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

PAPER # 133

A permissioned blockchain framework typically runs an efficient Byzantine distributed consensus protocol and is especially attractive to deploy trading and payment applications among mutually untrusted entities. Unfortunately, all existing permissioned blockchain frameworks adopt a sequential workflow of the consensus protocol and applications' transaction execution in order to suit both Internet and datacenter networks, making both the latency and throughput of these applications orders of magnitude worse than deploying them in traditional frameworks (e.g., in-datacenter stock exchange). We propose Bidl, the first permissioned blockchain framework dedicatedly deployed in approximately synchronous networks (typically, datacenter networks). We leverage the network ordering in such networks to create a *shepherded sequencer abstraction*. This abstraction enables the parallelism of the consensus protocol and transaction execution speculatively, and then securely commits the transactions. Compared with Hyperledger Fabric (HLF), the most notable and one of the fastest permissioned blockchain frameworks, Bidl's new parallel workflow improves applications' latency by 73.7% and their throughput by up to 9.1×. Bidl's performance is comparable to traditional trading frameworks. Bidl's code and results are released on github.com/sigmetrics21/bidl.

## 1 INTRODUCTION

Cross-enterprise trading applications (e.g., stock exchange [1, 6, 8, 10, 59] and online payment [12, 18, 23]) facilitate instant recording, clearing, or settling of transactions between mutually untrusted participants (e.g., banks and merchants). These trading applications are often deployed in *approximately synchronous* (AS) networks, including a single datacenter or multiple datacenters connected with dedicated network cables [44, 47, 51, 78, 95]. To process transactions from a large number of participants with real-time commit latency (e.g., tens of milliseconds [56, 102]), these trading applications usually have performance demands. For instance, the Hong Kong stock exchange framework typically runs at a trading transaction rate of 60K txns/s [5].

The prosperity of blockchains attracts industries and academia to develop new blockchain frameworks (e.g., Libra [18]) for various trading applications without involving a centralized authority. A permissioned blockchain framework (e.g., Libra [18], Hyperledger Fabric [26] and Quorum [9]) is especially suitable to realize this development. Specifically, unlike Bitcoin (a permissionless blockchain [76]), a *permissioned* blockchain is a shared ledger that runs among a set of *identified*, mutually untrusted participants as member nodes, and runs a fast Byzantine-fault-tolerant (BFT) consensus protocol (e.g., BFT-SMaRt [29]) among the nodes to commit transactions with high performance and energy efficiency [26]. For instance, a Singapore company is developing a novel stock trading system based on the notable permissioned blockchain system Hyperledger Fabric (HLF) [26] with the aim of deploying the system within a datacenter [82].

All existing permissioned blockchain systems adopt sequential workflows: at a high level, according to whether the consensus protocol is invoked before or after the transaction execution, blockchain workflows are divided into two categories: *consensus-execute* (e.g., Ethereum [93] and Quorum) or *execute-consensus* (e.g., HLF [26]). The most notable permissioned blockchain system HLF follows an *execute-consensus* workflow to process transactions. In HLF, a member node (for short, a node) running on a dedicated computer first executes transactions, then invokes a BFT consensus protocol to order and pack the transactions' execution results into blocks (for short, block

Author's address: Paper # 133.

consensus). No single participant has the privilege to tamper with the execution results of the transactions. Due to the nodes' explicit identities, a permissioned blockchain can run traditional BFT consensus protocols (e.g., PBFT [36] and BFT-SMaRt [29]) and can effectively detect malicious nodes. Overall, all existing permissioned blockchain systems are designed for both the lossy Internet and the AS networks (typically, datacenter networks), so their sequential workflow is essential to handle lost or reordered transactions. Specifically, some nodes may receive reordered transactions and executed them, so all nodes' execution results must subsequently go through a BFT protocol, otherwise they may commit different transactions and violate a blockchain's strong consistency [36, 76].

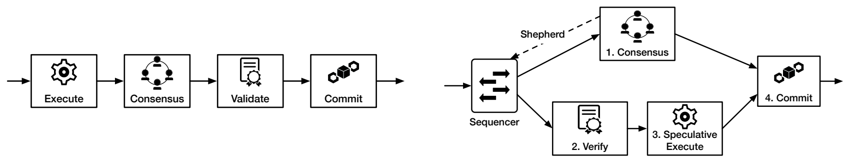
Recently, to achieve high performance, more and more blockchain systems and applications are being deployed in AS networks [63, 68, 72]. Moreover, the leading cloud providers such as Amazon [13], IBM [14], and Microsoft [20] also provide built-in permissioned blockchain frameworks (especially HLF) for cloud tenants in their datacenters.

Unfortunately, even being deployed in the fastest datacenter network (e.g.,100Gbps bandwidth, 0.1ms RTT), the performance of existing permissioned blockchains is orders of magnitude lower than the performance of traditional trading systems (e.g., Hong Kong stock exchange aims to process 60K txns/s [5]). For instance, Quorum's workflow achieves an end-to-end throughput of merely 1.6K transactions/s with an end-to-end latency of about 200ms in a datacenter [27]. Another instance is HLF: we ran HLF with its mainstream block consensus protocol BFT-SMaRt [29] (four consensus nodes) in our own cluster with 40Gbps network. HLF's workflow achieves an end-to-end throughput of merely 3K transactions/s, and the end-to-end latency of a transaction is about 100ms (§6).

In this paper, we owe the low performance of permissioned blockchains running in AS networks to their sequential workflows (will be illustrated in §2.1). At a detailed level, HLF's workflow is *execute* → *consensus* → *validate* → *commit* (Figure 1a): each node in HLF first executes all received transactions, orders the execution results of these transactions through a BFT protocol among nodes, validates whether there are conflicts in the ordered transactions' execution results, and commits the valid transactions. Each of the first three steps consumes tens of milliseconds, causing a long end-to-end application latency.

Moreover, to ensure that all nodes can commit the same execution results, the second step of HLF needs to perform BFT consensus on the entire block containing all the execution results of thousands of transactions, and the execution result of each transaction contains thousands of bytes with HLF's essential metadata [26]. Worse, in a general BFT protocol, since it is essential for a specific BFT consensus node (e.g., the consensus leader [29, 36, 53, 61]) to distribute all the transactions with all payloads (all the execution results) to all consensus nodes (can be hundreds [53]), HLF's sequential workflow restricts the entire system's throughput to at most a few thousands of transactions per second (§6.2). Overall, we believe existing permissioned blockchains' sequential workflow is the key reason that makes applications' throughput and latency unsatisfiable even being deployed in fast AS networks.

Inspired by existing ultra-fast distributed consensus protocols dedicatedly developed for AS networks (e.g., Speculative Paxos [78], NoPaxos [71], and Eris [70]), our key observation is that the network ordering in AS networks also has great potential to parallelize the sequential workflows of existing permissioned blockchains, greatly improving both application throughput and latency. When a permissioned blockchain is dedicatedly deployed in an AS network, the network ordering has the potential to move a part of the duties of the BFT consensus protocol (i.e., transaction ordering and distribution) to the routing layer: we can run a dedicated computer as a *sequencer*, which adds sequence numbers to all incoming transactions and re-routes all transactions to all nodes by routing-aware multicast. If the sequencer is never faulty (i.e., always sends the same sequence of transactions to all nodes), all nodes can receive all transactions in almost the same complete order (no gap). All nodes can safely execute and commit the transactions on their own without



(a) HLF's sequential workflow (b) Bidl's new parallel workflow

Fig. 1. The workflows of Bidl and HLF.

performing any consensus. When a node detects a gap in its received transactions' sequence, it can just request other nodes for the lost transactions.

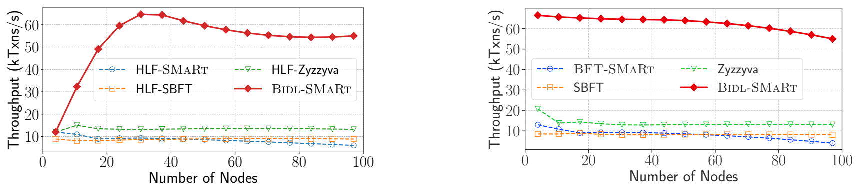
However, all these notable work (e.g., Speculative Paxos [78], NoPaxos [71], and Eris [70]) targets only the crash fault tolerance model [24, 64], and cannot be applied to permissioned blockchains which require the BFT model [36]. For instance, the sequencer can be malicious. A malicious sequencer may send transactions with inconsistent sequence numbers and contents to different nodes [36, 87]. Moreover, any blockchain node can be malicious. A malicious node can forge the transactions from the sequencer and distribute them to other nodes. With the malicious sequencer or malicious nodes, different nodes may receive inconsistent transactions and commit different transactions, causing the blockchain saved by different nodes to be inconsistent (i.e., the blockchain forks [92]). A strawman approach to defend against malicious nodes forging the transactions from the sequencer is to force the sequencer to sign on the added sequence number on every transaction; these signatures will greatly downgrade the throughput of the sequencer (e.g., a 16-core computer can only sign 8K txns/s with the ECDSA signature [83]).

In summary, realizing a practical (high-performance) permissioned blockchain is especially challenging: we must include a secure sequencer protocol without letting the sequencer sign on the sequence numbers. Without the signatures, a node cannot distinguish whether a transaction is from the sequencer or malicious nodes, so this protocol must have new mechanisms to prevent both malicious nodes and sequencer.

In this paper, we present Bidl[1], the first high-performance permissioned blockchain framework, which takes the first step to explore the network ordering mechanism in one AS network and is compatible with a blockchain's BFT model. We present a new *shepherded sequencer* abstraction: the sequencer adds sequence numbers without signatures to all received transactions sent from all clients and broadcasts the transactions to all nodes. Upon receiving transactions from the sequencer, Bidl nodes conduct light-weight verifications (e.g., valid client transactions). As shown in Figure 1b, this abstraction enables a new parallel workflow in Bidl: nodes speculatively execute the received transactions if no gap exists in transactions' sequence number space; in parallel, nodes also invoke a regular BFT protocol to agree on the blocks as well as to shepherd the transactions sent by the sequencer. Specifically, we develop a *denylist-based sequencer view change protocol* to efficiently replace a malicious sequencer with another one and to keep Bidl's workflow at high performance, while making the entire workflow comply with a permissioned blockchain's BFT model and strong consistency.

We implemented Bidl on the HLF 1.3 [4] codebase. Bidl has a modular architecture (§4), so existing BFT consensus protocols can be easy to deploy in the Bidl framework with moderate engineering efforts. We compared Bidl with the HLF [26] permissioned blockchain framework in the same AS network. We ran three popular BFT consensus protocols, including BFT-SMaRt [29], SBFT [53], and Zyzzyva [61], on both the two frameworks and measured their performance with

[1]Blockchain-powered In-Datacenter Ledger



(a) End-to-end throughput of HLF and BIDL. (b) Throughput of block consensus protocols.

Fig. 2. End-to-end and consensus throughput of HLF and BIDL with three block consensus protocols.

typical workloads. Evaluation results show that:

(1) BIDL is efficient in AS networks. BIDL achieves up to 73.7% lower latency and 9.1× higher throughput than HLF (Figure 2a).
(2) BIDL's block consensus performance is scalable to the number of consensus nodes. When running BFT-SMaRt with four consensus nodes, BIDL achieves 5.1× better throughput than HLF. When the number of nodes increases to 97, BIDL's throughput outperforms HLF by 9.2× (Figure 2b).
(3) BIDL's performance is robust on malicious sequencer and nodes.

Our major contribution is BIDL, the first high-performance permissioned blockchain framework designed for AS networks. BIDL exploits the network ordering in AS networks to deliver transactions, which enables the design of a new high-performance parallel workflow while complying with the BFT model of permissioned blockchains. This new workflow can be implemented with other permissioned blockchain frameworks (e.g., HLF [26] and Quorum [9]) for cloud providers (e.g., Amazon) when the entire blockchain framework runs within one AS network. Except for the datacenter networks, BIDL can also be deployed in other AS networks such as the Nasdaq's millimeter wave network [11] and the InfiniBand long-reach system [16].

The rest of the paper is organized as follows: §2 introduces the permissioned blockchains and systems in AS networks. §3 gives an overview of BIDL. §4 introduces BIDL's protocol. §5 presents implementation details. §6 shows our evaluation, and §7 concludes the paper.

## 2 RELATED WORK

Our goal is to design a more efficient permissioned blockchain by taking advantage of the features in AS networks. Before introducing our approach, we briefly discuss existing permissioned blockchains as well as the distributed systems designed for AS networks.

### 2.1 Permissioned Blockchains

A permissioned blockchain is a blockchain system where only nodes with known identities can add new transactions to the blockchain [26, 89]. Typically, permissioned blockchains run BFT consensus protocols to agree on the order of transactions in blocks. PBFT [36] is the first efficient BFT algorithm. BFT-SMaRt [29] implements the PBFT protocol in Java, which fully exploits the modern hardware. BFT-SMaRt outperforms PBFT in real deployments and is used as one of the mainstream consensus protocols in HLF [2].

Existing permissioned blockchains that built on top of BFT consensus protocols have two major limitations. First, their BFT consensus protocols suffer from the scalability problem: when the number of consensus nodes increases, the performance of a BFT protocol deteriorates dramatically. There have been many proposals created to optimize the scalability of BFT consensus protocols. One major cause of existing BFTs' poor scalability is their $O(n^2)$ message complexity, where $n$ is the number of consensus nodes. Zyzzyva [61], FastBFT [73], and SBFT [53] use message aggregation to reduce the message complexity to $O(n)$. Some studies improve BFT's scalability by reducing the resources required to tolerate $f$ faulty replicas. A2M [39], TrInc [67], CheapBFT [60], MinBFT [88], and Hybster [28] use the trusted components [41, 48] to reduce the number of required replicas from $3f+1$ to $2f+1$. Yin et al. propose a new architecture for BFT consensus protocol that separates consensus from execution [97], which reduces replication costs. Yin's new architecture can tolerate faults in half of the replicas that execute requests. Despite these advances on building new BFT consensus protocols, in general, scalability remains a fundamental problem for BFTs.

Second, existing blockchains, including both permissioned and permissionless blockchains, follow a sequential workflow. Specifically, the block consensus and transaction execution run in a sequential order. The sequential workflows can be divided into two categories: *consensus-then-execute* or *execute-consensus*. First, most blockchains (e.g., Bitcoin [76] and Ethereum [93]) follow the *consensus-execute* workflow: nodes first achieve the total order of transactions by consensus; then they execute transactions in this order. The second workflow is the *execute-consensus* workflow [26]: nodes first execute transactions and generate a proposal (execution results) for each transaction; then they order the proposals by consensus. Since both the transaction execution (e.g., smart contracts [26, 33]) and consensus in a permissioned blockchain often takes tens of milliseconds, this sequential workflow incurs high end-to-end latency (§6.3).

Our BIDL framework mitigates these two limitations by leveraging the network ordering in AS networks to improve the scalability of consensus and parallelize the consensus with the transaction execution. First, BIDL exploits the network ordering in an AS network to improve the scalability of BFT consensus protocols. To the best of our knowledge, existing BFTs rely on the consensus leader to deliver clients' requests to the other nodes. The consensus leader takes a much higher workload than non-leader nodes, and usually becomes the performance bottleneck. BIDL alleviates this load imbalance by utilizing the network ordering to order and deliver requests. The sequencer delivers transactions to all nodes using the UDP multicast. In an AS network, transactions can reach all nodes in almost the same order. This mechanism greatly improves the scalability of BFT protocols when they are running in the BIDL framework. For instance, when BFT-SMaRt runs in the BIDL framework with four consensus nodes, it can achieve 5.1× better throughput compared with running BFT-SMaRt alone in the same AS network (§6.2).

Second, the network ordering enables a new parallel workflow for BIDL. As depicted in Figure 1b, since the sequencer has already ordered the transactions, the execution module can directly execute transactions after verification, without waiting for the result of consensus. Therefore, the transaction execution can run in parallel with the block consensus to reduce the end-to-end latency.

In this paper, we choose to build a permissioned blockchain instead of a permissionless blockchain for two reasons. First, to handle Sybil attacks [45], permissionless blockchains require nodes who wish to participate in block production to provide either high computation resources (e.g., PoW [76]) or stake (e.g., PoS [93] and DPoS [94]), which are not applicable to our target applications. For instance, in a clearing system, there is no cryptocurrency, and high power consumption is unacceptable. On the other hand, in permissioned blockchains, node identities are controlled and managed, so malicious nodes are not able to perform Sybil attacks by forging identities, and the aforementioned requirements are thus unnecessary.

Second, our goal is to build a high-performance blockchain framework for trading systems. Permissioned blockchains use consensus protocols with explicit membership such as PBFT [36] and HotStuff [98], which generally provide higher throughput and lower latency than the consensus protocols of permissionless blockchains.

## 2.2 Applications Leveraging Approximately Synchronous Networks

The approximately synchronous (AS) networks have several features that are not available in asynchronous networks. Take the datacenter network as an example, there are two key features [78]. First, datacenter networks are reliable. Datacenter networks are built with structured topologies (e.g., fat-tree [25, 37, 49, 66]) using business equipment and modern network techniques. Thus, they can provide stable and optimal latency with extremely low packet loss/reorder rate (e.g., $10^{-5}$ [101]). Second, equipment (e.g., switches, servers) in a datacenter network belongs to a single administrative domain, making it possible to deploy in-network processing and routing rules with programmable switches [32] and SDN [62]. These key features of datacenter networks offer opportunities to design more efficient distributed systems.

**Programmable Switch.** With the development of the programmable switch, several recent studies have investigated moving application-level logic to network devices [38]. NetPaxos [43], CAANS [42], and NetChain [57] implement the entire Paxos protocol [64] in programmable switches.

**Network Ordering.** Speculative Paxos [78] assigns the duty of requests ordering to the network-level. It relies on the network to provide the best-effort ordering property. NoPaxos [71] takes one step further. NoPaxos demonstrates that building a network that can provide ordering guarantee is possible. This ordering guarantee in the network-level can simplify the Paxos crash-tolerant consensus protocol [64] while achieving high performance. The above two network ordering protocols are all implemented using SDN running in one datacenter.

Although these latest systems [71, 78] use the network ordering to speed up consensus, Bidl differs from them in three aspects. First, Bidl handles the Byzantine failures (i.e., malicious sequencer and malicious nodes), while existing systems only handle the crash failures. Second, Bidl's parallel workflow is tailored for blockchains, where transactions can invoke smart contracts. A smart contract is a piece of executable code stored on the blockchain which can be invoked by clients' transactions [26, 33]. The execution time of permissioned blockchains' smart contracts is often tens of milliseconds [26, 84], comparable to the latency of consensus. Therefore, parallelizing consensus and execution greatly reduces the end-to-end latency. In contrast, existing network ordering based replication systems are mainly designed for the key-value store replication; the time cost of processing a key-value request is often tens of microseconds, much shorter than the time cost of consensus. Third, Bidl leverages the *execute-then-commit* paradigm of smart contracts to eliminate rollbacks in all cases. Execution of a smart contract can be divided into two steps: *execute* and *commit*. Bidl first performs the *execute* step and only *commits* the execution results to the database when consensus succeeds. Existing systems [71, 78] adopt a *speculation-rollback* paradigm: they speculatively commit the requests and adopt a rollback when consensus fails. The *Speculation-rollback* paradigm is unsuitable for the consistency guarantee of financial transactions, as inconsistent transaction results may be temporarily visible to clients.

## 3 OVERVIEW

### 3.1 Deployment Requirements and Threat Model

Bidl requires all member nodes of its permissioned blockchain to be deployed in the same AS network: a single datacenter or multiple datacenters connected with dedicated network cables [35, 44, 47, 50, 52, 58, 65, 74, 75, 78, 79, 90, 91, 95, 96]. This is because Bidl uses the network ordering in AS networks to order and disseminate transactions. Some Internet-scale high-speed networks can also provide a similar guarantee [11, 16]. Bidl's clients can be deployed within or outside the AS network.

We denote a message $M$ signed by the private key of sender $s$ as $\langle M \rangle_{\delta_s}$. Message payload $p$ encrypted by the public key of sender $s$ is expressed by $p_{pk_s}$. Each node knows the public keys of the other nodes and the clients to verify signatures. We denote a message $M$ containing a MAC vector appropriate for verification by all nodes as $\langle M \rangle_{\vec{\mu}}$.

BIDL adopts the BFT model, where faulty components can behave arbitrarily. We do not trust the sequencer. A faulty sequencer may fail to deliver transactions, corrupt them, delay them, or deliver them out of order. Clients are not trusted. A malicious client may issue illegal transactions (e.g., transfer money to an unavailable account). BIDL runs BFT consensus and can tolerate at most $f = \lfloor \frac{n-1}{3} \rfloor$ faulty consensus nodes. Although BIDL's sequencer can be malicious, BIDL can provide the same security guarantee with HLF (§4.5.2). Although HLF uses Raft [22] (only tolerates the crash failures) as its default block consensus protocol, because we target the trading applications between mutually untrusted participants in a BFT model, we did not consider Raft in this paper. The adversary cannot break cryptographic techniques such as collision-resistant hashing, message authentication codes (MACs), encryption, and digital signatures. Each non-faulty node holds the public key of all nodes and its clients.

Although BIDL leverages the approximate network synchrony to speed up consensus as well as to parallelize consensus and execution, BIDL does not require absolute synchrony. Specifically, BIDL ensures **safety**: all non-faulty nodes (typically, $2f + 1$ out of all $3f + 1$ nodes) commit the same blocks containing the same totally ordered transactions no matter whether the AS network lost or reordered packets. Even if packet losses/reorders occur (e.g., the malicious sequencer drops transactions), BIDL's consensus module invokes BFT consensus such as BFT-SMaRt [29] to ensure the safety property (§4.4).

BIDL does rely on this approximate synchrony in an AS network to provide **liveness**: all valid clients' transactions sent to nodes are received within limited time, and all the transactions are eventually committed on-chain within limited time. In our evaluation, this limited time is tens of milliseconds.

## 3.2 BIDL's Workflow Overview

Existing blockchain systems such as HLF are designed for both AS networks and asynchronous networks (e.g., Internet-scale WAN networks). To achieve high performance, many permissioned blockchain systems are deployed in AS networks such as datacenters [13–15, 20, 63, 68]. However, even running in AS networks, existing permissioned blockchains still exhibit poor performance (e.g., HLF achieves only 3K txns/s in a single datacenter [26]), far from the requirements of many applications (e.g., trading systems usually require 20K txns/s [56]). We own the key reason of existing permissioned blockchain frameworks' low performance to their sequential workflows (§1).

BIDL leverages the network ordering in AS networks to develop a new parallel workflow. BIDL introduces a shepherded sequencer which adds sequence numbers to incoming client transactions and delivers transactions to all nodes using a routing level UDP multicast. The UDP multicast is group communication, where the destination of a message can be a group of nodes. With the UDP multicast, the sequencer can efficiently send each transaction to a large number of nodes in a single transmission. Copies are automatically created in network level switches which can process packets at line rate [17]. In an AS network, the packet loss/reorder rarely happens; all nodes can receive the same sequence of transactions in the same order with high probability.

BIDL's sequencer-based transaction delivery mechanism can greatly improve the end-to-end throughput and latency than those in sequential workflows (Figure 1a) due to two reasons. First, the sequencer is already shepherded in BIDL and can safely improve the throughput of the committed blocks. In an AS network, all nodes can receive the transactions from the sequencer in almost the same order, BIDL's BFT consensus nodes do not need to re-transmit the large number of transactions' execution results (§1) as in HLF, so the consensus content of BIDL's BFT consensus module contains only the transaction metadata (i.e., 32 bytes hash) indicating each transaction's order in each block. Since the performance of consensus is susceptible to the message size transmitted by the leader during consensus, this **consensus-on-hash** mechanism greatly improves the performance and scalability of block consensus (confirmed in §6.2). Second, BIDL's workflow can reduce the end-to-end latency. The transaction execution runs in **parallel** with the block consensus, leading to much lower end-to-end latency (see §6.3).

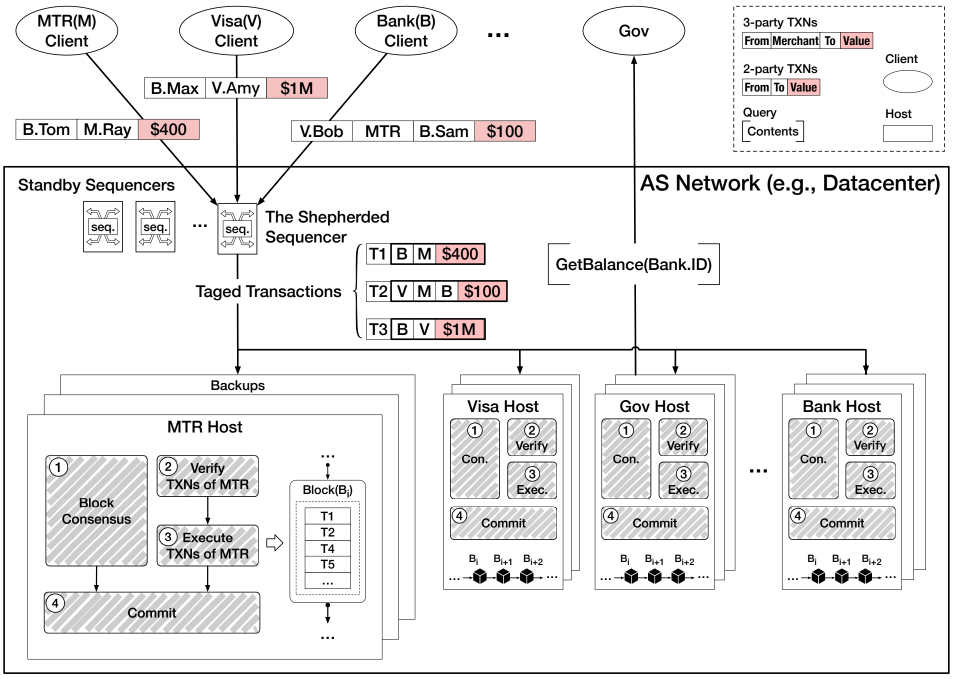


Fig. 3. Architecture of BIDL. BIDL's modules are shaded.

We now sketch the transaction processing workflow of BIDL. BIDL is permissioned: all entities (e.g., banks and merchants) that participate in the network are identified as BIDL members using a standard membership protocol of HLF. Each member node (for short, each node) runs on one computer. Figure 3 shows BIDL's architecture and BIDL's modules are shaded.

(1) **Consensus Module** runs block consensus among nodes. Nodes collect transactions from the sequencer. Once nodes have received enough transactions or timeout elapsed, they invoke a BFT consensus protocol to agree on the transaction hashes.
(2) **Verification Module** verifies received transactions from the sequencer to check whether they are valid (e.g., contain correct signatures of BIDL clients or other BIDL member nodes).
(3) **Execution Module** executes the verified transactions. Since the sequencer has already ordered the transactions, the execution module can directly and speculatively execute transactions after verification according to the orders issued by the sequencer without waiting for the result of consensus. Therefore, the transaction execution runs in parallel with the consensus module.
(4) **Commit Module** collects the results of the consensus module and the execution module. Each node checks if the hashes of its local executed transactions (from the execution module) are consistent with the transaction hashes in the block (from the consensus module). If the

hashes are consistent, a node directly commits the execution results to its local database. If the hashes are not consistent, a node will re-execute the transactions (§4.3).

The key distinctions between Bidl and existing permissioned blockchains (e.g., HLF [26], Ethereum [93], and Quorum [9]) are two-fold. First, in existing blockchain systems, the block consensus needs to perform consensus on large transaction payloads. The consensus nodes take too much workload, which greatly limits the consensus performance. Thanks to Bidl's shepherded sequencer, Bidl's consensus module agrees only on transaction hashes. Second, compared to the sequential workflow of HLF, the most popular and one of the fastest permissioned blockchains, Bidl's parallel workflow improves applications' latency by 73.7% and their throughput by up to 9.1×. Bidl's performance is comparable to traditional trading frameworks (§6).

Overall, the highlight of Bidl's parallel workflow stems from enforcing strict **Safety** and reasonable **Liveness** (§4.6). We illustrate in two aspects. First, Bidl ensures safety by running a regular BFT protocol in parallel with the speculative transaction execution and by committing transactions only if the speculatively executed transactions are consistent with the transactions' consensus results. Specifically, Bidl's commit phase commits transactions on-chain only when the total order of the speculative executed transactions and the total order of transactions ordered by the BFT protocol are identical. Even if the sequencer is malicious, the integration of the Bidl's BFT protocol and Bidl's commit phase collectively prevents the malicious sequencer affecting Bidl's safety (§4.4), making the sequencer at most be able to affect Bidl's liveness. Therefore, Bidl's entire parallel workflow complies with the BFT model and strong consistency of a permissioned blockchain.

Second, Bidl can provide reasonable liveness by efficiently detecting the malicious sequencer and switching to a new sequencer via its denylist based sequencer view change protocol (§4.5.1). For instance, if the sequencer sends inconsistent (e.g., incorrectly ordered) or forged transactions to different nodes and (at most) causes Bidl's commit phase to fail to commit a minor portion of transactions, this sequencer view change protocol (§4.5.1) can sensibly detect this minor throughput loss and be invoked.

## 4 PROTOCOL DESCRIPTION

In this section, we describe Bidl's runtime protocol. Bidl's detailed workflow is shown in Figure 4.

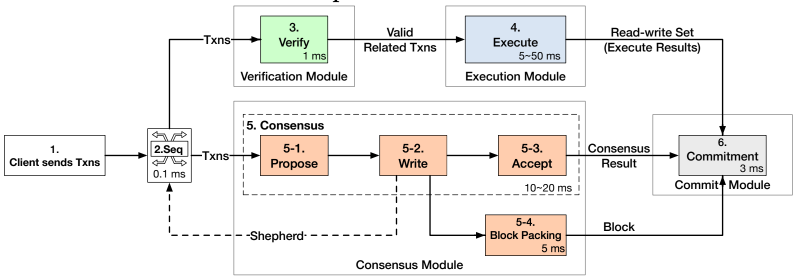


Fig. 4. Detailed workflow of Bidl.

### 4.1 Client Behavior

Transactions in a permissioned blockchain [26] can be divided into two categories: public transactions and private transactions. A public transaction's payload is visible to all participants. A private transaction's payload is only visible to related participants, which are specified in a *privateFor* field in a typical permissioned blockchain transaction format (defined below). A private transaction is only executed by nodes of related participants.

**Step 1: Clients submit transactions to the sequencer.** A client $c$ issues a new transaction $\langle \text{TXN}, \mathcal{P}, h_p, privateFor, t \rangle_{\delta_c}$ by sending the transaction to Bidl's sequencer group address. If the client does not receive a timely result from the related nodes, it will re-submit the transaction.

**Step 2: The sequencer broadcasts transactions to all nodes.** Client transactions sent to the sequencer group address are routed to the sequencer. The sequencer adds a sequence number to each incoming transaction and broadcasts the transaction to all nodes. A sequence number is a 2-tuple *s: (view-num, txn-num)*, where *view-num* is the view number, and *txn-num* is the transaction's serial number in the current view. Bidl uses a view-based approach. In each view, there is one active sequencer, which assigns a *view-num* to transactions. Nodes discard out of order transactions (e.g., from earlier views) and detect dropped transactions by noticing gaps in the sequence number. Considering the performance, sequencer neither signs nor adds MACs to transactions.

## 4.2 Verification

**Step 3: Nodes collect and verify transactions.** Upon receiving a transaction issued by client $c$ $tx : \langle\langle \textsc{txn}, \mathcal{P}, h_p, privateFor, t\rangle_{\delta_c}, s\rangle$ from the sequencer, node $n_i$ stores it in the *txns_recv* set, and starts to verify $tx$ by the following steps:

(1) **Denylist check.** If the client is not denylisted, proceeding to step (2). Otherwise, $n_i$ ignores this transaction. The denylist mechanism is discussed in §4.5.

(2) **Sequence check.** If the sequence number $s$ lies in the range of the current block, then go to step (3). Otherwise, if the transaction belongs to previous blocks, $n_i$ discards the transaction; if the sequence number is larger than the range of current block, $n_i$ caches the transaction for further processing; if the sequencer number is conflict with a previous transaction with different hash in the same block, all related nodes of these two transactions are suspected as malicious and are put in a local denylist. The denylist mechanism is discussed in §4.5.2.

(3) **Relevance check.** $n_i$ determines if it is related to the transaction by checking the transaction's *privateFor* field. If $n_i$ holds the transaction, then proceed to step (4). Otherwise, $n_i$ only stores the transaction in the *txns_recv* set and waits for block consensus.

(4) **Replay check.** $n_i$ checks the timestamp of the most recent transaction from client $c$. If the timestamp is smaller than $tx.t$, then proceeds to step (5). Otherwise, $tx$ is regarded as invalid.

(5) **Signature check.** If $\delta_c$ is valid, then the transaction $tx$ is regarded as valid.

## 4.3 Transaction Execution

**Step 4. Nodes execute related transactions.** After verification, nodes speculatively execute related transactions. Because the sequencer has already ordered the transactions, nodes can execute related transactions according to the sequence number without waiting for the consensus result. A rw-set of related transactions in this block is generated and will be committed or aborted in step 6.

Since transactions that do not have dependencies with each other can be executed in parallel, and thus, reducing total execution time. Let $tx_1$ and $tx_2$ denote two transactions. $tx_2$ depends on $tx_1$ if (1) both $tx_1$ and $tx_2$ have the same sender (e.g., $tx_1 : A \rightarrow B, tx_2 : A \rightarrow C$), or (2) receiver of $tx_1$ is the sender of $tx_2$ (e.g., $tx_1 : A \rightarrow B, tx_2 : B \rightarrow C$). Each node analyzes dependencies between transactions by constructing dependency graphs. To capture smart contracts' dependencies, a transaction invoking smart contract should provide hints about the variables to modify. Two options exist to implement the dependency graph: circular or non-circular:

(1) In circular dependency graphs, early transactions are allowed to rely on latter transactions in the same block. For a cyclic dependency with three transactions $tx_1$, $tx_2$ and $tx_3$, if transaction $tx_1$ depends on $tx_2$ and $tx_2$ depends on $tx_3$ and $tx_3$ depends on $tx_1$, then all three transactions must be committed or aborted atomically. A transaction cannot be executed until the dependency graphs have been fully constructed.

(2) In non-circular dependency graphs, a transaction $tx$ only depends on previous transactions. Therefore, a node can start to execute the transaction immediately after previous transactions $tx$ depends on have been executed, without waiting for latter transactions. With respect to performance, the non-circular setting is apparently superior. One side effect of this option is that transactions rely on latter ones may be rejected. However, this seldom happens and can be easily solved by client re-transmission.

## 4.4 Consensus

The consensus protocol ensures that each node holds the same sequence of transactions. BIDL's block consensus is pluggable. The consensus module can support generic BFT consensus protocols, and BIDL's codebase integrated three BFT consensus protocols, including BFT-SMaRt [29], SBFT [53],and Zyzzyva [61].

Compared to integrating BFT consensus protocols in existing permissioned blockchains' workflows (e.g., HLF's), integrating these protocols within BIDL's parallel workflow has three benefits. First, since BIDL has exploited the network ordering mechanisms to distribute transactions at its shepherded sequencer, BIDL nodes only need to perform a light-weight BFT consensus on transaction hashes (§3.2) instead of performing a BFT on the transactions' execution results, leading to much higher consensus performance in BIDL (§6.2). Second, BIDL makes the BFT consensus shepherd the sequencer and prevents the malicious sequencer from affecting BIDL's safety (§4.6). As discussed in §1, a malicious node may affect both the safety and the liveness of any workflows by broadcasting inconsistent (e.g., incorrectly ordered) transactions to different nodes. With the commit phase of BIDL's workflow, all nodes only commit transactions when they hold the same sequence of transactions. Therefore, the malicious sequencer can at most affect BIDL's liveness, but not safety. Third, even when the malicious sequencer causes inconsistent sequences of transactions on some nodes and makes these nodes fail to commit some speculative transactions (i.e., these nodes have to re-execute the transactions and try to commit later, see **Step 6**), BIDL re-selects a new sequencer (§4.5.1) upon detecting the malicious behaviors of the current sequencer.

In this section, we take BFT-SMaRt [29], a widely used BFT protocol in HLF practice, as an example to explain how to integrate BFT consensus protocols with BIDL.

**Step 5-1: Propose.** After receiving enough transactions for a block from the sequencer or timeout elapsed, the leader $n_L$ can start to perform consensus on and transaction hashes by sending a $\langle \text{PROPOSE}, v, b, h, \mathcal{H}, n_L \rangle_{\vec{\mu}}$ message to all nodes , where $v$ is the current view number, $b$ is the block number, $h$ is the block hash, $\mathcal{H}$ is a set indicates the hashes of included transactions. When running within BIDL, the consensus leader of BFT-SMaRt only needs to include transaction hashes rather than the entire transactions in the proposed block.

**Step 5-2: Write.** Following receipt of a $\langle \text{PROPOSE}, v, b, h, \mathcal{H}, n_L \rangle_{\vec{\mu}}$ message from the leader $n_L$, node $n_i$ checks every element of $\mathcal{H}$. If any transaction digests are different with $n_i$'s local transactions', $n_i$ requests the payloads of inconsistent transactions from $n_L$. If node $n_i$ has received all transactions indicated in the PROPOSE message, $n_i$ sends a $\langle \text{WRITE}, v, b, h, n_i \rangle_{\vec{\mu}}$ to all other nodes.

In addition, nodes shepherd the sequencer and detect the malicious nodes according to the transactions from the sequencer and the leader proposed blocks. If a node judges the current sequencer as malicious, it will invoke a sequencer view change to select another one (§4.5.1). Nodes add malicious nodes with misbehaviors to a global denylist and ignore all transactions from malicious nodes in the denylist (§4.5.2).

**Step 5-3: Accept.** Upon receipt of $2f$ matching $\langle \text{WRITE}, v, b, h, n_i \rangle_{\vec{\mu}}$ messages (consistent with the PROPOSE message) from non-leader nodes , $n_i$ broadcasts a $\langle \text{ACCEPT}, v, b, h, n_i \rangle_{\vec{\mu}}$ to other nodes. This step is the same with the origin BFT-SMaRt.

**Step 5-4. Nodes assemble blocks.** Because the consensus leader does not propose the entire block, each node needs to assemble blocks on their own. Each node assembles a block tentatively as soon as it has received the PROPOSE message according to the transaction hashes in the PROPOSE message. If consensus fails, the assembled block will be discarded in step 6.

**Step 6. Nodes commit valid transactions.** After receipt of $2f + 1$ matching ACCEPT messages from distinct nodes, each node checks the related transactions in the committed block. If all related transactions in the committed block have the same hashes with the transactions executed in step 4 (§4.3), node $n_i$ commits related transactions in the block by committing the execution results to the state database and adds the block to the ledger. Otherwise, if any transaction hashes are different, the node re-executes all related transactions in the committed block. In this situation, the parallel workflow falls back to the sequential workflow (i.e., the transaction execution is performed after the block consensus). However, this can happen only if any nodes or the sequencer are malicious. We propose a denylist mechanism to reduce this re-execution in §4.5.

## 4.5 Misbehavior Punishment

The above protocol can ensure safety among nodes. However, a wide range of misbehaviors of the malicious sequencer and malicious nodes can greatly degrade BIDL's performance. In this section, we present a new denylist-based view change protocol to handle the performance degradations.

First, since BIDL framework has exploited the network ordering mechanisms to distribute transactions at the routing level, BIDL nodes only need to perform a light-weight BFT consensus on transaction hashes, which leads to high consensus performance. Second, the BFT consensus shepherds the behaviors of the sequencer, and re-selects a new sequencer upon detecting the malicious behaviors of the current sequencer. The shepherded sequencer enables BIDL's novel parallel workflow where nodes can speculatively execute transactions according to the sequence numbers issued by the shepherd sequencer. Third, the BFT consensus prevents the malicious sequencer from affecting BIDL's safety. As discussed in §1, a malicious node can affect both the safety and the liveness of BIDL by broadcasting inconsistent transactions to different nodes. With the light-weight BFT consensus, all nodes only commit transactions when they hold the same sequence of transactions. Thus the malicious sequencer can only affect the system's liveness, but not safety.

*4.5.1 Shepherding the Sequencer.* BIDL has a default sequencer initially. If the sequencer is malicious, it can send inconsistent transactions to different nodes, drop or re-order transactions, and introduce gaps in the sequence number space, etc. All these misbehaviors can cause severe performance degradations. Rather than design specific mechanisms to handle different misbehaviors of the sequencer, we present a sequencer view change protocol for BIDL to replace the faulty sequencer with another one.

There exist several studies that aim to detect the performance degradations of the BFT consensus [36, 40], and perform view changes upon detecting performance degradations. Each node monitors the consensus throughput, and slowly raises the acceptable throughput level. If a node judges the consensus's performance is insufficient (e.g., lower than 70% throughput of the previous view), then the node initiates a sequencer view change.

We adopt this performance anomalies detection method in these two studies, but we differ with them in two aspects. First, BIDL performs a denylist-based sequencer view change protocol to proactively shepherd the sequencer, and select a new sequencer when performance degrades. Second, in a financial trading scenario, malicious member nodes and the sequencer may collude. Our protocol also proactively detects the malicious nodes and kicks them out of the permissioned blockchain's membership. Overall, BIDL is compatible with permissioned blockchain's BFT model and achieve high performance (§4.6).

We first present BIDL's sequencer view change protocol at a high level. Because the misbehavior of the malicious sequencers and malicious nodes is hard to distinguish in BFT model, BIDL nodes start from the performance metric. If the consensus throughput downgrades significantly, node $n_i$ suspects the shepherded sequencer as malicious and initiates a sequencer view change and stops processing transactions by sending a $\langle \textsc{view-change}, v+1, b, C, \mathcal{B}_L, i\rangle_{\delta_i}$ message to all nodes as well as the network controller (i.e., the network controller of SDN, see §5), where $v+1$ is the next view number, $b$ is the sequence number of the latest confirmed block known to $n_i$, $C$ is the set of $2f+1$ valid checkpoint messages, $\mathcal{B}_L$ is $n_i$'s local denylist of malicious nodes. If node $n_i$ judges $n_j$ to be malicious, $n_i$ adds $n_j$ to $\mathcal{B}_L$. We will describe how nodes maintain their local denylists in §4.5.2.

If the network controller receives $2f+1$ valid VIEW-CHANGE messages from different nodes for view $v+1$, this means $2f+1$ nodes agree that some problems occurred in view $v$. The network controller will start a new view. To start a new view, the network controller selects a new sequencer by updating the routing rules in switches, and broadcasts a $\langle \textsc{new-view}, v+1, \mathcal{V}, \mathcal{B}_G^{(v+1)}\rangle_{\delta_{nc}}$ message to all replicas, where $\mathcal{V}$ is the set of valid VIEW-CHANGE messages, $\mathcal{B}_G^{(v+1)}$ is the global denylist for view $v+1$. If a node appears for $2f+1$ times in the $\mathcal{B}_L$ of different VIEW-CHANGE messages, the network controller adds this node to $\mathcal{B}_G^{(v+1)}$. When a node is in $\mathcal{B}_G^{(v+1)}$, it cannot be the consensus leader, and transactions related to this node will be dropped by all nodes.

If the consensus performance does not drop significantly, even through a malicious sequencer can misbehave by drop/reorder a small portion of transactions, this does not affect the safety and liveness of the entire workflow (§4.6). Therefore, with this performance metric, we can effectively prevent the malicious sequencer from downgrades the system's performance.

#### 4.5.2 *Maintaining a Denylist of Malicious Nodes.*

For high performance in BIDL, the sequencer does not sign on transactions (§1), this also leaves the risk for malicious nodes to forge transactions from the sequencer. We now present how BIDL eliminates this risk.

By distributing forged transactions to the other nodes, a malicious node can cause conflicts in the other nodes' sequence number spaces (i.e., receive different transactions with the same sequence number). The problem can be even more severe when the malicious node is the consensus leader. A consensus leader can delay proposing, corrupt, and send inconsistent propose messages to different consensus nodes, etc. We propose a denylist mechanism for punishing the misbehaviors of malicious nodes.

Each node detects misbehaviors of other nodes from the received transactions, and adds misbehaved nodes into a locally maintained denylist $\mathcal{B}_L$, which will be merged into the global denylist $\mathcal{B}_G$ during the next view change (§4.5.1). However, sequence number conflicts of received transactions or performance degradations cannot only caused by malicious nodes but also by the malicious sequencer. How to distinguish which role is actually malicious becomes a subtle challenge.

We tackle this challenge by monitoring nodes' behavior across several sequencer view changes. The sequencer view change (§4.5.1) replaces the current sequencer with another one. Intuitively, if a node keeps misbehaving across several sequencer view changes, it has a high probability to be malicious.

Based on this intuition, we propose the following two-phase denylist mechanism. (1) Phase 1: If node $n_i$ detects another node $n_j$'s misbehavior for the first time, $n_i$ **suspects** $n_j$ as malicious, and continuously monitors $n_j$'s behavior. (2) Phase 2: If $n_j$ keeps being malicious across $s$ consecutive sequencer view changes, $n_i$ moves $n_j$ to the local denylist $\mathcal{B}_L$. In BIDL, we set $s=1$. A non-faulty node $n_i$ considers the following misbehaviors:

(1) if node $n_i$ receives two transactions with the same sequence number, $n_i$ will suspect related nodes of these two transactions as malicious;

(2) if any transaction hashes in a Propose message from the consensus leader is inconsistent with that in node $n_i$'s local storage, $n_i$ suspects related nodes of the transaction as malicious;
(3) if $n_i$ judges the current consensus throughput is not sufficient, it suspects the current consensus leader to be malicious.

Each node sets the initial value of its local denylist $\mathcal{B}_L$ as the view's global denylist $\mathcal{B}_G$ after sequencer view changes and maintains $\mathcal{B}_L$ continuously before the next sequencer view change. A malicious node can stay in $\mathcal{B}_L$ for at most $b_{BL}$ blocks. The size of $\mathcal{B}_L$ is no more than $f$, which is the maximum number of faulty nodes. If the list is full, to insert a new node, the first node added to the queue will be removed.

The two-phase denylist mechanism incurs low overhead for two reasons. First, it has low computation and communication overhead. The misbehavior detection is based on the performance monitoring data (i.e. consensus throughput) as well as the intermediate results of the transaction verification (§4.2), no extra computation or communication is incurred. Second, it has a low storage overhead. Each node is only required to maintain three lists: a local list of suspected nodes (Phase 1), a local denylist $\mathcal{B}_L$, and a global denylist $\mathcal{B}_G$. The number of records in each of the three lists is less than $f$, and each position only contains the meta-data (e.g., public key) of a malicious node.

Noted that any sequencer in BFT model will cause both safety and liveness issue, e.g., However, because of Bidl's new paralle workflow (Figure), sequencer from both safety liveness to only liveness, and we propose a protocol to solve the liveness issue.

In summary, Bidl provides both safety and reasonable liveness, even with malicious sequencers and malicious nodes. **Safety:** Although a malicious sequencer can send invalid or inconsistent transactions to nodes, Bidl invokes a standard BFT consensus protocol (§4.4) to ensures that all non-faulty nodes commit the same sequence of transactions and only valid transactions are committed to the blockchain. Moreover, the consensus protocol is light-weight, because nodes only need to achieve consensus on transaction hashes. Overall, the malicious sequencer or malicious nodes can only cause performance degradations (§4.6), and we propose a denylist-based sequencer view change protocol to solve the performance degradations (§4.5). **Liveness:** Many misbehaviors of a malicious sequencer can degrade the system performance. Bidl's view change protocol (§4.5.1) replaces the malicious sequencer with a new one upon detecting performance degradations. A malicious node can also harm the system's liveness by broadcasting transactions with forged sequence numbers, which introduces sequence number conflicts in other nodes' received transactions. In consequence, nodes need to request for retransmission of the transaction payloads, leading to performance degradations. Bidl's denylist mechanism (§4.5.2) adds the malicious nodes to a global denylist and discards all transactions from the malicious nodes.

## 4.6 Proof Sketch of Correctness

In this section, we proof that Bidl's workflow is compatible with the permissioned blockchains's BFT safety guarantee and can provide reasonable liveness even with the malicious sequencer and malicious nodes.

**Safety:** The safety of Bidl inherits from the BFT block consensus protocol. The BFT consensus protocol ensures that all non-faulty nodes commit the same totally ordered sequence of transactions [36]. Upon receiving transactions from the sequencer, Bidl nodes speculatively execute transactions in parallel with the standard BFT consensus, and only commit the execution results according to the consensus results after the consensus is finished. Therefore, all non-faulty nodes can commit the same sequence of transactions to the blockchain.

Bidl's denylist-based sequencer view change protocol (§4.5) does not affect the safety guarantee. This is because the protocol only prevents malicious nodes being the consensus leader and ignores the transactions from the malicious nodes.

**Liveness:** We first proof a key aspect of BIDL's liveness: a malicious node that is constantly sending forged transactions and causing performance degradations will eventually be added to the local denylists by at least $2f + 1$ nodes. The proof consists of two steps:

Supposing a malicious node $m$ is constantly sending forged transactions to a subset of nodes $Q$.

(1) Non-faulty nodes in $Q$ will add the malicious node to their local denylists. Non-faulty nodes in $Q$ can detect these forged transactions according to the conflicts in the sequence number space. Because the malicious node is one of the related nodes of the conflict transactions, it will be added to the denylists of non-faulty nodes in $Q$.

(2) Non-faulty nodes not in $Q$ will add the malicious node to their local denylists. If a node $n_i \in Q$ becomes the consensus leader, it selects transactions from its memory pool to propose for consensus. There are two cases: (1) If $n_i$ proposes the malicious node's forged transactions in consensus, nodes not in $Q$ judge that the forged transactions included in the proposed block are different from the transactions in their local storage. They request re-transmissions of the entire block (causing performance degradations) and add the malicious node to their local denylists. (2) If $n_i$ does not propose the forged transactions, there will be no block re-transmission: performance of consensus is not affected.

In summary, if a malicious node sends forged transactions and causes re-transmissions, all non-faulty nodes (in $Q$ or not in $Q$) will add the malicious node to their local denylists. Because the number of non-faulty nodes is more than $2f + 1$, the malicious node will be added to the local denylists by at least $2f + 1$ nodes. If a malicious node is in the denylist of $2f + 1$ node, the malicious node will be added to the global denylist $\mathcal{B}_G$ after the next sequencer view change, all the node's transactions are ignored, and it cannot be the consensus leader, and the malicious node cannot force frequent view changes by sending forged transactions or delaying PROPOSE messages.

Overall, BIDL has reasonable liveness because malicious nodes will be denylisted and cannot degrade BIDL's performance. The evaluation shows that BIDL can handle the performance degradation caused by malicious nodes and sequencers (§6.5).

## 5 IMPLEMENTATION

The network ordering is implemented using SDN [62] in datacenters. SDN moves the network's control logic from low-level hardware-based network devices to high-level software-based controllers running on conventional end-hosts. Components that make up an SDN network are separated into three layers: application layer (nodes of BIDL), control layer (network controllers), and infrastructure layer (switches or routers).

To resist attacks on communication links, the secure sockets layer (SSL) is used to create encrypted channels between servers that enables authentication and encryption for all communications. Equipment at different levels can verify the management server which they are communicating with is trusted and has permission to modify their states.

To resist Byzantine attacks on controllers, BIDL uses a Byzantine-replicated controller group [46, 69, 80]. Each switch is controlled by multiple controllers. The controller group ensures that switches' routing tables are correctly updated even some compromised controllers issue false instructions.

The sequencer implemented with the programmable switch [31] can provide the best performance. Because we don't have a programmable switch hardware in our cluster, we built a software-based prototype on conventional end-host using Intel's Data Plane Development Kit [7, 77] to simulate the programmable switch. Prior work [71] shows that this software-based method achieves similar performance to hardware programmable switches. For 1k bytes transactions, this DPDK implementation adds about 20 μs to the transfer delay.

## 6 EVALUATION

In this section, we experimentally evaluate our prototype implementation of BIDL. We ran all experiments in a cluster with 10 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 is about 0.2ms.

We compared BIDL with HLF 1.3 [4], an open-sourced and most popular permissioned blockchain framework. To the best of our knowledge, HLF reports the highest performance results among all permissioned blockchain frameworks. HLF is used by various cloud providers (e.g., IBM blockchain [14] and AWS blockchain [13]), companies [19], and governments [10]. Nodes use LevelDB as the state database. To prevent the performance bottleneck caused by the block storage, each node stores blocks in-memory rather than on disk [54]. Although Libra [18], a permissioned blockchain framework, releases the code of its consensus protocol, its workflow is unknown and not open-sourced, so we did not evaluated Libra with BIDL. Since is Libra is designed for both Internet and AS networks, we believe its workflow is sequential. Quorum [9], another permissioned blockchain framework, is built on Ethereum [93] and achieves less than 1K txns/s, so this section compares the fastest permissioned blockchain framework (HLF, about 5K txns/s) with BIDL. Quorum evaluated the same targeted trading and payment applications (§1) of our paper. These applications contain only private transactions (§4.1) that involve a small number of related participants. For fair comparison, we made both BIDL and HLF aware of private transactions: each node only validates and executes related private transactions and all public transactions.

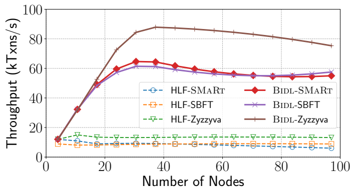
We ran BIDL and HLF with three Byzantine consensus protocols, including BFT-SMaRt [29], SBFT [53], and Zyzzyva [61]. BFT-SMaRt and Zyzzyva are evaluated with $n = 3f + 1$ consensus nodes, where $f$ ranges from 1 to 32. We implemented the batch mode optimization [61] of Zyzzyva and select a non-leader node to collect consensus nodes' responses to the clients and to send the *commit* messages for each block. We ran SBFT with $n = 3f + 2c + 1$ nodes, where $f$ ranges from 1 to 32. SBFT contains $c + 1$ collectors, and we set $c = 0$ in all experiments. If not otherwise specified, the default transaction size is 1 kbytes and the default batch size of consensus protocols is 500, which are typical for HLF [26, 83]. The primary questions we want to evaluate are as follows:

Because participants are mutually untrusted, the block consensus is performed among all participants. For BIDL, each participant has one node that performs both the transaction executions and the block consensus. For HLF, each participant has one consensus node (only performs the block consensus) and one normal node (only performs the transaction execution).
§6.1: How is BIDL's performance compared to other permissioned blockchains?
§6.2: How BIDL's network ordering can improve the performance of existing BFT protocols?
§6.3: How BIDL's parallel workflow affects a transaction's confirm latency?
§6.4: How robust are BIDL and other permissioned blockchains to packet losses?
§6.5: How robust is BIDL to faulty sequencers and nodes?

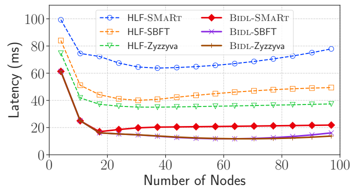
### 6.1 Compare BIDL's End-to-end Performance with HLF

In this experiment, we evaluate the end-to-end performance of BIDL and HLF with different number of nodes $n$. We start from 4 nodes and increase the number of nodes to 97. In BIDL, all the $n$ nodes participate in consensus. For HLF, there are $n$ consensus nodes and $n$ normal nodes.

Figure 5a shows the end-to-end throughput of BIDL and HLF. We evaluated BIDL and HLF with three consensus protocols: BFT-SMaRt, SBFT, and Zyzzyva. BIDL shows an average of 5.1× higher throughput than HLF. This improvement is mainly caused by two reasons. First, BIDL's network ordering reduces the workload of the consensus leader, making different BFT consensus protocols more efficient when running in BIDL (§4.4). BIDL pushes the duty of disseminating transaction payloads to the highly-efficient network-level switches. For each block, the consensus leader



(a) End-to-end throughput of HLF and Bidl.



(b) End-to-end latency of HLF and Bidl.

Fig. 5. End-to-end performance comparison of HLF and Bidl.

Table 1. End-to-end latency breakdown of HLF-Smart and BIDL-Smart.

| **HLF-Smart** | | | | | | | **Bidl-Smart** | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Number of Nodes** | **4** | **7** | **13** | **25** | **49** | **97** | **Number of Nodes** | **4** | **7** | **13** | **25** | **49** | **97** |
| Endorse (ms) | 9.15 | 8.55 | 7.89 | 7.47 | 7.19 | 5.47 | Consensus (ms) | 10.31 | 12.07 | 10.02 | 12.83 | 14.88 | 16.37 |
| Consensus (ms) | 38.63 | 42.38 | 50.58 | 51.26 | 53.25 | 68.48 | Ver & Exec (ms) | 56.26 | 33.37 | 14.71 | 9.24 | 5.64 | 5.66 |
| Validate (ms) | 51.49 | 29.63 | 15.78 | 8.12 | 4.19 | 3.92 | Commit (ms) | 5.65 | 5.84 | 5.50 | 4.80 | 4.61 | 4.89 |
| End-to-end (ms) | 99.27 | 80.56 | 74.25 | 66.85 | 64.63 | 77.87 | End-to-end (ms) | 61.91 | 39.21 | 20.21 | 17.63 | 19.49 | 21.26 |

delivers the transaction hashes to all nodes. However, in HLF, the consensus leader transfers the entire block containing transaction payloads as well as the endorsements (execution results) [26]. A transaction hash is only tens of bytes in length (e.g., a SHA-256 hash is 32 bytes long), much smaller than the transaction payloads or endorsements (e.g., several kilobytes in length [2]). When a BFT consensus protocol runs in Bidl, the consensus leader takes much lower workload, which leads to better throughput. Second, the commitment phase (step 5 in §4.4) of Bidl is more light-weight. Upon receiving a new block, an HLF node needs to verify all transactions in this block, which greatly limits HLF's peak throughput. A Bidl node, however, verifies only related transactions.

Bidl's end-to-end throughput first increases with the number of consensus nodes, then decreases as the number of nodes continues to increase. This is because when there are fewer nodes, each node needs to verify and execute more transactions. If a block contains 500 transactions, and there are four nodes, each node needs to verify and execute 125 transactions. Transaction execution is the major performance bottleneck. With the increasing number of nodes, the amount of transactions processed by each node decreases, leading to higher throughput. When the number of nodes continues to increase, block consensus becomes the major performance bottleneck.

Figure 5b is the end-to-end latency of a single transaction in Bidl and HLF. To make a fair comparison, we eliminated the latency caused by client-server communications. When there are four nodes, Bidl and HLF show a similar end-to-end latency. With the increasing number of nodes, Bidl can achieve 62.2% lower latency than HLF on average. To understand why Bidl has better end-to-end latency than HLF, we recorded the time taken for each step of HLF and Bidl, as shown in Table 1. After analyzing their transaction flows, we identify two reasons that may lead to Bidl's low latency. First, highly efficient consensus: time consuming of Bidl's consensus is much lower than HLF's consensus. Second, Bidl's parallel workflow reduces reduce the end-to-end latency by paralleling the block consensus and the transaction execution (§6.3).

## 6.2 Effectiveness of Bidl's Network Ordering on BFTs

Figure 6 shows the performance gains obtained by introducing network ordering to three BFT protocols. With network ordering, the performance and scalability of three BFT protocols is improved. When running BFT-SMaRt with four consensus nodes, BFT-SMaRt with the network ordering achieves 5.1× better throughput. When the number of nodes increases to 97, BFT-SMaRt with network ordering achieves 9.2× better throughput.

To understand Bidl's consensus latency, we recorded the time for each phase of consensus. Figure 7 shows the latency breakdown of BFT-SMaRt running with or without Bidl. Because BFT-SMaRt (§2.1) runs a three-phase commit protocol, its latency is divided into three parts. As shown in Figure 7b, when BFT-SMaRt was run in one datacenter (but it runs alone without involving Bidl), the *propose* phase of BFT-SMaRt is the dominant factor of its consensus latency, because the leader needs to broadcast the blocks containing the whole transactions, while the *write* and *accept* phase only transmit the hashes. Message transferred in *propose* is proportional to the number of nodes, which leads to poor scalability. When BFT-SMaRt is deployed with Bidl (Figure 7a), the duty of distributing transaction payloads is offloaded to the highly efficient network switches, so the amount of data transferred in *Propose* is greatly decreased. Evaluation results of SBFT and Zyzzyva show a similar result: if these BFT consensus protocols are deployed with Bidl, their performance improves much.

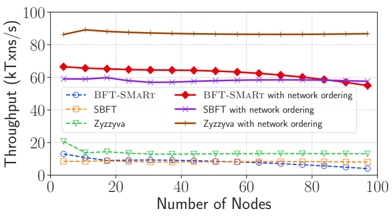
## 6.3 Effectiveness of Bidl's Parallel Workflow

To understand how Bidl's parallel workflow affects transactions' end-to-end latency, we implemented a prototype of Bidl using the sequential workflow. Each node starts to execute transactions after the block consensus has completed. Figure 8 shows the latency comparison of the sequential and the parallel workflow. Since BFT-SMaRt is recognized as the most widely used BFT consensus protocol [34], we run BFT-SMaRt for block consensus by default. Evaluation results show that Bidl's parallel workflow achieves up to 32.3% lower latency.

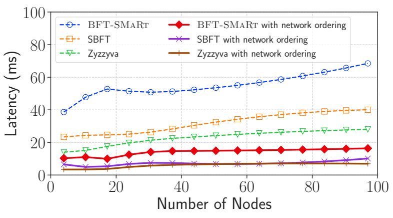
## 6.4 Robustness of Bidl and HLF on Packet Losses

To investigate Bidl and HLF's performance in asynchronous networks, we run 25 normal nodes (to eliminate the bottleneck associated with the transaction execution) and four (f=1) consensus nodes for both systems. We set packet loss rates ranging from 0% to 10% for all nodes (both normal nodes and consensus nodes). The performance of Bidl and HLF is shown in Figure 9. It can be observed that the performance of both Bidl and HLF deteriorates with the increasing packet loss rate.

Bidl can achieve much better performance than HLF when the packet loss rate is low (<6%). With 2% packet losses, Bidl achieves 3.4× higher throughput and 64.4% lower latency than HLF. This is because, even with packet losses, the consensus leader in Bidl still transmits less data

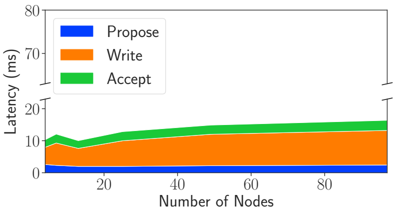


(a) Consensus throughput of BFTs.

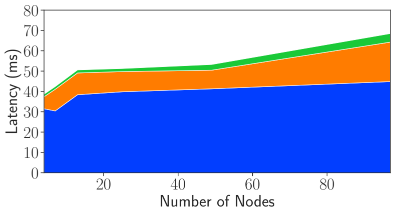


(b) Consensus latency of BFTs.

Fig. 6. Effectiveness of Bidl's network ordering on the performance of BFTs.



(a) BFT-SMaRt with network ordering.



(b) BFT-SMaRt.

Fig. 7. Latency breakdown of BFT-SMaRt with or without network ordering.

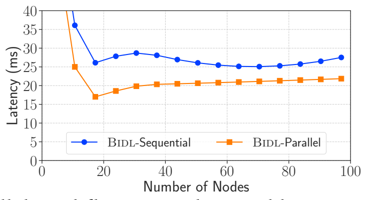


Fig. 8. Effectiveness of BIDL's parallel workflow on end-to-end latency. Both run BFT-SMaRt for consensus.

than in HLF. BIDL leverages the sequencer to disseminate transactions, and the consensus leader proposes only transaction hashes. When a consensus node detects packet losses (i.e., hashes of transactions from the sequencer are inconsistent with transaction hashes in leader proposed blocks), the node requests for retransmissions of lost transactions from the consensus leader. Only a small subset of transactions is retransmitted. However, in HLF, the consensus leader needs to transfer all transactions to all consensus nodes.

Although BIDL's performance is constantly better than HLF with increasing packet loss rates, the performance gap between BIDL and HLF gets smaller when the packet loss rate becomes large (>6%). This happens because the consensus leader needs to retransmit more lost transactions to other consensus nodes, and the execution module needs to re-execute all transactions if the speculatively executed transactions are inconsistent with the transactions in blocks from the consensus module.

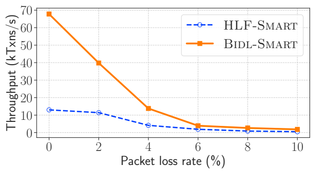
Most common network environments can achieve packet loss rates lower than 6%. For a single datacenter, the packet loss rate is about $10^{-5}$ [55]. For the cross datacenter deployment, multiple datacenters at different geolocations are connected with dedicated, high-speed and long haul fibers, thus the inter-datacenter communications can also achieve a low packet loss rate of no more than 0.1% [55, 99]. For the public Internet, the normal packet loss rate is less than 5% [21, 30].

## 6.5 Robustness of BIDL on Malicious Nodes and Sequencer

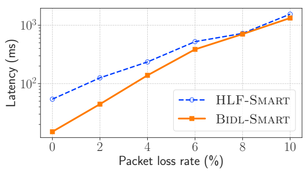
BIDL's network ordering uses the sequencer to order and distribute client transactions to all nodes. A malicious sequencer can send inconsistent transactions to different nodes, drop/reorder transactions, etc. Because the sequencer does not sign on sequence numbers, a malicious node can forge transactions from the sequencer and distribute them to other nodes. These misbehaviors may cause block retransmissions (step 5-2 in §4.4), which greatly degrade BIDL's performance.

BIDL uses the denylist-based sequencer view change to handle the malicious sequencer and malicious nodes (§4.5). Upon detecting performance degradations, nodes initiate a sequencer view change, During the view change, malicious nodes are denylisted, and the network controller modifies the routing rules in the network switches which route transactions to a new sequencer.

We measured the throughput of BIDL during sequencer view changes (Figure 10). We triggered sequencer view changes by letting a malicious node keep broadcasting forged transactions to all nodes. Due to block re-transmissions caused by these forged transactions, the throughput decreases (Point 1). Nodes detect this performance degradation. They suspect that either the sequencer or the malicious node (related node of the conflict transactions) may be malicious and initiate a sequencer view change (Point 2). During a sequencer view change, the throughput of BIDL drops to zero and



(a) End-to-end throughput.



(b) End-to-end latency (y-axis is log-scaled).

Fig. 9. Performance of BIDL and HLF with packet losses.

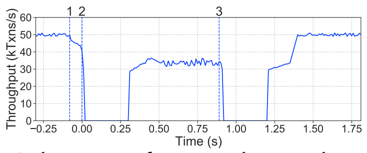


Fig. 10. Robustness of Bidl under a malicious node.

takes approximately 300 ms to resume normal operations. Most of this delay is caused by the route update. After the first view change, the malicious node keeps broadcasting forged transactions. Nodes detect this cross view change misbehavior and add the malicious node to the denylist. Due to the performance degradation caused by block re-transmissions, another view change is initiated (Point 3). After this view change, the malicious node is added to the global denylist. Nodes drop transactions from the malicious node; the system again achieves the peak throughput.

## 6.6 Discussion

Bidl has two limitations. First, Bidl's parallel workflow is designed for deterministic smart contracts: with the same input, a deterministic smart contract's execution results on different nodes are consistent. HLF's sequential workflow supports non-deterministic smart contracts by conducting BFT consensus on the contracts' execution results. In modern performance-sensitive and security-critical blockchain applications (e.g., trading systems [85, 86]), non-determinism in smart contracts may cause both security issues and performance degradations [81, 100]. Therefore, most existing blockchain systems (e.g., Ethereum [93], Quorum [9], and Corda [3]) and Bidl support only deterministic smart contracts.

Second, compared to existing permissioned blockchain frameworks' sequential workflows, Bidl's parallel workflow can achieve both much better throughput and latency for private transactions (e.g., related to two nodes, see §4.1). For public transactions, any permissioned blockchains' workflows, including Bidl's and HLF's, will be bounded by the transaction execution module (Figure 1), so the throughput of all the workflows will trivially equal to the throughput of the transaction execution module. In this case, the throughput of Bidl's workflow and HLF's workflow will be the same. Nevertheless, the end-to-end latency of Bidl's workflow can still be about 60ms lower than HLF's, because Bidl's workflow masks the consensus module's latency, while HLF's consensus latency is about 60ms (Figure 6).

Overall, Bidl is complementary to HLF: when all smart contracts are deterministic and the system is deployed in an AS network (e.g., datacenters), Bidl can achieve better performance than HLF for both private and public transactions. Therefore, Bidl is appropriate for enterprise applications which require secure, high-performance transaction processing in an AS network. HLF is designed for both AS networks and asynchronous networks and supports non-determinism in smart contracts.

## 7 CONCLUSION

We present Bidl, the first blockchain-powered distributed ledger optimized for AS networks. Bidl achieves high performance and scalability by network-application co-design. In Bidl, the network layer is responsible for ordering and distributing transactions. The network ordering enables a new parallel workflow and greatly reduces the overhead of consensus. Evaluation shows that Bidl achieves up to 9.1× better throughput and 73.7% lower latency compared to Hyperledger Fabric. 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's code and evaluation results are released on github.com/sigmetrics21/bidl.

## REFERENCES

[1] 2013. 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

[2] 2018. Byzantine fault-tolerant ordering service for Hyperledger Fabric. https://github.com/bft-smart/fabric-orderingservice/wiki

[3] 2018. Deterministic Contract Verification. https://www.corda.net/blog/deterministic-contract-verification

[4] 2018. Hyperledger Fabric 1.3. https://github.com/hyperledger/fabric/releases/tag/v1.3.0

[5] 2018. 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

[6] 2019. ASX is replacing CHESS with distributed ledger technology (DLT) developed by Digital Asset. https://www.asx.com.au/services/chess-replacement.htm

[7] 2019. DPDK: Home. https://www.dpdk.org

[8] 2019. Exchange Trading & Matching Technology System – Nasdaq. https://www.nasdaq.com/solutions/trading-and-matching-technology

[9] 2019. Quorum. https://github.com/jpmorganchase/quorum

[10] 2019. Singapore Exchange. https://www2.sgx.com

[11] 2019. Wireless Express Connect - Nasdaq. https://www.nasdaqtrader.com/content/Productsservices/trading/CoLo/ExpressConnectFS.pdf

[12] 2020. Alipay. https://intl.alipay.com/

[13] 2020. Amazon Managed Blockchain. https://aws.amazon.com/managed-blockchain/

[14] 2020. Blockchain Platform. https://cloud.ibm.com/catalog/services/blockchain-platform

[15] 2020. Blockchain Platform. https://www.oracle.com/application-development/cloud-services/blockchain-platform/

[16] 2020. InfiniBand Long-Reach and Long-Haul Systems. https://www.mellanox.com/products/long-reach?mtag=long_haul_systems_ov

[17] 2020. IP Multicast. https://en.wikipedia.org/wiki/IP_multicast

[18] 2020. Libra. https://libra.org

[19] 2020. MedicalChain. https://medicalchain.com/en/home/hyperledger/

[20] 2020. Microsoft Azure Blockchain. https://azure.microsoft.com/en-us/solutions/blockchain/

[21] 2020. Packet loss. https://en.wikipedia.org/wiki/Packet_loss

[22] 2020. The Ordering Service. https://hyperledger-fabric.readthedocs.io/en/release-2.2/orderer/ordering_service.html

[23] 2020. VISA. https://usa.visa.com/run-your-business/small-business-tools/retail.html

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

[25] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review*, Vol. 38. ACM, 63–74.

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

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

[28] Johannes Behl, Tobias Distler, and Rüdiger Kapitza. 2017. Hybrids on steroids: SGX-based high performance BFT. In *Proceedings of the Twelfth European Conference on Computer Systems*. ACM, 222–237.

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

[30] Michael S Borella, Debbie Swider, Suleyman Uludag, and Gregory B Brewster. 1998. Internet packet loss: Measurement and implications for end-to-end QoS. In *Proceedings of the 1998 ICPP Workshop on Architectural and OS Support for Multimedia Applications Flexible Communication Systems. Wireless Networks and Mobile Computing (Cat. No. 98EX206)*. IEEE, 3–12.

[31] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. 2014. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 87–95.

[32] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 99–110.

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

[34] Christian Cachin and Marko Vukolić. 2017. Blockchain consensus protocols in the wild. *arXiv preprint arXiv:1707.01873* (2017).

[35] Yi Cao, Javad Nejati, Aruna Balasubramanian, and Anshul Gandhi. 2019. ECON: Modeling the network to improve application performance. In *Proceedings of the Internet Measurement Conference*. 365–378.

[36] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *OSDI*, Vol. 99. 173–186.

[37] Yang Chen, Jie Wu, and Bo Ji. 2018. Virtual network function deployment in tree-structured networks. In *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. IEEE, 132–142.

[38] Sean Choi, Boris Burkov, Alex Eckert, Tian Fang, Saman Kazemkhani, Rob Sherwood, Ying Zhang, and Hongyi Zeng. 2018. FBOSS: building switch software at scale. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 342–356.

[39] Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. 2007. Attested append-only memory: Making adversaries stick to their word. In *ACM SIGOPS Operating Systems Review*, Vol. 41. ACM, 189–204.

[40] Allen Clement, Edmund L Wong, Lorenzo Alvisi, Michael Dahlin, and Mirco Marchetti. 2009. Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults.. In *NSDI*, Vol. 9. 153–168.

[41] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016, 086 (2016), 1–118.

[42] Huynh Tu Dang, Pietro Bressana, Han Wang, Ki Suh Lee, Hakim Weatherspoon, Marco Canini, Fernando Pedone, and Robert Soulé. 2016. Network hardware-accelerated consensus. *arXiv preprint arXiv:1605.05619* (2016).

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

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

[45] John R Douceur. 2002. The sybil attack. In *International workshop on peer-to-peer systems*. Springer, 251–260.

[46] Karim ElDefrawy and Tyler Kaczmarek. 2016. Byzantine fault tolerant software-defined networking (SDN) controllers. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 2. IEEE, 208–213.

[47] Minghong Fang and Jia Liu. 2020. Toward Low-Cost and Stable Blockchain Networks. *arXiv preprint arXiv:2002.08027* (2020).

[48] Andrew Ferraiuolo, Andrew Baumann, Chris Hawblitzel, and Bryan Parno. 2017. Komodo: Using verification to disentangle secure-enclave hardware from software. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 287–305.

[49] Chenfei Gao, Vahid Rajabian-Schwart, Weiyi Zhang, Guoliang Xue, and Jian Tang. 2018. How would you like your packets delivered? An SDN-enabled open platform for QoS routing. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.

[50] Chenfei Gao, Weiyi Zhang, Jian Tang, Rakesh K Sinha, and Kostas N Oikonomou. 2017. DEMUR: Dependable Multipath Routing in Software Defined Networking for ISP Backbone. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 1–6.

[51] Kristen Gardner, Samuel Zbarsky, Sherwin Doroudi, Mor Harchol-Balter, and Esa Hyytia. 2015. Reducing latency via redundant requests: Exact analysis. *ACM SIGMETRICS Performance Evaluation Review* 43, 1 (2015), 347–360.

[52] Mark Glavind, Niels Christensen, Jiri Srba, and Stefan Schmid. 2020. Online Dynamic B-Matching with Applications to Reconfigurable Datacenter Networks. In *Proc. 38th International Symposium on Computer Performance, Modeling, Measurements and Evaluation (PERFORMANCE)*.

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

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

[55] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, et al. 2015. 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*. 139–152.

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

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

[58] Yuchen Jin, Sundararajan R, Ganesh Ananthanarayanan, Junchen Jiang, Venkat Padmanabhan, Manuel Schroder, Matt Calder, and Arvind Krishnamurthy. 2019. Zooming in on Wide-area Latencies to a Global Cloud Provider. In *ACM SIGCOMM*. https://www.microsoft.com/en-us/research/publication/zooming-in-on-wide-area-latencies-to-a-global-cloud-provider/

[59] Alun John. 2018. Hong Kong exchange turns to blockchain to open up Chinese shares. https://tinyurl.com/y2t8ofwb

[60] Rüdiger Kapitza, Johannes Behl, Christian Cachin, Tobias Distler, Simon Kuhnle, Seyed Vahid Mohammadi, Wolfgang Schröder-Preikschat, and Klaus Stengel. 2012. CheapBFT: resource-efficient byzantine fault tolerance. In *Proceedings of the 7th ACM european conference on Computer Systems*. ACM, 295–308.

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

[62] Diego Kreutz, Fernando Ramos, Paulo Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. 2014. Software-defined networking: A comprehensive survey. *arXiv preprint arXiv:1406.0440* (2014).

[63] Lucas Kuhring, Zsolt István, Alessandro Sorniotti, and Marko Vukolić. 2018. StreamChain: Rethinking Blockchain for Datacenters. *arXiv* (2018), arXiv–1808.

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

[65] Anh Le, Arash Saber Tehrani, Alexandros Dimakis, and Athina Markopoulou. 2016. Recovery of packet losses in wireless broadcast for real-time applications. *IEEE/ACM Transactions on Networking* 25, 2 (2016), 676–689.

[66] Charles E Leiserson. 1985. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE transactions on Computers* 100, 10 (1985), 892–901.

[67] Dave Levin, John R Douceur, Jacob R Lorch, and Thomas Moscibroda. 2009. TrInc: Small Trusted Hardware for Large Distributed Systems.. In *NSDI*, Vol. 9. 1–14.

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

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

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

[71] Jialin Li, Ellis Michael, Naveen Kr Sharma, Adriana Szekeres, and Dan RK Ports. 2016. Just Say NO to Paxos Overhead: Replacing Consensus with Network Ordering.. In *OSDI*. 467–483.

[72] Xueping Liang, Sachin Shetty, Deepak Tosh, Charles Kamhoua, Kevin Kwiat, and Laurent Njilla. 2017. 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)*. IEEE, 468–477.

[73] Jian Liu, Wenting Li, Ghassan O Karame, and N Asokan. 2018. Scalable byzantine consensus via hardware-assisted secret sharing. *IEEE Trans. Comput.* 68, 1 (2018), 139–151.

[74] Long Luo, Klaus-Tycho Foerster, Stefan Schmid, and Hongfang Yu. 2020. Deadline-Aware Multicast Transfers in Software-Defined Optical Wide-Area Networks. *IEEE Journal on Selected Areas in Communications* (2020).

[75] Ajay Mahimkar, Angela Chiu, Robert Doverspike, Mark D Feuer, Peter Magill, Emmanuil Mavrogiorgis, Jorge Pastor, Sheryl L Woodward, and Jennifer Yates. 2011. Bandwidth on demand for inter-data center communication. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*. 1–6.

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

[77] Gergely Pongrácz, László Molnár, and Zoltán Lajos Kis. 2013. Removing roadblocks from SDN: OpenFlow software switch performance on Intel DPDK. In *2013 Second European Workshop on Software Defined Networks*. IEEE, 62–67.

[78] Dan RK Ports, Jialin Li, Vincent Liu, Naveen Kr Sharma, and Arvind Krishnamurthy. 2015. Designing Distributed Systems Using Approximate Synchrony in Data Center Networks.. In *NSDI*. 43–57.

[79] Shahrooz Pouryousef, Lixin Gao, and Arun Venkataramani. 2020. Towards Logically Centralized Interdomain Routing. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*. 739–757.

[80] Ermin Sakic, Nemanja Deric, Endri Goshi, and Wolfgang Kellerer. 2019. P4BFT: Hardware-Accelerated Byzantine-Resilient Network Control Plane. *arXiv preprint arXiv:1905.04064* (2019).

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

[82] Peter Shen. 2019. Investing In A Blockchain Future At Singapore Exchange. https://tinyurl.com/y2fbknms

[83] Joao Sousa, Alysson Bessani, and Marko Vukolic. 2018. 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)*. IEEE, 51–58.

[84] Harish Sukhwani, Nan Wang, Kishor S Trivedi, and Andy Rindos. 2018. Performance Modeling of Hyperledger Fabric (Permissioned Blockchain Network). In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*. IEEE, 1–8.

[85] Alex Tapscott and Don Tapscott. 2017. How blockchain is changing finance. *Harvard Business Review* 1, 9 (2017), 2–5.

[86] Philip Treleaven, Richard Gendal Brown, and Danny Yang. 2017. Blockchain technology in finance. *Computer* 50, 9 (2017), 14–17.

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

[88] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung, and Paulo Verissimo. 2009. Minimal Byzantine fault tolerance: Algorithm and evaluation. (2009).

[89] Marko Vukolić. 2017. Rethinking permissioned blockchains. In *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*. ACM, 3–7.

[90] Muhammad Wajahat, Aditya Yele, Tyler Estro, Anshul Gandhi, and Erez Zadok. 2020. Analyzing the distribution fit for storage workload and Internet traffic traces. *Performance Evaluation* (2020), 102121.

[91] Ye Wang, Hao Wang, Ajay Mahimkar, Richard Alimi, Yin Zhang, Lili Qiu, and Yang Richard Yang. 2010. R3: resilient routing reconfiguration. In *Proceedings of the ACM SIGCOMM 2010 conference*. 291–302.

[92] Nick Webb. 2018. A fork in the blockchain: income tax and the bitcoin/bitcoin cash hard fork. *North Carolina Journal of Law & Technology* 19, 4 (2018), 283.

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

[94] Brent Xu, Dhruv Luthra, Zak Cole, and Nate Blakely. 2018. Eos: An architectural, performance, and economic analysis.

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

[96] Zhenjie Yang, Yong Cui, Xin Wang, Yadong Liu, Minming Li, Shihan Xiao, and Chuming Li. 2019. Cost-efficient scheduling of bulk transfers in inter-datacenter WANs. *IEEE/ACM Transactions on Networking* 27, 5 (2019), 1973–1986.

[97] Jian Yin, Jean-Philippe Martin, Arun Venkataramani, Lorenzo Alvisi, and Mike Dahlin. 2003. Separating agreement from execution for byzantine fault tolerant services. In *ACM SIGOPS Operating Systems Review*, Vol. 37. ACM, 253–267.

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

[99] Gaoxiong Zeng, Wei Bai, Ge Chen, Kai Chen, Dongsu Han, Yibo Zhu, and Lei Cui. 2019. Congestion Control for Cross-Datacenter Networks. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. IEEE, 1–12.

[100] Shenbin Zhang, Ence Zhou, Bingfeng Pi, Jun Sun, Kazuhiro Yamashita, and Yoshihide Nomura. 2019. A Solution for the Risk of Non-deterministic Transactions in Hyperledger Fabric. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 253–261.

[101] Danyang Zhuo, Monia Ghobadi, Ratul Mahajan, Klaus-Tycho Förster, Arvind Krishnamurthy, and Thomas Anderson. 2017. Understanding and mitigating packet corruption in data center networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 362–375.

[102] Jia Zou, Gong Su, Arun Iyengar, Yu Yuan, and Yi Ge. 2011. Design and Analysis of a Distributed Multi-leg Stock Trading System. In *2011 31st International Conference on Distributed Computing Systems*. IEEE, 13–24.