the number of organizations continued to increase, the consensus became the major performance bottleneck of BIDL's workflow.

FastFabric-Smart results (in milliseconds)						
Number of Orgs	**4**	**7**	**13**	**25**	**49**	**97**
P1: Endorse	9.15	8.55	7.89	7.47	7.19	6.47
P2: Consensus	10.39	11.61	9.6	12.97	15.38	16.21
P3: Validate	51.49	29.63	15.78	10.12	8.19	6.92
End-to-end (P1+P2+P3)	71.03	49.79	33.27	30.56	30.76	29.6

Table 2. End-to-end latency breakdown of FastFabric-SMaRt.

BIDL-Smart results (in milliseconds)						
Number of Orgs	**4**	**7**	**13**	**25**	**49**	**97**
P1: Consensus	10.31	12.07	10.02	12.83	14.88	16.37
P2: Ver & Exec	59.26	35.47	17.91	10.24	7.64	7.56
P3: Persist	0.54	1.07	1.3	1.83	1.97	2.09
P4: Execution (P2+P3)	59.8	36.54	19.21	12.07	9.61	9.65
P5: Commit	2.65	2.84	3.13	2.88	2.61	2.89
End-to-end (Max(P1, P4) + P5)	62.45	39.38	22.34	15.71	17.49	19.26

Table 3. End-to-end latency breakdown of BIDL-SMaRt.

6.2 Robustness on Malicious Nodes

Table 4 shows the performance of BIDL and baseline systems in the presence of malicious participant cases. N/A means a framework is not designed to support a case. For each experiment, we tested the cases that we believe to be the worst case for BIDL and relevant frameworks. All blockchain frameworks ran four BFT or CFT consensus nodes and 50 normal nodes to eliminate the bottleneck of transaction execution. We reported each system's effective throughput (i.e., number of valid client transactions committed per second) when the system stabilized after each attack was conducted.

Malicious leader. We first explored the impact of the malicious leader on BIDL and HLF. For HLF, the leader refers to the consensus leader. We made the malicious leader propose a series of invalid transactions (random characters) during consensus and observed their peak effective throughput.

As shown in Table 4, HLF and BIDL maintained stable throughput on malicious cases. In these two frameworks, the BFT consensus leader disseminates all transaction payloads to other consensus nodes. Therefore, all consensus nodes can verify transaction payloads, and initiate a view change upon detecting invalid transactions.

FastFabric is not designed to support this case because it targets the deployment scenarios with a few mutually trusted or accountable companies (explicitly stated in its paper [51]), so for high performance, FastFabric leverages a single trusted orderer to send transaction hashes to the consensus protocols and to dissimilates transaction payloads to

Blockchain Frameworks	**Effective Throughput (kTxns/s)**		
	S1 Fault Free	**S2** Malicious Leader	**S3** Malicious Broadcaster
StreamChain [65]	2.73	N/A	N/A
HLF [31]	9.25	9.25	9.25
FastFabric [51]	29.32	N/A	N/A
BIDL w/o denylist	41.67	41.67	10.75
BIDL	41.67	41.67	41.67

Table 4. Observed effective throughput of blockchain frameworks in three scenarios. **S1:** fault-free case. **S2:** the malicious leader proposes a series of invalid transactions. **S3:** a malicious broadcaster broadcasts a carefully crafted series of transactions to all nodes.

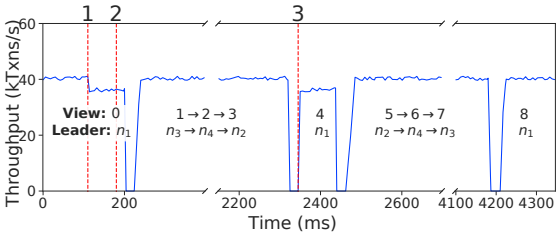


Figure 7. Robustness of Bidl under a malicious node.

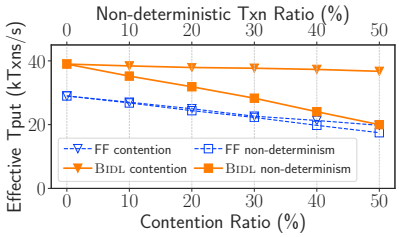


Figure 8. Robustness of Bidl and FF with different workloads.

normal nodes. In FastFabric, since the orderer never disseminates transaction payloads to consensus nodes, if the orderer is malicious and sends invalid hashes to consensus nodes, the consensus nodes cannot handle this case.

Overall, although consensus-on-hash can greatly improve the performance of BFT protocols in fault-free cases, the leader must be shepherded (§4.5) to achieve robust performance in malicious scenarios.

Malicious broadcaster. We implemented a malicious broadcaster that keeps broadcasting crafted transactions (at the same time with the Bidl sequencer) to randomly selected nodes. HLF's performance was stable because HLF only accepts transactions from the consensus leader in TCP connections. This experiment also evaluated the effectiveness of Bidl's denylist protocol (§4.6): if the denylist protocol is disabled, Bidl's performance dropped dramatically (Table 4).

To further understand the effectiveness of Bidl's denylist protocol, we recorded the real-time throughput of Bidl (denylist enabled) after the attack was injected, as shown in Figure 7. We made a smart adversary broadcast crafted transactions signed by a malicious client c_m in only a correct consensus node n_1's views, trying to escape from Bidl's denylist protocol (§4.6).

When the adversary started to broadcast crafted transactions (Point 1), these crafted transactions caused conflicts in some nodes' sequence number spaces. These nodes requested retransmissions of conflicting transactions, causing Bidl's throughput to decrease, and the consensus nodes initiated a view change (Point 2). The view change took approximately 30ms to finish. The adversary detected that n_1 was no longer the leader and stopped broadcasting crafted transactions.

After n_1 became the leader again (Point 3), the adversary restarted to broadcast crafted transactions, which triggered Bidl's proactive view change protocol (§4.5), so the adversary's crafted transactions caused conflicts in the next view (n_2 being the leader). Since c_m's transactions caused conflicts in $f + 1 = 2$ views with different leaders, c_m was added to the denylist (§4.6), and normal nodes do not speculate on the transactions from c_m. In subsequent views with n_1 being the leader (e.g., view 8), Bidl maintained the peak throughput even if the adversary continued to broadcast crafted transactions signed by c_m.

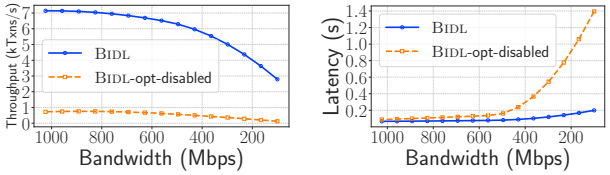
Overall, Bidl is complementary to existing permissioned blockchain frameworks. StreamChain is most suitable for being deployed with a small number of organizations with extremely low latency requirements; FastFabric can achieve safety and very high performance in the presence of mild adversaries (e.g., company insiders) that do not target the system's performance; HLF is general for different deployment scenarios, within or outside datacenters. Bidl achieves high performance among many organizations in a datacenter or datacenters connected with dedicated cables (§2), and Bidl retains stable high performance even some participants try to corrupt its performance. Bidl is the most suitable for diverse applications that desire high performance and security requirements, such as cross-enterprise trading [72, 83], stock exchange [4, 8], supply chains [65], and voting [34, 55, 58].

6.3 Non-deterministic and Contended Workloads

Non-deterministic workload. We tested the performance of Bidl and FastFabric (FF) under non-deterministic workloads. We developed a non-deterministic smart contract that *creates* an account with a random balance. When executing a non-deterministic transaction invoking this smart contract, different nodes may generate inconsistent balances. Note that such transactions should generally be avoided in blockchain systems and are usually regarded as bugs [96].

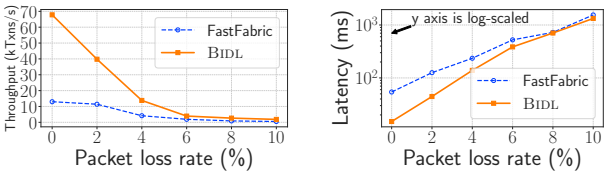
We varied the ratio of non-deterministic transactions among all transactions. Since in FastFabric and Bidl, non-deterministic transactions do not affect the commit latency of other transactions, so we focus on the throughput results. As can be seen from Figure 8, the effective throughput of both Bidl and FastFabric decreased with increasing non-deterministic ratios, but the drop of Bidl was faster. This is because, in FastFabric, non-deterministic transactions are early-aborted after endorsement without going through the entire workflow; while in Bidl, non-deterministic transactions with inconsistent execution results are regarded as aborted in the commit phase (§4.4). Note that Bidl's non-determinism handling protocol (§4.4) makes normal nodes commit transactions only when the account balances are the same; this prevents non-determinism making the local state of Bidl's normal nodes diverge, and we observed only varied performance effects.

High-contention workload. We then evaluated Bidl and FastFabric on workloads with different contention rates (i.e., skewness). We set 1% accounts as *hot* accounts, and each money transfer transaction has a certain probability of accessing the hot accounts, controlled by the *Contention Ratio* parameter. We varied the contention ratio from 0% to 50% according to real-world blockchain applications [65, 81, 85]. As shown in Figure 8, Bidl showed better throughput with



(a) End-to-end throughput. **(b)** End-to-end Latency.

Figure 9. Performance in the cross-datacenter deployment.



(a) End-to-end throughput. **(b)** End-to-end latency.

Figure 10. Performance of Bidl and HLF with packet losses.

increasing contention ratios. This is because FastFabric endorses transactions in parallel before ordering them, and most contending transactions accessing the same account end up being aborted in the validation phase. In contrast, Bidl eliminates the aborts caused by contention because Bidl speculatively executes transactions according to their sequence numbers (§4.3), and normal nodes in each organization execute transactions that access the same accounts in the same order. This may lead to a bit longer execution latency compared to non-contending transactions in **Phase 4**, which was often masked by the BFT consensus phase with more consensus nodes (Table 3).

6.4 Performance over Multiple Datacenters

One limitation of Bidl is that Bidl requires the triangle inequality property (§2) for a low false-positive rate of its denylist protocol (but Bidl does not need such a requirement for liveness or safety). However, as mainstream cloud providers [21, 22, 25] are actively deploying dedicated network cables among their datacenters, Bidl can also be deployed across datacenters among which triangle inequality holds. In addition, some modern Internet-scale high-speed networks can also satisfy the property (e.g., Nasdaq's millimeter wave network [9] and InfiniBand long-reach system [14]).

To evaluate Bidl's performance across many datacenters when the cross-datacenter network bandwidth becomes the bottleneck. We ran a four-datacenter setting in our 20-server cluster using Linux traffic control by limiting the bandwidth and latency among servers. Each datacenter has five servers. Servers within the same datacenter are connected with 40Gbps network with 0.2ms RTT. We set the RTT between any two servers from different datacenters as 20ms, a typical value for inter-datacenter networks [35]. We deployed four consensus nodes and 50 normal nodes and chose BFT-SMaRt as the consensus protocol.

As shown in Figure 9, Bidl's performance dropped slowly when the bandwidth became more stringent. We also collected Bidl's performance when two optimizations, IP-multicast and consensus-on-hash, were disabled (i.e., Bidl-opt-disabled). Bidl achieved much higher performance for all bandwidth setups than Bidl-opt-disabled. The performance gain of Bidl over Bidl-opt-disabled became more obvious when the bandwidth was more stringent. This is because when Bidl's shepherded leader was disseminating a transaction using IP multicast to nodes in another datacenter, the transactions were transferred only once on the inter-datacenter network. This reduces the bandwidth consumption compared to Bidl-opt-disabled, where the consensus leader invokes N TCP transmissions to disseminate the large Propose message to N consensus nodes in another datacenter. The transactions are transferred N times on the inter-datacenter network, which incurs significant bandwidth consumption and degrades the performance.

In summary, Bidl is suitable for deploying in multiple datacenters due to the two optimizations of IP-multicast and consensus-on-hash. Although these two optimizations were exploited by existing systems, Bidl is the first permissioned blockchain framework that can ensure *stable* high end-to-end performance in the presence of malicious participants (§6.2).

Performance on packet losses. We also evaluated the performance of Bidl and FastFabric on packet losses. As shown in Figure 10, Bidl's performance gain over FastFabric was substantial when the packet loss rate was low (< 2%) and became less obvious when the packet loss rate kept increasing. Overall, Bidl is robust to real-world packet losses because the packet loss rate is about 10^{-5} [52, 103] within a datacenter and no more than 10^{-3} [1, 10, 56, 61, 93] for datacenters connected with dedicated cables.

7 Conclusion

We present Bidl, the first blockchain-powered distributed ledger optimized for datacenter networks. Bidl leverages the network ordering in a datacenter network to enable a new shepherded parallel workflow that performs the consensus in parallel with the transaction execution. Evaluation on three notable permissioned blockchains and four BFT protocols shows that Bidl is highly efficient and is robust to malicious participants. This study demonstrates the benefits of co-design permissioned blockchain with the underlying network, providing a new approach for building in-datacenter Byzantine blockchain systems. Bidl is open source. Bidl's code is released on github.com/hku-systems/bidl.

Acknowledgments

We thank our shepherd, Ittay Eyal, and all anonymous reviewers for their helpful comments. This work is funded by the research grants from two Huawei Flagship Research Grants 2018 and 2021, HKU-SCF FinTech Academy R&D Funding Scheme, HK RGC GRF (17202318, 17207117), HK RGC ECS (27200916), NSFC Grant 61972332, and a Croucher Innovation Award.

References

[1] Huawei network planning. https://support.huawei.com/view/contentview!getFileStream.action?mid=SUPE_DOC&viewNid=EDOC1000092270&nid=EDOC1000092270&partNo=j005&type=htm.

[2] HKEx Data Centre and Hosting Services. https://www.hkex.com.hk/-/media/HKEX-Market/Services/Connectivity/Hosting-Services/Subscriber-Notices-and-Guidance-Note/Samuel-Wong_Hosting-Ecosystem-2013.pdf, 2013.

[3] Hyperledger Fabric 1.3. https://github.com/hyperledger/fabric/releases/tag/v1.3.0, 2018.

[4] With turnover close to HK$200b, new HKEX platform can handle 60,000 trades per second. https://www.scmp.com/business/investor-relations/ipo-quote-profile/article/2130344/new-hkex-securities-trading-platform, 2018.

[5] ASX is replacing CHESS with distributed ledger technology (DLT) developed by Digital Asset. https://www.asx.com.au/services/chess-replacement.htm, July 2019.

[6] DPDK: Home. https://www.dpdk.org, 2019.

[7] Exchange Trading & Matching Technology System – Nasdaq. https://www.nasdaq.com/solutions/trading-and-matching-technology, 2019.

[8] Singapore Exchange. https://www2.sgx.com, 2019.

[9] Wireless Express Connect - Nasdaq. https://www.nasdaqtrader.com/content/Productsservices/trading/CoLo/ExpressConnectFS.pdf, 2019.

[10] Alibaba cloud network faq. https://partners-intl.aliyun.com/help/doc-detail/40637.htm#section-t34-uni-zg6, 2020.

[11] Amazon Managed Blockchain. https://aws.amazon.com/managed-blockchain/, 2020.

[12] Blockchain Platform. https://cloud.ibm.com/catalog/services/blockchain-platform, 2020.

[13] Hyperledger Caliper Benchmarks. https://github.com/hyperledger/caliper-benchmarks/tree/master/benchmarks/scenario/smallbank, 2020.

[14] InfiniBand Long-Reach and Long-Haul Systems. https://www.mellanox.com/products/long-reach?mtag=long_haul_systems_ov, 2020.

[15] Ip multicast. https://en.wikipedia.org/wiki/IP_multicast, 2020.

[16] Libra. https://libra.org, 2020.

[17] MedicalChain. https://medicalchain.com/en/home/hyperledger/, 2020.

[18] Microsoft Azure Blockchain. https://azure.microsoft.com/en-us/solutions/blockchain/, 2020.

[19] Nasdaq Co-Location. https://www.nasdaq.com/solutions/nasdaq-co-location, 2020.

[20] SmallBank Benchmark. https://hstore.cs.brown.edu/documentation/deployment/benchmarks/smallbank/, 2020.

[21] Amazon Global Network. https://aws.amazon.com/about-aws/global-infrastructure/global_network/?nc1=h_ls, 2021.

[22] Azure Global Network. https://azure.microsoft.com/en-us/global-infrastructure/global-network/#documentation, 2021.

[23] Cloud Ping. https://www.cloudping.co/grid, 2021.

[24] Cosmos Validators Overview. https://hub.cosmos.network/main/validators/overview.html, 2021.

[25] Google Cloud Infrastructure. https://cloud.google.com/infrastructure, 2021.

[26] Quorum. https://consensys.net/quorum, 2021.

[27] Tape. https://github.com/Hyperledger-TWGC/tape, 2021.

[28] The Solidity Contract-Oriented Programming Language. https://github.com/ethereum/solidity, 2021.

[29] Marcos Kawazoe Aguilera, Wei Chen, and Sam Toueg. Failure detection and consensus in the crash-recovery model. In *International Symposium on Distributed Computing*, pages 231–245. Springer, 1998.

[30] Yair Amir, Brian Coan, Jonathan Kirsch, and John Lane. Byzantine replication under attack. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, pages 197–206. IEEE, 2008.

[31] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, page 30. ACM, 2018.

[32] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The next 700 bft protocols. *ACM Transactions on Computer Systems (TOCS)*, 32(4):1–45, 2015.

[33] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. Rbft: Redundant byzantine fault tolerance. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*, pages 297–306. IEEE, 2013.

[34] Ahmed Ben Ayed. A conceptual secure blockchain-based electronic voting system. *International Journal of Network Security & Its Applications*, 9(3):01–09, 2017.

[35] Peter Bailis, Aaron Davidson, Alan Fekete, Ali Ghodsi, Joseph M Hellerstein, and Ion Stoica. Highly available transactions: Virtues and limitations. *Proceedings of the VLDB Endowment*, 7(3):181–192, 2013.

[36] Mahesh Balakrishnan, Dahlia Malkhi, Vijayan Prabhakaran, Ted Wobbler, Michael Wei, and John D Davis. {CORFU}: A shared log design for flash clusters. In *9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, pages 1–14, 2012.

[37] Arati Baliga, I Subhod, Pandurang Kamat, and Siddhartha Chatterjee. Performance evaluation of the quorum blockchain platform. *arXiv preprint arXiv:1809.03421*, 2018.

[38] Robert P Bartlett III and Justin McCrary. How rigged are stock markets? evidence from microsecond timestamps. *Journal of Financial Markets*, 45:37–60, 2019.

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

[40] Andrew Brook. Evolution and practice: Low-latency distributed applications in finance: The finance industry has unique demands for low-latency distributed systems. *Queue*, 13(4):40–53, 2015.

[41] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37), 2014.

[42] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[43] Allen Clement, Edmund L Wong, Lorenzo Alvisi, Michael Dahlin, and Mirco Marchetti. Making byzantine fault tolerant systems tolerate byzantine faults. In *NSDI*, volume 9, pages 153–168, 2009.

[44] Huynh Tu Dang, Daniele Sciascia, Marco Canini, Fernando Pedone, and Robert Soulé. Netpaxos: Consensus at network speed. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, page 5. ACM, 2015.

[45] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. *ACM SIGOPS operating systems review*, 41(6):205–220, 2007.

[46] Ankush Desai, Sanjit A Seshia, Shaz Qadeer, David Broman, and John C Eidson. Approximate synchrony: An abstraction for distributed almost-synchronous systems. In *International Conference on Computer Aided Verification*, pages 429–448. Springer, 2015.

[47] Minghong Fang and Jia Liu. Toward low-cost and stable blockchain networks. *arXiv preprint arXiv:2002.08027*, 2020.

[48] Kristen Gardner, Samuel Zbarsky, Sherwin Doroudi, Mor Harchol-Balter, and Esa Hyytia. Reducing latency via redundant requests:

Exact analysis. *ACM SIGMETRICS Performance Evaluation Review*, 43(1):347–360, 2015.

[49] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68, 2017.

[50] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael K Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. Sbft: a scalable decentralized trust infrastructure for blockchains. *arXiv preprint arXiv:1804.01626*, 2018.

[51] Christian Gorenflo, Stephen Lee, Lukasz Golab, and Srinivasan Keshav. Fastfabric: Scaling hyperledger fabric to 20,000 transactions per second. *arXiv preprint arXiv:1901.00910*, 2019.

[52] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, et al. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 139–152, 2015.

[53] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. Peerreview: Practical accountability for distributed systems. *ACM SIGOPS operating systems review*, 41(6):175–188, 2007.

[54] Biao Han, Xiangrui Yang, and Xiaoyan Wang. Dynamic controller-switch mapping assignment with genetic algorithm for multi-controller sdn. In *2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pages 980–986. IEEE, 2019.

[55] Rifa Hanifatunnisa and Budi Rahardjo. Blockchain based e-voting recording system design. In *2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA)*, pages 1–6. IEEE, 2017.

[56] Osama Haq, Mamoon Raja, and Fahad R Dogar. Measuring and improving the reliability of wide-area cloud paths. In *Proceedings of the 26th International Conference on World Wide Web*, pages 253–262, 2017.

[57] Joel Hasbrouck and Gideon Saar. Low-latency trading. *Journal of Financial Markets*, 16(4):646–679, 2013.

[58] Friðrik Þ Hjálmarsson, Gunnlaugur K Hreiðarsson, Mohammad Hamdaqa, and Gísli Hjálmtýsson. Blockchain-based e-voting system. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 983–986. IEEE, 2018.

[59] Xin Jin, Xiaozhou Li, Haoyu Zhang, Nate Foster, Jeongkeun Lee, Robert Soule, Changhoon Kim, and Ion Stoica. Netchain: Scale-free sub-rtt coordination. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 35–49, 2018.

[60] Manos Kapritsos, Yang Wang, Vivien Quema, Allen Clement, Lorenzo Alvisi, and Mike Dahlin. All about eve: Execute-verify replication for multi-core servers. In *10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*, pages 237–250, 2012.

[61] Archana Kesavan. Comparing the network performance of aws, azure and gcp, 2019.

[62] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583–598. IEEE, 2018.

[63] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: speculative byzantine fault tolerance. *ACM SIGOPS Operating Systems Review*, 41(6):45–58, 2007.

[64] Ramakrishna Kotla and Michael Dahlin. High throughput byzantine fault tolerance. In *International Conference on Dependable Systems and Networks, 2004*, pages 575–584. IEEE, 2004.

[65] Lucas Kuhring, Zsolt István, Alessandro Sorniotti, and Marko Vukolić. Streamchain: Rethinking blockchain for datacenters. *arXiv*, pages arXiv–1808, 2018.

[66] Leslie Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[67] Haochen Li, Keke Gai, Zhengkang Fang, Liehuang Zhu, Lei Xu, and Peng Jiang. Blockchain-enabled data provenance in cloud datacenter reengineering. In *Proceedings of the 2019 ACM International Symposium on Blockchain and Secure Critical Infrastructure*, pages 47–55, 2019.

[68] He Li, Peng Li, Song Guo, and Amiya Nayak. Byzantine-resilient secure software-defined networks with multiple controllers in cloud. *IEEE Transactions on Cloud Computing*, 2(4):436–447, 2014.

[69] Jialin Li, Ellis Michael, and Dan RK Ports. Eris: Coordination-free consistent transactions using in-network concurrency control. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 104–120. ACM, 2017.

[70] Jialin Li, Ellis Michael, Naveen Kr Sharma, Adriana Szekeres, and Dan RK Ports. Just say no to paxos overhead: Replacing consensus with network ordering. In *OSDI*, pages 467–483, 2016.

[71] Jialin Li, Jacob Nelson, Ellis Michael, Xin Jin, and Dan RK Ports. Pegasus: Tolerating skewed workloads in distributed storage with in-network coherence directories. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, pages 387–406, 2020.

[72] Zhetao Li, Jiawen Kang, Rong Yu, Dongdong Ye, Qingyong Deng, and Yan Zhang. Consortium blockchain for secure energy trading in industrial internet of things. *IEEE transactions on industrial informatics*, 14(8):3690–3700, 2017.

[73] Xueping Liang, Sachin Shetty, Deepak Tosh, Charles Kamhoua, Kevin Kwiat, and Laurent Njilla. Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 468–477. IEEE, 2017.

[74] Alex Manuskin, Michael Mirkin, and Ittay Eyal. Ostraka: Secure blockchain scaling by node sharding. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 397–406. IEEE, 2020.

[75] Justin Meza, Tianyin Xu, Kaushik Veeraraghavan, and Onur Mutlu. A large scale study of data center network reliability. In *Proceedings of the Internet Measurement Conference 2018*, pages 393–407, 2018.

[76] Ciamac C Moallemi and Mehmet Saglam. The cost of latency. *SSRN eLibrary*, 2010.

[77] Satoshi Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system. 2008.

[78] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, pages 305–319, 2014.

[79] Dan RK Ports, Jialin Li, Vincent Liu, Naveen Kr Sharma, and Arvind Krishnamurthy. Designing distributed systems using approximate synchrony in data center networks. In *NSDI*, pages 43–57, 2015.

[80] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, and Alex C Snoeren. Passive realtime datacenter fault detection and localization. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 595–612, 2017.

[81] Pingcheng Ruan, Dumitrel Loghin, Quang-Trung Ta, Meihui Zhang, Gang Chen, and Beng Chin Ooi. A transactional perspective on execute-order-validate blockchains. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 543–557, 2020.

[82] Signe Rüsch, Ines Messadi, and Rüdiger Kapitza. Towards low-latency byzantine agreement protocols using rdma. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 146–151. IEEE, 2018.

[83] Moein Sabounchi and Jin Wei. Towards resilient networked micro-

grids: Blockchain-enabled peer-to-peer electricity trading mechanism. In *2017 IEEE Conference on Energy Internet and Energy System Integration (EI2)*, pages 1–5. IEEE, 2017.

[84] Vikram Saraph and Maurice Herlihy. An empirical study of speculative concurrency in ethereum smart contracts. *arXiv preprint arXiv:1901.01376*, 2019.

[85] Ankur Sharma, Felix Martin Schuhknecht, Divya Agrawal, and Jens Dittrich. Blurring the lines between blockchains and database systems: the case of hyperledger fabric. In *Proceedings of the 2019 International Conference on Management of Data*, pages 105–122, 2019.

[86] Peter Shen. Investing In A Blockchain Future At Singapore Exchange, July 2019.

[87] Joao Sousa and Alysson Bessani. From byzantine consensus to bft state machine replication: A latency-optimal transformation. In *2012 Ninth European Dependable Computing Conference*, pages 37–48. IEEE, 2012.

[88] Joao Sousa, Alysson Bessani, and Marko Vukolic. A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform. In *2018 48th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pages 51–58. IEEE, 2018.

[89] Chrysoula Stathakopoulou, Tudor David, Matej Pavlovic, and Marko Vukolić. Mir-bft: High-throughput robust bft for decentralized networks. *arXiv preprint arXiv:1906.05552*, 2019.

[90] Hadar Sufiev, Yoram Haddad, Leonid Barenboim, and José Soler. Dynamic sdn controller load balancing. *Future Internet*, 11(3):75, 2019.

[91] Harish Sukhwani, Nan Wang, Kishor S Trivedi, and Andy Rindos. Performance modeling of hyperledger fabric (permissioned blockchain network). In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–8. IEEE, 2018.

[92] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, and Lau Cheuk Lung. Spin one's wheels? byzantine fault tolerance with a spinning primary. In *2009 28th IEEE International Symposium on Reliable Distributed Systems*, pages 135–144. IEEE, 2009.

[93] Guohui Wang and TS Eugene Ng. The impact of virtualization on network performance of amazon ec2 data center. In *2010 Proceedings IEEE INFOCOM*, pages 1–9. IEEE, 2010.

[94] Jiaping Wang and Hao Wang. Monoxide: Scale out blockchains with asynchronous consensus zones. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, pages 95–112, 2019.

[95] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[96] Kazuhiro Yamashita, Yoshihide Nomura, Ence Zhou, Bingfeng Pi, and Sun Jun. Potential risks of hyperledger fabric smart contracts. In *2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 1–10. IEEE, 2019.

[97] Tian Yang, Robert Gifford, Andreas Haeberlen, and Linh Thi Xuan Phan. The synchronous data center. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, pages 142–148, 2019.

[98] Jingjing Yao, Ping Lu, Long Gong, and Zuqing Zhu. On fast and coordinated data backup in geo-distributed optical inter-datacenter networks. *Journal of Lightwave Technology*, 33(14):3005–3015, 2015.

[99] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus in the lens of blockchain. *arXiv preprint arXiv:1803.05069*, 2018.

[100] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.

[101] Aydan R Yumerefendi and Jeffrey S Chase. Strong accountability for network storage. *ACM Transactions on Storage (TOS)*, 3(3):11–es, 2007.

[102] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 931–948, 2018.

[103] Gaoxiong Zeng, Wei Bai, Ge Chen, Kai Chen, Dongsu Han, Yibo Zhu, and Lei Cui. Congestion control for cross-datacenter networks. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pages 1–12. IEEE, 2019.

[104] Jiashuo Zhang, Jianbo Gao, Zhenhao Wu, Wentian Yan, Qize Wo, Qingshan Li, and Zhong Chen. Performance analysis of the libra blockchain: An experimental study. In *2019 2nd International Conference on Hot Information-Centric Networking (HotICN)*, pages 77–83. IEEE, 2019.

[105] Jia Zou, Gong Su, Arun Iyengar, Yu Yuan, and Yi Ge. Design and analysis of a distributed multi-leg stock trading system. In *2011 31st International Conference on Distributed Computing Systems*, pages 13–24. IEEE, 2011.