Towards Improving Throughput and Scalability of DAG-based BFT

Regular paper

***Abstract*—DAG-based BFT protocols face scalability issues due to bandwidth-heavy data replication across all participants. Fortunately, replicating data to a smaller sub-committee with an honest majority, even with high-probability guarantees, suffices.**

In this work, we introduce *tribe-assisted reliable broadcast*, a new primitive that enforces reliable broadcast (RBC) properties within a smaller sub-committee (a clan) of the network, referred to as the tribe. Using this primitive, we develop two efficient DAG-based BFT protocols. We first design an efficient single-clan protocol, where a single clan is elected from the tribe and data is disseminated exclusively to this designated clan using tribe-assisted RBC. We then extend this approach to a multi-clan setting, where multiple clans are elected, and data dissemination is confined to each respective clan using the same mechanism. Our experimental results show that both protocols significantly improve throughput and latency compared to current DAG-based BFT methods, even at moderately large scales.

I. Introduction

Byzantine fault-tolerant state machine replication (BFT SMR) serves as the core building block for blockchain systems. BFT SMR allows a group of n parties to reach consensus on a sequence of values, even in the presence of up to f Byzantine parties, who may act in arbitrarily malicious ways. As blockchains strive toward greater decentralization, it becomes critical for these protocols to scale to hundreds of nodes while maintaining high throughput and low latency.

Similar to many recent low-latency real-world blockchain designs, we consider the partially synchronous network [20], where BFT SMR requires $f < n/3$. Traditional partially synchronous BFT SMR protocols can achieve a commit latency as low as 3δ (where δ represents the actual network delay) [9], [10], [18] and also achieve linear communication complexity [27], [44]. However, their throughput is significantly constrained, achieving only around 3K TPS [3]. This limitation is largely due to their reliance on a *single proposer* design: only a designated (rotating or fixed) leader proposes transactions and disseminates substantial amounts of data across the network at a time, which creates a bottleneck and hinders throughput.

A novel approach called DAG-based BFT [24], [25], [36], [38] has recently emerged to address this challenge. This approach enables every party to propose concurrently, thereby maximizing bandwidth utilization and resulting in improved throughput. Additionally, the state-of-the-art DAG-based BFT protocols such as Mysticeti [5], Sailfish [36], and Shoal++ [4], have demonstrated a commit latency of 3δ, matching the single-proposer BFT protocols while achieving higher throughput. Indeed, the experimental analyses demonstrate that these multi-proposer DAG-based BFTs offer significantly better throughput without affecting the latency for moderate network sizes. However, as the system scales, these protocols often suffer a significant drop in throughput, particularly due to bandwidth-intensive replication of data across all parties. Is this decline in throughput *inherent* to DAG-based protocols as the system scales? This paper seeks to answer this question.

Technical challenge. Most DAG-based BFT protocols [4], [36]–[38] rely on a reliable broadcast (RBC) primitive [8], [28] to disseminate their proposals (referred to as vertices), ensuring non-equivocation and guaranteed delivery of these proposed vertices to all parties. As a result, these protocols incur a communication complexity of at least $\Omega(n^2\ell + \kappa n^3)$, where ℓ represents the size of input transactions and κ is the security parameter. The proposed vertices typically contain large number of transactions to promote good throughput i.e., $\ell > \kappa n$. Consequently, due to bandwidth-heavy replication of large data across all parties, they create a bottleneck as the system scales. Additionally, once ordered, every party must execute all transactions, further increasing the bottleneck as the system expands.

Key idea. We observe that only the consensus/ordering phase requires a super-majority of honest parties with $n > 3f$ participating parties with at most $f < n/3$ [20] (assuming partially synchrony). Once a total ordering of the proposed vertices is agreed upon, it is not necessary for all parties to execute the transactions in these vertices. Equivocation is impossible after transactions are ordered, and the committee responsible for executing transactions only requires an honest majority to function correctly [43]. Particularly, in a (sub-) system with $n_c < n$ parties and $f_c < n_c$ being faulty, a client only needs consistent responses from $f_c + 1$ parties to ensure their transaction has been safely executed. To address the possibility of up to f_c faulty parties failing to execute or sending inconsistent responses, we need $n_c - f_c \geq f_c + 1$, and thus $n_c \geq 2f_c + 1$. As a result, it is sufficient to disseminate the transaction to a committee with $n_c \geq 2f_c + 1$ parties, ensuring that at least $f_c + 1$ honest parties execute the transaction and respond to the client.

Furthermore, if parties are selected uniformly at random, we can form significantly smaller committees while maintaining the honest majority assumption with only a negligible probability of failure statistically. For instance, in a system of $n = 500$ parties and $f = 166$ faulty parties, a committee of just $n_c = 184$ members suffices to ensure an honest majority with a negligible failure probability of 10^{-9}, based on the

hypergeometric probability distribution. We present committee sizes at various system sizes in Figure 1. Leveraging this observation, we elect smaller committees and confine the dissemination of larger payloads to these committees, thereby reducing bandwidth usage and enhancing throughput and scalability. To simplify the explanation, we refer to the entire set of n parties as a *tribe* and the honest majority committee of size n_c as a *clan*.

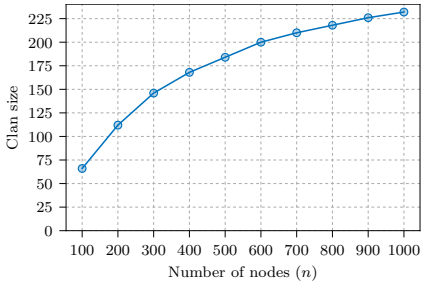


Fig. 1: Clan sizes required to ensure an honest majority with failure probability below 10^{-9}.

A straw-man approach and further challenges. With a clan, it is relatively straightforward to disseminate the data solely to the clan and collect proof of data availability (PoA), ensuring that at least one honest party has received the data (with high probability) [29]. This PoA can then be provided to any SMR protocol to establish a global ordering. Since consensus on metadata is generally less costly, this method reduces overhead. However, this method introduces additional latency due to its inherently sequential nature. Disseminating data and collecting the PoA takes at least 2δ, and even if the PoA is included in the next proposed consensus block, there is an average queuing delay of 1δ. Finally, the commit latency for the proposed block is at least 3δ, resulting in a total latency of at least 6δ.

In contrast, state-of-the-art DAG-based protocols [4], [5] avoid the additional latency overhead from a separate data dissemination layer by embedding transactions directly into vertices. By conducting data dissemination and consensus concurrently, these protocols minimize latency. However, most DAG-based BFT protocols rely on an RBC primitive which delivers the proposed vertices to all parties in the network. As mentioned before, this can be bandwidth intensive, making existing RBC mechanisms unsuitable for our specific requirements. This paper focuses on improving the data dissemination layer while preserving the latency performance of current DAG-based BFT protocols.

A. Our contributions

In this paper, we make three key contributions toward building a throughput-efficient and scalable DAG-based BFT SMR:

1. Tribe-assisted Byzantine reliable broadcast. As a foundational building block, we first introduce an asynchronous primitive that ensures agreement and eventual data delivery within a clan. It is well established that for agreement under partial synchrony, $f < n/3$ is necessary [20], a condition that cannot be satisfied within a clan that only has an honest majority. To achieve agreement within a clan, we leverage the support of the entire tribe. In this regard, we refer to this primitive as *tribe-assisted* reliable broadcast. We present two variations of this protocol–the first protocol terminates in three rounds under an honest sender and is *signature-free*, while the second protocol reduces termination to just two rounds using signatures.

2. Single-clan DAG-based BFT. To address the scalability challenge, we propose electing a smaller clan of parties that retains an honest majority with high probability. By utilizing the tribe-assisted reliable broadcast (RBC), we restrict data dissemination to this designated clan while ensuring that consensus on metadata occurs across the entire network in parallel, thereby avoiding additional latency overhead. To ensure transaction validation, only the parties in the clan are permitted to act as proposers, enabling them to validate and propose transactions effectively. We refer to this protocol as the single-clan DAG-based BFT. This method significantly reduces bandwidth consumption by limiting data dissemination to the designated clan, which improves throughput and scalability. Additionally, this approach is versatile and can be integrated into existing DAG-based BFT protocols to enhance throughput and scalability while preserving security guarantees equivalent to systems where data dissemination and execution involve all parties.

3. Multi-clan DAG-based BFT. We extend the single-clan DAG-based BFT to a multi-clan setting, where the entire tribe is partitioned into multiple disjoint clans, each maintaining the honest majority assumption with negligible error probability. Each clan independently disseminates and executes proposed data and is responsible for responding to the clients. This design builds upon the single-clan approach by enabling more efficient bandwidth utilization across the system, thereby improving both throughput and scalability. The multi-clan protocol is particularly well-suited for shared sequencers [32], [40], [41], which order transactions from independent applications, and it can also enhance performance in state-sharded blockchains [26], [45], [46].

Implementation and evaluation. We implemented and evaluated the performance of both the single-clan and multi-clan protocols, comparing them against Sailfish [36], the current state-of-the-art DAG-based BFT protocol. Our results in a geo-distributed setting show that the single-clan DAG-based BFT significantly outperforms Sailfish in terms of throughput across all system sizes, despite having fewer proposers. This is primarily due to the reduced dissemination of data to fewer parties within the designated clan. Interestingly, despite having the same theoretical latency in terms of the number of rounds, the single-clan DAG-based BFT also exhibited lower latency than Sailfish. Additionally, we observed that the multi-clan DAG-based BFT outperforms both the single-clan DAG-based BFT and Sailfish in terms of throughput, benefiting from the efficient distribution of data across multiple clans.

Organization. In Section II, we present the system model and preliminaries. In Section III, we define the tribe-assisted reliable broadcast primitive and discuss the tribe-assisted RBC protocol. We present a round efficient variation of the primitive in Section IV. Sections V and VI present the single-clan and the multi-clan DAG-based BFT protocols respectively. Section VII presents an evaluation of both of these protocols. Finally, we discuss related works in Section VIII.

II. Preliminaries

We consider a system $\mathcal{P} := P_1, \ldots, P_n$ consisting of n parties out of which up to $f = \lfloor \frac{n-1}{3} \rfloor$ parties can be Byzantine, meaning they can behave arbitrarily. The model of corruption is static i.e., the adversary picks the corrupted parties before the start of protocol execution. A party that is not faulty throughout the execution is considered to be *honest* and executes the protocol as specified.

We consider the partial synchrony model of Dwork et al. [20]. Under this model, the network starts in an initial state of asynchrony during which the adversary may arbitrarily delay messages sent by honest parties. However, after an unknown time called the *Global Stabilization Time* (GST), the adversary must ensure that all messages sent by honest parties are delivered to their intended recipients within Δ time of being sent. We use δ to characterize the actual (variable) transmission latencies of messages and observe that $\delta \leq \Delta$ after GST. Additionally, we assume the local clocks of the parties have *no clock drift* and *arbitrary clock skew*.

We employ digital signatures and a public-key infrastructure (PKI) to safeguard against spoofing, replay attacks, and to ensure message authenticity. A message x digitally signed by party P_i using its private key is denoted as $\langle x \rangle_i$, while $\langle x \rangle$ refers to an unsigned message x transmitted over an authenticated channel. Additionally, we represent the hash of an input x by $H(x)$, where H is the hash function. For simplicity, we use the same parameter κ to denote both the hash size and the signature size.

Tribe and clans. We define the entire system of n nodes as the *tribe*, where up to $f < \frac{n}{3}$ nodes may exhibit Byzantine faults. A sub-committee within this system, where the honest majority assumption holds, is referred to as a *clan*. In a clan, the number of nodes is represented as n_c, with at most $f_c < \frac{n_c}{2}$ being Byzantine, except with a negligible probability of error.

Problem definition. Following earlier works [24], [36], [38], we focus on the Byzantine Atomic Broadcast (BAB) problem as defined below:

Definition 1 (Byzantine atomic broadcast [24], [38])**.** *In a BAB, each honest party $P_i \in \mathcal{P}$ can call a_bcast$_i(m, r)$ to propagate its input m in some round $r \in \mathbb{N}$. Each party P_i then outputs a_deliver$_i(m, r, P_k)$, where $P_k \in \mathcal{P}$ represents the sender of the message. A Byzantine atomic broadcast protocol satisfies the following properties:*

- ***Agreement.*** *If an honest party P_i outputs a_deliver$_i(m, r, P_k)$, then every other honest party P_j eventually outputs a_deliver$_j(m, r, P_k)$.*
- ***Integrity.*** *For every round $r \in \mathbb{N}$ and party $P_k \in \mathcal{P}$, an honest party P_i outputs a_deliver$_i$ at most once regardless of m.*
- ***Validity.*** *If an honest party P_k calls a_bcast$_k(m, r)$ then every honest party eventually outputs a_deliver(m, r, P_k).*
- ***Total order.*** *If an honest party P_i outputs a_deliver$_i(m, r, P_k)$ before a_deliver$_i(m', r', P_\ell)$, then no honest party P_j outputs a_deliver$_j(m', r', P_\ell)$ before a_deliver$_j(m, r, P_k)$.*

III. Tribe-assisted Reliable Broadcast

In this section, we introduce a primitive that ensures all honest parties within a clan reach agreement on a value and guarantees eventual delivery with the support of the entire tribe. We refer to this primitive as *tribe-assisted reliable broadcast*. We first formally define the tribe-assisted reliable broadcast primitive. We denote the parties in a clan by $\mathcal{P}_c$ which has an honest majority (except with negligible error probability).

Definition 2 (Tribe-assisted reliable broadcast)**.** *Let a designated sender P_k invokes r_bcast$_k(m, r)$ to propagate its input m in some round $r \in \mathbb{N}$. Each party P_i outputs r_deliver$_i(y, r, P_k)$, where $y = m$ when $P_i \in \mathcal{P}_c$ and $y = H(m)$ when $P_i \notin \mathcal{P}_c$, P_k is the designated sender and r is the round number in which sender P_k sent the message m. The tribe-assisted reliable broadcast primitive satisfies the following properties:*

- ***Validity.*** *If an honest party P_k calls r_bcast$_k(m, r)$ then each honest party P_i eventually outputs r_deliver$_i(y, r, P_k)$ where $y = m$ when $P_i \in \mathcal{P}_c$ and $y = H(m)$ when $P_i \notin \mathcal{P}_c$.*
- ***Agreement.*** *If an honest party P_i outputs r_deliver$_i(y, r, P_k)$, then each honest party P_j eventually outputs r_deliver$_j(y, r, P_k)$ where $y = m$ when $P_i \in \mathcal{P}_c$ and $y = H(m)$ when $P_i \notin \mathcal{P}_c$.*
- ***Integrity.*** *For every round $r \in \mathbb{N}$ and party $P_k \in \mathcal{P}$, an honest party P_i outputs r_deliver$_i$ at most once regardless of m.*

Next, we present a candidate protocol in Figure 2. This construction is based on Bracha's RBC protocol [8]. Similar to Bracha RBC, this protocol is signature-free and requires three rounds in the good case when the sender is honest. Alternatively, we can extend the RBC protocol of Abraham et al. [2] to achieve better latency, which will be discussed later in Section IV.

Protocol details. In this protocol, the sender P_k with input value m sends $\langle \mathsf{VAL}, m, r \rangle$ to each party $P_i \in \mathcal{P}_c$ for some round r, while it sends only the digest (i.e., $H(m)$) to the parties outside the clan. Upon receiving the value m, each party $P_i \in \mathcal{P}_c$ sends $\langle \mathsf{ECHO}, H(m), r \rangle$. Parties outside the clan send $\langle \mathsf{ECHO}, H(m), r \rangle$ upon receiving just the digest.

An honest party then sends a READY message for the value m when it receives $2f+1$ distinct $\langle \mathsf{ECHO}, H(m), r \rangle$ messages, with at least $f_c + 1$ of those coming from parties in $\mathcal{P}_c$. Note that the clan has at most f_c Byzantine parties (except with negligible error probability). Thus, waiting for at least

$f_c + 1$ ECHO messages from parties in $\mathcal{P}_c$ ensures that at least one honest party in $\mathcal{P}_c$ has received the value m. This allows other parties in $\mathcal{P}_c$ to download value m at a later point when required. Additionally, a party can also send a READY message if they receive $f + 1$ READY messages for m and have not yet sent a READY message.

Upon receiving $2f + 1$ $\langle \mathsf{READY}, H(m), r \rangle$ messages, an honest party $P_i \in \mathcal{P}_c$ can deliver m if it has already received the value m. If it has not yet received the value, it can download m from other parties in $\mathcal{P}_c$ and then deliver it. As previously mentioned, if an honest party sends a $\langle \mathsf{READY}, H(m), r \rangle$ message, it guarantees that at least one honest party in $\mathcal{P}_c$ has received the value m, making it possible to retrieve m at this stage. Finally, parties outside the clan deliver $H(m)$ upon receiving $2f+1$ $\langle \mathsf{READY}, H(m), r \rangle$ messages.

Communication complexity. Let ℓ be the size of the value m. An honest sender sends the ℓ-bit message to parties in $\mathcal{P}_c$ and a κ-bit digest to parties in $\mathcal{P} \backslash \mathcal{P}_c$, resulting in a communication complexity of $O(n_c\ell + \kappa(n - n_c))$. The all-to-all multicast of ECHO and READY messages incurs a complexity of $O(\kappa n^2)$. Therefore, the overall communication complexity is $O(n_c\ell + \kappa n^2)$ in the good case when the sender is honest.

However, when the sender is Byzantine, only a subset of honest parties in $\mathcal{P}_c$ may receive the value m, but all honest parties could still gather $2f + 1$ READY messages for m. In this scenario, the honest parties in $\mathcal{P}_c$ would need to download the value m from other honest parties who have received it. To expedite the retrieval, the honest parties may request the value from a linear number of parties, which could lead to an increase in communication complexity, reaching up to $O(n_c^2\ell + \kappa n^2)$.

Remark on communication complexity. Our protocol allows parties to download or pull missing data. This feature could be exploited by Byzantine parties, who might repeatedly request the data, potentially leading to unbounded communication complexity. To mitigate such an attack, parties can be rate-limited, effectively bounding the communication complexity in practice.

Theoretical RBC protocols [8], [16], [28] typically only support data being pushed, without allowing parties to pull the data. Despite this, these protocols can still encounter unbounded communication in practice. The assumption is that parties communicate over reliable links, often implemented using the TCP protocol [11]. In such setups, a sender continues transmitting data until it receives an acknowledgment from the receiver. If the receiver is Byzantine, it may never send an acknowledgment, causing the sender to continuously transmit the data, leading to unbounded communication. However, in practical systems, it is usually assumed that Byzantine parties do not disrupt the network layer to trigger such unbounded communication. Under this assumption, our protocol provides comparable communication complexity guarantees in practice.

Additionally, theoretical RBC protocols often leverage erasure and error-correcting codes [33] to improve worst-case communication efficiency. While these techniques enhance communication in the worst case, they also introduce overhead in terms of encoding, decoding, and verifying the erasure-coded chunks [12], which can degrade performance under normal conditions. Since most nodes in practice are honest, many practical implementations [14], [34] avoid using erasure codes. Instead, the sender multicasts the proposed value to all parties, and the parties exchange ECHO and READY messages on the digest of the value and download missing values at a later point. This approach offers better performance in typical scenarios. Similarly, we present our tribe-assisted RBC in a setting without erasure and error-correcting codes, optimizing for practical performance.

A. Security Analysis

Lemma 1 (Validity)**.** *The protocol in Figure 2 satisfies Validity, except with a negligible error probability.*

Proof. Observe that an honest sender P_k sends $\langle \mathsf{VAL}, m, r \rangle$ to all parties in $\mathcal{P}_c$ and $\langle \mathsf{VAL}, H(m), r \rangle$ to all parties in $\mathcal{P} \setminus \mathcal{P}_c$. Consequently, all honest parties will eventually send $\langle \mathsf{ECHO}, H(m), r \rangle$. As a result, all honest parties will receive at least $2f+1$ distinct $\langle \mathsf{ECHO}, H(m), r \rangle$ messages, including at least f_c+1 from the parties in $\mathcal{P}_c$. Consequently, all honest parties will eventually send $\langle \mathsf{READY}, H(m), r \rangle$ to all parties, and all honest parties will receive $2f + 1$ $\langle \mathsf{READY}, H(m), r \rangle$ messages. Furthermore, honest parties in $\mathcal{P}_c$ will receive m from the sender P_k. Therefore, all honest parties will invoke r_deliver(y, r, P_k), where $y = m$ if $P_i \in \mathcal{P}_c$ and $y = H(m)$ if $P_i \notin \mathcal{P}_c$.

Note that honest parties may fail to receive at least $f_c + 1$ $\langle \mathsf{ECHO}, H(m), r \rangle$ messages only in the rare case where the clan has a dishonest majority, which occurs with negligible probability. Therefore, the validity property is maintained except with a negligible probability of failure. □

Claim 1. *No two honest parties will send* READY *message on conflicting values.*

Proof. Suppose an honest party P_i sends READY message for value m. This implies P_i must have received at least $2f + 1$ $\langle \mathsf{ECHO}, H(m), r \rangle$. A simple quorum intersection argument show that there cannot exist a set of at least $2f + 1$ ECHO messages on a conflicting value m'. Thus, no honest party sends a READY message for a conflicting message. □

Lemma 2 (Agreement)**.** *The protocol in Figure 2 satisfies Agreement, except with a negligible error probability.*

Proof. Suppose an honest party P_i outputs r_deliver$_i(y, r, P_k)$ (where $y = m$ when $P_i \in \mathcal{P}_c$ and $y = H(m)$ when $P_i \notin \mathcal{P}_c$). This implies P_i must have received at least $2f+1$ $\langle \mathsf{READY}, H(m), r \rangle$ messages, including at least $f+1$ $\langle \mathsf{READY}, H(m), r \rangle$ from honest parties. Since honest parties send $\langle \mathsf{READY}, H(m), r \rangle$ to all parties, all honest parties will eventually receive at least $f + 1$ $\langle \mathsf{READY}, H(m), r \rangle$. By Claim 1, no honest party sends READY message for a conflicting value. Thus, honest parties that have not sent

$r\_bcast_k(m, r)$

1) The sender P_k sends $\langle \mathsf{VAL}, m, r\rangle$ to each party $P_i \in \mathcal{P}_c$ and $\langle \mathsf{VAL}, H(m), r\rangle$ to party $P_i \notin \mathcal{P}_c$.
2) Upon receiving the first $\langle \mathsf{VAL}, y, r\rangle$, party P_i sends $\langle \mathsf{ECHO}, H(m), r\rangle$ to all parties (where $y = m$ when $P_i \in \mathcal{P}_c$ and $y = H(m)$ when $P_i \notin \mathcal{P}_c$).
3) Upon receiving $2f + 1$ distinct $\langle \mathsf{ECHO}, H(m), r\rangle$ messages with at least $f_c + 1$ $\langle \mathsf{ECHO}, H(m), r\rangle$ messages from $\mathcal{P}_c$, party P_i sends $\langle \mathsf{READY}, H(m), r\rangle$ to all parties.
4) Upon receiving $f + 1$ distinct $\langle \mathsf{READY}, H(m), r\rangle$ messages, if $\langle \mathsf{READY}, H(m), r\rangle$ has not been sent, party P_i sends $\langle \mathsf{READY}, H(m), r\rangle$ to all parties.
5) Upon receiving $2f + 1$ distinct $\langle \mathsf{READY}, H(m), r\rangle$ messages, party P_i performs the following:
   - If party $P_i \in \mathcal{P}_c$ and has received value m, invoke $r\_deliver_i(m, r, k)$; otherwise download value m from parties in $\mathcal{P}_c$ and invoke $r\_deliver_i(m, r, k)$.
   - If party $P_i \notin \mathcal{P}_c$, invoke $r\_deliver_i(H(m), r, k)$.

Fig. 2: Tribe-assisted Byzantine reliable broadcast based on [8]

READY message will eventually send $\langle \mathsf{READY}, H(m), r\rangle$. Consequently, all honest parties will eventually receive $2f+1$ $\langle \mathsf{READY}, H(m), r\rangle$ and honest parties in $\mathcal{P} \setminus \mathcal{P}_c$ will invoke r_deliver$(H(m), r, P_k)$.

Moreover, observe that at least one honest party must have sent $\langle \mathsf{READY}, H(m), r\rangle$ upon receiving $2f + 1$ $\langle \mathsf{ECHO}, H(m), r\rangle$ messages, including at least $f_c + 1$ $\langle \mathsf{READY}, H(m), r\rangle$ messages from parties in $\mathcal{P}_c$. This implies at least one honest party $P_j \in \mathcal{P}_c$ must have received the value m, except with negligible error probability. Consequently, other honest parties in $\mathcal{P}_c$ will download value m from P_j and invoke r_deliver(m, r, P_k), except with negligible error probability. □

Theorem 1. *The protocol in Figure 2 is a tribe-assisted reliable broadcast tolerating $t < n/3$ Byzantine faults satisfying Definition 2, except for a negligible probability of error.*

Proof. The integrity property is straightforward from the protocol, as a party can deliver a value at most once. The validity and agreement property follows from Lemma 1 and Lemma 2 respectively. □

IV. Round Optimal Tribe-assisted Reliable Broadcast

In this section, we present a tribe-assisted RBC protocol that completes in two rounds in the good-case which is optimal [2]. This construction is based on the round-optimal RBC of Abraham et al [2]. The protocol is presented in Figure 3.

Protocol details. All the messages exchanged in this protocol are signed. The sender P_k with input value m sends $\langle \mathsf{VAL}, m, r\rangle_k$ to each party $P_i \in \mathcal{P}_c$ for some round r, while it sends only the digest (i.e., $H(m)$) to the parties outside the clan. Upon receiving the value m, each party $P_i \in \mathcal{P}_c$ multicasts $\langle \mathsf{ECHO}, H(m), r\rangle_i$. Parties outside the clan send $\langle \mathsf{ECHO}, H(m), r\rangle$ upon receiving just the digest.

Upon receiving $2f + 1$ $\langle \mathsf{ECHO}, H(m), r\rangle$ messages with at least $f_c + 1$ of those coming from parties in $\mathcal{P}_c$ (denoted by $\mathcal{EC}_r(m)$, an honest party $P_i \in \mathcal{P}_c$ multicasts the $\mathcal{EC}_r(m)$. P_i can deliver m if it has already received the value m. If it hasn't yet received the value, it can download m from other parties in $\mathcal{P}_c$ and then deliver it. Finally, parties outside the clan deliver $H(m)$ upon receiving $\mathcal{EC}_r(m)$.

Communication complexity. Let ℓ represent the size of the value m. An honest sender transmits the ℓ-bit message to parties in $\mathcal{P}_c$ and a κ-bit digest to parties in $\mathcal{P} \setminus \mathcal{P}_c$, resulting in a communication complexity of $O(n_c\ell + \kappa(n - n_c))$. The all-to-all multicast of ECHO messages incurs a complexity of $O(\kappa n^2)$. When using the BLS multi-signature scheme, the size of $\mathcal{EC}_r(m)$ becomes $O(\kappa + n)$, and the all-to-all multicast of the $\mathcal{EC}_r(m)$ incurs $O(\kappa n^2 + n^3)$. Therefore, the overall communication complexity is $O(n_c\ell+\kappa n^2+n^3)$ in the good-case. While there is a cubic term in the communication complexity, the linear term associated with the BLS multi-signature is merely a bit vector indicating who voted, so it does not impose significant communication overhead.

In the worst case, if the sender is Byzantine, the communication complexity can increase to $O(n_c^2\ell + \kappa n^2 + n^3)$ when the honest parties request the value from a linear number of parties.

We present detailed proofs in Appendix A.

V. Single-clan DAG-based BFT

In this section, we introduce an efficient and scalable architecture for a DAG-based BFT protocol that limits data dissemination and execution to a single designated clan of parties while maintaining security guarantees comparable to those of a system where transaction dissemination and execution occur across the entire network.

Structural overview of DAG-based BFT. A DAG-based BFT protocol progresses through a series of *rounds*. In each round r, each party proposes a single vertex using the RBC primitive [8], [16] to ensure non-equivocation and guarantee that all honest parties eventually deliver the vertex. Each vertex v contains a block of transactions, and references to at least $2f + 1$ vertices from round $r - 1$ and up to f vertices from earlier rounds (i.e., rounds $< r - 1$), provided there is no path from v to these earlier vertices. These references form the edges in the DAG, with the references to $2f + 1$ round $r - 1$ vertices serving as strong edges and references to earlier rounds as weak edges. A path from vertex v_k to vertex v_ℓ following the strong edges is called a strong path. The strong edges and strong paths are crucial for committing vertices within the DAG, while all edges contribute to the total ordering of the vertices. The commit and total ordering rules are specific to each protocol. We encourage readers to refer to the respective protocols [36]–[38] for detailed information on these rules.

$r\_bcast_k(m, r)$

1) The sender P_k sends $\langle \mathsf{VAL}, m, r\rangle_k$ to each party $P_i \in \mathcal{P}_c$ and $\langle \mathsf{VAL}, H(m), r\rangle_k$ to party $P_i \notin \mathcal{P}_c$.
2) Upon receiving the first $\langle \mathsf{VAL}, y, r\rangle_k$, party P_i sends $\langle \mathsf{ECHO}, H(m), r\rangle_i$ to all parties (where $y = m$ when $P_i \in \mathcal{P}_c$ and $y = H(m)$ when $P_i \notin \mathcal{P}_c$).
3) Upon receiving $2f + 1$ distinct $\langle \mathsf{ECHO}, H(m), r\rangle_*$ messages with at least $f_c + 1$ $\langle \mathsf{ECHO}, H(m), r\rangle_*$ messages from $\mathcal{P}_c$ (denoted by $\mathcal{EC}_r(m)$), party P_i multicasts $\mathcal{EC}_r(m)$ and performs the following:
   - If party $P_i \in \mathcal{P}_c$ and has received value m, invoke $r\_deliver_i(m, r, k)$; otherwise download value m from parties in $\mathcal{P}_c$ and invoke $r\_deliver_i(m, r, k)$.
   - If party $P_i \notin \mathcal{P}_c$, invoke $r\_deliver_i(H(m), r, k)$.

Fig. 3: Tribe-assisted Byzantine reliable broadcast based on [2].

The references, along with the edges and paths they create, are essential for committing and totally ordering the vertices. Therefore, it is necessary to propagate these references to all parties in the system. However, the block of transactions only needs to be (reliably) disseminated to a designated clan of parties, who will execute the transactions and respond to the client. This observation is fundamental to designing an efficient and scalable DAG-based BFT.

Towards improving throughput with scalability. As previously mentioned, transaction blocks within a vertex are typically large, making their dissemination to all parties bandwidth-intensive and a key bottleneck for scalability and throughput. In contrast, references to vertices from earlier rounds are much smaller, especially in moderately-sized networks, and their dissemination introduces minimal overhead. Specifically, the payload size ℓ far exceeds the size of vertex references, i.e., $\ell >> \kappa n$, in such settings. To mitigate this bottleneck, we randomly elect a clan of parties with a high probability of maintaining an honest majority. The transaction block is then reliably disseminated only to this clan, significantly reducing overall bandwidth usage.

To further optimize the propagation of vertices, we modify the vertex structure to contain only the digest of the block of transactions. The updated data structures are shown in Figure 4. The actual block of transactions is placed in a separate *block* structure, which is sent exclusively to the designated clan, while the vertex itself is propagated to the entire tribe. For block dissemination, we can leverage our tribe-assisted RBC protocol, whereas the standard RBC protocol is used to propagate the vertex. However, this approach increases message (and computation) complexity.

Efficiently propagating the vertex and the block. To efficiently propagate both the vertex and block, we merge the two RBC instances. Let v be a vertex, b be its associated block, and $\mathcal{C}$ be the designated clan. The sender broadcasts vertex v to all parties in the tribe but sends block b only to members of $\mathcal{C}$. Members of $\mathcal{C}$ send an ECHO message (as part of the RBC) only after receiving both v and b, while parties outside $\mathcal{C}$ send an ECHO after receiving just v, which includes the digest of block b. The READY message is sent upon receiving $2f + 1$ ECHO messages with at least $f_c + 1$ from the clan $\mathcal{C}$, or upon receiving $f + 1$ READY messages. This combined approach maintains the guarantees of standard RBC, as $\mathcal{C}$ always includes at least $f_c + 1$ honest members, except with negligible error probability. It effectively integrates tribe-assisted RBC for block propagation with standard RBC for the vertex.

Obtaining single-clan DAG-based BFT. Our scaling technique can be adapted to existing DAG-based BFT protocols that rely on RBC, such as Bullshark [39], Shoal [37], and Sailfish [36], with a few key modifications. Existing protocols typically allow all parties to propose both blocks and vertices. However, the single-clan technique limits block proposals to parties within the clan.

While theoretically, all parties could propose blocks (of transactions) and disseminate them only to the clan to improve throughput, this is impractical. Block proposers are also responsible for validating transactions and ensuring sufficient gas is available for transaction execution. These tasks require executing transactions and maintaining state, which only clan members can do. Thus, in the single-clan approach, only clan members can propose transaction blocks, while all parties must continue proposing vertices since these are essential for committing blocks and vertices.

Another modification involves separating the dissemination of vertices and blocks: instead of broadcasting them together, they are propagated independently. Despite these adjustment, the DAG construction, commit, and ordering rules remain unchanged, adhering to the original protocol designs. After transactions are ordered, only the clan members execute them and respond to the client.

Additionally, the enhanced protocol maintains the good-case commit latency of the original protocol. For example, in Bullshark [39], the protocol would still commit the leader vertex (proposed by the round's leader) with a latency of two RBCs. Similarly, in Sailfish [36], the leader vertex would be committed with a latency of one RBC, plus 1δ.

When using tribe-assisted RBC to propagate vertices, parties may need to download missing blocks if the sender is Byzantine, which introduces additional latency. However, this does not affect the protocol's progress or commit latency. In standard DAG-based BFT protocols, the protocol advances to round $r + 1$ once a sufficient number of RBCs from round r have been delivered. In our enhanced protocol, it proceeds to the next round upon receiving $2f + 1$ READY messages from a sufficient number of RBCs, even before downloading the associated blocks. This ensures the protocol's progress remains unaffected.

Moreover, parties can commit the vertex, which includes the block digest, even before the block itself has been delivered.

Local variables:

struct vertex v: ▷ The struct of a vertex in the DAG

- $v.round$ - the round of v in the DAG
- $v.source$ - the party that broadcast v
- $v.block\_digest$ - the digest of the corresponding block of transactions
- $v.strongEdges$ - a set of vertices in $v.round - 1$ that represent strong edges
- $v.weakEdges$ - a set of vertices in rounds $< v.round - 1$ that represent weak edges
- $v.nvc$ - a no-vote certificate for $v.round - 1$ (if any)
- $v.tc$ - a timeout certificate for $v.round - 1$ (if any)

struct block b:

- $b.txn$ - a list of transactions

Fig. 4: Basic data structures. The core structure is adapted from Sailfish [36].

Typically, transaction execution lags behind consensus. Parties can begin requesting the missing blocks as soon as they receive $2f+1$ ECHO messages, including at least f_c+1 from the clan $\mathcal{C}$ i.e., before receiving $2f+1$ READY messages; ensuring that missing blocks are delivered before execution begins.

On the security of single-clan DAG-based BFT. Existing DAG-based BFT protocols [36]–[38] rely on theoretical RBC primitives [8], [16], [28] to propagate proposed vertices, which consist of both blocks of transactions and references. These RBC primitives guarantee agreement, (eventual) guaranteed delivery, and timely delivery (after GST). These properties are essential to ensure the security guarantees of the protocol, as outlined in Definition 1.

In our protocol, we use a tribe-assisted RBC to disseminate blocks exclusively to a designated clan. This mechanism retains the same agreement and eventual guaranteed delivery properties, except for a negligible probability of error. While parties may need to download missing blocks, introducing some additional latency, the missing blocks can be retrieved without affecting the overall progress of the protocol. Thus, tribe-assisted RBC does not compromise the security of protocols relying on it, except in the unlikely scenario of a dishonest majority forming within the clan, which has a negligible probability.

Communication complexity. Let ℓ denote the size of a block of transactions. Existing DAG-based BFT protocols incur a communication overhead of at least $O(n^2\ell + \kappa n^3)$. By leveraging tribe-assisted RBC for block dissemination and restricting payload proposals to the clan, our protocol reduces this overhead to $O(n_c^2\ell + \kappa n^3)$ in the best-case scenario.

Operating cost reduction. Disseminating large amounts of data across the entire network can be expensive due to higher data transfer fees between data centers. Additionally, machines with a larger number of cores are needed to execute transactions in parallel and reduce latency. As a result, executing transactions and disseminating data across the entire network can become cost-prohibitive. Our single-clan design mitigates this by confining data dissemination and transaction execution to a smaller, designated clan. Only this clan needs high-capacity machines, reducing overall system costs and transaction fees.

Remark. Our scaling techniques are applicable to DAG-based BFT protocols that rely on RBC, such as [4], [36]–[38]. Some other approaches, like Cordial Miners [25] and Mysticeti [5], utilize best-effort broadcast (BEB) to reduce latency in failure-free scenarios but are more vulnerable to increased latency under Byzantine behavior [4]. Our technique does not directly extend to such BEB-based protocols, and exploring how to support these protocols presents an interesting avenue for future work.

Statistical security analysis. When a single clan of n_c parties is randomly selected from a system of n total parties, of which f are Byzantine, the probability of forming a dishonest majority within the clan is given by the hypergeometric cumulative distribution function (CDF). This function calculates the likelihood of selecting more than half of the n_c parties as Byzantine. Specifically, the probability $\Pr(\text{dishonest majority})$ is:

$$\Pr(\text{dishonest majority}) = \sum_{k=\lceil \frac{n_c}{2} \rceil}^{n_c} \frac{\binom{f}{k}\binom{n-f}{n_c-k}}{\binom{n}{n_c}} \tag{1}$$

The value of n_c must be chosen sufficiently large relative to the total number of parties n and the number of Byzantine parties f so that the probability of forming a dishonest majority within the clan is below the desired security threshold μ. Specifically, we require:

$$\Pr(\text{dishonest majority}) \leq 2^{-\mu} \tag{2}$$

In Figure 1, we present the required clan size n_c necessary to ensure that the probability of forming a dishonest majority is below $10^{-9} \approx 2^{-30}$.

VI. Multi-clan DAG-based BFT

In the earlier design, a single clan handled block proposals, received and disseminated the block, and executed transactions. This design is effective for small systems where the clan size remains only marginally smaller than the tribe size. However, as the system scales, the clan size does not grow proportionally, as illustrated in Figure 1. This single clan design can restrict throughput in larger systems and fails to fully utilize the bandwidth of the entire network. To address this, we propose partitioning the tribe into multiple disjoint clans. Each clan independently disseminates, processes, and executes transactions within its group, while also responding to clients. This design maintains security guarantees equivalent to a system where transaction dissemination and execution are performed across the entire network.

Obtaining Multi-clan DAG-based BFT. We partition the tribe into multiple clans, denoted as $\mathcal{C}_1, \ldots, \mathcal{C}_q$, where q is determined based on the desired error probability. Unlike the single-clan approach, which restricts block proposals to the designated clan, the multi-clan design allows all parties to propose. Each proposer disseminates its payload exclusively within its respective clan, evenly distributing bandwidth consumption across clans. Transaction execution is also handled independently within each clan, balancing the processing load.

In the single-clan approach, only the designated clan members could propose transactions because parties outside the clan did not execute transactions, preventing proposers from validating them. In contrast, the multi-clan design ensures that all parties belong to a clan, execute transactions specific to their clan, and can validate and propose transactions within their respective clans.

As before, we adopt the modified data structure where the block of transactions is separated from the vertex, and the vertex structure includes only the block's digest (as depicted in Figure 4). Parties within a clan use tribe-assisted RBC to disseminate the block within their clan, while employing standard RBC to propagate the vertex to the entire tribe. These steps are integrated: a party in a clan sends an ECHO message only after receiving both the block and the vertex. As previously discussed, references to vertices are critical for committing proposed vertices and must be propagated to the entire tribe. Since these references are relatively small, disseminating this metadata to all parties introduces minimal overhead.

The core components of the DAG-based BFT protocol, such as DAG construction, vertex commitment, and ordering, remain unchanged. Although the block is disseminated exclusively within the respective clan, its vertex (and by extension, the block) is globally ordered, ensuring a consistent transaction sequence across the network. Once the protocol has ordered the vertices, the corresponding clan members execute the transactions and respond to the client.

The security of the multi-clan DAG-based BFT is derived from the security guarantees of the underlying DAG-based BFT protocol and the tribe-assisted RBC. Since tribe-assisted RBC may fail with a negligible probability, the same holds true for multi-clan DAG-based BFT, with failure occurring only with negligible probability.

Communication complexity. Let ℓ represent the size of a block of transactions. Existing DAG-based BFT protocols incur a minimum communication overhead of $O(n^2\ell + \kappa n^3)$. By utilizing tribe-assisted RBC for block dissemination and allowing all parties to propose payloads within their respective clans, our protocol reduces this overhead to $O(n_c n\ell + \kappa n^3)$ in the best-case scenario.

A. Applications

Shared-sequencers. The multi-clan DAG-based BFT design is specially well-suited for applications such as shared sequencers [32], [40], [41], which focus on ordering transactions from independent applications. Since these applications have no inter-dependencies, a designated clan can handle transactions for a specific application, processing and maintaining only the state relevant to that application. The recent surge in interest in shared sequencers highlights the potential of our multi-clan design as a promising solution.

State-sharded blockchains. The multi-clan DAG-based BFT is also applicable in state-sharded blockchains [15], [26], [46], where the user space of a blockchain is partitioned into multiple shards to group frequently interacting users within a shard. In such blockchains, each shard is managed by a dedicated clan. Intra-shard transactions are processed within the shard, while cross-shard transactions require synchronization across shards, handled by protocols like two-phase commit. Extensive literature exists on efficiently managing cross-shard transactions [15], [45], [46], which can be extended to multi-clan DAG-based protocols. Exploring this direction is left for future work.

B. Statistical security analysis.

In this sub-section, we calculate the probability of forming a clan with a dishonest majority when the number of clans is 2 and 3. This analysis can be generalized to compute the error probability for a larger number of clans using the same principles.

Analysis for 2 clans. Let $f = \lfloor \frac{n-1}{3} \rfloor$ represent the maximum number of Byzantine parties, and let $n_h = n - f$ denote the number of honest parties. When the tribe is divided into 2 clans, the total number of possible combinations to form two clans is given by:

$$N = \binom{n}{n_c} \tag{3}$$

Let w_1 and w_2 represent the number of Byzantine parties in clans $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively. To ensure an honest majority in both clans, the number of Byzantine parties in each clan must be between 0 and f_c, where f_c is the maximum number of Byzantine parties that a clan can tolerate and still maintain an honest majority, i.e., $0 \leq w_1, w_2 \leq f_c$. Given that there are f Byzantine parties in total, they must be split between $\mathcal{C}_1$ and $\mathcal{C}_2$, meaning that $w_1 + w_2 = f$. Let $\mathcal{W}$ represent all possible pairs (w_1, w_2) such that $0 \leq w_1, w_2 \leq f_c$ and $w_1 + w_2 = f$. The total number of valid combinations where both clans maintain an honest majority is then given by:

$$s = \sum_{(w_1,w_2)\in\mathcal{W}} \binom{f}{w_1}\binom{n_h}{n_c - w_1} \tag{4}$$

Accordingly, the probability of forming a clan with dishonest majority is given by:

$$\Pr(\text{dishonest majority}) = 1 - \frac{s}{N} \tag{5}$$

Analysis for 3 clans. When the entire tribe is divided into 3 clans, the total number of possible combinations to form three clans is given by:

$$N = \binom{n}{n_c}\binom{n - n_c}{n_c} \tag{6}$$

Let w_1, w_2 and w_3 represent the number of Byzantine parties in clans $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$ respectively. To ensure an honest majority in both clans, the number of Byzantine parties in each clan must be between 0 and f_c, i.e., $0 \leq w_1, w_2, w_3 \leq f_c$. Given that there are f Byzantine parties in total, they must be split between $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$, meaning that $w_1 + w_2 + w_3 = f$. Let $\mathcal{W}$ represent all possible pairs (w_1, w_2, w_3) such that $0 \leq w_1, w_2, w_3 \leq f_c$ and $w_1 + w_2 + w_3 = f$. The total number of valid combinations, s, where each clan maintains an honest majority is then given by:

$$\sum_{(w_1,w_2,w_3)\in\mathcal{W}} \binom{f}{w_1}\binom{n_h}{n_c - w_1}\binom{f - w_1}{w_2}\binom{n_h - (n_c - w_1)}{n_c - w_2} \tag{7}$$

We obtain the probability of forming a clan with dishonest majority by substituting the value of N (from Equation (6)) and s (from Equation (7)) into Equation (5).

The value of n_c must be chosen sufficiently large relative to the total number of parties n and the number of Byzantine parties f so that the probability of forming a dishonest majority within the clan is below the desired security threshold μ. Specifically, we require:

$$\Pr(\text{dishonest majority}) \leq 2^{-\mu} \tag{8}$$

Concrete numbers. Based on the above analysis, when the network size is 150, we can partition the network into two clans, with the probability of forming a clan with a dishonest majority being approximately $4.015 * 10^{-6}$. Similarly, when the network size is 387, we can divide the network into three clans, with the probability of forming a clan with a dishonest majority being approximately $1.11 * 10^{-6}$.

VII. Evaluation

We evaluate the performance of both the single-clan and multi-clan protocols, comparing their throughput and latency with the state-of-the-art DAG-based BFT protocol, Sailfish [36], which relies on RBC.

Implementation details. We use Sailfish as the underlying DAG-based protocol. Sailfish commits leader vertices with a latency of 3δ and non-leader vertices with 5δ. Multi-leader Sailfish supports multiple leaders per round, committing all leader vertices with 3δ latency. We modify the open-source Sailfish implementation [35] to develop both the single-clan and multi-clan versions, naming them single-clan Sailfish and multi-clan Sailfish, respectively.

To minimize latency, we use the round-optimal RBC protocol [2] for vertex propagation and a two-round tribe-assisted RBC protocol (cf. Section IV) for block propagation. For efficiency, we combine these RBC processes, where clan members send ECHO only after receiving both the vertex and block (detailed in Section V). Following Sailfish, this results in a commit latency of 1RBC + 1δ (i.e., 3δ). In our implementation of the round-optimal RBC [2], we avoid forwarding the sender's proposal and instead download missing proposals off the critical path of consensus if needed, reducing communication overhead in failure-free cases. As noted earlier, retrieving missing vertices off the critical path does not slow down consensus or increase commit latency.

We used RocksDB for persistent storage of the consensus data and BLS multi-signatures [7] for authentication, significantly reducing the size of signatures that need to be multicast. While BLS multi-signatures are computationally expensive to verify, we optimize this by aggregating individual signatures without upfront verification and verifying only the aggregated signature. In case the aggregated signature fails verification due to a Byzantine party's incorrect signature, individual signatures are verified to identify and penalize the faulty party. This approach avoids the overhead of verifying individual signatures in the typical case where all nodes sign correctly.

TABLE I: Ping latencies (in ms) between GCP regions

	Destination*				
Source	us-e-1	us-w-1	eu-n-1	as-ne-1	au-se-1
us-east1-a	0.75	66.14	114.75	160.28	197.98
us-west1-a	66.15	0.66	158.13	89.56	138.33
europe-north1-a	115.40	158.38	0.69	245.15	295.13
asia-northeast1-a	159.89	90.05	246.01	0.66	105.58
australia-southeast1-a	197.60	139.02	294.36	108.26	0.58

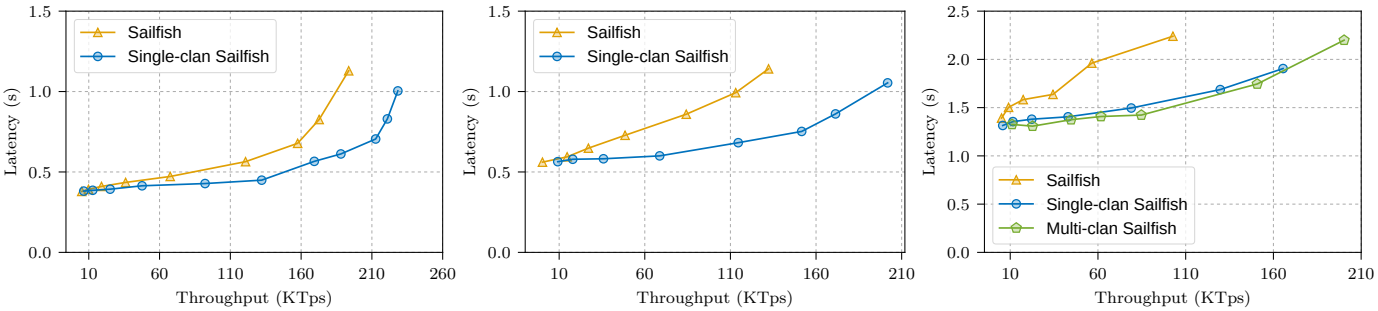
*Region names are abbreviated versions of the source regions.

Experimental setup. We carried out our evaluations on the Google Cloud Platform (GCP), distributing nodes evenly across five distinct GCP regions: us-east1-b (South Carolina), us-west1-a (Oregon), europe-north1-a (Hamina, Finland), asia-northeast1-a (Tokyo), and australia-southeast1-a (Sydney). We employed e2-standard-32 instances [30], each featuring 32vCPUs, 128GB of memory, and up to 16Gbps network bandwidth [31]. All nodes ran on Ubuntu 20.04, and we summarize round-trip latencies in Table I.

In our evaluations, each party generates a configurable number of transactions (512 random bytes each) for inclusion in its proposal, with a proposal containing up to 6000 transactions (i.e., 3 MB). Latency is measured as the average time between the creation of a transaction and its commit by all non-faulty nodes. Throughput is measured by the number of committed transactions per second. Note that our evaluation does not include transaction execution.

Methodology. In our evaluations, we gradually increased the number of input transactions per proposal. As shown in Figure 5, throughput increases with the load, accompanied by a slight increase in latency, up to a certain point before reaching saturation. Beyond this point, latency begins to rise while throughput either stabilizes or increases marginally.

Performance of single-clan Sailfish. Large-scale geo-distributed experiments are inherently costly. To reduce the expense of running these experiments, we opted for a slightly higher failure probability of $10^{-6} \approx 2^{-20}$, which remains reasonable for many practical applications. With this failure probability, we can have clans of 32, 60, and 80 nodes for



(a) At $n = 50$ (b) At $n = 100$ (c) At $n = 150$

Fig. 5: Throughput vs. latency at varying system sizes and varying input

system sizes of 50, 100, and 150 respectively.

We distributed clan nodes evenly across GCP regions instead of randomly sampling them to produce more uniform output. We then evaluated the performance of Sailfish and single-clan Sailfish for system sizes of 50, 100, and 150 nodes. The corresponding throughput and latency results are shown in Figures 5.

Sailfish exhibited better throughput for the same number of input transactions due to its higher number of proposers. However, its throughput quickly saturated, even with a smaller number of input transactions. In contrast, single-clan Sailfish initially showed lower throughput for the same input transactions per proposer, as illustrated in Figure 6. Nevertheless, single-clan Sailfish supported larger payload sizes and ultimately achieved higher throughput. As a result, single-clan Sailfish outperformed Sailfish in terms of throughput across all evaluated system sizes. This improvement is primarily attributed to the efficient dissemination of data only to the designated clan, significantly reducing bandwidth consumption. The difference in throughput becomes increasingly pronounced as the system scales, given that the clan size grows more slowly relative to the overall system size.

For a fixed system size, single-clan Sailfish showed lower commit latency compared to Sailfish. This is because only the designated clan received the full block of transactions, while the rest of the parties received only the vertex, allowing quicker delivery. In tribe-assisted RBC, only $f_c + 1$ ECHO messages are needed from the clan, with the rest coming from the tribe. Since the vertex propagates faster, tribe-assisted RBC finishes sooner than standard RBC, leading to better latency in single-clan DAG-based BFT.

Commit latency for both protocols increased as the system size grew, primarily due to the increased number of cryptographic operations. Specifically, the commit latency for minimal payload across all protocols was around 380ms at $n = 50$, while it increased to approximately 1392ms at $n = 150$. In our implementation, BLS signature aggregation was performed on a single thread while verification of the aggregated signature was performed in parallel. This contributed to a significant increase in latency at $n = 150$. There is potential to reduce these latencies with an optimized implementation.

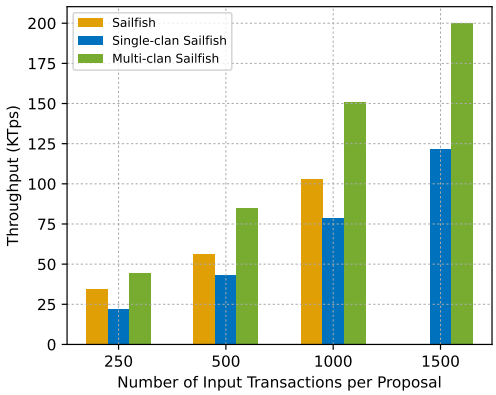


Fig. 6: Throughput vs. # of transactions at $n = 150$

Performance of multi-clan Sailfish. Next, we evaluate the performance of multi-clan Sailfish. To form multiple clans with an acceptable failure probability, the system size must be sufficiently large. For $n = 150$, we can form two clans with a failure probability of $4.015 \times 10^{-6} \approx 10^{-6}$, as analyzed in Section VI-B. Therefore, we evaluate the multi-clan protocol only at $n = 150$. We limit our evaluations to $n = 150$ due to the high cost of running these experiments. Figure 5c compares the throughput and latency of multi-clan Sailfish with Sailfish and single-clan Sailfish, while Figure 6 shows the throughput relative to the number of input transactions per proposal. Note that we did not evaluate Sailfish's throughput for 1500 transactions per proposal due to its significantly higher latency at 1000 transactions (as shown in Figure 5c).

As shown in Figure 6, multi-clan Sailfish achieves higher throughput for the same number of input transactions due to more efficient bandwidth use compared to Sailfish and single-clan Sailfish. However, multi-clan Sailfish incurs higher latency at the same input rate because all parties need to process blocks, unlike single-clan Sailfish, where parties outside the designated clan propose and process only the vertex and can send ECHO messages more quickly.

Note on scalability. Both single-clan and multi-clan Sailfish demonstrate stable performance even at moderately large system sizes (e.g., $n = 150$), whereas the throughput of

Sailfish quickly saturates. This stability is primarily due to more efficient bandwidth utilization in clan-based protocols. In this regard, our clan-based approach offers significantly better scalability, making it a promising solution for larger systems.

VIII. Related Work

An extensive body of research has focused on improving the BFT consensus. DAG-based BFT protocols have emerged as a promising solution to boost throughput while maintaining low latency. We review the most relevant works below.

Comparison with Arete [46]. Arete [46] also elects multiple sub-committees for data dissemination and execution while executing the consensus across the entire tribe. In their protocol, parties disseminate data solely to their respective clans and collect proof of availability (PoA). This PoA is then input to Jolteon [21], a traditional leader-based BFT SMR protocol. While this method reduces bandwidth consumption and enhances throughput and scalability, it introduces additional latency overhead due to the sequential nature of a separate data dissemination layer. Specifically, PoA generation incurs a latency of 2δ, while the queuing latency is at least 1δ, and the commit latency for Jolteon adds up to 5δ, resulting in a minimum total latency of 8δ.

We also identify an issue with the minimum shard size computation in Arete. They rely on the hypergeometric distribution to compute the size of multiple sub-committees such that each sub-committee has an honest majority with high probability. However, the hypergeometric distribution can only be applied to compute the size of a single committee required to ensure an honest majority. This is because the initial count of Byzantine parties is required to calculate the probability of achieving an honest majority within a selected committee. After the first committee is elected, this information is not available for subsequent committee selections, making the approach unsuitable for multiple committees.

To provide accurate probabilities, we instead calculate the total number of possible committees that can be formed and the subset of those committees that ensure an honest majority. By comparing these counts, we can determine the probability of multiple selected committees having an honest majority.

Comparison with committee-based consensus protocols. A long line of research [1], [6], [13], [22] focuses on electing a sub-committee of parties to execute the consensus protocol. This sub-committee must ensure an honest super majority to execute the consensus, leading to larger sub-committee sizes unless there are over 10,000 participants in the system. As a result, these approaches are mainly advantageous at much larger scales. Furthermore, these protocols typically offer only sub-optimal resilience. In contrast, our method selects a sub-committee solely for data dissemination and execution tasks, which only require an honest majority. This allows us to form much smaller committees, even in moderate system sizes. Meanwhile, the consensus protocol is executed across the entire tribe, ensuring optimal resilience. Our approach, therefore, offers improved performance even in moderate system sizes.

Comparison with Gearbox [17]. Another noteworthy protocol is Gearbox [17], which prioritizes safety at the cost of liveness. Gearbox forms smaller committees that may include a Byzantine super-majority. To commit a value, the protocol requires a higher number of consistent responses from committee members, which limits its resilience against liveness failures. Furthermore, to detect liveness failures, Gearbox relies on a separate consensus protocol that maintains both safety and liveness under $f < n/3$ Byzantine faults.

Comparison with Pando [42]. Pando elects a single committee for data dissemination and multiple sub-committees for consensus execution, all requiring an honest super-majority. This results in sub-optimal resilience. Its consensus mechanism resembles traditional leader-based BFT. Additionally, the PoA from data dissemination is used during the consensus phase, introducing extra latency due to the sequential nature of the process.

Comparison with state-sharded protocols. Several existing works [15], [26], [45] utilize state sharding to partition the ledger into multiple shards, each managed by a dedicated sub-committee executing the full consensus protocol on the relevant partitioned state. However, these protocols often rely on stronger assumptions such as synchrony (e.g., Rapidchain [45]) and provide only sub-optimal resilience [26], [45]. Our Multi-clan DAG-based BFT can also be applied in the state-sharded context, with consensus being executed across the entire tribe, thereby tolerating optimal Byzantine failures. A common challenge in sharded blockchains is efficiently handling cross-shard transactions. There is extensive literature on this topic [15], [26], [45] that could also be relevant to our multi-clan protocol.

Comparison with Autobahn [23] and Star [19]. Both Autobahn [23] and Star [19] include a separate data dissemination layer where data is broadcast to all parties, and a PoA with $f+1$ signatures is collected. This PoA is then used in a single-proposer BFT SMR protocol for ordering. As previously discussed, this approach introduces additional latency due to the separate dissemination layer. Moreover, these protocols do not restrict data dissemination to a smaller committee, further contributing to the overhead.

IX. Conclusion

We introduced tribe-assisted RBC, a novel primitive that enforces RBC properties within a designated clan. Leveraging this primitive, we developed two efficient DAG-based BFT protocols: single-clan and multi-clan DAG-based BFT. Our experimental evaluation demonstrated that both protocols significantly enhance throughput and latency compared to the state-of-the-art DAG-based BFT protocols, even at moderately large scales. While our exploration of the multi-clan DAG-based BFT primarily focused on its applicability to shared sequencers, an intriguing future direction would be to investigate its application in state-sharded blockchains.

References

[1] Ittai Abraham, TH Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 317–326, 2019.

[2] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Good-case latency of byzantine broadcast: A complete categorization. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 331–341, 2021.

[3] Salem Alqahtani and Murat Demirbas. Bottlenecks in blockchain consensus protocols. In *2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS)*, pages 1–8. IEEE, 2021.

[4] Balaji Aurn, Zekun Li, Florian Suri-Payer, Das Sourva, and Alexander Spiegelman. Shoal++: High throughput dag bft can be fast! *arXiv preprint 2405.20488*, 2024.

[5] Kushal Babel, Andrey Chursin, George Danezis, Lefteris Kokoris-Kogias, and Alberto Sonnino. Mysticeti: Low-latency dag consensus with fast commit path. In *Network and Distributed System Security Symposium (NDSS)*, 2025 (To appear).

[6] Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. Asynchronous byzantine agreement with subquadratic communication. In *Theory of Cryptography: 18th International Conference, TCC 2020, Durham, NC, USA, November 16–19, 2020, Proceedings, Part I 18*, pages 353–380. Springer, 2020.

[7] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 435–464. Springer, 2018.

[8] Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

[9] Jan Camenisch, Manu Drijvers, Timo Hanke, Yvonne-Anne Pignolet, Victor Shoup, and Dominic Williams. Internet computer consensus. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 81–91, 2022.

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

[11] Vinton Cerf and Robert Kahn. A protocol for packet network intercommunication. *IEEE Transactions on communications*, 22(5):637–648, 1974.

[12] Brian Cho and Alexander Spiegelman. Quorum store: How consensus horizontally scales on the aptos blockchain. https://medium.com/aptoslabs/quorum-store-how-consensus-horizontally-scales-on-the-aptos-blockchain-988866f6d5b0.

[13] Shir Cohen, Idit Keidar, and Alexander Spiegelman. Not a coincidence: Sub-quadratic asynchronous byzantine agreement whp. In *34th International Symposium on Distributed Computing*, 2020.

[14] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient bft consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 34–50, 2022.

[15] Sourav Das, Vinith Krishnan, and Ling Ren. Efficient cross-shard transaction execution in sharded blockchains. *arXiv preprint arXiv:2007.14521*, 2020.

[16] Sourav Das, Zhuolun Xiang, and Ling Ren. Asynchronous data dissemination and its applications. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2705–2721, 2021.

[17] Bernardo David, Bernardo Magri, Christian Matt, Jesper Buus Nielsen, and Daniel Tschudi. Gearbox: Optimal-size shard committees by leveraging the safety-liveness dichotomy. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 683–696, 2022.

[18] Isaac Doidge, Raghavendra Ramesh, Nibesh Shrestha, and Joshua Tobkin. Moonshot: Optimizing chain-based rotating leader bft via optimistic proposals. In *International Conference on Dependable Systems and Networks (DSN)*, 2024.

[19] Sisi Duan, Haibin Zhang, Xiao Sui, Baohan Huang, Changchun Mu, Gang Di, and Xiaoyun Wang. Dashing and star: Byzantine fault tolerance with weak certificates. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pages 250–264, 2024.

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

[21] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In *International Conference on Financial Cryptography and Data Security*, pages 296–315. Springer, 2022.

[22] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*, pages 51–68, 2017.

[23] Neil Giridharan, Florian Suri-Payer, Ittai Abraham, Lorenzo Alvisi, and Natacha Crooks. Autobahn: Seamless high speed bft. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pages 1–23, 2024.

[24] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is dag. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 165–175, 2021.

[25] Idit Keidar, Oded Naor, Ouri Poupko, and Ehud Shapiro. Cordial miners: Fast and efficient consensus for every eventuality. In *37th International Symposium on Distributed Computing (DISC 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023.

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

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

[28] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 31–42, 2016.

[29] Kamilla Nazirkhanova, Joachim Neu, and David Tse. Information dispersal with provable retrievability for rollups. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, pages 180–197, 2022.

[30] [n.d.]. Google Cloud Platform - general purpose machines. https://cloud.google.com/compute/docs/general-purpose-machines. [Online; accessed 06/03/2024].

[31] [n.d.]. Google Cloud Platform - Network Bandwidth. https://cloud.google.com/compute/docs/network-bandwidth. [Online; accessed 06/03/2024].

[32] Radius. Radius: Trustless shared sequencing layer, 2023. Accessed on Sept 30, 2024.

[33] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[34] Facebook Research. Bullshark github repository. https://github.com/facebookresearch/narwhal/tree/bullshark. [Online; accessed 30-September-2024].

[35] Nibesh Shrestha. Sailfish github repository. https://github.com/nibeshrestha/sailfish. [Online; accessed 30-September-2024].

[36] Nibesh Shrestha, Rohan Shrothrium, Aniket Kate, and Kartik Nayak. Sailfish: Towards improving the latency of dag-based bft. In *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2025 (To appear).

[37] Alexander Spiegelman, Balaji Aurn, Rati Gelashvili, and Zekun Li. Shoal: Improving dag-bft latency and robustness. In *International Conference on Financial Cryptography and Data Security*, 2024.

[38] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: Dag bft protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2705–2718, 2022.

[39] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: The partially synchronous version. *arXiv preprint arXiv:2209.05633*, 2022.

[40] Espresso Systems. The espresso sequencer, 2023. Accessed on Sept 30, 2024.

[41] Maven 11 Venture. The shared sequencer, 2023. Accessed on Sept 30, 2024.

[42] Xin Wang, Haochen Wang, Haibin Zhang, and Sisi Duan. Pando: Extremely scalable bft based on committee sampling. *Cryptology ePrint Archive*, 2024.

[43] Jian Yin, Jean-Philippe Martin, Arun Venkataramani, Lorenzo Alvisi, and Mike Dahlin. Separating agreement from execution for byzantine

fault tolerant services. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 253–267, 2003.

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

[45] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 931–948, 2018.

[46] Jianting Zhang, Zhongtang Luo, Raghavendra Ramesh, and Aniket Kate. Sharding smr with optimal-size shards for highly scalable blockchains. *arXiv preprint arXiv:2406.08252*, 2024.

APPENDIX A
SECURITY ANALYSIS OF ROUND OPTIMAL TRIBE-ASSISTED RBC

Lemma 3 (Validity)**.** *The protocol in Figure 3 satisfies Validity, except with a negligible error probability.*

Proof. Observe that an honest sender P_k sends $\langle \mathsf{VAL}, m, r\rangle$ to all parties in $\mathcal{P}_c$ and $\langle \mathsf{VAL}, H(m), r\rangle$ to all parties in $\mathcal{P} \setminus \mathcal{P}_c$. Consequently, all honest parties will eventually send $\langle \mathsf{ECHO}, H(m), r\rangle$. As a result, all honest parties will eventually receive at least $2f+1$ distinct $\langle \mathsf{ECHO}, H(m), r\rangle$ messages, including at least f_c+1 from the parties in $\mathcal{P}_c$. Furthermore, honest parties in $\mathcal{P}_c$ will receive m from the sender P_k. Therefore, all honest parties will invoke r_deliver(y, r, P_k), where $y = m$ if $P_i \in \mathcal{P}_c$ and $y = H(m)$ if $P_i \notin \mathcal{P}_c$.

Note that honest parties may fail to receive at least $f_c + 1$ $\langle \mathsf{ECHO}, H(m), r\rangle$ messages only in the rare case where the clan has a dishonest majority, which occurs with negligible probability. Therefore, the validity property is maintained except with a negligible probability of failure. □

Lemma 4 (Agreement)**.** *The protocol in Figure 3 satisfies Agreement, except with a negligible error probability.*

Proof. Suppose an honest party P_i outputs r_deliver$_i(y, r, P_k)$ (where $y = m$ when $P_i \in \mathcal{P}_c$ and $y = H(m)$ when $P_i \notin \mathcal{P}_c$). This implies P_i must have received $2f+1$ $\langle \mathsf{ECHO}, H(m), r\rangle$ messages, including at least $f_c + 1$ from the parties in $\mathcal{P}_c$. P_i must have sent $\mathcal{EC}_r(m)$ to all parties. Consequently, all honest parties will eventually receive $\mathcal{EC}_r(m)$ and honest parties in $\mathcal{P} \setminus \mathcal{P}_c$ will invoke r_deliver$(H(m), r, P_k)$.

Moreover, observe that at least one honest party $P_j \in \mathcal{P}_c$ must have received the value m, except with negligible error probability. Consequently, other honest parties in $\mathcal{P}_c$ will download value m from P_j and invoke r_deliver(m, r, P_k), except with negligible error probability. □

Theorem 2. *The protocol in Figure 3 is a tribe-assisted reliable broadcast satisfying Definition 2, except for a negligible probability of error.*

Proof. The integrity property is straightforward from the protocol, as a party can deliver a value at most once. The validity and agreement property follows from Lemma 3 and Lemma 4 respectively. □