Limitless Scalability: A High-Throughput and Replica-Agnostic BFT Consensus

Chenyu Zhang
Tianjin University
chenyu_zhang@tju.edu.cn

Xiulong Liu*
Tianjin University
xiulong_liu@tju.edu.cn

Hao Xu
Tianjin University
hao_xu@tju.edu.cn

Haochen Ren
Tianjin University
haochen_ren@tju.edu.cn

Muhammad Shahzad
North Carolina State University
mshahza@ncsu.edu

Guyue Liu
Peking University
guyue.liu@gmail.com

Keqiu Li
Tianjin University
keqiu@tju.edu.cn

***Abstract*—Traditional Byzantine Fault Tolerance (BFT) consensus protocols adopt a star topology with a leader to handle all message transmission, causing performance to degrade rapidly as replicas grow. Recently, many studies have sought to improve scalability by exploring multi-layer topology (e.g., tree structures) to reduce the leader's fanout. However, these approaches either depend on a polynomial fanout to preserve fault tolerance or are constrained by the impact of topology depth on throughput, ultimately leading to only modest scalability gains. To this end, we propose Tide, the first leader-based BFT protocol that maintains robust performance as replica count grows, which is enabled by our design of logarithmic-fanout topology and high-parallel pipelining. Tide utilizes redundant connections as a key insight in topology, reducing fanout without compromising resilience. Tide further introduces a novel pipelining where inter-layer interactions dynamically determine the degree of proposal parallelism, thereby decoupling throughput from topology depth. Real-world experiments with 100 cloud servers demonstrate that as the replica count scales from 100 to 1,000, state-of-the-art protocols experience a 65% to 90% decrease in throughput and a 50× increase in latency. In contrast, Tide maintains a replica-agnostic high throughput of around 50ktps, over 5x higher than others, while its latency remains at 0.3s-0.4s.**

I. Introduction

Byzantine Fault Tolerance (BFT) consensus [14], [56], [22] is widely utilized in distributed systems, particularly in the rapidly evolving blockchain ecosystem [12], [49], [5], [29]. BFT protocols ensure state consistency across N replicas, but their performance degrades significantly as the replica count grows, posing a major *scalability* challenge [7]. Traditional leader-based BFT protocols [54], [48], [32] employ a star topology, where a single leader communicates directly with all N replicas. As a result, the leader handles most of the communication load, while the other replicas have limited communication responsibilities. This indicates that utilizing idle replicas to balance the communication load across the system may offer opportunities for performance improvement. To this end, a recent line of work [31], [43], [36], [51], [24], [53], [11], [37] has adopted *multi-layer communication topologies*, where intermediate replicas relay messages on behalf of the leader, thereby reducing the leader's fanout. Despite this benefit, such designs introduce two key issues that must be addressed.

*Xiulong Liu is the corresponding author.

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.240101
www.ndss-symposium.org

Firstly, a smaller fanout requires involving more replicas as relays, which amplifies the impact of Byzantine faults. Therefore, the fanout must remain sufficiently large (e.g., polynomial in N) to meet fault tolerance requirements. Kauri [43], Omniledger [36], and Motor [35] utilize a tree topology in which Byzantine replicas can affect all replicas within the subtree. Therefore, they can only employ a three-layer tree with a fanout of $O(\sqrt{N})$, yet it still fails to operate under $N/3$ Byzantine faults. Crackle [51], [52] addresses this issue by frequently rotating the $\sqrt{N}$ replicas in the middle layer, but forming each QC (Quorum Certificate) incurs $O(\sqrt{N})$ communication steps. Committee consensus protocols such as Algorand [24], and [55], [39], as well as hierarchical consensus protocols like ORION [53] and [3], [38], make stronger security assumptions by requiring that the fraction of Byzantine replicas within each committee (or cluster) remains below a fixed threshold. This allows consensus to be achieved with a smaller fanout but significantly compromises fault tolerance. Tendermint [11] and Gosig [37] adopt gossip-based topologies [46], which reduce the fanout of block dissemination to $O(\log N)$. Nevertheless, the load for vote collection remains $O(N)$, since gossip operates as a unidirectional broadcast protocol.

Secondly, increasing the topology depth lengthens the message propagation path in a consensus round, which in turn increases the round latency and ultimately reduces the system throughput. In HotStuff with a star topology, a consensus round, from the leader's proposal generation to the collection of the corresponding QC, completes in just two communication steps. In comparison, protocols that employ a multi-layer topology with depth d require $2d$ communication steps to complete a consensus round. If we naively adopt the *pro-*

TABLE I: Comparison of Existing Protocols.

Protocol	**Communication Topology**				**Proposal Pipelining**		
	Fanout	$N/3$ **Fault Tolerance**	**Depth**	**Comm. Complexity per Block**	**Throughput Independent of Depth**	**Responsive Vote Collection**	**Comm. Steps per QC**
HotStuff [54]	$O(N)$	✓	1	$O(N)$	Yes	✓	$O(1)$
Hermes [31]	$O(N)$	✓	2	$O(c_1 N)$*	Yes	✓	$O(1)$
Omniledger [36]	$O(\sqrt{N})$	$\times$	2	$O(N)$	No	$\times$	$O(1)$
Kauri [43]	$O(\sqrt{N})$	$\times$	2	$O(N)$	Weak	$\times$	$O(1)$
Crackle [51]	$O(\sqrt{N})$	✓	2	$O(N)$	Yes	$\times$	$O(\sqrt{N})$
Algorand [24]	c_2*	$\times$	2	$O(N)$	Yes	✓	$O(1)$
ORION [53]	c_3*	$\times$	2	$O(N)$	Yes	✓	$O(1)$
Tendermint [11]	$O(\log N)$ / $O(N)$**	✓	$O(\log N)$ / 1	$O(N \log N)$	No	✓	$O(\log N)$
Gosig [37]	$O(\log N)$ / $O(N)$**	✓	$O(\log N)$ / 1	$O(N \log N)$	No	✓	$O(\log N)$
Tide(this paper)	$\boldsymbol{O(\log N)}$	✓	$\boldsymbol{O(\log N)}$	$\boldsymbol{O(N \log N)}$	**Yes**	✓	$\boldsymbol{O(\log N)}$

* c_1, c_2, c_3 are system parameters related to security.
** For gossip-based protocols, the fanout for block broadcast is $O(\log N)$, while vote collection still requires $O(N)$, and its depth remains 1.

posal pipelining from star-based protocols such as HotStuff, where the next round starts after completing the previous one, then even though a multi-layer topology reduces fanout, its additional layers increase the per-round latency, and thus may fail to improve throughput. Kauri [43] was the first to recognize this problem in tree-based topologies and proposed to improve proposal parallelism to mitigate the impact of topology depth on throughput. By pre-measuring network parameters, Kauri provides a solution that is well-suited for highly static environments. In contrast to Kauri, gossip-based protocols [11], [37] exhibit suboptimal performance in practice, as their throughput heavily depends on the topology depth; increasing depth significantly degrades throughput. This is because the randomized propagation prevents proposals from being disseminated in a stable and parallel manner, forcing the leader to wait for responses to a proposal from all replicas before issuing the next one. Ultimately, Gossip is a randomized, one-way broadcast protocol, which renders it unsuitable for vote-driven BFT consensus.

TABLE I summarizes the characteristics of existing protocols in terms of communication topology and pipelining, as discussed above. It can be observed that none of these protocols simultaneously achieve a logarithmic fanout topology and the pipelining design whose throughput is independent of topology depth. Consequently, their performance degrades rapidly with scale, resulting in poor scalability.

In this paper, we propose ***Tide***, the first leader-based BFT protocol that sustains high performance regardless of the number of replicas by eliminating the polynomial fanout and depth-induced throughput degradation inherent in prior multi-layer designs. Specifically, ***Tide constructs a topology with logarithmic fanout while tolerating*** $N/3$ ***Byzantine faults, and employs a highly parallel proposal pipelining that makes throughput independent of topology depth.***

Tide introduces a novel topology (left part of Fig. 1) that combines redundant inter-layer links for strong fault tolerance with geometrically expanding layers to ensure logarithmic fanout. Like tree structures, we organize replicas into multiple layers, where each layer contains κ times more replicas than the previous one. This geometric growth balances the communication load and keeps the fanout per replica small. Unlike traditional trees, however, each replica connects to ρ replicas in the upper layer (instead of just one), improving its likelihood of staying connected to the root (leader) despite Byzantine failures. Under this model (§ III-A), we qualitatively analyze how to select the ρ connections to maximize the probability that a replica remains connected to the leader, and derive the construction of the Tide Graph accordingly (§ III-B). Furthermore, we quantitatively prove that with $\kappa = 2$ and $\rho = \log_3 N + c(\varepsilon)$ ($c(\varepsilon)$ is a constant independent of N), the Tide Graph achieves the same level of availability as the star topology, i.e., tolerating up to $f = N/3$ Byzantine faults with fanout = $\rho \cdot \kappa = O(\log N)$ (§ III-C).

Building on the topology, Tide introduces a novel pipelining (right part of Fig. 1) where inter-layer interactions dynamically adjust the proposal parallelism, effectively decoupling throughput from topology depth. To enable parallel processing of multiple proposals within a multi-layer topology, rather than following HotStuff's approach of completing a consensus round of the previous proposal before moving to the next, Tide pioneers a QC-driven agreement approach (§ IV-B). Replicas maintain only three QC-related variables locally and do not track the phases (e.g., lock, commit) of individual proposals, thereby eliminating dependencies between proposals. Simultaneously, we design a passive cascade workflow (§ IV-C) that dynamically determines the frequency of parallel proposals based on inter-layer message exchanges. The replica passively waits for messages, triggering a fire-and-forget action upon reception, which generates response messages that further cascade to trigger actions in replicas across adjacent layers. This design ensures replicas (including the leader) base their decisions only on messages from neighbors.

For scalability evaluation, we performed large-scale real deployments with 100 cloud servers, comparing Tide against three state-of-the-art protocols: HotStuff [54], Kauri [43], and Crackle [51]. The results demonstrate that as the replica count increased from 100 to 1,000, the throughput of other approaches dropped significantly: HotStuff declined from 12.8k to 0.7k tps, Kauri from 20k to 2k tps, and Crackle from 28k to 10k tps – corresponding to a reduction of 65% to 90%, while the latency increased by up to 50×. In contrast, Tide can maintain a high throughput of around 50ktps with fluctuations of less than 10%, and the latency only experiences a slight increase, from 0.3s to 0.4s.

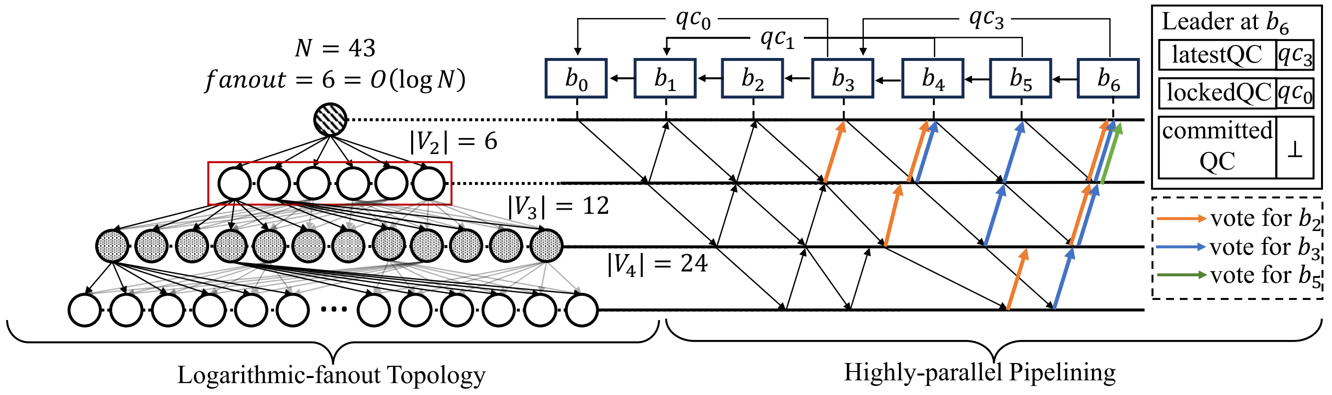


Fig. 1: Overall architecture diagram of Tide. In the topology on the left, when $\rho = 3 < \log_3 N, \kappa = 2$, the fanout $= \rho \cdot \kappa = 6$ In the pipelining on the right, after the leader sends b_5 to its 4 successors, once receives a vote for b_5 (together with vote for b_2, b_3 forming qc_2, qc_3), it proceeds to broadcast b_6, carrying the latest QC (qc_3).

Overall, the core contribution of ***Tide is achieving limitless scalability in BFT consensus, sustaining near-constant performance regardless of the number of replicas***, thanks to its logarithmic fanout topology and highly parallelized pipeline. Specifically, the advantages of Tide over the prior arts are four-fold: (i) compared with [54], [43], [51], [31], Tide achieves logarithmic fanout, ensuring the communication load of replicas remains almost unaffected by increases in the replica count; (ii) Compared with [43], [11], [37], Tide achieves throughput that is independent of topology depth, thereby allowing parallel proposals to be driven at the maximal rate supported by the system. (iii) compared with [43], [24], [53], Tide achieves $N/3$ Byzantine fault tolerance while using low fanout, ensuring that resilience is not compromised; and (iv) compared with [43], [51], Tide achieves responsiveness, reducing reliance on strong network assumptions.

II. Preliminaries

A. System Model

We describe the system model adopted by Tide, including the Byzantine fault model, partial synchrony assumption, and the setup for aggregated signature schemes.

Byzantine Fault Model. Our basic system model addresses the classical *State Machine Replication (SMR)* [44], [8] problem. We assume there are N replicas, denoted as $\mathcal{R} = \{r_1, r_2, \ldots, r_N\}$, with at most F Byzantine faulty [19] replicas, where $N = 3F + 1$, and the actual number of Byzantine replicas is f ($f \leq F$). All Byzantine replicas can behave arbitrarily without restrictions, while other correct replicas strictly adhere to the protocol.

Partial Synchrony Assumption. We use the partial synchrony assumption [21], where after an unknown bounded Global Stabilization Time (GST), messages are delivered within a known bounded time Δ. This means consensus progresses when the network is stable long enough, but consensus liveness isn't guaranteed if it fluctuates significantly (asynchronous). Similar assumptions [41] also ensure consensus liveness.

Aggregated Signature Setup. We assume a public key infrastructure (PKI) where each replica r_i possesses a private-public key pair (sk_i, pk_i), with pk_i being public. For a message m, r_i can use sk_i to generate a signature $\sigma_i(m)$, which can be verified by pk_i. Based on signature schemes like BLS [9] or Schnorr [45], multiple signatures can be aggregated into a single compact signature to reduce bandwidth overhead.

B. Tide's Architectural Distinction

We consider a standard BFT setting with clients and replicas. Clients submit transactions to replicas, and replicas run the consensus protocol to order and commit these transactions, typically by grouping them into blocks. The correctness of a BFT protocol is specified as follows:

- **Safety**: If b and b' are conflicting blocks, then they cannot be both committed, each by a correct replica.
- **Liveness**: If a transaction tx is submitted to all correct replicas, then all correct replicas will eventually commit the transaction tx.

Traditional PBFT-family protocols are typically designed under a star topology, where the leader of each view (i.e., epoch) directly communicates with all replicas and performs all coordination. Decades of research [11], [48], [32], [40], [54] have refined protocol logic, for example, reducing the number of commit phases, simplifying view change, or optimizing leader replacement. These efforts improve efficiency within the same architectural model but leave the underlying communication structure unchanged. In contrast, our work focuses not on modifying protocol rules, but on lifting existing BFT protocols to a scalable multi-layer topology, which requires redesigning communication patterns and proposal pipelining. Given its simplicity, we instantiate our design on HotStuff, but the architecture is protocol-agnostic and can be applied to other BFT protocols with minimal changes.

Tide does not alter the core design of HotStuff, including its three-phase commit, two-phase locking, and safety guarantees. Instead, its novelty lies in rearchitecting HotStuff's workflow onto a hierarchical log-fanout topology, enabling concurrent processing of multiple proposals while preserving all original correctness properties. To fully exploit this topology, Tide introduces a custom pipelining that is topology-aware, adaptive, and responsive. While Kauri can also combine pipelining with a hierarchical topology, its design depends on pre-measured environmental parameters (e.g., RTT, proposal processing time), assumes homogeneous latency, and relies on timers, making it fragile and non-responsive to dynamic network changes. In contrast, Tide's pipeline is entirely event-driven: replicas progress proposals purely upon message arrivals, without any timers, allowing automatic adjustment of concurrency levels under varying conditions. Therefore, Tide achieves high throughput independent of network depth while maintaining full responsiveness (see Table I and § IV-B).

C. Topology Availability

Redesigning the communication topology requires careful consideration of how its connectivity impacts liveness. In a star topology, this requirement is trivial: a correct leader can always directly reach all correct replicas. In a multi-layer topology, however, even if the leader is correct, its messages must traverse intermediate replicas. Faults on these routing paths can prevent proposals or votes from reaching a quorum, causing progress to stall despite a correct leader. Thus, liveness in such settings depends not only on the leader but on whether the communication topology itself remains connected in a way that allows quorum formation. To capture this more general requirement, we introduce a topology-centric notion, *available*, which characterizes whether a given view's communication structure is sufficiently connected for consensus to make progress. Intuitively, consensus can advance if and only if the topology for that view is available.

Definition 1 (Available)**.** *A communication topology for view v is said to be* **available** *if the leader in view v is correct and can successfully establish communication via the topology with at least $Q = N - F$ correct replicas. We refer to the probability that a randomly selected view is available as the* **availability** *of the topology.*

For any given topology, higher availability is naturally desirable. However, there is a theoretical upper bound given by the probability that the leader is correct, which equals $\frac{N-f}{N}$, assuming the leader is uniformly selected at random from all replicas. A smaller fanout restricts the leader's communication to fewer replicas, making it challenging to guarantee availability under higher fault thresholds. Therefore, this work aims to achieve availability close to the theoretical upper bound $\frac{N-f}{N}$, despite using only logarithmic fanout—a significant step toward improving scalability without compromising fault tolerance. Formally, for any $0 < \varepsilon \leq 1$, our design achieves availability at least $\frac{N-f}{N}(1-\varepsilon)$ with a fanout of $O(\log N) + c(\varepsilon)$, where $c(\varepsilon)$ is a constant depending solely on the desired security level ε, as shown in § III-C. For example, as shown in Table II, when $\varepsilon = 1/9$, $c(\varepsilon) = 2$; and when $\varepsilon = 1/27$, $c(\varepsilon) = 4$. This is independent of the values of N, f, and other parameters.

III. Logarithmic-fanout Topology

In this section, we introduce a logarithmic-fanout communication topology while tolerating $N/3$ Byzantine faults. We begin by constructing a topology model that formalizes the layering of replicas and their in-degree and out-degree. Based on this model, we derive the optimal interconnection strategy and propose a corresponding topology construction algorithm. Finally, we analyze the availability of our topology and show that it achieves the availability of $\frac{N-f}{N}(1-\varepsilon)$ for any $0 < \varepsilon \leq 1$ with $\rho = O(\log N) + c(\varepsilon)$, comparable to $\frac{N-f}{N}$ for the star topology, which is the most connected structure in leader-based consensus systems.

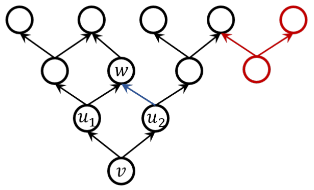
A. Topology Model

We choose a structured topology as the foundation of our design, as it better addresses the scalability and efficiency challenges inherent in randomized topologies. Although randomized communication schemes such as Gossip [46] can reduce the leader's fanout to $O(\log N)$, they lack deterministic message propagation paths, making it difficult to coordinate efficient proposal pipelining. Consequently, the proposal pace remains influenced by the entire network and cannot fully leverage the low fanout. In contrast, structured topologies define fixed propagation paths within a view, enabling predictable communication and allowing the proposal pace to be matched with the local fanout in the pipelining.

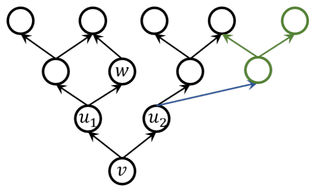
However, constructing such a topology introduces a core challenge: reducing fanout while preserving resilience. In a structured topology, a replica can influence all of its neighbors, amplifying the disruptive influence of Byzantine faults. Thus, the smaller the fanout, the more susceptible the system is to Byzantine attacks, which may prevent the leader from reaching enough correct replicas. For instance, in a star topology with fanout of N, communication remains unaffected by non-leader Byzantine replicas, as progress only depends on whether the leader is correct or not. Conversely, in a tree topology, a single Byzantine replica in an upper layer can affect all replicas in its subtree, isolating them from the leader.

In practice, all replicas are typically assigned randomly to a predefined topology for each view, and consensus proceeds depending on where the Byzantine replicas are placed. For each view, we randomly shuffle the replica set $\mathcal{R}$ into a permutation $\mathcal{R}_p = [r_{p_1}, r_{p_2}, \ldots, r_{p_N}]$ using a globally consistent random seed. For replica r_{p_i}, we map it to a vertex v_i in the graph, i.e., $r_{p_i} \longleftrightarrow v_i$. Through this one-to-one mapping, we obtain a vertex set $\mathcal{V} = \{v_1, v_2, \ldots, v_N\}$, where the number of vertices is N, with f randomly distributed fault vertices. We focus on how to establish edges between vertices in $\mathcal{V}$, forming a low-degree, highly connected graph.

We organize replicas into multiple layers, and use the set $\mathcal{V}_i$ to represent the vertices in the i-th layer. The first layer contains a single vertex ($|\mathcal{V}_1| = 1$) as the graph's root, corresponding to the leader. To reduce the fanout, we perform an exponential layering, where $|\mathcal{V}_i| = |\mathcal{V}_{i-1}| \cdot \kappa$ for $i > 2$. In terms of edge connections, we only allow connections to be established between adjacent layers. To enhance availability,



(a) v in ELG



(b) v in ELG$'$

Fig. 2: v in ELG$'$ has more predecessors at every layer.

for $\forall v \in V_i$, there can be at most ρ predecessor vertices in V_{i-1} connected to v. Therefore, the in-degree of each vertex is at most ρ, and the out-degree (fanout) is at most $\rho\kappa$. We investigate how to achieve maximum availability with the minimum fanout in this model. The formal constraints are given as follows:

Definition 2 (EP)**.** *We denote the exponential partition of vertex set* $\mathcal{V} = \{v_1, v_2, \ldots, v_N\}$ *as* $\mathrm{EP}(\mathcal{V}) = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_d\}$*, where* $d = |\mathrm{EP}(\mathcal{V})| = \lceil \log_\kappa \frac{(N-1)(\kappa-1)}{\rho\kappa} + 1 \rceil$*, and*

$$|\mathcal{V}_i| = \begin{cases} 1 & (i = 1) \\ \rho\kappa^{i-1} & (1 < i < d) \\ N - \sum_{j=1}^{d-1} |\mathcal{V}_j| & (i = d) \end{cases}$$

For simplicity, we assign the vertices in $\mathcal{V}$ *sequentially to* $\mathcal{V}_i \in \mathrm{EP}(\mathcal{V})$*, i.e.,* $\mathcal{V}_i = \{v_{s_i}, v_{s_i+1}, \ldots, v_{e_i}\}$*, where* $s_i = e_{i-1} + 1 (i > 1), e_i - s_i + 1 = |\mathcal{V}_i|$ *and for special* $s_1 = 1$*. We call the single vertex* $v_1 \in \mathcal{V}_1$ *as the* ***root****.*

Definition 3 (ELG)**.** *We denote the exponential layering graph as* $\mathrm{ELG} = \langle \mathrm{EP}(\mathcal{V}), \mathcal{E} \rangle$*, where* $\mathcal{E}$ *is the set of edges. The directed edge from vertex* u *to vertex* v *is denoted as* $(u, v) \in \mathcal{E}$*. We use* $\mathrm{Pred}(v) = \{u | (u, v) \in \mathcal{E}\}$*,* $\mathrm{Succ}(v) = \{w | (v, w) \in \mathcal{E}\}$ *to represent the predecessor and successor sets of a vertex* v*.* ELG *graph satisfies that for* $\forall\ \mathcal{V}_i \in \mathrm{EP}(\mathcal{V}) \quad \forall v \in \mathcal{V}_i$*:*

$$|\mathrm{Pred}(v)| \le \rho \ \wedge \mathrm{Pred}(v) \subseteq \mathcal{V}_{i-1} (i > 1)$$
$$\wedge\ |\mathrm{Succ}(v)| \le \rho\kappa \wedge \mathrm{Succ}(v) \subseteq \mathcal{V}_{i+1} (i < d)$$

For simplicity, if there is a mapping $\mathcal{F} : V \to \mathcal{P}(V)$ *(such as* Pred*), we use the multi-order mapping* $\mathcal{F}^0(v) = \{v\}, \mathcal{F}^1(v) = \mathcal{F}(v)$*, and for* $x > 1$*,* $\mathcal{F}^x(v) = \bigcup_{u \in \mathcal{F}^{x-1}(v)} \mathcal{F}(u)$*.*

B. Topology Construction

The ELG graph only specifies the layering of vertices and their degrees ($\rho\kappa$). We now explore how to construct the edge connections to maximize availability. Starting from the fundamental properties of the ELG graph, we will derive the characteristics that a availability-optimized connection scheme should possess. Finally, we present the construction of the Tide Graph (Definition 6) and prove that it satisfies the desired properties. Due to space constraints, some formal proofs are placed in the Appendix. We strive to provide intuitive explanations and illustrations in the main text.

First, we begin the analysis from a single vertex. For a vertex v in ELG, if there exists a path from the root to it such that all vertices along the path are not faulty, we say that v is *reachable*, and denote the probability of this event as *reachability*. Consider the following scenario: two predecessors u_1 and u_2 of v have a common predecessor w, as shown in Fig. 2a, and if w is not reachable, it will simultaneously affect both u_1 and u_2, thereby impacting v. Conversely, if u_1 and u_2 have no common predecessors, as shown in Fig. 2b, the reachability of v will be higher, since v has more paths to the leader. Therefore, the fewer the common predecessors between the predecessors of v, or in other words, the more predecessors v has at each layer, the higher its reachability will be, as proved in Theorem 4.

To maximize the reachability of each vertex and thereby enhance the availability of the topology, each vertex should be connected to the maximum theoretical number of predecessors at each layer. For $\forall v \in \mathcal{V}_i$, the number of its predecessors can grow exponentially by the factor ρ during the upward connection process, but it cannot exceed the total number of vertices at that layer. Thus, in the $(i-1)$-th layer, the maximum number of predecessors it can have is $\min(\rho^l, |\mathcal{V}_{i-l}|)$. We donate the property that every vertex has the *maximum number of predecessors* at every layer as MPRED:

Definition 4 (MPRED)**.** *For an* ELG*, we refer to the following property as* MPRED*:*

$$\forall i\ \forall v \in \mathcal{V}_i\ \forall l < i\ (|\mathrm{Pred}^l(v)| = \min(\rho^l, |\mathcal{V}_{i-l}|))$$

If an ELG does not satisfy MPRED, an alternative edge connection set must exist that increases the reachability for at least one vertex. In other words, there must exist another graph ELG$'$, in which there exists a vertex u such that the reachability of u in ELG$'$ is higher than in ELG, as proved in Theorem 5.

However, MPRED is a bottom-up property, which means achieving this property during a top-down connection process is difficult. Therefore, we introduce a symmetric property, MSUCC, which means that every vertex has the *maximum number of successors* at every layer.

Definition 5 (MSUCC)**.** *For an* ELG*, we refer to the following property as* MSUCC*:*

$$\forall i\ \forall v \in \mathcal{V}_i\ \forall l \le d - i\ (|\mathrm{Succ}^l(v)| = \min((\rho \cdot \kappa)^l, |\mathcal{V}_{i+l}|))$$

These two symmetric properties, MPRED and MSUCC, both represent the establishment of connections between a vertex and as many other vertices as possible. Therefore, they are essentially equivalent, as proven in Theorem 6. So we only need to focus on how to satisfy MSUCC when constructing the topology. Unlike MPRED, MSUCC is a top-down property so that we can ensure each layer maintains the maximum number of successors during the top-down connection process, ultimately achieving MSUCC.

Supposing graph ELG satisfies MSUCC, it means that for $\forall v \in \mathcal{V}_i$, the successors of v at any layer do not overlap ($|\mathrm{Succ}^l(v)| = (\rho \cdot \kappa)^l$) until their total number fully occupies an entire layer ($|\mathrm{Succ}^l(v)| = |\mathcal{V}_{i+l}|$). Since there are always connections between adjacent layers, if $|\mathrm{Succ}^l(v)| = |\mathcal{V}_{i+l}|$, then it must follow that $|\mathrm{Succ}^{l+1}(v)| = |\mathcal{V}_{i+l+1}|$. Therefore,

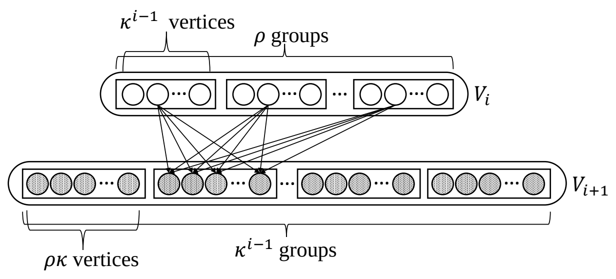


Fig. 3: Construction of the Tide Graph

we primarily focus on how to ensure $|\text{Succ}^l(v)| = (\rho \cdot \kappa)^l$ when $(\rho \cdot \kappa)^l < |\mathcal{V}_{i+l}|$. To achieve this, when expanding the successors of v downward, we only need to maintain the following two properties to ensure that v has no overlapping successors at each layer: (1) All of its successors at each layer are contiguous. (2) The successors of contiguous vertices are completely distinct.

Finally, our design is as follows: for two adjacent layers $\mathcal{V}_i$ and $\mathcal{V}_{i+1}$, we divide the vertices in $\mathcal{V}_i$ into ρ groups and the vertices in $\mathcal{V}_{i+1}$ into k groups. Subsequently, we connect the x-th vertex in each group of $\mathcal{V}_i$ to the x-th group in $\mathcal{V}_{i+1}$, as shown in Fig. 3. As a result, all vertices within the same group in $\mathcal{V}_i$ have distinct successors, while the successors of each individual vertex are contiguous. This satisfies properties (1) and (2) and ensures that the same properties hold at every subsequent layer, ultimately achieving MSUCC, as proven in Theorem 9. The formal definition is as follows:

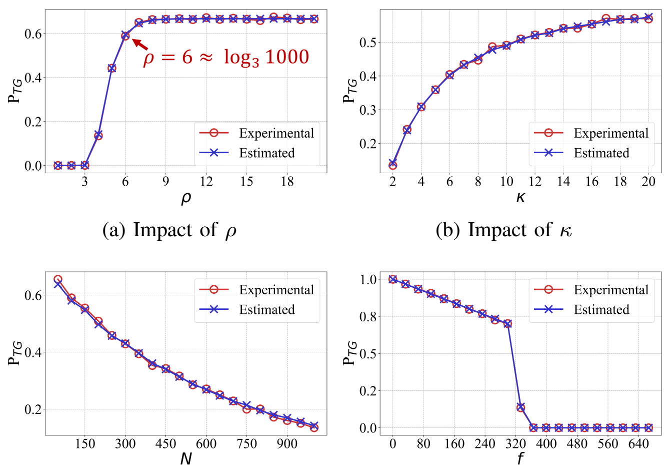
Definition 6 (TG)**.** *We refer to a graph* $\text{ELG} = \langle \text{EP}(\mathcal{V}), \mathcal{E}\rangle$ *as a **Tide Graph**, denoted as* $\text{TG} = \langle \text{EP}(\mathcal{V}), \mathcal{E}\rangle$*, if it satisfies: for* $\forall v_x \in \mathcal{V}_i \quad \forall v_y \in \mathcal{V}_{i+1}$

$$(v_x, v_y) \in \mathcal{E} \iff (x - s_i) \mod \kappa^{i-1} = \lfloor \frac{y - s_{i+1}}{\rho\kappa} \rfloor$$

C. Analysis of Availability

Although we have proven that the construction of TG satisfies the optimal property, this does not necessarily imply that its overall availability is sufficiently high. Therefore, we conduct a probabilistic estimation analysis to evaluate its availability. In our calculations, we will use some approximations, ultimately deriving an estimation formula, denoted as $\hat{\text{P}}_{\text{TG}}(N, f, Q, \rho, \kappa)$, which reflects the influence of five parameters on availability. Furthermore, we prove that when $\kappa = 2$ and $\rho = \log_3 N + \log_{1/3} \varepsilon$ (fanout = $\rho \cdot \kappa$), $\hat{\text{P}}_{\text{TG}} \geq \frac{N-f}{N}(1-\varepsilon)$ for any $0 < \varepsilon \leq 1$, indicating that the availability of the TG is on the same order of magnitude as that of a star topology in the worst case.

In large-scale samples, the difference between sampling with or without replacement is negligible [42]. Therefore, we approximate the probability of a vertex being faulty as $\hat{p}_f = \frac{f}{N}$, and assume it's independent between vertices. Since TG satisfies the MPRED property, the predecessors of vertices expand rapidly upwards. Therefore, we assume that the reachability of a vertex primarily depends on whether all



(a) Impact of ρ (b) Impact of κ

(c) Impact of N (d) Impact of f

Fig. 4: Estimated Values vs. Experimental Results.

the vertices in its first-order predecessors are faulty. Excluding the effect of whether the vertex itself is faulty, we assume that a vertex is reachable if not all of its first-order predecessors are faulty. The probability of this event is denoted as $\hat{p}_a = 1 - \hat{p}_f^{\rho}$. Furthermore, by observing the connection structure of TG, we can see that the vertices in each layer are divided into groups of size $s = \rho\kappa$. The predecessors of the vertices within each group are identical, thus their reachability is deterministically related. Therefore, when the root vertex can connect to x vertices, it is connected to x/s groups. Additionally, since the probability of a vertex being faulty is $\hat{p}_f$, $x/(s \cdot (1 - \hat{p}_f))$ groups need to be connected in order to reach x non-faulty vertices. We assume that each group has reachability $\hat{p}_a$ and that the reachability of different groups is independent, because the predecessors of any two groups have minimal overlap, which significantly reduces the likelihood of correlated failures. Consistently, the resulting probabilistic estimates match our experimental observations. The number of groups the root can connect to thus becomes a binomial distribution problem, where the probability of connecting to i groups is $\hat{p}_a^i \cdot (1 - \hat{p}_a^i)^{\frac{N}{s} - i}$. As mentioned earlier, to connect to Q non-faulty vertices, at least $Q/(s \cdot (1 - \hat{p}_f))$ groups need to be connected. Finally, we obtain the calculation method for $\hat{\text{P}}_{\text{TG}}$: with the condition that the root is not faulty, the probability of connecting to $Q/(s \cdot (1 - \hat{p}_f)) \leq i \leq N/s$ groups of vertices, i.e.,

$$\hat{\text{P}}_{\text{TG}} = (1 - \hat{p}_f) \cdot \sum_{i=Q/(s\cdot(1-\hat{p}_f))}^{N/s} \binom{\frac{N}{s}}{i} \hat{p}_a^i \cdot (1 - \hat{p}_a)^{\frac{N}{s} - i}$$

$$= \frac{N - f}{N} \sum_{i=\frac{QN}{\rho\kappa(N-f)}}^{\frac{N}{\rho\kappa}} \binom{\frac{N}{\rho\kappa}}{i} (1 - (\frac{f}{N})^{\rho})^i ((\frac{f}{N})^{\rho})^{\frac{N}{\rho\kappa} - i}$$

We validate the accuracy of our estimation through Monte Carlo simulation. As shown in Fig. 4, the blue line represents the estimated probability calculated using the formula, while the red line shows the results obtained from 100 million simulation experiments. We use $N = 1000, f = 333, Q = 667, \rho = 4, \kappa = 2$ as default values and vary one parameter at a time to observe its impact. The experimental results show that our estimation is very accurate.

TABLE II: The minimum ρ required for a given ε to satisfy $\hat{\mathrm{P}}_{\mathrm{TG}} > \frac{N-F}{N}(1-\varepsilon)$, under $N = 100$ and $N = 1000$.

ε	1/3	1/5	1/9	1/27	1/81	1/100	1/243	1/1000
$1-\varepsilon$	0.667	0.8	0.889	0.963	0.988	0.99	0.996	0.999
$\log_{1/3}\varepsilon$	1	1.465	2	3	4	4.192	5	6.288
$\frac{N-F}{N}(1-\varepsilon)$	0.444	0.533	0.593	0.642	0.658	0.660	0.664	0.666
$N = 100$ **and** $\log_3 N \approx 4.192$								
$\log_{\frac{1}{3}}\frac{\varepsilon}{N}$	5.192	5.657	6.192	7.192	8.192	8.384	9.192	10.480
ρ	4	4	5	6	6	7	7	8
$\hat{\mathrm{P}}_{TG}$	0.581	0.581	0.644	0.663	0.663	0.666	0.666	0.666
$N = 1000$ **and** $\log_3 N \approx 6.287$								
$\log_{\frac{1}{3}}\frac{\varepsilon}{N}$	7.288	7.753	8.288	9.288	10.288	10.480	11.288	12.575
ρ	6	6	6	7	8	8	9	10
$\hat{\mathrm{P}}_{TG}$	0.596	0.596	0.596	0.646	0.661	0.661	0.665	0.666

We now proceed to relax the expression in order to obtain a tractable lower bound. Noting that the formula contains a cumulative summation, we scale it by taking only the last term at $i = \frac{N}{\rho\kappa}$, which gives:

$$\hat{\mathrm{P}}_{\mathrm{TG}} \geq \frac{N-f}{N}(1-(\frac{f}{N})^{\rho})^{\frac{N}{\rho\kappa}}$$

Equality in the above inequality holds if and only if $f = F$ as $Q = N - F$, which corresponds to the worst-case scenario with $N/3$ Byzantine faults, and the system achieves its minimum availability. Based on the Maclaurin expansion of $(1-x)^{\alpha}$ with $0 \leq x \leq 1, \alpha > 0$, we have $(1-x)^{\alpha} \geq 1 - \alpha x$. Substituting $x = (\frac{f}{N})^{\rho}$ and $\alpha = \frac{N}{\rho\kappa}$, we obtain:

$$\hat{\mathrm{P}}_{\mathrm{TG}} \geq \frac{N-f}{N}(1-(\frac{f}{N})^{\rho} \cdot \frac{N}{\rho\kappa})$$

To further simplify, we use the setting in BFT, i.e., $N = 3F+1$ and $f \leq F$, which gives:

$$\hat{\mathrm{P}}_{\mathrm{TG}} \geq \frac{N-f}{N}(1-(\frac{F}{N})^{\rho} \cdot \frac{N}{\rho\kappa}) \geq \frac{N-f}{N}(1-(\frac{1}{3})^{\rho} \cdot \frac{N}{\rho\kappa})$$

Now, the estimation formula becomes very clear: the influence of N is linear, κ has an inverse proportional effect, and ρ has an exponential impact. The degree of TG is $\rho\kappa$, which determines the performance of the graph. While κ has a smaller impact on the probability, its effect on the graph's degree is similar to that of ρ. Therefore, we select it as the smallest usable constant, i.e., $\kappa = 2$. Next, for any $0 < \varepsilon \leq 1$, we can choose $\rho = \log_3 N + \log_{\frac{1}{3}} \varepsilon$ to ensure that $\hat{\mathrm{P}}_{\mathrm{TG}}$ is sufficiently large:

$$\begin{aligned}
&\hat{\mathrm{P}}_{\mathrm{TG}}(N, f \leq F, Q = N-F, \rho = \log_3 N + \log_{\frac{1}{3}} \varepsilon, \kappa = 2) \\
&\geq \frac{N-f}{N}(1-(\frac{1}{3})^{\log_3 N + \log_{\frac{1}{3}} \varepsilon} \cdot \frac{N}{\rho\kappa}) \\
&= \frac{N-f}{N}(1-\frac{\varepsilon}{N} \cdot \frac{N}{\rho\kappa}) = \frac{N-f}{N}(1-\frac{\varepsilon}{\rho\kappa}) > \frac{N-f}{N}(1-\varepsilon)
\end{aligned}$$

We note that the scaling step uses the inequality $\rho\kappa \geq 1$, and the relaxation becomes looser as ρ increases. Therefore, it suffices to choose $\rho < \log_3 N + \log_{\frac{1}{3}} \varepsilon$ to ensure that $\hat{\mathrm{P}}_{\mathrm{TG}} > \frac{N-f}{N}(1-\varepsilon)$. TABLE II reports the minimum ρ required for a given ε, under $N = 100$ and $N = 1000$. The results show that even when $N = 1000$, choosing $\rho = 6 \approx \log_3 N$ already ensures a TG availability of $\hat{\mathrm{P}}_{\mathrm{TG}} = 0.596$, which is close to the star topology's 0.66. If higher availability is desired, it suffices to increase ρ by a small constant. For practical parameter selection, one can substitute the known system parameters (e.g., $\kappa = 2, N = 100, F = 33, Q = 67$) into the estimated availability equation and incrementally increase ρ until $\hat{P}_{\mathrm{TG}} > \frac{N-F}{N}(1-\varepsilon)$. As shown in Table II, for $\varepsilon = 0.2, \rho = 4$ satisfies this condition, corresponding to a total fanout of $\kappa\rho = 8$ and an availability of $0.583 > \frac{2}{3}(1-0.2) = 0.533$. In other words, the topology achieves more than 80% of the availability of a star topology—the highest achievable level under the same system parameters. It is worth emphasizing that ε and N are independent variables. The parameter ε reflects the security level, i.e., how close the achieved availability is to that of a star topology. According to our theoretical analysis, increasing ρ by only a constant amount is sufficient to compensate for an exponential decrease in ε. For example, when ε decreases from $\frac{1}{9}$ to $\frac{1}{27}$, a threefold tightening of the availability gap, ρ only needs to increase by 1 to maintain the required security property.

In summary, setting $\rho = O(\log N)$ and $\kappa = 2$, which results in a fanout of $\rho\kappa = O(\log N)$, suffices to ensure that the availability of the topology in Tide reaches the same level as that of the star topology. This demonstrates that we can achieve logarithmic fanout without sacrificing the $N/3$ fault tolerance guarantee.

D. Practicality and Security Discussion

We now analyze the complexity of constructing the topology in practice. Although the topology may appear intricate, it only increases local computation without affecting communication complexity. In a star-based topology, for any view v, all replicas obtain a globally consistent random seed $seed_v$, used to select the leader. This randomness can be produced by a simple pseudorandom generator [27] or by a stronger source such as a distributed randomness beacon [16] or a Rabin dealer [34], ensuring globally verifiable and unbiased randomness. Replicas can derive $seed_v$ before entering view v, allowing them to locally determine their communication partners with only $O(1)$ computation, effectively forming a simple star-shaped topology. In contrast, in our design, each replica r_i performs $O(N)$ local computation before entering view v to derive a permutation from $seed_v$, which determines its position in the multi-layer topology without requiring additional communication. Importantly, the shape of the topology does not need to be recomputed for every view, as it depends only on the fixed parameters N, ρ, and κ. It can be constructed once during system initialization with a cost of $O(\rho\kappa N)$. Overall, the topology construction introduces only $O(N)$ local computation per view, a negligible cost in practice, since N is typically at most a few thousand.

Next, we discuss the potential security risks introduced by our multi-layer topology. While Byzantine attacks targeting non-leader replicas may influence consensus progress, we show that our topology preserves security comparable to that of a star topology. First, the communication topology does not affect safety: it only determines communication paths and does not change the commit rule. In the worst case, it may delay progress but cannot violate safety. Second, regarding liveness, we have shown earlier that the availability of our topology matches that of a star topology when Byzantine replicas are randomly distributed. This is ensured by assuming static Byzantine faults (consistent with most BFT protocols) and by randomly placing replicas in the topology for each view. Even under dynamic Byzantine attacks, a replica remains unaffected unless all of its ρ predecessors are simultaneously compromised. Even if such an attack succeeds, it only disrupts the current view by causing a timeout; the topology is rebuilt in the next view, rendering the attack ineffective. If an adversary were powerful enough to compromise ρ replicas in every view, directly attacking the leader would be an even simpler and more effective approach—one that no consensus protocol can prevent. Overall, although Tide introduces slightly higher security risks than HotStuff, these do not compromise safety and only minimally affect liveness.

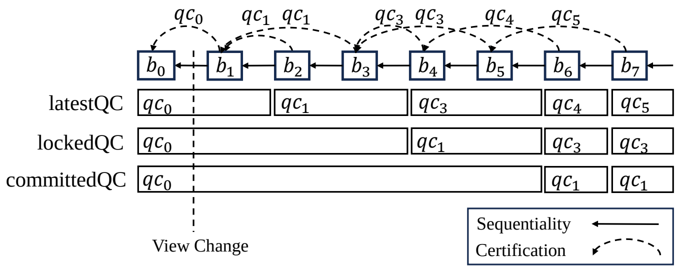
IV. Highly-parallel Pipelining

In this section, we introduce Tide's proposal pipelining, emphasizing the key differences from HotStuff and the additional considerations imposed by our multi-layer topology. Unlike HotStuff, Tide employs a QC-driven workflow to enable parallel processing of multiple proposals. As a trade-off, each view must be sufficiently long to allow messages to fully propagate through the multi-layer topology, ensuring that all correct replicas can participate in the pipeline. To achieve responsive and adaptive message propagation, we design a passive cascade workflow that dynamically adjusts the frequency of parallel proposals based on inter-layer message exchanges. Finally, we provide proofs of safety and liveness, extending HotStuff's guarantees to account for our topology and pipelining.

A. Data Structures

View&Seq. Like most previous works, we use $view$, an auto-increment positive integer, as the fundamental unit for advancing consensus. For a given $view$, all replicas can obtain a globally consistent topology (refer to § III), with the root of the topology serving as the leader. In a view, the leader can generate multiple proposals, distinguished by the sequence number seq. All data types, including $block$, $vote$, and qc, use a pair of $\langle view, seq\rangle$ to identify their temporal order. [1]

[1]For such two objects, obj_1 and obj_2, if they satisfy $obj_1.view > obj_2.view$ or $obj_1.view = obj_2.view \wedge obj_1.seq > obj_2.seq$, we say that obj_1 is more recent than obj_2.



qc_0
qc_1 qc_1
qc_3 qc_3
qc_4
qc_5
b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7
latestQC qc_0 qc_1 qc_3 qc_4 qc_5
lockedQC qc_0 qc_1 qc_3 qc_3
committedQC qc_0 qc_1 qc_1
View Change
Sequentiality
Certification

Fig. 5: QC-Driven Agreement of Tide.

Block. A block serves as an input proposal to the consensus system, denoted as $block = \langle view, seq, parent, qc, tx\_set\rangle$. The blocks form a chain structure, indexed by a unique $parent$. If $block_1$ is a direct or indirect parent of $block_2$, we refer to $block_1$ as an ancestor of $block_2$. $block.qc$ refers to a quorum certificate for an ancestor block. The tx_set maintains an ordered set of transactions, representing the full payload of the proposal.

Aggregated Vote. $av = \langle block, signature, replicas\rangle$ represents the aggregation of votes from all replicas in the set $av.replicas$ for the voting object $av.block$, and $av.signature$ represents the aggregated signature, which remains of constant size. The $view$ and seq of av are inherited from $av.block$. We group multiple av in a *map* for propagation, using $av.block$ as the key, and denote it as avm.

Quorum Certificate. When the number of voters in an aggregated vote av exceeds the quorum size Q, we term it a Quorum Certificate (QC), denoted as qc. This implies that a sufficient number of votes have been cast for $av.block$, forming an undeniable authentication.

State Variables. Each replica r_i locally stores three special QCs, which are key factors in the consensus process: $latestQC$, $lockedQC$, and $committedQC$, representing the latest progress, the locked position, and the committed block, respectively. In addition, $localAVM$ represents all buffered votes that the replica has not yet forwarded locally, and $latestBlock$ is the latest block received by the replica.

B. QC-driven Agreement

Previous works have employed a phase-driven approach, where the progress of consensus is determined by the phase a proposal has reached. In this model, the role of the QC is limited to advancing a proposal to the next phase. However, this design introduces a critical bottleneck in pipelined protocols: the next proposal must wait for the QC of the previous proposal before it can be initiated, resulting in prolonged intervals between consecutive proposals and causing replicas to remain idle. To address this issue, we propose a novel QC-driven agreement that only tracks the progress of consensus by maintaining the states of three QCs: $latestQC$, $lockedQC$, and $committedQC$, as illustrated in Fig. 5. Similar to traditional designs, a block still requires a three-hop QC to be committed. However, a fresh proposal only needs to include the latest QC known to the leader (e.g., in Fig. 5, block b_3 carries qc_1). This design eliminates the rigid constraint

Algorithm 1 QC-driven Agreement (for replica r_i)

```
 1: function ProposalRule()
 2:     $view \leftarrow curView$,        $curSeq \leftarrow curSeq + 1$
 3:     $parent \leftarrow latestBlock$,    $qc \leftarrow latestQC$
 4:     return $\langle view, curSeq, parent, qc, txSet \rangle$
 5: end function
 6: function VoteRule($block$)
 7:     if $lockedQC.block$ is an ancestor of $block$
 8:         or $block.qc$ is more recent than $lockedQC$
 9:     then return true    else return false
10: end function
11: function AdvanceRule($qc$)
12:     // Receives new qc: one-hop qc
13:     if $qc$ is more recent than $latestQC$
14:     then $latestQC \leftarrow qc$ else return
15:     // Lock rule: two-hop qc
16:     if $latestQC.view == latestQC.block.qc.view$ then
17:         $lockedQC \leftarrow latestQC.block.qc$
18:     else return
19:     // Commit rule: three-hop qc
20:     if $lockedQC.view == lockedQC.block.qc.view$ :
21:         $commitedQC \leftarrow lockedQC.block.qc$
22:         commit $commitedQC.block$ and all its ancestors
23: end function
```

between consecutive proposals, enabling unrestricted parallel processing of proposals. We describe the QC-driven agreement using three core rules in detail, as illustrated in Algorithm 1.

Proposal Rule. A fresh block inherits from the current $latestBlock$ and carries the $latestQC$, with the sequence $curSeq+1$. In this way, the leader generates a chain of blocks within a single view, with their seq increasing sequentially. Each block carries the latest QC, enabling rapid multi-hop QC propagation to advance consensus.

Vote Rule. When receiving a new $block$, it is necessary to determine whether to vote for it. Suppose the $block$ inherits from $lockedQC.block$, it indicates that the $block$ is consistent with the current locked state and complies with safety rules, thus it can be voted for. Besides, if the $block$ carries a QC that is more recent than the local $lockedQC$, it implies that other replicas have formed a conflict QC without the participation of r_i. This indicates that the number of correct replicas locked on the same path with $lockedQC$ is less than $F+1$, thus, the locked path of r_i is deprecated, and it can vote for the $block$ on a new path, thereby ensuring liveness.

Advance Rule. Upon receiving a new qc, the local $latestQC$ is updated first. Then, if the $latestQC$ forms a two-hop within the current view, the $lockedQC$ is updated. The corresponding block can be committed if the $latestQC$ forms a three-hop within the current view. Since we ensure that the newly generated block always carries the latest QC, the local updates to all QC states are linearly incremental, allowing the consensus to progress step by step.

Algorithm 2 Passive Cascade Workflow (for replica r_i)

```
 1: ▷ Network Message Triggered
 2: procedure ProcessBlock($block$)
 3:     if $block$ has been received then return
 4:     AdvanceRule($block.qc$)
 5:     if ¬ VoteRule($block$) then return
 6:     $latestBlock \leftarrow block$
 7:     ProcessAVM($r_i$ vote for $block$)
 8:     send $localAVM$ to all predecessors // Feedback Vote
 9:     $localAVM = \perp$
10:     send $block$ to all successors // Distribute Block
11: end procedure
12: procedure ProcessAVM($avm$)
13:     $localAVM \leftarrow MergeAVM(localAVM, avm)$
14:     if $r_i$ is not Leader then return
15:     $newQC \leftarrow$ GetQCFromAVM($localAVM$)
16:     AdvanceRule($newQC$)
17:     if received votes for $latestBlock$ then
18:         $newBlock \leftarrow$ ProposalRule()
19:         ProcessBlock($newBlock$)
20: end procedure
21: ▷ Local Timer Triggered
22: procedure ProcessTimeout()
23:     // Enter View Change
24:     $curView = curView + 1,\ curSeq = 0$
25:     Get Topology for $curView$
26:     // Send $newView$ message
27:     Send $\langle curView, r_i, latestQC \rangle$ to Leader in $curView$
28:     if $r_i$ is Leader for $curView$
29:         wait for $Q$ $newView$ messages
30:             AdvanceRule($newView.latestQC$)
31:         goto line 18 // Generate first block for $curView$
32: end procedure
```

C. Passive Cascade Workflow

Ensuring security while parallelizing proposal generation is only the first step. The timing of when the leader generates the next proposal, how replicas forward the proposal, and how votes are collected are equally important issues, as they will determine the final performance. We design a passive cascade workflow that enables the frequency of parallel proposals to adaptively match the fanout of the multi-layer topology. The replica passively waits for messages, triggering a fire-and-forget action upon reception, which generates response messages that further cascade to trigger actions in replicas across adjacent layers. This ensures that replicas make decisions based solely on messages from neighbors, making throughput independent of topology depth. Specifically, each replica only requires two procedures to handle all incoming blocks and votes (avm) messages from the network, and one additional procedure to handle local timeout, as shown in Algorithm 2. Among these, only the View Change (Line 23 in Algorithm 2) is triggered by a timer, which ensures that our pipelining remains responsive.

Process Block. When replica r_i receives a $block$ from a predecessor replica, it first checks if the $block$ has been previously received. Given that r_i has multiple predecessors, receiving the same block from different predecessors multiple times is normal, but r_i only processes it the first time. If it is a new block, it indicates progress at higher-level replicas, prompting r_i to advance. At this point, r_i sends its buffered votes $localAVM$ to all its predecessors, and broadcasts this new $block$ to all its successors.

Process AVM. When r_i receives a set of votes avm, it first merges avm with its $localAVM$. If r_i is the leader, it checks whether a new QC has been generated in $localAVM$, thereby advancing the consensus. At the same time, if votes for the $latestBlock$ from other replicas are received, a new block will be generated, which serves as the source of all blocks.

Process Timeout. Considering that we are using a partially synchronous model, we assume that each replica has a timer. If the $latestQC$ is not updated within a time interval Δ, a view change will be triggered. Replica r_i enters the next view and sends a $newView$ message to the next leader, which includes the $latestQC$ recorded by r_i. As the new leader, upon receiving Q $newView$ messages, the leader gathers the $latestQC$ and initiates the new view.

D. Safety and Liveness

Theorem 1. *(Safety) If b and b' are conflicting blocks, then they cannot be both committed, each by a correct replica.*

Proof. Unlike HotStuff, Tide allows multiple proposals to occur in parallel within the same view. To maintain safety in this setting, we assign a unique seq number to each proposal within a $view$, which totally orders the proposals. For each pair $(view, seq)$, at most one QC can be generated, since forming a QC requires $2F + 1$ votes and each correct replica can cast at most one vote for a given $(view, seq)$. This ensures that conflicting QCs cannot arise for the same sequence number, and thus the possibility of conflicting commits due to parallel proposals is prevented.

Without loss of generality, we can assume that b is committed by a correct replica and focus on discussing the following two cases. Case (1): $b.view = b'.view \wedge b.seq \leq b'.seq$. Among the ancestor blocks (include itself) of b', there must exist a block b^* such that $b^*.seq = b.seq$, and b^* conflicts with b. If b gets a valid QC, it indicates that at least $F + 1$ correct replicas have voted for it. Therefore, at most F byzantine replicas and the remaining F correct replicas could vote for b^*, making it impossible to produce a QC for b^*. Therefore, b^* and its descendant blocks cannot be committed. Case (2): $b.view < b'.view$. If at the end of $b.view$, b is committed by at least one correct replica, it implies that at least $F+1$ correct replicas have their $lockedQC.block$ as a descendant of b, and they will not vote for any block that conflicts with b. After this point, no new QC can be generated for any block that conflicts with b. Therefore, in $b'.view$, no conflicting block can be committed. □

Lemma 2. *After GST, there exists a bounded time period T_f such that if all correct replicas remain in view v during T_f and the topology for view v is available, then a decision is reached.*

Proof. At any given moment, for any correct replica, if Algorithm 1 line 17 is executed such that qc^* becomes its $lockedQC$, then there must be $F + 1$ correct replicas that have voted for qc^*, which implies that Algorithm 1, line 14 has also been executed. Therefore, the $latestQC$ of these $F+1$ replicas will certainly be more recent than qc^*, and at least one of them will be received by the new leader in view v, as the leader needs to collect Q $newView$ messages. As a result, once the leader derives the most recent QC from all $newView$ messages and proposes block b, the logical condition in line 8 of Algorithm 1 will always be satisfied, prompting all replicas to vote for b.

Since we use a more complex topology, we must prove that blocks can be disseminated and votes can be collected when the communication topology is available. First, among the leader's successors, there is at least one available replica, denoted as r^*. After the leader sends a block b to r^*, r^* will vote for b and return the vote to the leader. Once the leader receives this vote, the next block will be proposed. By continuously repeating this process, the leader will keep proposing blocks and disseminating them. From the perspective of block distribution, since any replica will immediately forward a received block to all of its successor replicas, the blocks proposed by the leader are guaranteed to be delivered to all available replicas. From the perspective of vote propagation, since a replica will forward all collected votes to its predecessor replicas upon receiving a new block, any vote will eventually be relayed back to the leader, driven by the continuous flow of block messages. In summary, unlike most other works that rely on time-based waiting within multi-layer topologies, we use block messages as the driving force. This ensures the distribution of blocks and the propagation of votes, ultimately guaranteeing the responsiveness of consensus. □

Theorem 3. *(Liveness) If a transaction tx is submitted to all correct replicas, then all correct replicas will eventually commit the transaction tx.*

Proof. Using a view-synchronization mechanism [10], or a simpler approach in PBFT [14] and HotStuff [54], where exponential back-off is used to adjust timeouts, we can ensure that all correct replicas will eventually have at least any bounded time period T_f overlap within the same view. On the other hand, given that an available topology exists, traversing all possible topologies will eventually yield an available one. In practice, selecting a topology at random is sufficient: according to the analysis in Section III-C, the probability of choosing an available topology is at least $\frac{N-F}{N}(1 - \varepsilon)$. At this point, the conditions of Lemma 2 are satisfied, and the correct leader will include the transaction tx in a block, ensuring that all correct replicas eventually commit tx. □

V. Optimization & Simulation

In this section, we introduce two practical optimizations. The first is to achieve improved resilience by utilizing idle replicas in the bottom layer without increasing fanout. The second is to achieve adaptive resilience by dynamically adjusting fanout based on the actual number of Byzantine faults.

Improved Resilience. There is a load imbalance in the topology introduced earlier, with the bottom-layer replicas largely idle. In the topology, all vertices have the same in-degree, which is ρ, but their out-degrees vary. The out-degree of vertices in the bottom layer (V_d) is 0, while the out-degree of other replicas is $\rho\kappa$. When $\kappa = 2$, the number of bottom-layer vertices is $N/2$, which indicates potential for improvement.

We introduce an additional redundancy parameter α, which indicates the number of times each replica appears in the topology. α ranges from 1 to κ. When $\alpha = 1$, it represents the topology structure we discussed earlier. When α is greater than 1, we attach the replicas that appear multiple times to the bottom layer of the topology. When $\alpha > 1$, we attach the repeatedly occurring replicas to an additional layer below the bottom layer. Since we use exponential layering, this additional layer can accommodate at most $(\kappa-1)\cdot N$ vertices. Since each replica appears α times, their in-degree increases from ρ to $\rho\alpha$, while the maximum out-degree remains $\rho\kappa$. The advantage of this method is that it does not require modifying the topology construction algorithm; only the input needs to be adjusted. In Section III-B, we generate one random permutation of R to construct the topology. Now, we generate α random sequences. The j-th ($1 \leq j \leq \alpha$) sequence is denoted as $\mathcal{R}_p^j = [r_{p_1}^j, r_{p_2}^j, \ldots, r_{p_N}^j]$. Use mapping $r_{p_i}^j \longleftrightarrow v_{(j-1)*N+i}$ to get vertex set $\mathcal{V} = \{v_1, v_2, \ldots, v_{\alpha N}\}$.

Adaptive Resilience. The larger the redundancy parameters ρ and α, the stronger the fault tolerance of the topology. However, this also increases the actual communication volume, as each replica forwards messages to ρ targets. During the actual operation of the system, only the actual number of Byzantine replicas f, is uncertain, which in turn affects the availability of the topology. Therefore, our approach is to dynamically adjust the topology parameters during system operation based on the actual consensus success rate. Using hypothesis testing, such as the Z-test[42], we can estimate the current topology's availability (donated as P_{TG}) by statistically analyzing the availability frequency of the topology over a past period (reflected by whether consensus can progress). For example, we can set the confidence threshold to 90%, aiming to adjust P_{TG} to lie between 0.5 and 0.8. If we believe that $P_{\text{TG}} < 0.5$, we increase the parameters; if we believe that $P_{\text{TG}} > 0.8$, we decrease the parameters.

Due to the minimal impact of the parameter κ on P_{TG} but its significant influence on communication complexity, we do not adjust κ; this paper defaults to using $\kappa = 2$. The impact of ρ is greater than that of α, so we prioritize adjusting α. Meanwhile, both ρ and α have clear upper and lower bounds, specifically $1 \leq \rho \leq \log_3 N$ and $1 \leq \alpha \leq \kappa$. When the parameters take the lower bound, the topology is a simple κ-ary tree; while the upper bound of the parameters is derived in III-C, ensuring the availability. Our specific rules are as follows. When increasing the parameters, if $\rho = \log_3 N$, we do nothing; else if $\alpha = \kappa$, we adjust $\rho = \rho+1$ and reset $\alpha = 1$; else we adjust $\alpha = \alpha + 1$. When decreasing the parameters, if $\rho = \alpha = 1$, we do nothing; else if $\alpha = 1$, we adjust $\rho = \rho - 1$ and reset $\alpha = \kappa$; else we adjust $\alpha = \alpha - 1$.

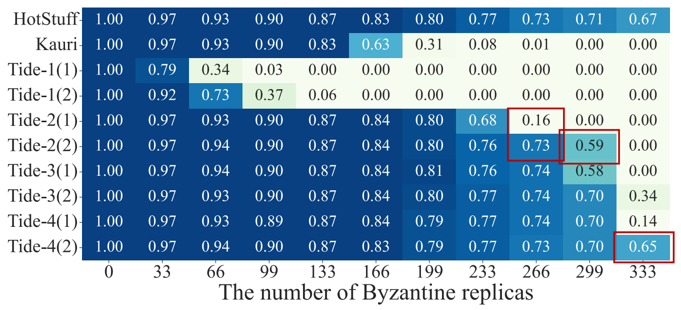


Fig. 6: The availability for different topologies, with varying numbers of Byzantine replicas. On the vertical axis, different parameters are represented in the form Tide-$\rho(\alpha)$. For example, Tide-3(2) represents ρ = 3, α = 2.

Simulation For Availability. We evaluate different values of ρ, α, and f to observe the effects of the two optimizations. As shown in Fig. 6, we plot a heatmap where the values represent the availability statistically recorded in 1,000,000 simulation experiments. We select a fixed $N = 1000$ as an example, where the horizontal axis represents f, and the vertical axis represents different topologies. The first row represents the star topology, exemplified by HotStuff. It is the simplest topology, and its availability (donated as P_{HS}) is the highest, as it is only dependent on whether the leader is a Byzantine replica. As f increases, $P_{HS} = \frac{N-f}{N}$ decreases linearly. This is the theoretical upper bound for all topologies. The second row represents the tree topology, exemplified by Kauri, which has poor resilience. The availability of Kauri (donated as P_{KA}) decreases rapidly as f increases. When $f = 266$, P_{KA} is only 0.01, which can be considered negligible. The remaining rows show the performance of Tide under different parameters. For a fixed ρ, a larger α significantly enhances the resilience of the Tide topology. For example, when $f = 266$, $\rho = 2$, and $\alpha = 1$, $P_{\text{TG}} = 0.16$, whereas for $\alpha = 2$, $P_{\text{TG}} = 0.73$. At the same time, for relatively small f, only small values of ρ and α are required to ensure the availability of the topology. For example, when $f = 333$, $\rho = 4$ and $\alpha = 2$ are required to ensure $P_{\text{TG}} \geq 0.5$, whereas for $f = 299$, $\rho = 2$ and $\alpha = 2$ are sufficient.

VI. Implementation & Evaluation

We implemented Tide and conducted evaluations across several aspects. Experimental results showed that Tide's performance was almost unaffected by the number of replicas, effectively overcoming the scalability bottleneck.

Implementation. We implemented Tide[2], HotStuff[54], Kauri[43], and Crackle[51] using the Go language with 5000 lines of code. Our implementation is built on top of the open-source Bamboo framework[3] [23], which provides a general benchmarking environment for Chained-BFT consensus. We implemented aggregated signatures based on the BLS12-381, which is also used by Ethereum. To conduct large-scale experiments, we implemented comprehensive Bash scripts to facilitate batch control of the servers. The experiments were conducted on 100 cloud servers within a local area network, each equipped with 8 CPUs (Intel(R) Xeon(R) Gold 6278C @ 2.60GHz) and 16 GiB of memory, running Ubuntu 20.04, and we used Netem [28] to introduce control over network round-trip time (RTT).

Setup. We primarily focus on the following five parameters, with their default values set as: the number of replicas N is 100, the number of Byzantine faults f is 0, the block size (number of transactions) is 400, the payload size of a transaction is 128 bytes, and the RTT between replicas is 10 ms. Each time, we vary one of these parameters to explore its impact on performance.

Metrics. We use two key metrics to evaluate performance: throughput (ktps), which measures the number of kilo-transactions committed per second, and latency (ms), which represents the average time taken from transaction generation to commitment. Due to the significant differences in performance metrics under different parameters, **the y-axis is on a logarithmic scale**.

Evaluation of Scalability. We first conducted an evaluation of scalability separately for small-scale ($N \leq 100$) and large-scale ($N \geq 100$) scenarios as shown in Fig. 7 and Fig. 8. To demonstrate the impact of pipelining on performance, we implement a variant of Tide, ***Tide-NP***, which adopts the same logarithmic fanout topology (§ III) but lacks the highly-parallel pipelining design (§ IV). Tide-NP employs a pipeline directly adapted from HotStuff, where the leader broadcasts a proposal, waits for a quorum of votes, and then proceeds to the next. As a result, its throughput is affected by the depth of the topology.

As the number of replicas increases, the throughput of other protocols experiences a significant decline, while Tide is only minimally affected. When $N = 100$, Tide's throughput reaches 54ktps, more than twice that of the other protocols. As N increases from 100 to 1000, Tide's throughput decreases by less than 10%, while other protocols experience a decline of 65%-90%, making Tide's throughput more than 5 times higher than that of the other protocols. This significant performance advantage stems from differences in topology and pipeline design. HotStuff uses a star topology with $O(N)$ fanout, causing throughput to drop quickly as N increases. Kauri and Crackle adopt tree topologies with $O(\sqrt{N})$ fanout, offering better scalability. However, Kauri's pipeline parallelism relies on prior profiling and is less effective than Crackle. Tide further reduces fanout to $O(\log N)$ and employs a highly-parallel

[2]https://github.com/glimmerzcy/Tide-Public

[3]https://github.com/gitferry/bamboo

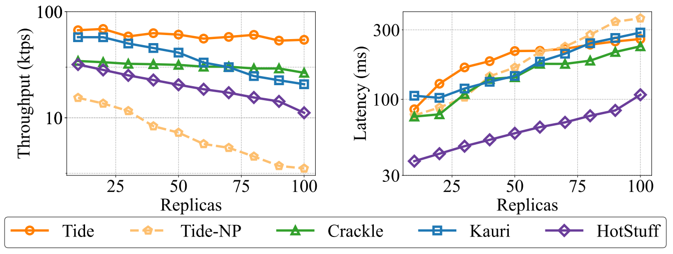


Fig. 7: Performance in Samll-Scale($N \leq 100$)

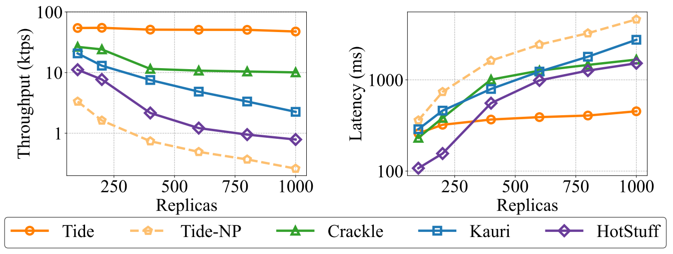


Fig. 8: Performance in Large-Scale($N \geq 100$)

pipeline, enabling it to maintain high throughput even at large scales. Despite using an $O(\log N)$ fanout, Tide-NP suffers from low throughput because it lacks parallel pipelining. Each proposal must go through $\log N$ communication steps before the leader can collect votes, resulting in significant latency caused by the depth of the topology.

Meanwhile, Tide's latency also remains largely stable, increasing only from 0.3s to 0.4s, as shown in Fig. 8, while the latency of other protocols increased by more than 10 times. The latency of Tide is primarily dictated by the logarithmic depth of its communication topology. Since $\log N$ grows slowly, the increase in latency is marginal when N increases from 100 to 1000. However, when $N < 100$, the growth of $\log N$ is relatively faster, so Tide does not exhibit a latency advantage in this regime, as shown in Figure 7.

In summary, Tide demonstrates remarkable stability in terms of both throughput and latency, overcoming the performance limitations imposed by the leader bottleneck. In contrast, Tide-NP exhibits significantly lower performance, illustrating that a low fanout alone is insufficient. If throughput is constrained by the depth of the communication topology, overall system performance can degrade, which is also a common issue in gossip-based protocols [11], [37].

Evaluation of Resilience. We secondly conducted an evaluation of resilience, testing performance under different Byzantine numbers. We selected $N = 100$ and $N = 400$ for comparison and conducted two sets of experiments. In each set, the number of Byzantine replicas was increased from 0 to $N/3$, as shown in Fig. 9 and 10. Kauri, due to its tree topology, suffers from crashes as the number of Byzantine faults increases, failing to maintain a usable topology. Although Crackle addresses the resilience issue of tree topology by using intermediate-layer rotation, allowing it to operate with

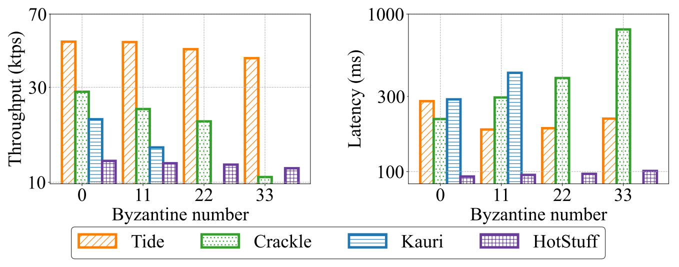


Fig. 9: Impact of Byzantine Faults ($N = 100$)

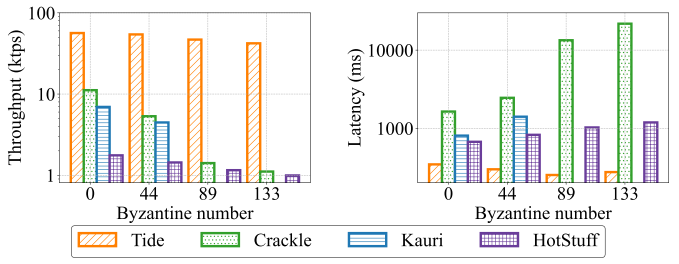


Fig. 10: Impact of Byzantine Faults ($N = 400$)

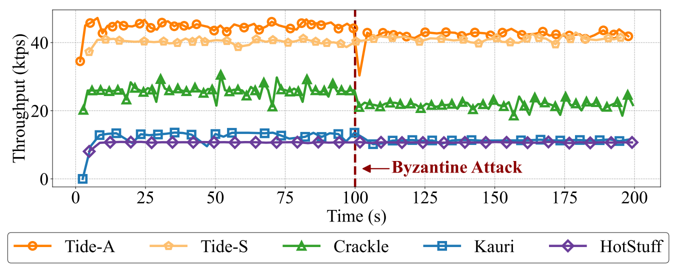


Fig. 11: Throughput under Byzantine Attack

$f = N/3$, its performance remains low. Since Crackle lacks responsiveness, as the number of Byzantine faults increases, more voting collections rely on timers, which leads to a decrease in performance. When $f = N/3$, Crackle's throughput is nearly identical to that of HotStuff. In comparison, Tide is only minimally affected by the number of Byzantine faults, with performance decreasing by less than 20%, as Tide requires at most $O(\log N)$ fanout. Although HotStuff has a low absolute performance, it is almost unaffected by the number of Byzantine faults due to its star topology, which provides the best resilience. The experiments show that both Kauri and Crackle have fragile resilience to Byzantine faults, while HotStuff exhibits low performance. Only Tide maintains high performance across different environments.

Evaluation of Adaptive Resilience. Section V introduces adaptive resilience by dynamically adjusting the fanout in Tide. We evaluate this using sudden Byzantine attacks, and the results are shown in Fig. 11. Tide-A represents Tide with adaptive resilience, where the parameters ρ and α are dynamically adjusted during runtime. Tide-S, on the other hand, uses static parameters to ensure maximum fault tolerance, with $\rho = 4 \approx log_3 100, \alpha = 1$. The system starts with zero Byzantine faults, and Tide-A's throughput is significantly higher than Tide-S, as it operates with a smaller fanout. At the 100th second, 11 random Byzantine faults were injected. After a few seconds of performance fluctuation, Tide-A adjusted to the appropriate parameters ($\rho = 3$) and maintained a performance slightly higher than Tide-S. Although Tide-S's performance is slightly lower, it maintains higher stability, being almost unaffected by the Byzantine attack.

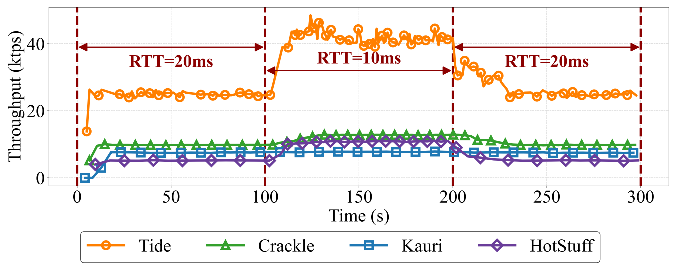


Fig. 12: Throughput under Network Fluctuation

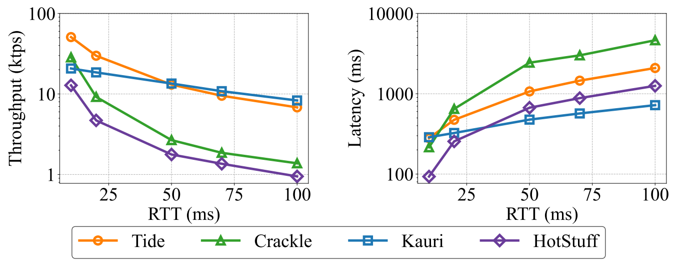


Fig. 13: Impact of RTT

Evaluation of Responsiveness. Responsiveness reflects the system's ability to advance consensus in line with the actual network delay RTT. In protocols without responsiveness, such as Kauri and Crackle, replicas rely on a preset Δ to drive consensus, preventing performance improvements when the network conditions improve. We set the number of Byzantine faults to 11, and through the silence of Byzantine replicas, the non-responsive voting collection in the protocol is affected. As shown in Fig. 12, we start the system with an RTT of 20ms, and adjust the RTT to 10ms between 100s and 200s. As the actual RTT decreased, the throughput of both HotStuff and Tide approximately doubled, whereas Kauri and Crackle remained largely unaffected.

Impact of RTT. RTT is one of the key factors influencing the performance of consensus protocols. We conducted an experiment where RTT was varied from 10ms to 100ms, as shown in Fig. 13. The results show that Crackle's latency is most affected by RTT, as its communication steps per QC scale with $\sqrt{N}$, which is designed to enhance resilience. In terms of throughput, the performance of all four protocols decreases as RTT increases. Tide and Kauri, due to their smaller fanout, experience a more moderate decline.

VII. Related Work

As the first practical BFT protocol, PBFT[14] adopts a leader-based paradigm, in which a designated replica is responsible for coordinating proposal and commitment in each consensus round. While this design simplifies coordination, it inevitably introduces a performance bottleneck and load imbalance, as the leader becomes the central point of both computation and communication. To mitigate these issues, a line of research has focused on restructuring the communication topology and building pipelined execution schemes that align with the underlying topology, allowing non-leader replicas to share part of the leader's workload and thereby enhance scalability. Along this line, the proposed Tide protocol is the first to reduce the workload of all replicas to a logarithmic level, making system performance almost independent of the number of replicas.

Works such as Kauri [43] and [36], [15], [35] utilize the tree topology to reduce fanout in which Byzantine replicas can affect all replicas within the subtree. Therefore, they can only use a three-layer tree with the fanout $\sqrt{N}$ to reduce the impact of Byzantine replicas. However, it is still difficult for the system to tolerate $N/3$ faults. Crackle [51] addresses this issue by frequently rotating the $\sqrt{N}$ replicas in the middle layer, but increases the Communication Steps per QC to $O(\sqrt{N})$. In terms of the pipelining, Kauri [43] and Crackle [51] are unable to achieve responsive vote collection. As long as one Byzantine replica does not return its vote, the consensus can only proceed according to the pre-set timer's value (Δ). Hermes[31] addresses leader resource limitations by introducing a forwarding layer with c replicas, reducing the leader's fanout to c. However, each forwarding replica has a fanout of N, becoming a new bottleneck. Tendermint [11], Gosig [37], and TSBFT [50] use the Gossip algorithm [46] to achieve proposal broadcasting with logarithmic fanout. Gossip addresses the fault tolerance issue when fanout is small by using randomized multi-round propagation. However, due to its randomness, message propagation becomes uncontrollable and unpredictable. As a result, the leader cannot deterministically know the progress of the current proposal's propagation. The leader must wait until all replicas have received the proposal before sending the next one. This ultimately leads to the proposal pace being dependent on all replicas rather than just the fanout. Therefore, although its fanout appears to be very small, in practice, it is not fully utilized.

More recently, an alternative line of research has explored multi-leader and leaderless paradigms, where multiple replicas concurrently propose and confirm blocks to overcome the inherent limitations of single-leader designs. Some studies [47], [26], [2] achieve moderate performance gains by introducing multiple leaders, each running its own consensus instance. However, this approach introduces significant challenges, such as transaction duplication and difficulties in maintaining a consistent global order. In contrast, DAG-based [18], [33], [17], [30] and ACS-based [41], [25], [20], [57] protocols enable all replicas to participate equally and concurrently in driving consensus, thereby fully utilizing system-wide resources, conceptually similar to how Tide achieves balanced participation within a single-leader framework. Nevertheless, most of these protocols were originally designed for asynchronous networks rather than the partially synchronous setting considered in this work. As a result, they typically incur higher communication complexity and latency due to their reliance on costly primitives such as Reliable Broadcast [1] and Common Coin [13], and they often require large batch sizes (e.g., $O(n \log n)$) to amortize cryptographic overhead. Concurrent with our work, Mysticeti [6] and Shoal++ [4] further optimize DAG-based designs for partially synchronous settings, demonstrating impressive performance improvements. Furthermore, inspired by the separation of transaction dissemination and ordering introduced in Narwhal [18], Tide can be naturally integrated with DAG-based approaches.

VIII. Conclusion

Tide is the first to overcome the scalability bottleneck of the leader-based BFT protocols, with performance invariance to replica count. We construct a robust communication topology that can tolerate $N/3$ Byzantine faults with a fanout of $O(\log N)$. Meanwhile, we design an efficient proposal pipelining, making the throughput independent of the depth of the topology. Furthermore, we implement two practical optimizations that further reduce communication complexity. Finally, we conducted large-scale experiments with up to 1000 replicas. The results show that the performance of Tide is almost unaffected by the number of replicas. With Tide, the scale of replicas no longer limits BFT consensus.

Acknowledgement

This work is supported by the National Natural Science Foundation of China under Grant Nos. 62372324 and 62502338, the China Postdoctoral Science Foundation under Grant Number 2025M771566, and the Postdoctoral Fellowship Program of China Postdoctoral Science Foundation under Grant Number GZB20250403.

References

[1] Nicolas Alhaddad, Sourav Das, Sisi Duan, Ling Ren, Mayank Varia, Zhuolun Xiang, and Haibin Zhang. Balanced byzantine reliable broadcast with near-optimal communication and improved computation. In *Proc. of the ACM PODC*, pages 399–417, 2022.

[2] Salem Alqahtani and Murat Demirbas. Bigbft: A multileader byzantine fault tolerance protocol for high throughput. In *Proc. of the IEEE IPCCC*, pages 1–10, 2021.

[3] Mohammad Javad Amiri, Ziliang Lai, Liana Patel, Boon Thau Loo, Eric Lo, and Wenchao Zhou. Saguaro: An edge computing-enabled hierarchical permissioned blockchain. In *Proc. of the IEEE ICDE*, pages 259–272, 2023.

[4] Balaji Arun, Zekun Li, Florian Suri-Payer, Sourav Das, and Alexander Spiegelman. Shoal++: High throughput dag bft can be fast and robust! In *Proc. of the USENIX NSDI*, pages 813–826, 2025.

[5] Diem Association. The diem blockchain, 2020. https://developers.diem.com/docs/technical-papers/the-diem-blockchain-paper/.

[6] Kushal Babel, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Arun Koshy, Alberto Sonnino, and Mingwei Tian. Mysticeti: Reaching the limits of latency with uncertified dags. In *Proc. of the ISOC NDSS*, 2025.

[7] Christian Berger, Signe Schwarz-Rüsch, Arne Vogel, Kai Bleeke, Leander Jehl, Hans P Reiser, and Rüdiger Kapitza. Sok: Scalability techniques for bft consensus. In *Proc. of the IEEE ICBC*, pages 1–18, 2023.

[8] Alysson Bessani, João Sousa, and Eduardo EP Alchieri. State machine replication for the masses with bft-smart. In *Proc. of the IEEE/IFIP DSN*, pages 355–362, 2014.

[9] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *Proc. of the Springer Asiacrypt*, pages 514–532, 2001.

[10] Manuel Bravo, Gregory Chockler, and Alexey Gotsman. Making byzantine consensus live. *Springer Distributed Computing*, 35(6):503–532, 2022.

[11] Ethan Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, University of Guelph, 2016.

[12] Christian Cachin et al. Architecture of the hyperledger blockchain fabric. In *Proc. of the ACM DCCL*, volume 310, pages 1–4, 2016.

[13] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography. In *Proc. of the ACM PODC*, pages 123–132, 2000.

[14] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *Proc. of the USENIX OSDI*, pages 173–186, 1999.

[15] Aleksey Charapko, Ailidani Ailijiang, and Murat Demirbas. Pigpaxos: Devouring the communication bottlenecks in distributed consensus. In *Proc. of the ACM SIGMOD*, pages 235–247, 2021.

[16] Kevin Choi, Aathira Manoj, and Joseph Bonneau. Sok: Distributed randomness beacons. In *Proc. of the IEEE S&P*, pages 75–92, 2023.

[17] Xiaohai Dai, Chaozheng Ding, Hai Jin, Julian Loss, and Ling Ren. Ipotane: Achieving the best of all worlds in asynchronous bft. *IACR Cryptol. ePrint Arch.*, 2024.

[18] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient bft consensus. In *Proc. of the Eurosys*, pages 34–50, 2022.

[19] Kevin Driscoll, Brendan Hall, Håkan Sivencrona, and Phil Zumsteg. Byzantine fault tolerance, from theory to reality. In *Proc. of the Springer SAFECOMP*, pages 235–248, 2003.

[20] Sisi Duan, Michael K Reiter, and Haibin Zhang. Beat: Asynchronous bft made practical. In *Proc. of the ACM CCS*, pages 2028–2041, 2018.

[21] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *ACM Journal of the ACM*, 35(2):288–323, 1988.

[22] Liaoliao Feng, Xiang Fu, Huaimin Wang, Keming Wang, Peichang Shi, Feng Jiang, and Moheng Lin. From pbft to the present: a thorough overview of blockchain consensus protocols. *Springer Science China Information Sciences*, 69(1):111102, 2026.

[23] Fangyu Gai, Ali Farahbakhsh, Jianyu Niu, Chen Feng, Ivan Beschastnikh, and Hao Duan. Dissecting the performance of chained-bft. In *Proc. of the IEEE ICDCS*, pages 595–606, 2021.

[24] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proc. of the ACM SOSP*, pages 51–68, 2017.

[25] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo: Faster asynchronous bft protocols. In *Proc. of the ACM CCS*, pages 803–818, 2020.

[26] Suyash Gupta, Jelle Hellings, and Mohammad Sadoghi. Rcc: Resilient concurrent consensus for high-throughput secure transaction processing. In *Proc. of the IEEE ICDE*, pages 1392–1403, 2021.

[27] Johan Håstad, Russell Impagliazzo, Leonid A Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[28] Stephen Hemminger et al. Network emulation with netem. In *Proc. of the Linux conf au*, volume 5, page 2005, 2005.

[29] Dengcheng Hu, Jianrong Wang, Xiulong Liu, Hao Xu, Xujing Wu, Muhammad Shahzad, Guyue Liu, and Keqiu Li. Ladder: A convergence-based structured {DAG} blockchain for high throughput and low latency. In *Proc. of the USENIX NSDI*, pages 779–794, 2025.

[30] Yi Hua, Xiulong Liu, Hao Xu, Chenyu Zhang, Licheng Wang, and Keqiu Li. Fastdag: A low-latency and parallel wave-execution consensus with a double-layer dag. In *Proc. of the Springer NPC*, pages 88–100, 2025.

[31] Mohammad M Jalalzai, Chen Feng, Costas Busch, Golden G Richard, and Jianyu Niu. The hermes bft for blockchains. *IEEE Transactions on Dependable and Secure Computing*, 19(6):3971–3986, 2021.

[32] Mohammad M Jalalzai, Jianyu Niu, Chen Feng, and Fangyu Gai. Fast-hotstuff: A fast and resilient hotstuff protocol. *arXiv preprint arXiv:2010.11454*, 2020.

[33] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is dag. In *Proc. of the ACM PODC*, pages 165–175, 2021.

[34] John Kelsey, Luís TAN Brandão, Rene Peralta, and Harold Booth. A reference for randomness beacons: Format and protocol version 2. Technical report, National Institute of Standards and Technology, 2019.

[35] Eleftherios Kokoris-Kogias. Robust and scalable consensus for sharded distributed ledgers. *Cryptology ePrint Archive*, 2019.

[36] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *Proc. of the IEEE S&P*, pages 583–598, 2018.

[37] Peilun Li, Guosai Wang, Xiaoqi Chen, Fan Long, and Wei Xu. Gosig: A scalable and high-performance byzantine consensus for consortium blockchains. In *Proc. of the ACM SoCC*, pages 223–237, 2020.

[38] Wenyu Li, Chenglin Feng, Lei Zhang, Hao Xu, Bin Cao, and Muhammad Ali Imran. A scalable multi-layer pbft consensus for blockchain. *IEEE Transactions on Parallel and Distributed Systems*, 32(5):1146–1160, 2020.

[39] Xiulong Liu, Zhiyuan Zheng, Hao Xu, Zhelin Liang, Gaowei Shi, Chenyu Zhang, and Keqiu Li. Enabling consistent sensing data sharing among iot edge servers via lightweight consensus. *IEEE Transactions on Computers*, 2025.

[40] Dahlia Malkhi and Kartik Nayak. Hotstuff-2: Optimal two-phase responsive bft. *Cryptology ePrint Archive*, 2023.

[41] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proc. of the ACM CCS*, pages 31–42, 2016.

[42] David S Moore and George P McCabe. *Introduction to the practice of statistics.* WH Freeman/Times Books/Henry Holt & Co, 1989.

[43] Ray Neiheiser, Miguel Matos, and Luís Rodrigues. Kauri: Scalable bft consensus with pipelined tree-based dissemination and aggregation. In *Proc. of the ACM SOSP*, pages 35–48, 2021.

[44] Fred B Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, 1990.

[45] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Springer Journal of Cryptology*, 4:161–174, 1991.

[46] Devavrat Shah et al. Gossip algorithms. *Foundations and Trends® in Networking*, 3(1):1–125, 2009.

[47] Chrysoula Stathakopoulou, Tudor David, and Marko Vukolic. Mir-bft: High-throughput bft for blockchains. *arXiv preprint arXiv:1906.05552*, 92, 2019.

[48] Xiao Sui, Sisi Duan, and Haibin Zhang. Marlin: Two-phase bft with linearity. In *Proc. of the IEEE/IFIP DSN*, pages 54–66, 2022.

[49] Harmony Team. Harmony technical whitepaper, 2019. https://harmony.one/whitepaper.pdf.

[50] Junfeng Tian, Jin Tian, and Hongwei Xu. Tsbft: A scalable and efficient leaderless byzantine consensus for consortium blockchain. *Elsevier Computer Networks*, 222:109541, 2023.

[51] Hao Xu, Xiulong Liu, Chenyu Zhang, Wenbin Wang, Jianrong Wang, and Keqiu Li. Crackle: A fast sector-based bft consensus with sublinear communication complexity. In *Proc. of the IEEE INFOCOM*, 2024.

[52] Hao Xu, Chenyu Zhang, Xiulong Liu, Yiran Lv, Shiyu Gan, Liehuang Zhu, and Keqiu Li. A fast and practical sector-based bft consensus with sublinear communication complexity. ieee transactions on networking. *IEEE Transactions on Networking*, 2026.

[53] Wassim Yahyaoui, Jeremie Decouchant, Marcus Völp, and Joachim Bruneau-Queyreix. Tolerating disasters with hierarchical consensus. In *Proc. of the IEEE INFOCOM*, 2024.

[54] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proc. of the ACM PODC*, pages 347–356, 2019.

[55] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *Proc. of the ACM CCS*, pages 931–948, 2018.

[56] Gengrui Zhang, Fei Pan, Yunhao Mao, Sofia Tijanic, Michael Dang'ana, Shashank Motepalli, Shiquan Zhang, and Hans-Arno Jacobsen. Reaching consensus in the byzantine empire: A comprehensive review of bft consensus algorithms. *ACM Computing Surveys*, 56(5):1–41, 2024.

[57] Haibin Zhang and Sisi Duan. Pace: Fully parallelizable bft from reproposable byzantine agreement. In *Proc. of the ACM CCS*, pages 3151–3164, 2022.

Appendix

We provide several proofs to help illustrate the properties of the topology discussed in the main text.

A. Properties of the ELG

We use the sample space Ω to represent there are f randomly chosen faulty vertices out of N total vertices. We use the event $\mathrm{NF}(v)$ to denote that vertex v is **not faulty**. If there exists a path from the root v_1 to v such that all vertices along the path are not faulty, we say that vertex v is **reachable**, and this event is denoted as $\mathrm{RV}(v)$.

Theorem 4. *The more predecessors vertex $v \in \mathcal{V}_l$ has at every layer (such as v in Fig. 2), the larger $P(\mathrm{RV}(v))$ becomes, i.e., for* ELG *and* ELG$'$*, $\forall i < l\ \forall U \subseteq \mathrm{Pred}^{l-i}(v)$ holds:*

$$U \subseteq \mathrm{Pred}'^{l-i}(v) \wedge \mathrm{Pred}_S(U) \subseteq \mathrm{Pred}'_S(U) \\ \implies \mathrm{P}(\mathrm{RV}'(v)) \geq \mathrm{P}(\mathrm{RV}(v))$$

Proof. First, we define some random variables and events. We use the event $\mathcal{A}(U, V)$ to represent that all vertices in U are non-faulty, and the other vertices are faulty, i.e.,

$$\mathcal{A}(U, V) = (\bigcap_{u \in U} \mathrm{NF}(u)) \cap (\bigcap_{u \in V-U} \overline{\mathrm{NF}(u)})$$

We use $\mathrm{RV}_S(V)$ to represent that at least one vertex in the set V is reachable, i.e.,

$$\mathrm{RV}_S(V) = \bigcup_{v \in V} \mathrm{RV}(v)$$

For a set SV_i consisting of vertices from the same layer V_i ($SV_i \subseteq V_i$), the following relationship holds:

$$\mathrm{RV}_S(SV_i) = \bigcup_{U \in \mathcal{P}(SV_i)} \mathcal{A}(U, SV_i) \cap \mathrm{RV}_S(\mathrm{Pred}_S(U))\ \ (i > 1)$$

We use the random variable $\mathrm{NF}_S(\omega, V)$ to represent the number of non-faulty vertices in a set. We limit the number of faulty vertices at each layer to $f_1, f_2, \dots, f_d (d = |\mathrm{EP}(\mathcal{V})|)$, such that the events of faulty vertices at each layer are independent. We denote this condition as $\Omega_{f_1, f_2, \dots, f_d}$, and all subsequent probability calculations will be performed under this condition, i.e.,

$$\Omega_{f_1, f_2, \dots, f_d} = (\bigcap_{i=0}^{d} \mathrm{NF}_S(\omega, V_i) = |V_i| - f_i) \cap (\sum f_i = f)$$

Under this condition, whether a vertex in a given layer is faulty is independent of whether other layers are reachable. Therefore, $\mathrm{P}(\mathcal{A}(U, SV_i)|\Omega_{f_1, f_2, \dots, f_d})$ and $\mathrm{P}(\mathrm{NF}_S(\mathrm{Pred}_S(U))|\Omega_{f_1, f_2, \dots, f_d})$ are independent probabilities. Therefore, we obtain the following probability equality:

$$\mathrm{P}(\mathrm{RV}_S(SV_i)|\Omega_{f_1, f_2, \dots, f_d}))(i > 1) \\ = \sum_{U \in \mathcal{P}(SV_i)} (\mathrm{P}(\mathcal{A}(U, SV_i)|\Omega_{f_1, f_2, \dots, f_d}) \cdot \\ \mathrm{P}(\mathrm{RV}_S(\mathrm{Pred}_S(U))|\Omega_{f_1, f_2, \dots, f_d})\ )$$

At this point, we have obtained a formula for calculating the reachability of a set. Although it is not possible to directly compute probabilities from it, we can identify some meaningful relationships. The first part of the formula $\mathrm{P}(\mathcal{A}(U, SV_i)|\Omega_{f_1, f_2, \dots, f_d})$ depends solely on the size of SV_i, meaning that the larger the enumerated set, the better when calculating probabilities. The second part $\mathrm{P}(\mathrm{NF}_S(\mathrm{Pred}_S(U))|\Omega_{f_1, f_2, \dots, f_d})$ directly depends on the size of the predecessor set of U, indicating that a larger number of predecessors is advantageous. Thus, we prove the following relationship using mathematical induction:

$$\forall i < l\ \forall SV_i \subseteq \mathrm{Pred}^{l-i}(v) \implies \\ \mathrm{P}(\mathrm{RV}'_S(SV_l)|\Omega_{f_1, f_2, \dots, f_d}) \geq \mathrm{P}(\mathrm{NF}_S(SV_l)|\Omega_{f_1, f_2, \dots, f_d})$$

First, for $i = 1$, if $SV_1 = \{v_1\}$:

$$\mathrm{P}(\mathrm{RV}'_S(\{v_1\})|\Omega_{f_1, f_2, \dots, f_d}) = \mathrm{P}(\mathrm{NF}_S(\{v_1\})|\Omega_{f_1, f_2, \dots, f_d}) \\ = \mathrm{P}(\mathrm{RV}(v_1)|\Omega_{f_1, f_2, \dots, f_d}) = \mathrm{P}(\mathrm{NF}(v_1)|\Omega_{f_1, f_2, \dots, f_d}) = \frac{1 - f_1}{1}$$

Else, if $SV_1 = \emptyset$:

$$\mathrm{P}(\mathrm{RV}'_S(\{v_1\})|\Omega_{f_1, f_2, \dots, f_d}) = \mathrm{P}(\mathrm{NF}_S(\{v_1\})|\Omega_{f_1, f_2, \dots, f_d}) = 0$$

Second, suppose for i:

$$\mathrm{P}(\mathrm{RV}'_S(SV_i)|\Omega_{f_1, f_2, \dots, f_d}) \geq \mathrm{P}(\mathrm{NF}_S(SV_i)|\Omega_{f_1, f_2, \dots, f_d})$$

Then, for $i + 1$:

$$\forall U \in \mathcal{P}(SV_{i+1})\ \ \wedge \mathrm{Pred}_S(U) \subseteq \mathrm{Pred}'_S(U) \subseteq \mathcal{V}_i \\ \implies \mathrm{P}(\mathrm{RV}'_S(\mathrm{Pred}'_S(U))|\Omega_{f_1, f_2, \dots, f_d}) \\ \geq \mathrm{P}(\mathrm{NF}_S(\mathrm{Pred}_S(U))|\Omega_{f_1, f_2, \dots, f_d}) \\ \implies \mathrm{P}(\mathrm{RV}'_S(SV_{i+1})|\Omega_{f_1, f_2, \dots, f_d}) \\ \geq \mathrm{P}(\mathrm{NF}_S(SV_{i+1})|\Omega_{f_1, f_2, \dots, f_d})$$

Finally, for $SV_l = \{v\}$, we get:

$$\mathrm{P}(\mathrm{RV}'_S(\{v\})|\Omega_{f_1, f_2, \dots, f_d}) \geq \mathrm{P}(\mathrm{NF}_S(\{v\})|\Omega_{f_1, f_2, \dots, f_d})$$

At the same time, the following equation clearly holds:

$$\mathrm{P}(\mathrm{RV}(v)) = \mathrm{P}(\mathrm{NF}_S(\{v\})) \\ = \sum_{\Omega_{f_1, f_2, \dots, f_d} \subseteq \Omega} \mathrm{P}(\mathrm{NF}_S(\{v\})|\Omega_{f_1, f_2, \dots, f_d}) \cdot \mathrm{P}(\Omega_{f_1, f_2, \dots, f_d})$$

Therefore:

$$\mathrm{P}(\mathrm{RV}'(v)) \geq \mathrm{P}(\mathrm{RV}(v))$$

□

Theorem 5.

$$(\mathrm{ELG} \implies \neg\mathrm{MPRED}) \\ \implies \exists \mathrm{ELG}' \exists u\ s.t.\ \mathrm{P}(\mathrm{RV}'(u)) > \mathrm{P}(\mathrm{RV}(u))$$

Proof. We plan to identify the vertices that violate MPRED and then construct a better graph. Since ELG does not satisfy MPRED, there must exist a vertex u as the first vertex to violate this property. That is, the vertices in the first a layers satisfy the MPRED condition, while u satisfies the condition up to the b-th order predecessors, but fails to do so in the $(b+1)$-th, i.e., $\exists u \in \mathcal{V}_a\ \exists b < a$ such that:

$$\begin{aligned}&\forall i < a\ \forall v \in \mathcal{V}_i\ \forall l < i\ |\operatorname{Pred}^l(v)| = \min(\rho^l, |\mathcal{V}_{i-l}|)\\ \wedge\ &\forall l \leq b\ |\operatorname{Pred}^l(u)| = \min(\rho^l, |\mathcal{V}_{a-l}|)\\ \wedge\ &|\operatorname{Pred}^{b+1}(u)| < \min(\rho^{b+1}, |\mathcal{V}_{a-b-1}|)\end{aligned}$$

Our goal is to swap some edges related to u in $\mathcal{E}$, to form a new graph $\mathrm{ELG}' = \langle \mathrm{EP}(\mathcal{V}), \mathcal{E}' \rangle$, such that $\mathrm{P}(\mathrm{RV}'(u)) > \mathrm{P}(\mathrm{RV}(u))$.

Since $|\operatorname{Pred}^b(u)|$ reaches the maximum, but $|\operatorname{Pred}^{b+1}(u)|$ does not. There must exist two vertices $w_1^b, w_2^b \in \operatorname{Pred}^b(u)$ that share at least one common predecessor $w_1^{b+1} \in \operatorname{Pred}^{b+1}(u)$, thereby disrupting the upward extension of the predecessors of u, and at least one vertex $w_2^{b+1} \in \mathcal{V}_{a-b-1}$ is not in $\operatorname{Pred}^{b+1}(u)$, i.e., $\exists w_1^b, w_2^b \in \operatorname{Pred}^b(u)\ \exists w_1^{b+1}, w_2^{b+1} \in \mathcal{V}_{a-b-1}$ such that:

$$\begin{aligned}w_1^{b+1} &\in (\operatorname{Pred}(w_1^b) \cap \operatorname{Pred}(w_2^b))\\ w_2^{b+1} &\in (\mathcal{V}_{a-b-1} \setminus \operatorname{Pred}(^{b+1}(u))\end{aligned}$$

And we donated the predecessor of w_1^{b+1} and w_2^{b+1} as:

$$\operatorname{Pred}(w_j^{b+1}) = \{w_{j,i}^{b+2} \mid 1 \leq i \leq \rho\}\ (1 \leq j \leq 2)$$

For any $w_{1,i}^{b+2}$, its successors cannot all lie within $\operatorname{Pred}^{b+1}(u)$. Otherwise, there must $\exists w^1 \in \operatorname{Pred}(u)$ such that $\operatorname{Pred}^b(u^1) < \rho^b$. A rigorous proof is provided in the appendix. We donated these vertices as $w_{1,i}^{b+1}$, i.e., for $\forall w_{1,i}^{b+2} (1 \leq i \leq \rho)$:

$$\exists w_{1,i}^{b+1} \in \operatorname{Succ}(w_{1,i}^{b+2}) \implies w_{1,i}^{b+1} \notin \operatorname{Pred}^{b+1}(u)$$

We construct the following edge set: $E_1 \subseteq \mathcal{E}$ represents the current connection scheme. By swapping some of the connections, we obtain $E_1' \subseteq \mathcal{E}'$, such that the predecessors of w_2^{b+1} and w_1^{b+1} are exactly the same.

$$\begin{aligned}E_1 &= \{(w_{1,i}^{b+2}, w_{1,i}^{b+1}), (w_{2,i}^{b+2}, w_2^{b+1}) \mid 1 \leq i \leq \rho\}\\ E_1' &= \{(w_{1,i}^{b+2}, w_2^{b+1}), (w_{2,i}^{b+2}, w_{1,i}^{b+1}) \mid 1 \leq i \leq \rho\}\end{aligned}$$

Next, we construct the following edge set $E_2 \subseteq \mathcal{E}, E_2' \subseteq \mathcal{E}'$ to make w_2^{b+1} a predecessor of u by connecting it to w_2^b.

$$\begin{aligned}E_2 &= \{(w_1^{b+1}, w_2^b), (w_2^{b+1}, w_3^b)\},\\ E_2' &= \{(w_2^{b+1}, w_2^b), (w_1^{b+1}, w_3^b)\}\end{aligned}$$

Formally, the relationship between E' and E is as follows:

$$\mathcal{E}' = (\mathcal{E} \setminus (E_1 \cup E_2)) \cup (E_1' \cup E_2')$$

In the new graph $\mathrm{ELG}' = \langle \mathrm{EP}(\mathcal{V}), \mathcal{E}' \rangle$, the number of paths from u to the root has obviously increased, which implies that the reachability also increases. In E, both w_1^b and w_2^b are connected to w_1^{b+1}, and whether they are connected depends entirely on w_1^{b+1}. In E', w_1^b and w_2^b are connected to w_1^{b+1} and w_2^{b+1} respectively, and their predecessors are exactly the same. Specifically, in the event $\mathrm{RV}(u)$, any path passing through (w_1^{b+1}, w_2^b) can be mapped to a path passing through (w_2^{b+1}, w_2^b) in $\mathrm{RV}'(u)$. However, under the new connection set, when both w_1^b and w_2^b are usable, the reachability of any vertex in w_1^{b+1} or w_2^{b+1} allows them to connect to higher layers, which is not the case in ELG. That is, $\mathrm{RV}'(u)$ contains more usable paths than $\mathrm{RV}(u)$, in the case:

$$\mathrm{NF}(w_1^b) \cap \mathrm{NF}(w_2^b) \cap \overline{\mathrm{NF}(w_2^{b+1})} \cap \mathrm{NF}(w_2^{b+2})$$

In summary, we construct a better graph $\mathrm{ELG}' = \langle \mathrm{EP}(\mathcal{V}), \mathcal{E}' \rangle$, such that $\mathrm{P}(\mathrm{RV}'(u)) > \mathrm{P}(\mathrm{RV}(u))$.

□

Theorem 6.

$$\mathrm{MSUCC} \implies \mathrm{MPRED}$$

Proof. Proving the original proposition is equivalent to proving its contrapositive. Therefore, we start with the contrapositive of MPRED:

$$\neg\ \mathrm{MPRED} \equiv \exists i\ \exists v \in V_i\ \exists l < i (|\operatorname{Pred}^l(v)| < \min(\rho^l, |V_{i-l}|))$$

To express concisely, let's assume $\exists i\ \exists v \in V_i\ \exists l < i$ such that $|\operatorname{Pred}^l(v)| < \min(\rho^l, |V_{i-l}|)$, and then proceed with the proof. Noting the presence of the min symbol, we divide the proof into two cases.

Case 1. First, consider the case when $\rho^l < |V_{i-l}|$. It can be proven that $\exists u \in \operatorname{Pred}^l(v)$ for which $\neg$MSUCC holds. All successors of any $u \in \operatorname{Pred}^l(v)$ must be disjoint, but in this situation they necessarily intersect; otherwise the predecessors of v would not be insufficient. Let l' be the last layer in which v has a full set of predecessors. In this layer, there must exist two parent nodes that point to the same successor. Therefore, there exists a vertex $u \in \operatorname{Pred}^l(v)$ such that $\operatorname{Succ}^l(u) < (\rho \cdot \kappa)^l$. Since $|V_i| = |V_{i-l}| \cdot \kappa^l$, we have $(\rho \cdot \kappa)^l < |V_i|$, which implies $|\operatorname{Succ}^l(u)| < \min\left((\rho \cdot \kappa)^l,\ |V_i|\right)$.

Case 2: $\rho^l > |V_{i-l}|$. Second, when $\rho^l > |V_{i-l}|$, it can be proven that there $\exists u \in (V_{i-l} - \operatorname{Pred}^l(v))$ such that $\neg$MSUCC holds. Since $u \notin \operatorname{Pred}^l(v)$, we have $v \notin \operatorname{Succ}^l(u)$. Because $v \in |V_i|$, it follows that $|\operatorname{Succ}^l(u)| < |V_i|$. At this point, since $(\rho \cdot \kappa)^l > |V_i|$, we obtain $|\operatorname{Succ}^l(u)| < \min\left((\rho \cdot \kappa)^l,\ |V_i|\right)$.

Combining the proofs of both cases, we have demonstrated the validity of the contrapositive of the original proposition, which is equivalent to proving the original proposition.

$$\neg\ \mathrm{MPRED} \implies \neg\ \mathrm{MSUCC}$$

□

B. Properties of the TG

By observing the Definition 6 of the Tide Graph, we can see that there is some continuity in the properties of the vertices. As shown in Fig. 14, two consecutive blue vertices in $\mathcal{V}_i$ have consecutive successors; similarly, two consecutive green vertices also have consecutive successors, if we treat $\mathcal{V}_{i+1}$ as a circular layer. We define this continuity as:

Definition 7. *(CC) For a subset $U \subseteq \mathcal{V}_i$, if U satisfies the following property, we say that U is congruently contiguous in $\mathcal{V}_i$, denoted as $\mathrm{CC}(U, \mathcal{V}_i)$:*

$$\begin{aligned}&\exists p_1, p_2, ..., p_{|U|} \implies U = \{v_{p_1}, v_{p_2}, ..., v_{p_{|U|}}\}\\ &\quad \wedge\ p_x - s_i + 1 \equiv p_{x+1} - s_i \mod \kappa^{i-1} (1 \leq x < |U|)\end{aligned}$$

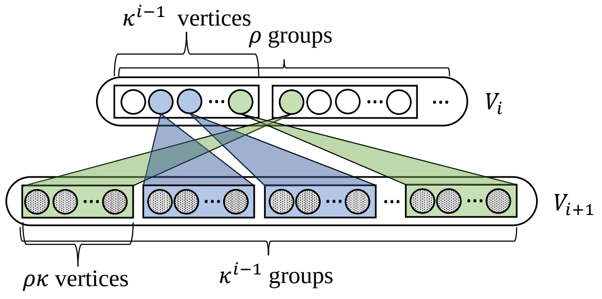


Fig. 14: The continuity of the Tide Graph

For a TG, there are two important properties of CC, proved in Lemma 7 and Lemma 8, namely:

$$\mathrm{CC}(U, \mathcal{V}_i) \implies \mathrm{CC}(\mathrm{Succ}_S(U), \mathcal{V}_{i+1})$$
$$\mathrm{CC}(U, \mathcal{V}_i) \implies |\,\mathrm{Succ}_S(U)| = \min(|U| * \rho\kappa, |\mathcal{V}_{i+1}|)$$

Lemma 7. *For* TG $=<\mathrm{EP}(\mathcal{V}), \mathcal{E}>$*, the following property holds:*

$$\mathrm{CC}(U, V_i) \implies \mathrm{CC}(\mathrm{Succ}_S(U), V_{i+1})$$

Proof. By Definition 6, the successor vertices of a vertex v_{p_x} in U can be represented as:

$$\mathrm{Succ}(v_{p_x}) = \{v_{p_{x,y}} \mid p_{x,y} = b_x * \rho\kappa + s_{i+1} + y,$$
$$b_x = (p_x - s_i) \mod \kappa^{i-1}, 0 \leq y < \rho\kappa\}$$

First, we only consider a single vertex v_{p_x} in U:

$$\begin{aligned}&p_{x,y} + 1 = p_{x,y+1} (0 \leq y < \rho\kappa - 1)\\ &\implies p_{x,y} - s_{i+1} + 1 \equiv p_{x,y+1} - s_{i+1} \mod \kappa^i\\ &\implies \mathrm{CC}(\mathrm{Succ}(v_{p_x}), V_{i+1})\end{aligned}$$

Furthermore, we prove that the successors of two adjacent vertices v_{p_x} and $v_{p_{x+1}}$ in U are congruently contiguous:

$$\begin{aligned}&b_x + 1 \equiv b_{x+1} \mod \kappa^{i-1}\\ &\implies (b_x * \rho\kappa + \rho\kappa - 1) + 1 \equiv b_{x+1} * \rho\kappa + 0 \mod \kappa^i\\ &\implies p_{x,\rho\kappa-1} - s_{i+1} + 1 \equiv p_{x+1,0} - s_{i+1} \mod \kappa^i\\ &\implies \mathrm{CC}(\mathrm{Succ}(v_{p_x}) \;\cup\; \mathrm{Succ}(v_{p_{x+1}}),\; V_{i+1})\end{aligned}$$

Therefore, the following sequence ensures that $\mathrm{Succ}_S(U)$ satisfies the property of being congruently contiguous:

$$p_{1,0}, p_{1,1}, ..., p_{1,\rho\kappa-1}, ..., p_{x,y}, ..., p_{|U|,0}, p_{|U|,1}, ..., p_{|U|,\rho\kappa-1}$$

□

Lemma 8. *For* TG $=<\mathrm{EP}(\mathcal{V}), \mathcal{E}>$*, the following property holds:*

$$\mathrm{CC}(U, V_i) \implies |\,\mathrm{Succ}_S(U)| = \min(|U| * \rho\kappa, |V_{i+1}|)$$

Proof. This lemma means that when U is congruently contiguous in V_i, its set of successor vertices can reach a maximum value, which is the minimum between $|U| \times \rho\kappa$ and $|V_{i+1}|$. The choice of this minimum value depends on the relationship between $|U|$ and κ^{i-1}, so we prove this in two cases.

Case 1: $|U| < \kappa^{i-1}$. For any two distinct vertices in U, it can be proven that their sets of successors have an empty intersection:

$$\begin{aligned}&\forall v_{p_e}, v_{p_f} \in U (f > e) \;\wedge\; f - e < |U|\\ &\implies f - e < \kappa^{i-1} \;\wedge\; p_e - s_i + (f - e) \equiv p_f - s_i \mod \kappa^{i-1}\\ &\implies p_e - s_i \not\equiv p_f - s_i \mod \kappa^{i-1}\\ &\implies \mathrm{Succ}(v_{p_e}) \cap \mathrm{Succ}(v_{p_f}) = \varnothing\\ &\implies |\,\mathrm{Succ}_S(U)| = \sum_{v \in U} |\,\mathrm{Succ}(v)| = |U| * \rho\kappa\end{aligned}$$

Case 2: $|U| \geq \kappa^{i-1}$. In this case, all vertices in V_{i+1} are successors of U. This is equivalent to proving that for any $v_y \in V_{i+1}$, $v_y \in \mathrm{Succ}_S(U)$.

$$\begin{aligned}&\forall v_y \in V_{i+1} \;\wedge\; \mathrm{CC}(U, V_i)\\ &\implies \exists v_{p_x} \in U \text{ s.t. } (p_x - s_i) \mod \kappa^{i-1} = \lfloor \frac{y - s_{i+1}}{\rho\kappa} \rfloor\\ &\implies v_y \in \mathrm{Succ}_S(U) \implies |\,\mathrm{Succ}_S(U)| = |V_{i+1}|\end{aligned}$$

□

Based on this, we ultimately provide proof that TG satisfies MSUCC, which also implies that it satisfies MPRED.

Theorem 9.

$$\mathrm{TG} \implies \mathrm{MSUCC}$$

Proof. Our approach is to prove, for $\forall i \forall v \in \mathcal{V}_i$, using the transitivity of properties (Lemma 7, 8) related to CC (Definition 7), that the successors of v at each layer satisfy the MSUCC principle through mathematical induction. For the zero-order successors of v, it is evident that:

$$\mathrm{CC}(\mathrm{Succ}^0(v), \mathcal{V}_i) \wedge |\,\mathrm{Succ}^0(v)| = \min((\rho\kappa)^0, |\mathcal{V}_i|)$$

Next, we assume that the l-th order successors of v satisfy:

$$\mathrm{CC}(\mathrm{Succ}^l(v), \mathcal{V}_{i+l}) \wedge |\,\mathrm{Succ}^l(v)| = \min((\rho\kappa)^l, |\mathcal{V}_{i+l}|)$$

Since $\mathrm{Succ}_S(\mathrm{Succ}^l(v)) = \mathrm{Succ}^{l+1}(v)$, we can deduce that the $(l+1)$-th order successors of v satisfy:

$$\begin{aligned}&\mathrm{CC}(\mathrm{Succ}^l(v), \mathcal{V}_{i+l})\\ &\implies \mathrm{CC}(\mathrm{Succ}^{l+1}(v), \mathcal{V}_{i+l+1})\\ &\quad \wedge\; |\,\mathrm{Succ}^{l+1}(v)| = \min(|\,\mathrm{Succ}^l(v)| * \rho\kappa, |\mathcal{V}_{i+l+1}|)\\ &\qquad = \min((\rho\kappa)^{l+1}, |\mathcal{V}_{i+l+1}|)\end{aligned}$$

□