A Data Flow Framework with High Throughput and Low Latency for Permissioned Blockchains

Zhenxing Hu[‡§], Shengjie Guan[‡§], Wenbo Xu[§], Zhen Xiao[‡]
Jie Shi[§], Pengze Li[‡], Qiuyu Ding[‡§], Hui Ding[§] and Chao Zeng[§]
[‡]*School of Computer Science, Peking University, China*
[§]*Ant Group, China*
[‡]{*hzx,xiaozhen, lipengze*}*@pku.edu.cn*, [‡]{*guanshengjie, dingqiuyu*}*@stu.pku.edu.cn*
[§]{*xuwenbo.xwb, tonglin.dh, xiaoyou.zc*}*@antgroup.com*, [‡]*shijie.s@alibaba-inc.com*

***Abstract*—In permissioned blockchains, the bandwidth of consensus nodes is mainly consumed by transaction ordering and block distribution; hence, the allocation of consensus nodes' bandwidth makes a significant difference to the system throughput. Previous research focuses on the consensus layer and attempts to optimize consensus protocols to improve throughput, which, however, neglects the impact of data distribution on the throughput and transfers performance bottlenecks to the network layer. In fact, the overall throughput of permissioned blockchains is co-determined by data production in the consensus layer and data distribution in the network layer. This paper proposes a novel data flow framework composed of *Predis* and *Multi-Zone*. The former is a data production strategy for permissioned blockchains that employ leader-based BFT protocols and the latter, its corresponding network topology. *Predis* enables each consensus node to contribute its idle bandwidth for block content pre-distribution so that a much higher volume of transactions can be confirmed in one consensus round, significantly increasing consensus efficiency. *Multi-Zone* is a network topology to distribute blocks. It can regulate the bandwidth consumption of consensus nodes at a certain value during data distribution and effectively reduce block propagation latency. To test our framework, we implement *Predis* based on Hotstuff and PBFT, respectively, and experiments show that *Predis* significantly improves their throughput by 300% to 800%. *Multi-Zone* is implemented on BFT-SMaRt and compared with random and star network topologies, and it is shown that *Multi-Zone* holds excellent scalability and the capability of reducing block propagation latency by at least 50%.**

***Index Terms*—permissioned blockchains, throughput, latency**

I. INTRODUCTION

The permissioned blockchain, a distributed ledger that can serve as a trusted ledger among mutually untrusted parties, is widely adopted in business cooperations [1]. Data production and distribution, as two major processes in permissioned blockchains, exert significant impact on the system's throughput. The former is located in the consensus layer, where consensus nodes pack transactions into a candidate block and reach a consensus on its content, while the latter finds itself in the network layer, where consensus nodes consume their own bandwidth to distribute new blocks to full nodes.

PBFT (Practical Byzantine Fault Tolerance) [2] protocol is widely used in mainstream permissioned blockchains, such as [3]–[5]. It achieves $O(n^2)$ message complexity due to all-to-all communication. Since then, PBFT has been extended in several aspects (such as [6]–[9]). HotStuff [10] is another milestone. It utilizes the chain structure and all-to-one voting to reach $\mathcal{O}(n)$ message complexity and better scalability. From then on, new proposals, such as Fireledger [11], Kauri [12], Stratus [13], Leopard [14] and Narwhal [15], have been advanced to improve throughput even further.

The improvement in the consensus layer may transfer the throughput bottleneck to the network layer. To improve throughput, recent solutions such as Narwhal [15], Stratus [13], and Leopard [14] attempt to design blocks capable of containing more transactions, which leads to larger blocks whose propagation latency climbs linearly once their size surpasses a certain limit [16], [17]. In addition, the rising quantity of nodes, which usually leads to a longer propagation latency, needs to be accompanied by a higher value of block interval. For example, Bitcoin [18] sets up a block interval of 10 minutes, provided that a block can propagate to most of the nodes within this block interval. Data distribution is a crucial factor affecting throughput, data propagation latency, and data availability [19]–[21], yet research to appreciate its significance remains to be covered.

In terms of data distribution, Compact [22], Xtreme [23], Graphene [24], and Dino [25] compress block size, enabling blocks to accommodate more transactions without prolonging propagation latency. Fibre [26], Bloxroute [27], and Marlin [28] employ a token incentive mechanism to introduce relay nodes for the reduction of distribution latency. There exist other proposals for improving the efficiency of data distribution as well, such as [29]–[37]. These optimizations require a better data production strategy to cooperate with to unleash their potentials to the fullest.

The throughput of data production is mainly determined by the bandwidth utilization of consensus nodes and block propagation latency. On the other hand, data distribution occupies the bandwidth of consensus nodes and determines block propagation latency. Thus, its improvement requires a solution of high-throughput data production and low-latency data distribution. These two mutually influencing processes co-determine the overall throughput of permissioned blockchains. With this insight in mind, we propose a new data production strategy *Predis* and its corresponding distribution mechanism *Multi-Zone* for permissioned blockchains that adopt leader-

2575-8411/23/$31.00 ©2023 IEEE
DOI 10.1109/ICDCS57875.2023.00088

based BFT consensus protocols (such as PBFT and HotStuff). *Predis* and *Multi-Zone* constitute our data flow framework for permissioned blockchains.

Predis allows each consensus node to continuously bundle up transactions and multicast each of such bundles to other nodes. In the end, parallel bundle chains constitute each node's transaction pool (mempool). At the start of every consensus round, the leader cuts a slice of each bundle chain, constructs a *Predis* block, and multicasts it to other nodes. The *Predis* block is tiny and carries no transactions (block body) but just metadata containing rules of block building and mapping into multiple transactions. In this way, a large number of transactions could be confirmed in one round.

Multi-Zone is devised to reach two goals: to keep the bandwidth consumption by consensus nodes from growing in distributing blocks as the number of full nodes rises; to harness the bandwidth of full nodes effectively to scale down block propagation latency. In *Multi-Zone*, the network is divided into more than one zone. Each zone has a fixed number of relayers (special full nodes) to receive stripes from consensus nodes and forward them to other nodes. After receiving enough stripes, a node can decode them to get a bundle and save that bundle into its bundle chains. Once a new block is produced, consensus nodes send its *Predis* block to the relayers in each zone, which will in turn pass it on to other full nodes. A node can then rebuild a new block with a *Predis* block and bundle chains in its mempool. By virtue of the minimal sizes of *Predis* blocks and efficacious data distribution in *Multi-Zone*, new blocks can be swiftly propagated to the entire network.

It is duly noticed that Narwhal [15], Leopard [14], Stratus [13] and *Predis* make each consensus nodes pack transactions into a microblock (similar to the bundle in *Predis*) and multicast them to others. At the start of every consensus round, the leader puts microblock identifiers instead of transactions into a proposal (also called a candidate block) and multicasts it to others. The idea that decouples the distribution of transactions from the consensus process can effectively improve throughput. However, there will then arise the issue of data availability. Faced with this problem, Narwhal employs reliable broadcast (RBC) in which every node needs to collect $n_c - f$ certificates of the current microblock before piggybacking them into the next microblock; Leopard adopts RBC for the first consensus round, after which the leader organizes the microblock identifiers along with their certificates into a proposal; Stratus introduces a provably available broadcast (PAB) primitive in which every producer collects $f + 1$ certificates after multicasting a microblock to guarantee that at least one honest node holds the microblock. In contrast to the above operations, *Predis* discards the RBC primitive. Instead, a mempool is devised as a chained structure to arrive at the equivalent outcome of RBC. Therefore, *Predis* can be counted as the simplest solution in which every node continuously produces bundles and piggybacks its latest bundle height on its bundle to secure data availability. Furthermore, with the rising volume of transactions, the sizes of proposals in *Predis* (*Predis* blocks) remain almost unchanged, while those of its counterparts manifest linear growth.

In summary, this paper makes the following contributions:

1) We propose *Predis*, which enables consensus nodes to contribute idle bandwidths for the pre-distribution of block content and scales down the complexity of bandwidth consumption in candidate block distribution from linear level to constant level.
2) *Multi-Zone* is then advanced in our research. It divides the network into several zones and combines the multicast tree [38] with erasure coding [39] to decrease bandwidth spikes in data distribution and reduce data propagation latency.
3) To test its performance, *Predis* is applied to PBFT and HotStuff. Experimental results show that *Predis* delivers a significant improvement in throughput from 300% to 800%. Besides, we implement *Multi-Zone* based on BFT-SMaRt [6]. Compared with star and random network topologies, *Multi-Zone* proves to hold excellent scalability and reduce block propagation latency by at least 50%.

II. SYSTEM MODEL

There are three roles in a permissioned blockchain: client, full node, and consensus node. A client signs its transaction and submits it to a full node, which maintains the history of the ledger and stands at the service of clients. A consensus node is a special kind of full node that packs transactions into blocks and extends the ledger. We assume the client is honest, and the system contains N full nodes, with at most f malicious nodes. The number of consensus nodes is $n_c = 3f + 1, (n_c \leq N)$.

Designed for leader-based BFT consensus protocols, *Predis* inherits the Byzantine threat and communication models [2], e.g., malicious nodes may produce arbitrary values, delay or omit messages, and collude with each other yet can not forge the signatures of honest nodes. Further, the network is partially synchronous, whereby a known upper bound Δ on the time for message transmission holds after a Global Stabilization Time (GST), which is unknown.

In the network layer, malicious full nodes can only delay or omit messages, which will be tackled by failure probability. Since there are at most f malicious nodes, a node can resist Eclipse attack [40] by connecting to more than f other nodes. In addition, nodes in permissioned blockchains do not communicate with nodes without permission. Therefore, attacks involving the control of a massive amount of nodes (such as [40], [41]) are ignored.

III. PREDIS

Leader-based consensus protocols [2], [10], [12] can not make good use of bandwidth for two reasons. For one, only in the proposing stage is leader's bandwidth utilized — for the rest of the process, it remains idle. For another, as for non-leader nodes, their bandwidth almost stays inactive throughout the whole round of consensus. Although Narwhal, Leopard, and Stratus tried to change this situation, the solution that we

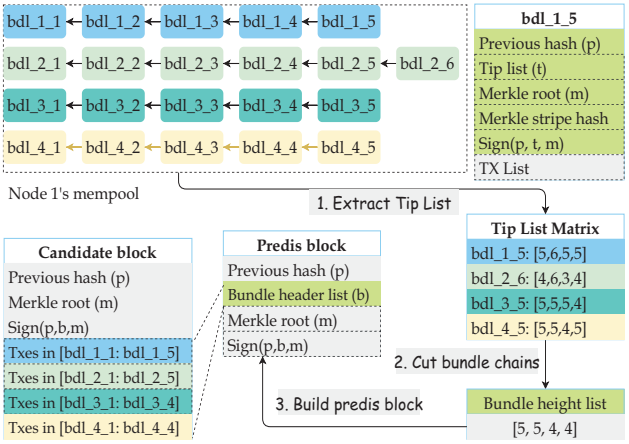


Fig. 1. An example of *Predis* block packing.

provide in this paper, *Predis*, can achieve the equivalent effect with a simpler mechanism.

A. Parallel Bundle Chains

In *Predis*, transactions are unceasingly packed into bundles, which, structured like blocks, contain hashes pointing to their parents. Every bundle belongs to one parent and assumes one child. Once a bundle is produced, the producer will have it signed and multicast it to other consensus nodes. Eventually, every node possesses n_c parallel bundle chains.

The structure of a bundle is presented in Fig.1, *bdl*_1_5, where the green part represents its header, the gray region, its body. As can be seen, a header is constituted by its parent hash (previous hash), a tip list, a Merkle root, and the signature of its producer. The tip list contains the height of the latest bundle that the producer received, e.g., node 1's mempool at the top left of Fig.1. When node 1 packs the next bundle, its tip list is [5, 6, 5, 5], which means that, for the first bundle chain, the bundle producer has received bundles whose heights are no higher than 5, and for the second chain, no higher than 6, and so forth. A node checks a bundle upon its reception. For a valid bundle:

1) Its parent is valid. If its parent does not arrive together, the node requests that from its producer.
2) Transactions in the TX list are valid.
3) Its tip list is more updated than that of the parent bundle.
4) There exist no other bundles in conflict with it.

If a bundle is judged as valid, it will be saved into the mempool. A conflict bundle is another valid bundle that has a different bundle header but shares the same parent bundle hash with the current one. As a malicious node may pack conflict bundles and forward them around, a ban list is established in *Predis* to exclude nodes producing conflict bundles. Once two conflict bundles are detected, an honest node will multicast them to other nodes and register its producer into the ban list. Eventually, a detected malicious node will be itemized in the ban list of all honest nodes.

B. Predis Block

As *Predis* organizes each node's mempool into parallel bundle chains, at the outset of every consensus round, for each bundle chain, the leader cuts a slice of each bundle chain and takes the last bundle header to construct a new proposal, which is called *Predis* block. For a bundle chain, we suppose all of the bundles of height lower than h have already been confirmed in the last round. To cut that bundle chain, the leader extracts the tip list from the latest bundle on each chain and cuts at height $h'(h' >= h)$, which corresponds to the minimum height of bundles received by the fastest $n_c - f$ nodes (including the leader itself). Besides, as mentioned above, the ban list is to eliminate malicious nodes, meaning that neither will an honest node cut bundle chains produced by banned nodes nor will it vote for a *Predis* block containing bundles produced by banned nodes.

Fig.1 illustrates how leader node 1 packs a *Predis* block from parallel bundle chains in its mempool. In this example, there are four consensus nodes (four bundle chains), and the packing goes as follows:

1) The leader extracts the tip list from each bundle chain's latest bundle and a tip list matrix is thus produced, which represents the newest bundles received by every consensus node. In our example, tip lists are extracted by leader node 1 from bdl_1_5, bdl_2_6, bdl_3_5, and bdl_4_5.
2) The resultant matrix in turn becomes the basis for the leader to cut its bundle chains. In our example, leader node 1 cuts according to the aforementioned cutting rule, and a list of bundle height is obtained: [5, 5, 4, 4], meaning that bundles before and including bdl_1_5, bdl_2_5, bdl_3_4, and bdl_4_4 will be put into the candidate block.
3) The corresponding bundle headers of the bundle height list are put into the Bundle header list, and the leader compute the Merkle root for transactions in the bundle height list. Finally, the previous block hash is written into a *Predis* block and signed.
4) The leader multicasts that *Predis* block to other nodes.

The node performs the following check once it receives a *Predis* block:

1) It has received the parent block, otherwise, it will initiate a parent request towards other nodes.
2) The *Predis* block does not contain bundles produced by nodes in its ban list. Further, there exist no other bundles in conflict with bundle in the *Predis* block, otherwise the node multicast conflict bundles to other nodes and register its producer into the ban list.
3) The node has all of the bundles referred to in the bundle header list, and the signature of the *Predis* block is valid. If some bundles are missing, the node will request them from the bundle producer and other available nodes.
4) The node collects all transactions according to the bundle header list, computes the Merkle root, and then

checks if the computed Merkle root is equal to that in the *Predis* block.

If the *Predis* block is judged as valid, the node will proceed to the voting stage; while if the block turns out to be illegal, the node will refuse to vote.

C. Predis's Safety

In the following paragraphs, b_{ij}^h denotes the ith bundle chain at height h that node j received. The collision-resistant hash function D used in *Predis* guarantees that if $D(x) = D(y)$, then $x = y$. We use b.header to denote the header of a bundle and b.body to denote its body.

Theorem 3.1: **Bundle header's consistency.** If b_{ij}^h and b_{ik}^h are both valid and b_{ij}^h.header = b_{ik}^h.header, then b_{ij}^h = b_{ik}^h.

Proof 3.1: The bundle header has a Merkle root computed by transactions in the body. According to the hash function, it can be proved that b_h^{ij}.body = b_{ik}^h.body, therefore, b_{ij}^h = b_{ik}^h.

Theorem 3.2: **Bundle's consistency.** If b_{ij}^h and b_{ik}^h are valid, and $b_{ij}^h = b_{ik}^h$, then $b_{ij}^{h'} = b_{ik}^{h'} (h' \leq h)$.

Proof 3.2: As $b_{ij}^h = b_{ik}^h$ and they are both valid, they share the same parent bundle hash. Furthermore, as b_{ij}^h is valid, so is b_{ij}^{h-1}, and Theorem 3.1 decides that $b_{ij}^{h-1} = b_{ik}^{h-1}$, which are also valid. Thus, it is arrived at that $b_{ij}^{h'} = b_{ik}^{h'} (h' \leq h)$.

Theorem 3.3: **Predis's consistency.** If an honest leader multicasts a *Predis* block, any two of the honest nodes who vote for it will build up two candidate blocks of the same content.

Proof 3.3: Suppose that the leader and two honest nodes are i, j, and k, respectively. Without loss of generality, b_{xi}^h is used to represent a bundle in the *Predis* block. If node j votes for that block, then b_{xi}^h.header = b_{xj}^h.header. According to Theorem 3.1, plus that $b_{xi}^h.body = b_{xj}^h$.body, we have $b_{xi}^h = b_{xj}^h$. Then, with Theorem 3.2, $b_{xj}^{h'} = b_{xk}^{h'}, (h' \leq h)$. Therefore, for all bundle chains that *Predis* block refers to, j and k can construct two identical candidate blocks.

D. Predis's Liveness

Predis can potentially harm the system's liveness, but it has features to handle these threats. The first case is when the leader can not manage to collect enough votes for a *Predis* block, and subsequently, the whole system is put to a forced halt. To deal with this breakdown, every node sets up a timer upon the arrival of a new bundle or a new *Predis* block, and when time is out, it can suspect and replace the leader. An honest node always multicasts its bundles to other nodes, implying that if the leader is in a state of halt, at least $n_c - f$ honest nodes will have it replaced. In the second case, some bundles are missing in the received *Predis* block. Faced with this situation, the node will refuse to vote for this block and initiate requests for the missing ones from bundle producers and other available nodes, which are obtainable from the latest tip lists of bundles in its bundle chains. The cutting rule determines that, when an honest node assumes leadership, it will always cut the chain at a height where at least $n_c - f$ nodes have received the corresponding bundle. This ensures that a node can always access missing bundles from $n_c - 2f$ honest nodes. If it is a malicious leader, it might not be able to collect enough votes, presenting then the first situation above.

E. Potential Attacks

Forking Attack. When an honest node detects conflict bundles, it will spread conflict bundle headers around and enter the involved bundle producer into ban list. Besides, honest leaders will never cut bundle chains in ban lists, and will never contribute their vote to a *Predis* block containing bundles generated by banned nodes. When a node is banned for a period of time, it has the option to propose a new genesis bundle to rejoin the bundle producing process, and others can remove it from the ban list.

Censorship Attack. If a client's transactions stay unconfirmed for longer than usual, the client will notify its full nodes to consign these transactions to another consensus node. A transaction will eventually be packed into a bundle after at most $f + 1$ attempts. If a malicious node refuses to pack a bundle into a *Predis* block, the honest node will suspect and replace it when the timer of the bundle expires.

Duplicate Transactions. The performance of *Predis* can be subject to deterioration out of Byzantine clients' dispatch of identical transactions to multiple nodes. This type of attack also exists in Narwhal, Stratus, and Leopard. One possible counter-measure is transaction partition in Mir-BFT [42]. However, we leave the exploration of applying this solution to future research.

F. Performance Analysis

Throughput. Suppose there are n_c consensus nodes, the uploading bandwidth of the ith ($i \in [1, nc]$) node is x_i B/s, the average propagation delay is ls seconds, each transaction is b bytes, and each consensus round costs t_c seconds. In *Predis*, it takes at least ls seconds for node i to distribute a bundle b_i_n, and the receiver j spends another ls seconds on distributing a bundle b_j_m that contains a tip list to confirm the reception of b_i_n. Thus, only bundles produced $2 \times ls$ seconds before can be packed into a *Predis* block by the leader.

Propose, *Write*, and *Accept* are used to represent three phases in PBFT, and P_i, W_i, and A_i ($i \geq 1$) denote each phase in the ith consensus round, respectively. We suppose each phase costs ls seconds, and each consensus round of PBFT is $3 \times ls$ seconds. When P_2 starts, bundles produced in P_1 can be packed into a *Predis* block. When $P_i (i \geq 3)$ starts, bundles produced between W_{i-2} and P_{i-1} can be packed into a *Predis* block, which demonstrates that a *Predis* block contains bundles produced within exactly one consensus round. As a node needs to distribute a bundle to $n_c - 1$ nodes, the throughput of n_c nodes during one consensus round is T bytes, as shown in Eq 1. The TPS of *Predis*-based PBFT (hereafter, P-PBFT) is in Eq.2, which shows that the throughput of *Predis* will decline gracefully when the number of nodes increases.

$$T = \sum_{i=1}^{n_c} \frac{x_i \cdot t_c}{n_c - 1} \tag{1}$$

$$TPS = \frac{T}{b \cdot t_c} = \sum_{i=1}^{n_c} \frac{x_i}{b \cdot (n_c - 1)} \tag{2}$$

We use *Propose* and *Vote* to represent procedures in chained HotStuff, and P_i and V_i $(i \geq 1)$ denote phases in the ith consensus round. When $P_i (i \geq 3)$ phase starts, bundles produced between P_{i-2} and V_{i-2} can be packed into a *Predis* block. Like in PBFT, a *Predis* block in HotStuff contains bundles produced within exactly one round. Thus, the TPS of n_c nodes during one round of HotStuff can be denoted by Eq 1, which implies that P-PBFT has the same TPS as *Predis*-based HotStuff (hereafter, P-HS), except that the latter manages a lower latency.

Nevertheless, the experiments in Section V can hardly reach the throughput in Eq.2 because:

1) The pre-condition for a leader to cut a bundle is that at least $n_c - f$ nodes have received it;
2) Voting and replying to clients also occupy bandwidth, making it unlikely to channel all bandwidth for bundle production;
3) Implementation and parameter-setting also affect the throughput.

Block Size. As a *Predis* block carries no transactions, its size does not increase with the rising volume of transactions of its corresponding candidate block, thus countable as a constant. Suppose there are n_{tx} transactions in a candidate block and n_c consensus nodes; *Predis* reduces the leader's bandwidth consumption of candidate-block dispatch from $\mathcal{O}(n_c \times n_{tx})$ to $\mathcal{O}(n_c)$, another manifestation of *Predis*'s superiority over Narwhal, Stratus, and Leopard. In section IV, we will show that when combines *Predis* and *Multi-Zone*, it also costs $\mathcal{O}(n_c)$ bandwidth to send a new block to full nodes.

IV. Multi-Zone

A. Overview of Multi-Zone

Existing permissioned blockchains underestimate the impacts of network topology on the system throughput and usually adopt star or random topologies. In star topology, when the number of nodes increases, the bandwidth overhead of consensus nodes grows linearly, and the throughput decreases significantly. The random network topology, widely employed in large-scale blockchains such as Bitcoin and Ethereum [43], usually uses the gossip [44] for data dissemination. In permissioned blockchains, nodes remain online unless crash failures happen, and their network environment is more static than in public blockchains. As a result, data propagation latency may be the most critical in permissioned blockchains, motivating us to design *Multi-Zone* to introduce multicast trees [38] for the swift distribution of data produced in *Predis*.

Multi-Zone divides the network into n_z areas, each of which counts as a zone. Their quantity is determined upon the initial construction of permissioned blockchain, and their division is based on the locality or connectivity of nodes, for instance, west-coast or east-coast zones. There are three types of full nodes in *Multi-Zone*, consensus nodes, relayer nodes

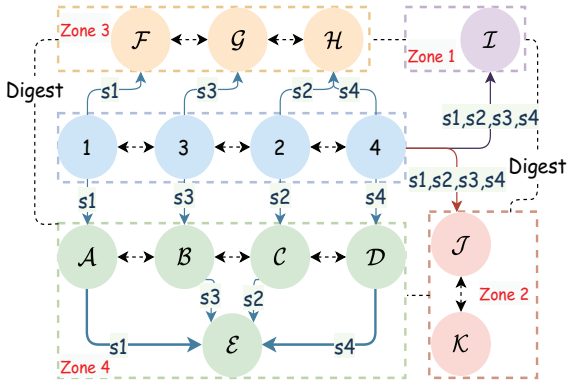


Fig. 2. *Multi-Zone*'s topology. s_i denote ith stripe.

(hereafter, relayer), and ordinary full nodes (hereafter, ordinary nodes). Relayers receive data from consensus nodes and forward data to other relayers and ordinary nodes. Ordinary nodes receive data from other ordinary nodes and relayers.

Fig.2 shows a *Multi-Zone* network with four consensus nodes (1,2,3, and 4) and four zones. Nodes connected with dashed double arrows constitute a full mesh topology. Upon receiving a bundle, a consensus node decodes that bundle into several stripes and send them to relayers, and relayers forward stripes to other nodes. Upon the reception of enough stripes, a node can successfully decode a bundle. When there are enough relayers in a zone, other nodes do not need to connect to consensus nodes but to subscribe for data to relayers. For example, ordinary node $\mathcal{E}$ can four stripes from $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$. To keep the network robust, nodes in a zone can connect to those in neighbor zones and exchange data digest with gossip protocols. In the case that some data are missing, a receiver will pull the original data from a digest sender.

B. Robustness Analysis

The most straightforward method of data distribution is to apply a hash function mapping every zone to a consensus node. Every consensus node sends data to one relayer, which forwards data to ordinary nodes. However, if a malicious consensus node declines to send data, a relayer will have no data to receive. The counter strategy is to ensure that at least $q_c (q_c > f)$ consensus nodes should send data to one relayer, and we should also factor in the problem of single point of failure, where the network will be rendered fragile if there is only one relayer for one zone.

As *Multi-Zone* is located in the network layer, node failure probability is employed to attend to malicious behaviors such as message delay or omission. It is supposed that the node failure probability of an honest node is p_h, and that of a malicious node is $p_b (p_b = 1)$, and the general failure probability of a node is p_c, as in Eq.3. According to [45], the annual failure rate of a server is approximately 3%, which is why we consider p_c as approximately equal to $\frac{f}{N}$.

$$p_c = \frac{f}{N} \times p_b + (1 - \frac{f}{N}) \times p_h \approx \frac{f}{N} \tag{3}$$

Algorithm 1: Check and become a relayer

```
   Data: Node 𝒪 and Consensus node set 𝒩_c
1  𝒮_r ← 𝒪.GetZoneRelayers(𝒪.GetZone());
2  𝒮_p ← A set of |𝒩_c| stripes;
3  for relayer ℛ ∈ 𝒮_r do
4  |  S_set ← ℛ.RelayedStripes();
5  |  s = 𝒪.GetStripeCanSub(S_set, max = |S_set|/2);
6  |  𝒪.Send(subscribe(s), ℛ);
7  |  𝒮_p ← (𝒮_p − s);
8  end
9  if 𝒮_p ≠ ∅ then
10 |  for stripeId s ∈ 𝒮_p do
11 |  |  𝒪.Send(subscribe(s), s);
12 |  end
13 if 𝒪 received accept-subscribe msg then
14 |  𝒪.SetStripeSender(msg);
15 |  𝒪.AddRelayedStripe(𝒮_p);
16 |  if msg.getSender() ∈ 𝒩_c then
17 |  |  𝒪.SetRelayer(True);
18 |  |  𝒪.Send(relayerAlive(𝒮_p), 𝒪.ZoneNeighbors());
```

$$(\frac{f}{N})^{n_{zr}} \leq p_r \tag{4}$$

We suppose there are n_{zr} relayers in a zone and a robustness threshold probability p_r. When $(p_c)^{n_{zr}} \leq p_r$, there is a high probability that ordinary nodes can always receive data from at least one relayer. n_{zr} can be computed from f, N, p_r, and Eq 4. In *Multi-Zone*, we set that $n_{zr} = n_c$ and $q_r = n_c$, allowing a node to receive data from relayers with a probability higher than 99.98% when $n_c \geq 4$. This setting enables *Multi-Zone* to distribute data using the multicast tree [38] and erasure encoding [39] to reduce the bandwidth overhead of consensus nodes in data distribution.

C. Network Construction

When a node joins a permissioned blockchain network for the first time, it usually generates a transaction and sends it to the consensus node to register its identity. Then, consensus nodes pack that transaction into a new block and distribute it to other nodes. After receiving the new block, a node will discover the newcomer and accepts its network connection. For any two nodes, the order of appearance of registered transactions in the blockchain can be used as a basis to decide which node joined the network earlier.

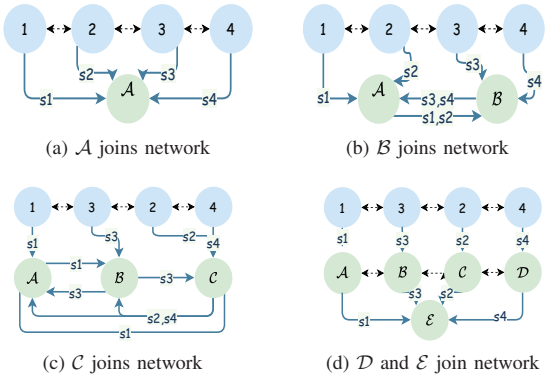
Algorithm 1 shows the logic of a node on whether it will become a relayer or not. When a node joins the network for the first time, it delivers a *getRelayer* message to its neighbors to acquire the current set of relayers. From line 3 to line 7: $\mathcal{O}$ can subscribe for at most half of the stripes to each relayer. A *subscribe* message contains stripes for which the sender subscribes. When the stripe sender crashes, a node can re-subscribe for that stripe to another node. From line 9 to line 13: if there are still some stripes with no senders, node $\mathcal{O}$ will subscribe for those stripes to consensus nodes and become a relayer. From line 14 to 22: when node $\mathcal{O}$ receives *accept-subscribe* messages, it records its stripe sender. An *accept-subscribe* message means that the sender accepts node $\mathcal{O}$'s *subscribe* request. If the message sender is a consensus node, it becomes a relayer. A relayer periodically multicasts *relayerAlive* messages to its neighbors, which contains the identity of the relayer and its relayed stripes.

D. Data Flow in Multi-Zone

In *Multi-Zone*, consensus nodes only forward stripes and *Predis* blocks to relayers. When a consensus node receives a new bundle, it encodes that bundle into n_c stripes with erasure coding, and the ith consensus node sends the ith stripe to its subscribers. In sending a stripe, the sender should attach the bundle header and a Merkle proof of the stripe to detect whether the stripe is tampered (The Merkle Stripe hash in Fig.1 is used to examine the correctness of a stripe). Hence, a relayer can decode a bundle by receiving stripes from all consensus nodes in parallel, mitigating the influence of node failure. However, if all relayers in a zone receive stripes directly from consensus nodes, it will impose a heavy bandwidth burden on them. *Multi-Zone*, on the contrary, allows relayers in a zone to send stripes to each other to reduce the bandwidth overhead of consensus nodes.

Fig.3 shows an example of how relayers forward data to each other. In Fig.3(a), when node $\mathcal{A}$ joins the network, it subscribes to four consensus nodes 1, 2, 3, and 4 for four stripes. In Fig.3(b), node $\mathcal{B}$ enters and subscribes to $\mathcal{A}$ for half of the stripes (1 and 2) and subscribes to nodes 3 and 4 for the other two stripes. When node $\mathcal{B}$ accepts *accept-subscribe* messages from consensus nodes, it becomes a relayer and multicasts a *relayerAlive* message to other nodes. *Multi-Zone* sets up a subscription preference for each relayer that they subscribe to other relayers. When node $\mathcal{A}$ receives a *relayerAlive* message, it subscribes to $\mathcal{B}$ for stripes (3 and 4) and sends an *un-subscribe* message to nodes 3 and 4, respectively. In Fig.3(c), when node $\mathcal{C}$ arrives, it subscribes to node $\mathcal{A}$ and $\mathcal{B}$ for stripes 1 and 3, respectively, and it subscribes to node 2 and 4 for stripes 2 and 4. When node $\mathcal{A}$ and $\mathcal{B}$ receive *relayerAlive* messages from node $\mathcal{C}$, they will subscribe to node $\mathcal{C}$ for stripes 2 and 4. When there are n_c relayers in a zone, other newcomer nodes connect and subscribe to those relayers for stripes. In Fig.3(d), ordinary node $\mathcal{E}$ can receive four stripes from four relayers. If a relayer reaches its upper limit of the manageable amount of subscriptions, it will send a *reject-subscribe* message to decline the following subscription requests. This message will contain the available children of the relayer, to which the declined can subscribe.

A *Predis* block has only several bundle headers, and it is far less than the size of a bundle. Therefore, each consensus node sends *Predis* block to relayers when a new block is produced, and then the relayers forward the *Predis* block to ordinary nodes. In addition, every node can rebuild the original block with *Predis* block and bundle chains in its mempool and the missing bundles can be acquired from *Predis* block senders.



(a) $\mathcal{A}$ joins network (b) $\mathcal{B}$ joins network

(c) $\mathcal{C}$ joins network (d) $\mathcal{D}$ and $\mathcal{E}$ join network

Fig. 3. Network construction in *Multi-Zone*.

There are two strategies of transaction dissemination in *Multi-Zone*. The first one allows every full node to make a connection with a consensus node and delivers transactions to it. The second path is that, the client selects a consensus node and writes the identity of the target consensus node on its transaction. Afterwards, the client sends the transaction to its full node, which helps spread transactions to other nodes. Eventually, the target consensus node will receive the transaction.

E. Fix the Number of Relayers

In *Multi-Zone*, when a relayer is ready to leave the network, it will send a *leave* message to the earliest-joining node among its subscribers; while upon the departure of an ordinary node, it will multicast a *leave* message to its subscribers. On the receiving end of *leave* messages, if the sender is found to be a relayer, the receiver will subscribe to consensus nodes for stripes and becomes a new relayer; if the sender turns out to be an ordinary node, the receiver will resort to other available nodes for stripe subscription. To prove its online status, a node is required to send heartbeat messages periodically to its neighbors, and a node will disconnect itself with its neighbor if the neighbor's heartbeat is time-out.

Furthermore, a node conducts periodical checks on the number of relayers in its zone according to *relayerAlive* messages it received. If the number of relayers is less than n_c, the node will become a new relayer. This way, when a relayer crashes, there will be more than one node to fill up the vacancy. In *Multi-Zone*, the number of relayers is maintained at n_c through nodes' periodical multicasting and processing of *relayerAlive* messages. In addition, a relayer will adjust itself to be an ordinary node when it finds itself to be redundant, as shown in Algorithm 2. In line 4, we can see that, if a node multicasts a *relayerAlive* message that contains an empty stripe set, it means that the relayer becomes an ordinary node. From line 7 to line 13: when two relayers relay the same stripes, the one that enters earlier will only relay one stripe and subscribe for other stripes to the later-entering one. If $\mathcal{O}$ relays only one kind of stripe, any node that enters later than $\mathcal{O}$ will subscribe to $\mathcal{O}$. This specific design directs redundant relayers to turn into ordinary nodes and subscribe for stripes to other relayers. Lines 14 to 18 tell that a relayer will subscribe to another relayer if the current one does not relay anymore. Lines 21 to 23 show that, once a relayer finds that it stops relaying, it will become an ordinary node and multicast a *relayerAlive* message with an empty stripe set.

Algorithm 2: Process *relayerAlive* message.

```
   Data: Node O, relayerAlive message M
1  R ← M.GetRelayer();
2  S_r ← R ⋃ O.GetZoneRelayers(O.GetZone());
3  P_o, P_m ← O.RelayedStripes(),R.RelayedStripes();
4  if P_m = ∅ then
5  |  O.removeRelayer(R);
6  else if O.IsRelayer() then
7  |  T_o, T_m ← O.GetJoinTime(), M.GetJoinTime();
8  |  P_om ← P_o ⋂ P_m;
9  |  if T_o ≤ T_m or |P_m| = 1 then
10 |  |  if T_o ≤ T_m then
11 |  |  |  P_om -= 1;
12 |  |  O.Send(subscribe(P_om), R);
13 |  |  P_o -= P_m;
14 |  for stripe s ∈ P_m do
15 |  |  sd ← O.GetStripeSender(s);
16 |  |  if s ∉ sd.RelayedStripes() then
17 |  |  |  O.Send(subscribe(s), R);
18 |  end
19 O.UpdateRelayer(R);
20 O.Send(relayerAlive(M), O.ZoneNeighbors());
21 if O.IsRelayer() and P_o = ∅ then
22 |  O.SetRelayer(false);
23 |  O.Send(relayerAlive(∅), O.ZoneNeighbors());
```

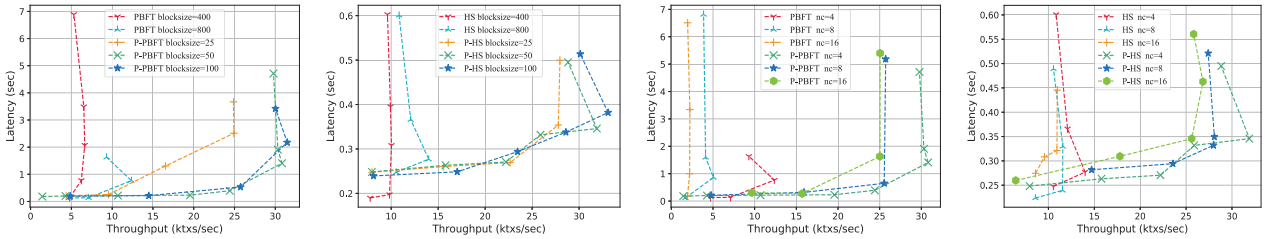
F. Backup connections

In *Multi-Zone*, a node can subscribe for n_c stripes to n_c nodes, although decoding a bundle only takes $n_c - f$ stripes. The rest of the f connections can be used to enhance the network's robustness and attenuate the impact of node churn. A relayer may receive stripes from only one consensus node but can still connect to other consensus nodes for data pulling. Besides, every ordinary node establishes connection (called backup connection) with nodes in neighboring zones. Along such a connection, nodes deliver data digests to synchronize with each other on the state of the ledger.

V. EVALUATION

We implement *Predis* on BFT-SMaRt [6] and HotStuff [46], and utilize ECS of Alibaba to examine its performance in LAN and WAN environments. Equipped with 4 cores and 8 GB memory, each instance is ecs.c7.xlarge. In the experiments, unless otherwise specified, the following configurations are adopted:

1) In WAN, the network bandwidth is 100 Mbps, and the instances are located in 4 regions: Ulanqab (CN-



(a) Parameter Effects on P-PBFT (b) Parameter Effects on P-HS (c) Compare P-PBFT and PBFT (d) Compare P-HS and HS

Fig. 4. *Predis*'s improvement on PBFT and HotStuff.

north), Shanghai (CN-east), Chengdu (CN-southwest), and Shenzhen (CN-south).

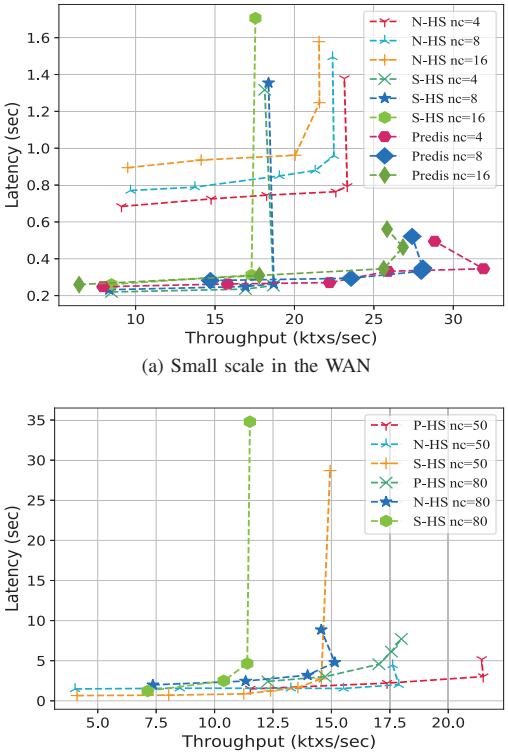
2) In LAN, the traffic control command is used to set a latency of 25 ms and a bandwidth of 100 Mbps for each instance to emulate the WAN environment;
3) The result of every experiment is the average of 5 runs, and every data point is measured when the system is running stable.
4) Every bundle has 50 transactions and every transaction has 512 bytes.

A. *Predis*

Predis's throughput will be shown in throughput-latency experiments, in which latency refers to the time elapsed from when a client sends a transaction to replicas to when the client receives a reply.

Improvement on PBFT and HotStuff. Fig.4 shows *Predis*'s improvement of PBFT and HotStuff in WAN environment. Fig.4(a) and (b) show the throughputs of PBFT, HotStuff, P-PBFT, and P-HS with different bundle sizes and batch sizes when $n_c = 4$. Bundle size refers to the volume of transactions in a bundle, while batch size refers to that in a batch or block. As can be seen, in PBFT and HotStuff, the throughput of the 800-batch size is about 3K∼6K higher than that of the 400-batch size. In *Predis*, the throughput under 50-bundle size is approximately equal to that under 100-bundle size, and is about 5K∼7K higher than that under 25-bundle size. Although different parameter settings lead to different levels of throughput, *Predis* can always achieve a much higher level than PBFT and HotStuff (almost 300%). According to above experimental results and experiments in [10], the following experiments are set with a bundle size of 50 and a batch size of 800.

Fig.4(c) and (d) show the throughput of *Predis* when the number of consensus nodes, n_c, is 4, 8, and 16, respectively. In *Predis*, a newcomer node consumes the bandwidth of others, degrading throughput, yet simultaneously, it also contributes its bandwidth to bundle production, which improves throughput. As a consequence, as we can see, as n_c rises, *Predis*'s throughput declines slowly, which is in consistent with our analysis in Eq.2, while PBFT and HotStuff manifest rapid decreases in throughput under the same condition. Evidently, compared with P-PBFT, P-HS can reach a higher level of



(a) Small scale in the WAN

(b) Large scale in the LAN

Fig. 5. Compare *Predis* with Narwhal and Stratus.

throughput. This is because HotStuff has a lower message complexity in the favor of P-HS's utilization of more bandwidth for bundle production. On the whole, when n_c is 4, 8, and 16, it is observed that P-PBFT's throughput is about 300%∼800% of PBFT's, and P-HS's throughput is about 200% of HotStuff's. This experiment demonstrates that *Predis* can provide PBFT and HotStuff with better scalability.

Comparison with SOTA. As Narwhal and Stratus are open-sourced state-of-the-art (SOTA) protocols, we conduct comparisons between them and *Predis* in both WAN and LAN environment. To ensure the fairness of the comparison, one worker is employed in Narwhal, Stratus, and *Predis* for transaction packing, and every bundle or microblock contains at most 50 transactions. As for the number of microblock

identifiers, the default settings (1000) of Narwhal and Stratus are adopted.

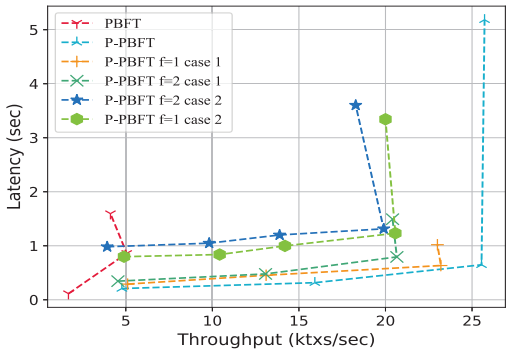
Fig.5 shows experiment results in the WAN and LAN. In both environments, *Predis* displays superior performance in terms of throughput and latency. In Narwhal and Stratus, the leader needs to put the identifier of each microblock into a proposal, which implies that the size of a proposal increases with the rising number of identifiers. As a result, they have to restrict the number of identifiers in a proposal in case large-size proposals prolong consensus time and degrade throughput. However, in *Predis*, as n_c bundle headers are always enough to represents cutting bundles, the leader can put bundles into a *Predis* block as many as possible. Therefore, once n_c is fixed, the bandwidth overhead of the leader for broadcasting a *Predis* block will be a constant. In our experiment, it is observed that when n_c is 80, a *Predis* block mapping into 50,000 transactions is no more than 2.5 KB, which is far less than the sizes of such blocks in Narwhal and Stratus (30 KB).

In *Predis*, the leader's bandwidth overhead of broadcasting proposals will not increase when the transaction volume in proposal increases, which allows a *Predis* block to aggressively contain more transactions and do not worry about increasing transmission delay of proposals. In addition, Narwhal and Stratus use RBC and PAB to guarantee data availability, that is to say, every node replies a certificate upon the reception of a microblock, while those in *Predis* do not need to, enabling *Predis* a lower message complexity and higher degree of bandwidth utilization. The two factors mentioned above constitute the reason why *Predis* has a higher throughput (4K∼6K higher) than Narwhal and Stratus. As shown in, Narwhal has the highest latency because every node has to collect $n_c - f$ certificates; Stratus has lower latency because its node needs to collect $f + 1 (< n_c - f)$ certificates; and Predis has the lowest latency because nodes do not need to collect certificates at all.

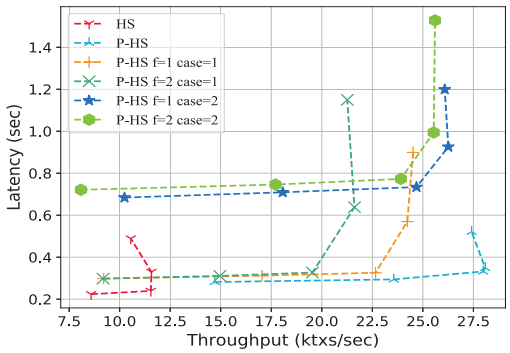
***Predis* under Faults.** We present two typical cases where malicious nodes cause the deterioration of *Predis*'s performance: (1) Malicious nodes neither produce bundles nor vote; (2) Malicious nodes refuse to vote and randomly send bundles to $n_c - f - 1$ nodes. In the latter case, the leader can not receive quorum votes before the missing bundles are found by certain nodes. Fig.6 shows *Predis*'s throughput under a system breakdown with 8 nodes. In case 1, the throughput of $8 - f$ working nodes is about $\frac{8-f}{8}$ of that under a normal situation. In case 2, it costs at least one extra round trip time to obtain missing bundles, leading to a higher latency. However, the throughput in case 2 is higher than that in case 1 because the malicious node is still producing bundles. Its throughput is less than that in the normal situation since sending missing bundles consumes the nodes' bandwidth. It should be noted that if it costs too much time to fetch those missing bundles, nodes may switch the view to maintain the system's liveness.

B. Multi-Zone

In this section, *Multi-Zone*'s impact on throughput and block propagation latency will go under test in the LAN environment. We realize *Multi-Zone* on BFT-SMaRt, and employ an open-source Reed-Solomon implementation [47] to encode bundles.



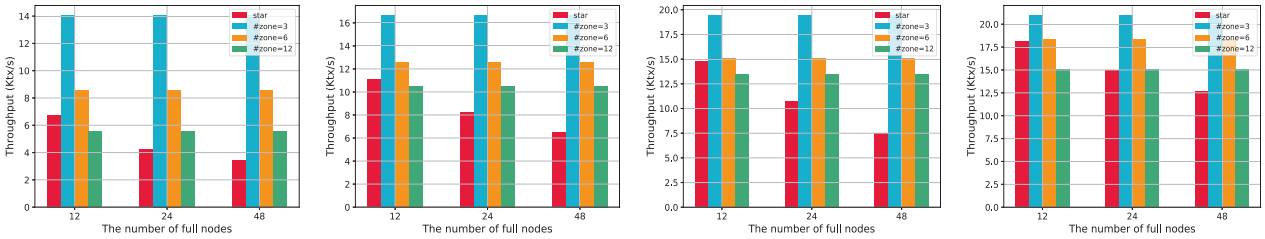
(a) P-PBFT under faults.



(b) P-HS under faults.

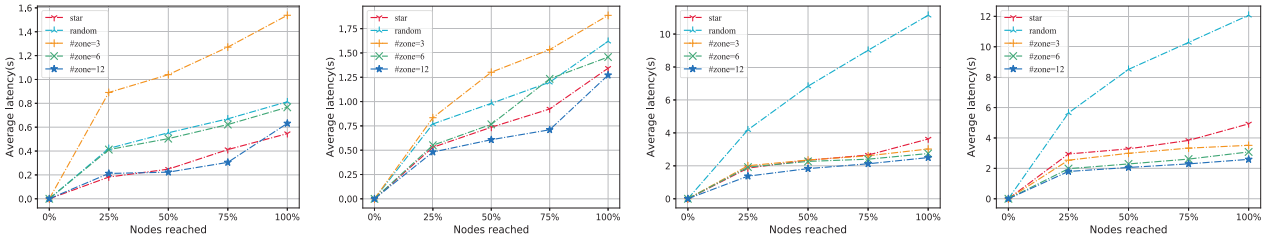
Fig. 6. *Predis* under faults with 8 nodes.

Effect on Throughput. Since the throughput of the random topology is determined by the tunable number of full nodes in connection to consensus nodes, we only compare the effects of the star and *Multi-Zone* on the throughput of the consensus layer. In our experiment, we fix the transaction generation rate at 26,000 txes/s, and conduct throughput comparison while the number of consensus and full nodes in the network is being raised up. As we can see in Fig.7, in the star topology, the throughput declines almost linearly as full nodes grow in number, because every newcoming full node will consume a piece of the bandwidth of the consensus layer. On the other hand, the throughput of *Multi-Zone* is determined by the number of zones, and when the zone number is a fixed value, its throughput will not decrease as the number of full nodes increases. When there are more than 24 full nodes, the throughput of a 12-zone *Multi-Zone* is always higher than that of the star topology. Contrary to our intuition, when the number of full nodes is fixed, the throughput of star and *Multi-Zone* increase as n_c climbs. For example, in *Multi-Zone*, the throughput when $n_c = 8$ is 3K higher than that when $n_c = 4$, similarly, the throughput when $n_c = 32$ is about 3K higher than that when $n_c = 16$ in the star topology. The reason behind is that, in *Predis*, a larger n_c implies more consensus nodes



(a) $n_c = 4$ (b) $n_c = 8$ (c) $n_c = 16$ (d) $n_c = 32$

Fig. 7. Compare the impact of Multi-Zone and star on throughput.



(a) block size <= 1MB (b) block size <= 5MB (c) block size <= 10MB (d) block size > 10MB

Fig. 8. Block propagation latency of star, random and Multi-Zone.

can contribute their bandwidth to the consensus process and share the bandwidth overhead of data distribution. Overall, this experiment demonstrates *Multi-Zone* can alleviate the throughput degradation caused by the increasing number of full nodes.

Effect on Block Propagation Latency. Random topologies are widely used in blockchain networks. In our experiment, we implemented random topologies on BFT-SMaRt and compared their block propagation latencies to those of *Multi-Zone* and the star topology. To compare with our data flow framework, consensus nodes in the star and random topologies do not send bundles but rather send complete blocks to full nodes every time a new block is produced. As a large number of consensus nodes requires more full nodes to form a proper comparison, we employ eight nodes to form a small scale of consensus nodes, and 100 nodes to form a large scale of full nodes, the remaining nodes are clients. Since Fair and Efficient Gossip (FEG) [48] is the state-of-the-art protocol in Hyperledger Fabric, we implemented and used it in the random topology for block distribution in our experiment. Parameters in the FEG protocol are set to the best values described in the paper. In addition, every node is set to be randomly connected to 8 nodes, a typical configuration in Bitcoin, Ethereum, and other random networks. As the fan-out parameter in the random topology is set to 4, we let every non-consensus node have at most 24 subscribers in *Multi-Zone* so that nodes in the two topologies have identical bandwidth overheads. The average block propagation latency of 100 blocks is measured.

Fig.8 shows the average propagation latency of blocks propagated to different percentages of nodes. Fig.8(a) and (b) show that the star topology has the shortest latency when the block size is smaller than 5 MB. When the block size is under 1 MB, the latency of random topology is shorter than that of *Multi-Zone* with 3 zones due to its larger fan-out number. However, as can be seen in Fig.8(c) and (d), when the block size exceeds 5 MB (the maximum block is 40 MB), *Multi-Zone*'s latency is the shortest. In addition, *Multi-Zone* with 12 zones has the shortest latency because increasing the number of zones leads to a lower height of multicast trees and block propagation latency, which is about 50% and 18% of the latency of the star and random topologies.

In *Multi-Zone*, each bundle is distributed when it is produced, and when a new block is produced, each node only needs to send a *Predis* block to other nodes, which is why *Multi-Zone*'s block propagation latency increases slowly as the block size increases. On the country, it costs more bandwidth and longer time to send a block in the star and random topologies. When the block is larger than 5 MB, the random topology exhibits the poorest latency performance because it randomly chooses several nodes and will ignore sending blocks to some nodes. We observe that the FEG protocol may waste bandwidth by sending duplicate data and is not efficient when distributing large blocks. This experiment shows that *Multi-Zone* helps the consensus layer and the network layer obtain a higher throughput and a shorter block propagation latency, respectively. For the computing overhead of encoding and decoding bundles, we observe that decoding and encoding only costs several microseconds, which is acceptable.

VI. RELATED WORK

As this paper presents a novel data flow framework encompassing data production in the consensus layer and data

distribution in the network layer, we provide an overview of related work from the perspectives of data distribution and data production, respectively.

A. Data Production.

Predis is specifically designed for leader-based Byzantine Fault Tolerance (BFT) protocols in a partially synchronous network. Consequently, our primary focus is on examining related work that is based on the partially synchronous network. Leader-based BFT protocols have received intensive research attention and many works have been dedicated to improving its performance [11]–[15], [49]. For example, Prime [50] comes up with the concepts of local order and global order to provide resilience against a malicious leader who degrades the throughput of PBFT. Kauri [12] improves the throughput of HotStuff by leveraging dissemination/aggregation trees to upgrade the bandwidth utilization of the leader. Compared to Prime and Kauri, *Predis* demonstrates a higher degree of bandwidth utilization and throughput since it allows all nodes to contribute bandwidth to the consensus process.

However, Narwhal, Stratus, and Leopard also enable all nodes to participate in block production. In comparison to these works, *Predis* offers two advantages: (1) *Predis* employs a single-chain structure for its mempool, achieving the same effect as reliable broadcast with lower message complexity overhead; (2) *Predis* avoids the issue of block size increasing linearly with the number of transactions it contains. This means that as the transaction volume increases, the sizes of proposals in *Predis* (referred to as *Predis* blocks) remain stable,unlike the linear growth seen in its counterpart models. These design choices in *Predis* contribute to higher throughput and lower latency.

Mir-BFT [42] and ISS [49] take a different approach by transitioning from a single leader consensus to a multiple leaders consensus model. As a result, all nodes are involved in block production, leading to a higher throughput. In comparison, *Predis* still adheres to the setting of having a single leader, resulting in a lower voting message complexity and higher bandwidth utilization.

B. Data Distribution.

Lots of research [17], [22], [48] have demonstrated the importance of block propagation latency for blockchain security and data availability. Existing work tries to improve block dissemination efficiency by optimizing data dissemination strategies, data compression approaches, and network topologies. To ameliorate data dissemination strategy, Kadcast [29], [30] proposes a new block propagation protocol based on Kademlia [51]. PiChu [31] and Velocity [32] disseminate a block by dividing it into several chunks to reduce the overhead of low-bandwidth nodes. Drabkin [52] provides a scalable Byzantine resilience dissemination approach. In comparison to these works, *Multi-Zone* absorbs their core ideas and becomes a more comprehensive scheme.

For a more efficient way of block compression, Compact [22] and XThin [53] adopt short transaction hashes to replace transactions. Graphene [24] introduces the invertible bloom lookup table [54] to compress the block. Dino [25] provides a new protocol that transfers the block construction rules to reduce the bandwidth consumption of transmitting a block to a constant complexity. In comparison to these works, *Predis* and *Multi-Zone* can also reduce the block transmission overhead to the constant level, similar to Dino. However, they only focus on the block synchronizing, while our work focus on the entire data flow of the blockchain.

To improve the network topology, Fiber [26] acts as a relay network to speed up block propagation for Bitcoin. BloXroute [27] and Marlin [28] employ a token incentive scheme to introduce relay nodes, reducing data dissemination latency. However, these solutions might lead to the centralization of the network and threaten the system's security. Additionally, other works [34]–[37] require nodes to conduct periodic probes and keep their neighbors updated. In comparison, *Predis* and *Multi-Zone* constitute a holistic data flow framework that optimizes all aspects of the work involved. *Predis* improves throughput by designing a new mempool with a chain structure. *Multi-Zone* divides the network into several zones to reduce the bandwidth overhead of the consensus layer in data distribution and improve network scalability. Additionally, it enhances data dissemination efficiency by using multicast trees and erasure coding to pre-distribute the content of a block (bundles) before the block is produced.

VII. CONCLUSION

In this paper, we argue that the throughput of permissioned blockchains is determined by both data production and distribution. This perspective motivates us to propose a new data flow framework for permissioned blockchains, in which, *Predis* enables each consensus node to contribute its idle bandwidth to pre-distribute block content in parallel, and *Multi-Zone* reduces the bandwidth overhead of consensus nodes in block distribution and significantly reduces block propagation latency. We hereby provide a novel viewpoint to design the data distribution strategy for a large-scale blockchain network.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their comments. This work was supported by Ant Group Research Fund. The corresponding author is Zhen Xiao.

REFERENCES

[1] ISO, 2022, https://www.iso.org/obp/ui/#iso:std:iso:tr:3242:ed-1:v1:en.

[2] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, no. 1999, 1999, pp. 173–186.

[3] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, 2018, pp. 1–15.

[4] Libra, 2022, https://libra.org.

[5] A. Group, 2022, https://antchain.antgroup.com/.

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

[7] M. Eischer and T. Distler, "Latency-aware leader selection for geo-replicated byzantine fault-tolerant systems," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2018, pp. 140–145.

[8] G. S. Veronese, M. Correia, A. N. Bessani, and L. C. Lung, "Spin one's wheels? byzantine fault tolerance with a spinning primary," in *2009 28th IEEE International Symposium on Reliable Distributed Systems*. IEEE, 2009, pp. 135–144.

[9] P.-L. Aublin, S. B. Mokhtar, and V. Quéma, "Rbft: Redundant byzantine fault tolerance," in *2013 IEEE 33rd International Conference on Distributed Computing Systems*. IEEE, 2013, pp. 297–306.

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

[11] Y. Buchnik and R. Friedman, "Fireledger: a high throughput blockchain consensus protocol," *arXiv preprint arXiv:1901.03279*, 2019.

[12] R. Neiheiser, M. Matos, and L. Rodrigues, "Kauri: Scalable bft consensus with pipelined tree-based dissemination and aggregation," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 35–48.

[13] F. Gai, J. Niu, I. Beschastnikh, C. Feng, and S. Wang, "Scaling blockchain consensus via a robust shared mempool," 2022. [Online]. Available: https://arxiv.org/abs/2203.05158

[14] K. Hu, K. Guo, Q. Tang, Z. Zhang, H. Cheng, and Z. Zhao, "Leopard: Towards high throughput-preserving bft for large-scale systems," in *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2022, pp. 157–167.

[15] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and tusk: a dag-based mempool and efficient bft consensus," in *Proceedings of the Seventeenth European Conference on Computer Systems*, 2022, pp. 34–50.

[16] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *IEEE P2P 2013 Proceedings*. IEEE, 2013, pp. 1–10.

[17] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer *et al.*, "On scaling decentralized blockchains," in *International conference on financial cryptography and data security*. Springer, 2016, pp. 106–125.

[18] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Manubot, Tech. Rep., 2019.

[19] M. Castro, M. B. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman, "An evaluation of scalable application-level multicast built using peer-to-peer overlays," vol. 2, pp. 1510–1520, 2003.

[20] D. Kostic, R. Braud, C. E. Killian, E. Vandekieft, and A. Vahdat, "Maintaining high bandwidth under dynamic network conditions," in *Proceedings of the 2005 USENIX Annual Technical Conference, April 10-15, 2005, Anaheim, CA, USA*, 2005.

[21] Magharei, N, Rejaie, R, Guo, and Y, "Mesh or multiple-tree: A comparative study of live p2p streaming approaches," -, 2007.

[22] M. Corallo, "Bip152: Compact block relay," *https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki*, 2016.

[23] P. Tschipper. (2016) Buip010 xtreme thinblocks. [Online]. Available: https://bitco.in/forum/threads/buip010-passed-xtreme-thinblocks.774/

[24] A. P. Ozisik, G. Andresen, B. N. Levine, D. Tapp, G. Bissias, and S. Katkuri, "Graphene: efficient interactive set reconciliation applied to blockchain propagation," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 303–317.

[25] Z. Hu and Z. Xiao, "Dino: A block transmission protocol with low bandwidth consumption and propagation latency," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 1319–1328.

[26] J. Toomim, "Fast internet bitcoin relay engine (fibre). 2017. homepage," August1, 2017.

[27] U. Klarman, S. Basu, A. Kuzmanovic, and E. G. Sirer, "bloxroute: A scalable trustless blockchain distribution network whitepaper," *IEEE Internet Things J.*, 2018.

[28] M. Labs. (2019) Design and analysis of a decentralized relay network. [Online]. Available: https://www.marlin.pro/whitepaper

[29] E. Rohrer and F. Tschorsch, "Kadcast: A structured approach to broadcast in blockchain networks," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, 2019, pp. 199–213.

[30] R. Elias and T. Florian, "Kadcast-ng: A structured broadcast protocol for blockchain networks," Cryptology ePrint Archive, Report 2021/996, 2021, https://ia.cr/2021/996.

[31] K. Ayinala, B.-Y. Choi, and S. Song, "Pichu: accelerating block broadcasting in blockchain networks with pipelining and chunking," in *2020 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 2020, pp. 221–228.

[32] N. Chawla, H. W. Behrens, D. Tapp, D. Boscovic, and K. S. Candan, "Velocity: Scalability improvements in block propagation through rateless erasure coding," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2019, pp. 447–454.

[33] M. Jin, X. Chen, and S.-J. Lin, "Reducing the bandwidth of block propagation in bitcoin network with erasure coding," *IEEE Access*, vol. 7, pp. 175 606–175 613, 2019.

[34] M. F. sallal, G. Owenson, and M. Adda, "Proximity awareness approach to enhance propagation delay on the bitcoin peer-to-peer network," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 2411–2416.

[35] Y. Mao, S. Deb, S. B. Venkatakrishnan, S. Kannan, and K. Srinivasan, "Perigee: Efficient peer-to-peer network design for blockchains," in *Proceedings of the 39th Symposium on Principles of Distributed Computing*, ser. PODC '20, 2020, pp. 428–437.

[36] B. Xue, Y. Mao, S. B. Venkatakrishnan, and S. Kannan, "Goldfish: Peer selection using matrix completion in unstructured p2p network," *arXiv preprint arXiv:2303.09761*, 2023.

[37] D. Vyzovitis, Y. Napora, D. McCormick, D. Dias, and Y. Psaras, "Gossipsub: Attack-resilient message propagation in the filecoin and eth2. 0 networks," *arXiv preprint arXiv:2007.02754*, 2020.

[38] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth multicast in cooperative environments," *ACM SIGOPS operating systems review*, vol. 37, no. 5, pp. 298–313, 2003.

[39] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[40] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 129–144.

[41] J. R. Douceur, "The sybil attack," in *Peer-to-Peer Systems: First InternationalWorkshop, IPTPS 2002 Cambridge, MA, USA, March 7–8, 2002 Revised Papers 1*. Springer, 2002, pp. 251–260.

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

[43] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[44] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, "Bimodal multicast," *ACM Transactions on Computer Systems (TOCS)*, vol. 17, no. 2, pp. 41–88, 1999.

[45] B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," in *International Conference on Dependable Systems & Networks*, 2006.

[46] HotStuff, 2022. [Online]. Available: https://github.com/relab/hotstuff

[47] ReedSolomon, 2022. [Online]. Available: https://github.com/Backblaze/JavaReedSolomon

[48] N. Berendea, H. Mercier, E. Onica, and E. Riviere, "Fair and efficient gossip in hyperledger fabric," *arXiv preprint arXiv:2004.07060*, 2020.

[49] C. Stathakopoulou, M. Pavlovic, and M. Vukoli, "State-machine replication scalability made simple (extended version)," *arXiv e-prints*, 2022.

[50] Y. Amir, B. Coan, J. Kirsch, and J. Lane, "Prime: Byzantine replication under attack," *IEEE transactions on dependable and secure computing*, vol. 8, no. 4, pp. 564–577, 2010.

[51] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *International Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 53–65.

[52] V. Drabkin, R. Friedman, and M. Segal, "Efficient byzantine broadcast in wireless ad-hoc networks," in *2005 International Conference on Dependable Systems and Networks (DSN'05)*. IEEE, 2005, pp. 160–169.

[53] J. Toomim, "Benefits of ltor in block entropy encoding," 2018.

[54] M. T. Goodrich and M. Mitzenmacher, "Invertible bloom lookup tables," in *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2011, pp. 792–799.