# Scalable and Probabilistic Leaderless BFT Consensus through Metastability

Team Rocket*
t-rocket@protonmail.com

Revision: 04/18/2019 20:49:12 UTC

***Abstract*—This paper introduces a family of leaderless Byzantine fault tolerance protocols, built around a metastable mechanism via network subsampling. These protocols provide a strong probabilistic safety guarantee in the presence of Byzantine adversaries while their concurrent and leaderless nature enables them to achieve high throughput and scalability. Unlike blockchains that rely on proof-of-work, they are quiescent and green. Unlike traditional consensus protocols where one or more nodes typically process linear bits in the number of total nodes per decision, no node processes more than logarithmic bits. It does not require accurate knowledge of all participants and exposes new possible tradeoffs and improvements in safety and liveness for building consensus protocols.**

**The paper describes the Snow protocol family, analyzes its guarantees, and describes how it can be used to construct the core of an internet-scale electronic payment system called Avalanche, which is evaluated in a large scale deployment. Experiments demonstrate that the system can achieve high throughput (3400 tps), provide low confirmation latency (1.35 sec), and scale well compared to existing systems that deliver similar functionality. For our implementation and setup, the bottleneck of the system is in transaction verification.**

## I. Introduction

Achieving agreement among a set of distributed hosts lies at the core of countless applications, ranging from Internet-scale services that serve billions of people [12], [30] to cryptocurrencies worth billions of dollars [1]. To date, there have been two main families of solutions to this problem. Traditional consensus protocols rely on all-to-all communication to ensure that all correct nodes reach the same decisions with absolute certainty. Because they require quadratic communication overhead and accurate knowledge of membership, they have been difficult to scale to large numbers of participants. On the other hand, Nakamoto consensus protocols [8], [24], [26], [35], [43]–[46], [53]–[55] have become popular with the rise of Bitcoin. These protocols provide a probabilistic safety guarantee: Nakamoto consensus decisions may revert with some probability $\varepsilon$. A protocol parameter allows this probability to be rendered arbitrarily small, enabling high-value financial systems to be constructed on this foundation. This family is a natural fit for open, permissionless settings where any node can join the system at any time. Yet, these protocols are costly, wasteful, and limited in performance. By construction, they cannot quiesce: their security relies on constant participation by miners, even when there are no decisions to be made. Bitcoin currently consumes around 63.49 TWh/year [20], about twice as all of Denmark [14]. Moreover, these protocols suffer from an inherent scalability bottleneck that is difficult to overcome through simple reparameterization [17].

This paper introduces a new family of consensus protocols called Snow. Inspired by gossip algorithms, this family gains its properties through a deliberately metastable mechanism. Specifically, the system operates by repeatedly sampling the network at random, and steering correct nodes towards a common outcome. Analysis shows that this metastable mechanism is powerful: it can move a large network to an irreversible state quickly, where the irreversibility implies that a sufficiently large portion of the network has accepted a proposal and a conflicting proposal will not be accepted with any higher than negligible ($\varepsilon$) probability.

Similar to Nakamoto consensus, the Snow protocol family provides a probabilistic safety guarantee, using a tunable security parameter that can render the possibility of a consensus failure arbitrarily small. Unlike Nakamoto consensus, the protocols are green, quiescent and efficient; they do not rely on proof-of-work [23] and do not consume energy when there are no decisions to be made. The efficiency of the protocols stems partly from removing the leader bottleneck: each node requires $\mathcal{O}(1)$ communication overhead per round and $\mathcal{O}(\log n)$ rounds in expectation, whereas classical consensus protocols have one or more nodes that require $\mathcal{O}(n)$ communication per round (phase). Further, the Snow family tolerates discrepancies in knowledge of membership, as we discuss later. In contrast, classical consensus protocols require the full and accurate knowledge of $n$ as its safety foundation.

Snow's subsampled voting mechanism has two additional properties that improve on previous approaches for consensus. Whereas the safety of quorum-based approaches breaks down immediately when the predetermined threshold $f$ is exceeded, Snow's probabilistic safety guarantee degrades smoothly when Byzantine participants exceed $f$. This makes it easier to pick the critical threshold $f$. It also exposes new tradeoffs between safety and liveness: the Snow family is more efficient when the fraction of Byzantine nodes is small, and it can be parameterized to tolerate more than a third of the Byzantine nodes by trading off liveness.

To demonstrate the potential of this protocol family, we illustrate a practical peer-to-peer payment system, Avalanche. In effect, Avalanche executes multiple Snowball instances with the aid of a Directed Acyclic Graph (DAG). The DAG serves to piggyback multiple instances, reducing the cost from $\mathcal{O}(\log n)$ to $\mathcal{O}(1)$ per node and streamlining the path where there are

[*] `0x 8a5d 2d32 e68b c500 36e4 d086 0446 17fe 4a0a 0296 b274 999b a568 ea92 da46 d533`

no conflicting transactions.

Overall, the main contribution of this paper is to introduce a brand new family of consensus protocols, based on randomized sampling and metastable decision. The next section provides the model, goals and necessary assumptions for the new protocols. Section III gives intuition behind the protocols, followed by their full specification, Section IV provides methodology used by our formal analysis of safety and liveness in Appendix A, Section V describes Avalanche, a Bitcoin-like payment system, Section VI evaluates Avalanche, Section VII presents related work, and finally, Section VIII summarizes our contributions.

## II. Model and Goals

### a) Key Guarantees

*Safety*: Unlike classical consensus protocols, and similar to longest-chain-based consensus protocols such as Nakamoto consensus [43], we adopt an $\varepsilon$-safety guarantee that is probabilistic. In practice, this probabilistic guarantee is as strong as traditional safety guarantees, since appropriately small choices of $\varepsilon$ can render consensus failure negligible, lower than the probability of hardware failure due to random events.

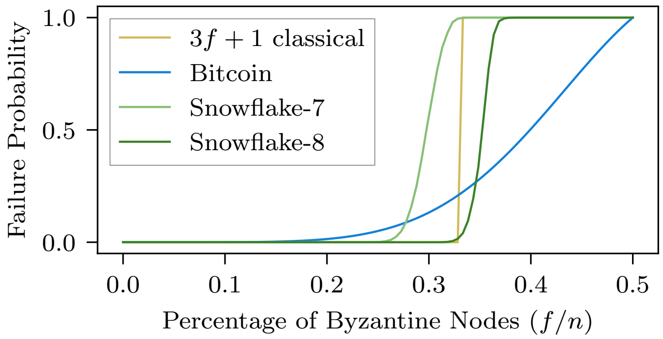


Fig. 1: The relation between $f/n$ and the probability of system safety failure (decision of two conflicting proposals), given a choice of finality. Classical BFT protocols that tolerate $f$ failures will encounter total safety failure when the threshold is exceeded even by one additional node. The Bitcoin curve shows a typical finality choice for Bitcoin where a block is considered final when it is "burried" in a branch having 6 additional blocks compared to any other competing forks. Snowflake belongs to the Snow family, and it is configured with $k = 10$, $\beta = 150$. Snowflake-7,8 uses $\alpha = 7$ and $\alpha = 8$ respectively.

*Liveness*: All our protocols provide a non-zero probability guarantee of termination within a bounded amount of time. This bounded guarantee is similar to various protocols such as Ben-Or [7] and longest-chain protocols. In particular, for Nakamoto consensus, the number of required blocks for a transaction increases exponentially with the number of adversarial nodes, with an asymptote at $f = n/2$ wherein the number is infinite. In other words, the time required for finality approaches $\infty$ as $f$ approaches $n/2$ (Figure 3). Furthermore, the required number of rounds is calculable ahead of time, as to allow the system designer to tune liveness at the expense of safety. Lastly, unlike traditional consensus protocols and similar to Nakamoto, our protocols benefit from lower adversarial presence, as discussed in property P3 below.

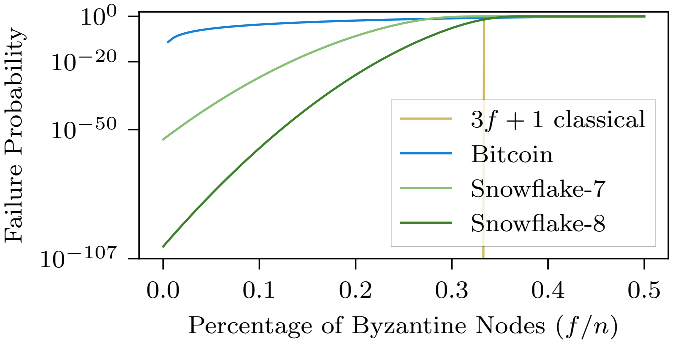


Fig. 2: Figure 1 with log-scaled y-axis.

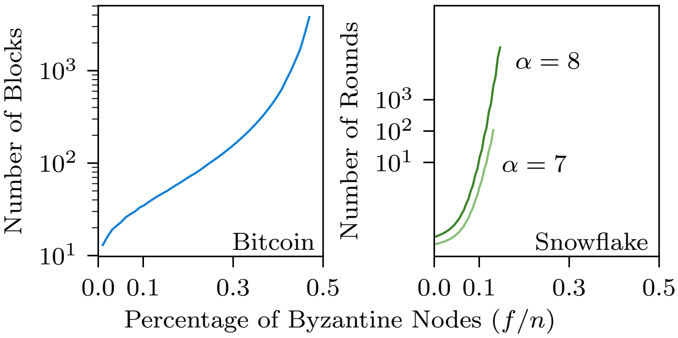


Fig. 3: The relation between $f/n$ and the convergence speed, given $\varepsilon = 10^{-20}$. The left figure shows the expected number of blocks to guarantee $\varepsilon$ in Bitcoin, which, counter to commonly accepted folk wisdom, is not a constant 6, but depends on adversary size to withhold the same $\varepsilon$. The right figure shows the maximum number of rounds required by Snowflake, where being different from Bitcoin, the asymptote is below 0.5 and varies by the choice of parameters.

*Formal Guarantees*: Let the system be parameterized for an $\varepsilon$ safety failure probability under a maximum expected $f$ number of adversarial nodes. Let $\mathcal{O}(\log n) < t_{max} < \infty$ be the upper bound of the execution of the protocols. The Snow protocols then provide the following guarantees:

**P1. Safety.** When decisions are made by any two correct nodes, they decide on conflicting transactions with negligible probability ($\leq \varepsilon$).

**P2. Liveness (Upper Bound).** Snow protocols terminate with a strictly positive probability within $t_{max}$ rounds.

**P3. Liveness (Lower Bound).** If $f \leq \mathcal{O}(\sqrt{n})$, then the Snow protocols terminate with high probability ($\geq 1-\varepsilon$) in $\mathcal{O}(\log n)$ rounds.

### b) Network

In the standard definition of asynchrony [7], message transmission is finite, but the distribution is undefined. This implies that the scheduling of message transmission itself could behave arbitrarily, and potentially even maliciously. We use a modified version of this model, which is well-accepted [6], [22], [25], [33], [39] in the analysis of epidemic networks and gossip-based stochastic systems. In particular, we fix the distribution of message delay to that of the exponential distribution. We note that, just like in the standard asynchronous model, there is a strictly non-zero probability that any correct node may execute its next local round only after an arbitrarily large amount of time has passed. Furthermore, we also note that scheduling only applies to correct nodes, and the adversary may execute arbitrarily, as discussed later.

*c) Achieving Liveness*

Classical consensus that works with asynchrony does not get stuck in a single phase of voting because the vote initiator always polls votes from all known participants and wait for $n - f$ responses. In our system, however, nodes operate via subsampling, hence it is possible for a single sample to select a majority of adversarial nodes, and therefore the node gets stuck waiting for the responses. To ensure liveness, a node should be able to wait with some timeout. Therefore, our protocols are synchronous in order to guarantee liveness. Lastly, it is worth noting that Nakamoto consensus is synchronous, in which the required difficulty of proof-of-work is dependent on the maximum network delay [44].

*d) Adversary*

The adversarial nodes execute under their own internal scheduler, which is unbounded in speed, meaning that all adversarial nodes can execute at any infinitesimally small point in time, unlike correct nodes. The adversary can view the state of every honest node at all times and can instantly modify the state of all adversarial nodes. It cannot, however, schedule or modify communication between correct nodes. Finally, we make zero assumptions about the behavior of the adversary, meaning that it can choose any execution strategy of its liking. In short, the adversary is computationally bounded (it cannot forge digital signatures) but otherwise is point-to-point informationally unbounded (knows all state) and round-adaptive (can modify its strategy at any time).

*e) Sybil Attacks*

Consensus protocols provide their guarantees based on assumptions that only a fraction of participants are adversarial. These bounds could be violated if the network is naively left open to arbitrary participants. In particular, a Sybil attack [21], wherein a large number of identities are generated by an adversary, could be used to exceed the adversarial bound.

A long line of work, including PBFT [13], treats the Sybil problem separately from consensus, and rightfully so, as Sybil control mechanisms are distinct from the underlying, more complex agreement protocol[1]. Nakamoto consensus, for instance, uses proof-of-work [4] to limit Sybils, which requires miners to continuously stake a hardware investment. Other protocols, discussed in Section VII, rely on proof-of-stake or proof-of-authority. The consensus protocols presented in this paper can adopt any Sybil control mechanism, although proof-of-stake is most aligned with their quiescent operation. One can use an already established proof-of-stake based mechanism [27]. The full design of a peer-to-peer payment system incorporating staking, unstaking and minting mechanism is beyond the scope of this paper, whose focus is on the core consensus protocol.

*f) Flooding Attacks*

Flooding/spam attacks are a problem for any distributed system. Without a protection mechanism, an attacker can generate large numbers of transactions and flood protocol data structures, consuming storage. There are a multitude of techniques to deter such attacks, including network-layer protection, proof-of-authority, local proof-of-work and economic mechanisms. In Avalanche, we use transaction fees, making such attacks costly even if the attacker is sending money back to addresses under its control.

*g) Additional Assumptions*

We do not assume that all members of the network are known to all participants, but rather may temporarily have some discrepancies in network view. We quantify the bounds on the discrepancy in Appendix A-F. We assume a safe bootstrapping mechanism, similar to that of Bitcoin, that enables a node to connect with sufficiently many correct nodes to acquire a statistically unbiased view of the network. We do not assume a PKI. Finally, we make standard cryptographic assumptions related to digital signatures and hash functions.

## III. Protocol Design

We start with a non-BFT protocol called Slush and progressively build up to Snowflake and Snowball, all based on the same common majority-based metastable voting mechanism. These protocols are single-decree consensus protocols of increasing robustness. We provide full specifications for the protocols in this section, and defer the analysis to the next section, and present formal proofs in the appendix.

### *A. Slush: Introducing Metastability*

The core of our approach is a single-decree consensus protocol, inspired by epidemic or gossip protocols. The simplest protocol, Slush, is the foundation of this family, shown in Figure 4. Slush is *not* tolerant to Byzantine faults, only crash-faults (CFT), but serves as an illustration for the BFT protocols that follow. For ease of exposition, we will describe the operation of Slush using a decision between two conflicting colors, red and blue.

In Slush, a node starts out initially in an uncolored state. Upon receiving a transaction from a client, an uncolored node updates its own color to the one carried in the transaction and initiates a query. To perform a query, a node picks a small, constant sized ($k$) sample of the network uniformly at random, and sends a query message. Upon receiving a query, an uncolored node adopts the color in the query, responds with that color, and initiates its own query, whereas a colored node simply responds with its current color. Once the querying node collects $k$ responses, it checks if a fraction $\geq \alpha$ are for the same color, where $\alpha > \lfloor k/2 \rfloor$ is a protocol parameter. If the $\alpha$ threshold is met and the sampled color differs from the node's

[1] This is not to imply that every consensus protocol can be coupled/decoupled with every Sybil control mechanism.

```
procedure onQuery($v, \text{col}'$)
    if col $= \perp$ then col := col$'$
    respond($v, \text{col}$)
procedure slushLoop($u, \text{col}_0 \in \{\texttt{R}, \texttt{B}, \perp\}$)
    col := col$_0$ // initialize with a color
    for $r \in \{1 \ldots m\}$ do
        // if $\perp$, skip until onQuery sets the color
        if col $= \perp$ then continue
        // randomly sample from the known nodes
        $\mathcal{K}$ := sample($\mathcal{N} \backslash u, k$)
        $P$ := [query($v, \text{col}$)    for $v \in \mathcal{K}$]
        for col$' \in \{\texttt{R}, \texttt{B}\}$ do
            if $P$.count(col$'$) $\geq \alpha$ then
                col := col$'$
    accept(col)
```

Fig. 4: Slush protocol. Timeouts elided for readability.

own color, the node flips to that color. It then goes back to the query step, and initiates a subsequent round of query, for a total of $m$ rounds. Finally, the node decides the color it ended up with at time $m$.

Slush has a few properties of interest. First, it is almost *memoryless*: a node retains no state between rounds other than its current color, and in particular maintains no history of interactions with other peers. Second, unlike traditional consensus protocols that query every participant, every round involves sampling just a small, constant-sized slice of the network at random. Third, Slush makes progress under any network configuration (even fully bivalent state, i.e. 50/50 split between colors), since random perturbations in sampling will cause one color to gain a slight edge and repeated samplings afterwards will build upon and amplify that imbalance. Finally, if $m$ is chosen high enough, Slush ensures that all nodes will be colored identically with high probability (whp). Each node has a constant, predictable communication overhead per round, and $m$ grows logarithmically with $n$.

The Slush protocol does not provide a strong safety guarantee in the presence of Byzantine nodes. In particular, if the correct nodes develop a preference for one color, a Byzantine adversary can attempt to flip nodes to the opposite so as to keep the network in balance, preventing a decision. We address this in our first BFT protocol that introduces more state storage at the nodes.

## B. Snowflake: BFT

Snowflake augments Slush with a single counter that captures the strength of a node's conviction in its current color. This per-node counter stores how many consecutive samples of the network by that node have all yielded the same color. A node accepts the current color when its counter exceeds $\beta$, another security parameter. Figure 5 shows the amended protocol, which includes the following modifications:
1) Each node maintains a counter $cnt$;
2) Upon every color change, the node resets $cnt$ to 0;
3) Upon every successful query that yields $\geq \alpha$ responses for the same color as the node, the node increments $cnt$.

```
procedure snowflakeLoop($u, \text{col}_0 \in \{\texttt{R}, \texttt{B}, \perp\}$)
    col := col$_0$, cnt := 0
    while undecided do
        if col $= \perp$ then continue
        $\mathcal{K}$ := sample($\mathcal{N} \backslash u, k$)
        $P$ := [query($v, \text{col}$)    for $v \in \mathcal{K}$]
        for col$' \in \{\texttt{R}, \texttt{B}\}$ do
            if $P$.count(col$'$) $\geq \alpha$ then
                if col$' \neq$ col then
                    col := col$'$, cnt := 0
                else
                    if ++cnt $> \beta$ then accept(col)
```

Fig. 5: Snowflake.

```
procedure snowballLoop($u, \text{col}_0 \in \{\texttt{R}, \texttt{B}, \perp\}$)
    col := col$_0$, lastcol := col$_0$, cnt := 0
    $d[\texttt{R}]$ := 0, $d[\texttt{B}]$ := 0
    while undecided do
        if col $= \perp$ then continue
        $\mathcal{K}$ := sample($\mathcal{N} \backslash u, k$)
        $P$ := [query($v, \text{col}$)    for $v \in \mathcal{K}$]
        for col$' \in \{\texttt{R}, \texttt{B}\}$ do
            if $P$.count(col$'$) $\geq \alpha$ then
                $d[\text{col}']$++
                if $d[\text{col}'] > d[\text{col}]$ then
                    col := col$'$
                if col$' \neq$ lastcol then
                    lastcol := col$'$, cnt := 0
                else
                    if ++cnt $> \beta$ then accept(col)
```

Fig. 6: Snowball.

When the protocol is correctly parameterized for a given threshold of Byzantine nodes and a desired $\varepsilon$-guarantee, it can ensure both safety (P1) and liveness (P2, P3). As we later show, there exists an irreversible state after which a decision is inevitable. Correct nodes begin to commit past the irreversible state to adopt the same color, whp. For additional intuition, which we do not expand in this paper, there also exists a phase-shift point, where the Byzantine nodes lose ability to keep network in a bivalent state.

## C. Snowball: Adding Confidence

Snowflake's notion of state is ephemeral: the counter gets reset with every color flip. Snowball augments Snowflake with *confidence counters* that capture the number of queries that have yielded a threshold result for their corresponding color (Figure 6). A node decides if it gets $\beta$ consecutive chits for a color. However, it only changes preference based on the total accrued confidence. The differences between Snowflake and Snowball are as follows:
1) Upon every successful query, the node increments its confidence counter for that color.
2) A node switches colors when the confidence in its current color becomes lower than the confidence value of the new color.

# IV. Analysis

Due to space limits, we move some core details to Appendix A, where we show that under certain independent and distinct assumptions, the Snow family of consensus protocols provide safety (P1) and liveness (P2, P3) properties. In this section, we summarize our core results and provide some proof sketches.

#### *a) Notation*

Let the network consist of a set $n$ nodes (represented by set $\mathcal{N}$), where $c$ are correct nodes (represented by set $\mathcal{C}$) and $f$ are Byzantine nodes (represented by set $\mathcal{B}$). Let $u, v \in \mathcal{C}$ refer to any two correct nodes in the network. Let $k, \alpha, \beta \in \mathbb{Z}^{+}$ be positive integers where $\alpha > \lfloor k/2 \rfloor$. From now on, $k$ will always refer to the network sample size, where $k \leq n$, and $\alpha$ will be the majority threshold required to consider the voting experiment a "success". In general, we will refer to $\mathcal{S}$ as the state (or configuration) of the network at any given time.

#### *b) Modelling Framework*

To formally model our protocols, we use continuous-time Markov processes (CTMC). The state space is enumerable (and finite), and state transitions occur in continuous time. CTMCs naturally model our protocols since state transitions do not occur in epochs and in lockstep for every node (at the end of every time unit) but rather occur at any time and independently of each other.

We focus on binary consensus, although the safety results generalize to more than two values. We can think of the network as a set of nodes either colored red or blue, and we will refer to this configuration at time $t$ as $\mathcal{S}_t$. We model our protocols through a continuous-time process with two absorbing states, where either all nodes are red or all nodes are blue. The state space $\mathcal{S}$ of the stochastic process is a condensed version of the full configuration space, where each state $\{0, \ldots, n\}$ represents the total number of blue nodes in the system.

The simplification that allows us to analyze this system is to obviate the need to keep track of all of the execution paths, as well as all possible adversarial strategies, and rather focus entirely on a single state of interest, without regards to how we achieve this state. More specifically, the core extractable insight of our analysis is in identifying the *irreversibility* state of the system, the state upon which so many correct nodes have usurped either red or blue that reverting back to the minority color is highly unlikely.

### *A. Safety*

#### *a) Slush*

Unless explicitly stated, we assume that $\mathcal{L}(u) = \mathcal{N}$ for all $u \in \mathcal{N}$. We model the dynamics of the system through a continuous-time process where two states are absorbing, namely the all-red or all-blue state[2]. Let $\{X_{t \geq 0}\}$ be the random variable that describes the state of the system at time $t$, where $X_0 = \{0, \ldots, c\}$. We begin by immediately discussing the most important result of the safety dynamics of our processes: the *reversibility* probabilities of the **Slush** process. All the other formal results in this paper are, informally speaking, intuitive derivations and augmentations of this result.

**Theorem 1.** *Let the configuration of the system at time $t$ be $\mathcal{S}_t = n/2 + \delta$, meaning that the network has drifted to $2\delta$ more blue nodes than red nodes ($\delta = 0$ means that red and blue are equal). Let $\xi_\delta$ be the probability of absorption to the all-red state (minority). Then, for all $0 \leq \delta \leq n/2$, we have*

$$\begin{aligned} \xi_\delta &\leq \left(\frac{1/2 - \delta/n}{\alpha/k}\right)^{\alpha} \left(\frac{1/2 + \delta/n}{1 - \alpha/k}\right)^{k-\alpha} \\ &\leq e^{-2((\alpha/k)-(1/2)+(\delta/n))^2 k} \end{aligned} \quad (1)$$

*Proof.* This bound follows from the Hoeffding-derived tail bounds of the hypergeometric distribution by Chvatal [15]. □

We note that Chvatal's bounds are introduced for simplicity of exposition and are extremely weak. We leave the full closed-form expression in Theorem 2 to the appendix, which is also significantly stronger than the Chvatal bound. Nonetheless, using the loose Chvatal bound, we make the key observation that as the drift $\delta$ increases, given fixed $\alpha$ and $k$, the probability of moving towards the minority value decreases *exponentially fast* (in fact, even faster, since there is a quadratic term in the inverse exponent). Additionally, the same result holds for increasing $\alpha$ given a fixed $k$.

The outcomes of this theorem demonstrate a key property: once the network loses full bivalency (i.e. $\delta > 0$), it tends to topple and converge rapidly towards the majority color, unable to revert back to the minority with significant probability. This is the fundamental property exploited by our protocols, and what makes them secure despite only sampling a small, constant-sized set of the network. The core result that follows for the safety guarantees in Snowflake is in finding regions (given specific parameter choices) where the reversibility holds with no higher than $\varepsilon$ probability even under adversarial presence.

#### *b) Snowflake*

For Snowflake, we relax the assumption that all nodes are correct and assume that some fraction of nodes are adversarial. In Slush, once the network gains significant majority support for one proposal (e.g., the color blue), it becomes unlikely for a minority proposal (e.g., the color red) to ever become decided in the future (irreversibility). Furthermore, in Slush nodes simply have to execute the protocol for a deterministic number of rounds, $m$, which is known ahead of protocol execution. When introducing adversarial nodes with arbitrary strategies, however, nodes cannot simply execute the protocol for a deterministic number of rounds, since the adversary may nondeterministically affect the value of $m$. Instead, correct nodes must implement a mechanism to *explicitly* detect that irreversibility has been reached. To that end, in Snowflake, every correct node implements a decision function, $\mathcal{D}(u, \mathcal{S}_t, blue) \rightarrow \{0, 1\}$, which is a random variable that outputs 1 if node $u$ detects that the network has reached an irreversibility state at time $t$ for blue. The decision mechanism is probabilistic, meaning that it can fail, although it is designed to do so with negligible probability. We now sketch the proof of Snowflake.

*Proof Sketch.* We define safety failure to be the event wherein any two correct nodes $u$ and $v$ decide on blue and red, i.e. $\mathcal{D}(u, \mathcal{S}_t, blue) \rightarrow 1$ and $\mathcal{D}(v, \mathcal{S}_{t'}, red) \rightarrow 1$, for any two

[2] Note that, in reality, we do not require that all nodes be the same color in order to ensure that we decide on that color, only $n - \alpha - 1$. This is only a simplification in our description.

times $t$ and $t'$. We again model the system as a continuous time random process. The state space is defined the same way as in Slush. However, we note some critical subtleties. First, unlike in Slush, where it is clear that, once nodes are the same color, a decision has been made, this is no longer the case for Snowflake. In fact, even if all correct nodes accept a color, it is entirely possible for a correct node to switch again. Second, we also have to consider the decision mechanism $\mathcal{D}(*)$. To analyze, we obviate the need to keep track of all possible network configurations under all possible adversarial strategies, and assume that a node $u$ first decides on blue. Then, conditioned on the state of the network upon $u$ deciding, we calculate the probability that another node $v$ decides red, which is a function of both the probability that the network reverts towards a minority blue state and that $v$ decides at that precise state. We show that under appropriate choices of $k$, $\alpha$, and $\beta$, we can construct highly secure instances of Snowflake (i.e. safety failure with probability $\leq \varepsilon$) when the network reaches some bias of $\delta$, as shown in Figure 7. A concrete example is provided in Figure 1.

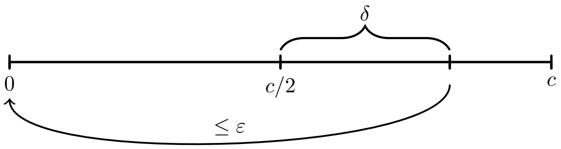


Fig. 7: Representation of the irreversibility state, which exists when – even under $f$ Byzantine nodes – the number of blue correct nodes exceeds that of red correct nodes by more than $2\delta$.

*c) Snowball*

Snowball is an improvement over Snowflake, where random perturbations in network samples are reduced by introducing a limited form of history, which we refer to as confidence. The fundamental takeaway is that the history enables Snowball to provide stronger security against safety failures than Snowflake.
*Proof Sketch.* We structure the model via a game of balls and urns, where each urn represents one of the correct nodes, and the ball counts correspond to confidences in either color. Using this model, the analysis applies martingale concentration inequalities to prove that once the system has reached the irreversibility state, then the growth of the confidence of the majority decided color will perpetually grow and drift further away from those of the minority color, effectively rendering reversibility less likely over time. If the drifts ever revert, then reversibility analysis becomes identical to that of Snowflake. Since now the adversary must overcome the confidence drifts, as well as the irreversibility dynamics, the security of Snowball is strictly stronger than that of Snowflake.

## B. Liveness

We assume that the observed adversarial presence $0 \leq f' \leq n(k - \alpha - \psi)/k \leq f$, where we refer to $\psi$ as the buffer-zone. The bigger $\psi$, the quicker the ability of the decision mechanism to finalize a value. If, of course, $\psi$ approaches zero or becomes negative, then we violate the upper bound of adversarial tolerance for the parameterized system, and thus the adversary can, with high probability, stall termination by simply choosing to not respond, although the safety guarantees may still hold.

Assuming that $\psi$ is strictly positive, termination is strictly finite under all network configurations where a proposal has at least $\alpha$ support. Furthermore, not only is termination finite with probability one, we also have a strictly positive probability of termination within any bounded amount of time $t_{max}$, as discussed in Lemma 4, which follows from Theorem 3. This captures liveness property P2.
*Proof Sketch.* Using the construction of the system to prove irreversibility, we characterize the distribution of the average time spent (sojourn times) at each state before the system terminates execution by absorption at either absorbing state. The termination time is then a union of these times.

For non-conflicting transactions, since the adversary is unable to forge a conflict, the time to decision is simply the mixing time of the network starting from a configuration where every correct node is uninitialized.
*Proof Sketch.* Mixing times for gossip is well characterized to be as $\mathcal{O}(\log n)$, and this result holds for all our protocols.

Liveness guarantees under a fully bilavent network configuration reduce to an optimal convergence time of $\mathcal{O}(\log n)$ rounds if the adversary is at most $\mathcal{O}(\sqrt{n})$, for $\alpha = \lfloor k/2 \rfloor + 1$. We leave additional detains to Lemma 5. When the adversary surpasses $\mathcal{O}(\sqrt{n})$ nodes, the worst case number of rounds increases polynomially, and as $f$ approaches $n/2$ it approaches exponential convergence rates.
*Proof Sketch.* We modify Theorem 3 to include the adversary, which reverts any imbalances in the network by keeping network fully bivalent.

*a) Multi-Value Consensus*

Our binary consensus protocol could support multi-value consensus by running logarithmic binary instances, one for each bit of the proposed value. However, such theoretical reduction might not be efficient in practice. Instead, we could directly incoporate multi-values as multi-colors in the protocol, where safety analysis could still be generalized.

As for liveness, we sketch a leaderless initialization mechanism, which in expectation uses $\mathcal{O}(\log n)$ rounds under the assumption that the network is synchronized. Every node operates in three phases: in the first phase, it gossips and collects proposals for $\mathcal{O}(\log n)$ rounds, where each round lasts for the maximum message delay; in the second phase, each node stops collecting proposals, and instead gossips all new values for an additional $\mathcal{O}(\log n)$ rounds; in the third phase, each node samples the proposals it knows of locally, checking for values that have an $\alpha$ majority, ordered deterministically, such as by hash values. Finally, a node selects the first value by the order as its initial state when it starts the subsequent consensus protocol. In a cryptocurrency setting, the deterministic ordering function would incorporate fees paid out for every new proposal, which means that the adversary is financially limited in its ability to launch a fairness attack against the initialization. While the design of initialization mechanisms is interesting, note that it is not necessary for a decentralized payment system, as we show in Section V.

Finally, we discuss churn and view discrepancies in the appendix.

## V. Peer-to-Peer Payment System

We have implemented a bare-bones payment system, Avalanche, which supports Bitcoin transactions. In this section, we describe the design and sketch how the implementation can support the value transfer primitive at the center of cryptocurrencies. Deploying a full cryptocurrency involves bootstrapping, minting, staking, unstaking, and inflation control. While we have solutions for these issues, their full discussion is beyond the scope of this paper, whose focus is centered on the novel Snow consensus protocol family.

In a cryptocurrency setting, cryptographic signatures enforce that only a key owner is able to create a transaction that spends a particular coin. Since correct clients follow the protocol as prescribed and never double spend coins, in Avalanche, they are guaranteed both safety and liveness for their *virtuous* transactions. In contrast, liveness is not guaranteed for *rogue* transactions, submitted by Byzantine clients, which conflict with one another. Such decisions may stall in the network, but have no safety impact on virtuous transactions. We show that this is a sensible tradeoff, and that resulting system is sufficient for building complex payment systems.

### A. Avalanche: Adding a DAG

Avalanche consists of multiple single-decree Snowball instances instantiated as a multi-decree protocol that maintains a dynamic, append-only directed acyclic graph (DAG) of all known transactions. The DAG has a single sink that is the *genesis vertex*. Maintaining a DAG provides two significant benefits. First, it improves efficiency, because a single vote on a DAG vertex implicitly votes for all transactions on the path to the genesis vertex. Second, it also improves security, because the DAG intertwines the fate of transactions, similar to the Bitcoin blockchain. This renders past decisions difficult to undo without the approval of correct nodes.

When a client creates a transaction, it names one or more *parents*, which are included inseparably in the transaction and form the edges of the DAG. The parent-child relationships encoded in the DAG may, but do not need to, correspond to application-specific dependencies; for instance, a child transaction need not spend or have any relationship with the funds received in the parent transaction. We use the term *ancestor set* to refer to all transactions reachable via parent edges back in history, and *progeny* to refer to all children transactions and their offspring.

The central challenge in the maintenance of the DAG is to choose among *conflicting transactions*. The notion of conflict is application-defined and transitive, forming an equivalence relation. In our cryptocurrency application, transactions that spend the same funds (*double-spends*) conflict, and form a *conflict set* (shaded regions in Figure 11) , out of which only a single one can be accepted. Note that the conflict set of a virtuous transaction is always a singleton.

```
procedure INIT
    $\mathcal{T} := \varnothing$  // the set of known transactions
    $\mathcal{Q} := \varnothing$  // the set of queried transactions
procedure ONGENERATETX(data)
    edges $:= \{T' \leftarrow T : T' \in \textsc{parentSelection}(\mathcal{T})\}$
    $T := \mathrm{Tx}(\mathrm{data}, \mathrm{edges})$
    ONRECEIVETX($T$)
procedure ONRECEIVETX($T$)
    if $T \notin \mathcal{T}$ then
        if $\mathcal{P}_T = \varnothing$ then
            $\mathcal{P}_T := \{T\}$, $\mathcal{P}_T$.pref $:= T$
            $\mathcal{P}_T$.last $:= T, \mathcal{P}_T$.cnt $:= 0$
        else $\mathcal{P}_T := \mathcal{P}_T \cup \{T\}$
        $\mathcal{T} := \mathcal{T} \cup \{T\}$, $c_T := 0$.
```

Fig. 8: Avalanche: transaction generation.

```
procedure AVALANCHELOOP
    while true do
        find $T$ that satisfies $T \in \mathcal{T} \wedge T \notin \mathcal{Q}$
        $\mathcal{K} := \textsc{sample}(\mathcal{N} \backslash u, k)$
        $P := \sum_{v \in \mathcal{K}} \textsc{query}(v, T)$
        if $P \geq \alpha$ then
            $c_T := 1$
            // update the preference for ancestors
            for $T' \in \mathcal{T} : T' \xleftarrow{*} T$ do
                if $d(T') > d(\mathcal{P}_{T'}.\mathrm{pref})$ then
                    $\mathcal{P}_{T'}$.pref $:= T'$
                if $T' \neq \mathcal{P}_{T'}$.last then
                    $\mathcal{P}_{T'}$.last $:= T'$, $\mathcal{P}_{T'}$.cnt $:= 0$
                else
                    ++$\mathcal{P}_{T'}$.cnt
        // otherwise, $c_T$ remains 0 forever
        $\mathcal{Q} := \mathcal{Q} \cup \{T\}$  // mark T as queried
```

Fig. 9: Avalanche: the main loop.

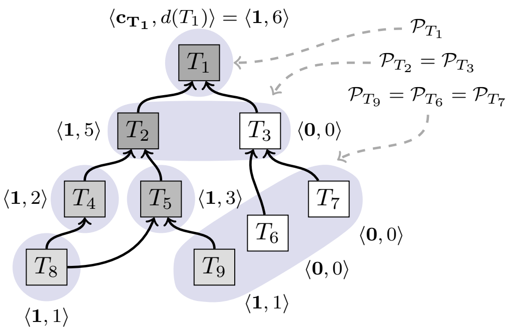
Avalanche embodies a Snowball instance for each conflict set. Whereas Snowball uses repeated queries and multiple counters to capture the amount of confidence built in conflicting transactions (colors), Avalanche takes advantage of the DAG structure and uses a transaction's progeny. Specifically, when a transaction $T$ is queried, all transactions reachable from $T$ by following the DAG edges are implicitly part of the query. A node will only respond positively to the query if $T$ and its entire ancestry are currently the preferred option in their respective conflict sets. If more than a threshold of responders vote positively, the transaction is said to collect a *chit*. Nodes then compute their *confidence* as the total number of chits in the progeny of that transaction. They query a transaction just once and rely on new vertices and possible chits, added to the progeny, to build up their confidence. Ties are broken by an initial preference for first-seen transactions. Note that chits are decoupled from the DAG structure, making the protocol immune to attacks where the attacker generates large, padded subgraphs.

### B. Avalanche: Specification

Each correct node $u$ keeps track of all transactions it has learned about in set $\mathcal{T}_u$, partitioned into mutually exclusive conflict sets $\mathcal{P}_T, T \in \mathcal{T}_u$. Since conflicts are transitive, if $T_i$

```
function isPreferred($T$)
    return $T = \mathcal{P}_T$.pref
function isStronglyPreferred($T$)
    return $\forall T' \in \mathcal{T}, T' \overset{*}{\leftarrow} T$ : isPreferred($T'$)
function isAccepted($T$)
    return
        $((\forall T' \in \mathcal{T}, T' \leftarrow T$ : isAccepted($T'$))
          $\wedge |\mathcal{P}_T| = 1 \wedge d(T) > \beta_1)$  // safe early commitment
        $\vee(\mathcal{P}_T$.cnt $> \beta_2)$  // consecutive counter
procedure onQuery($j, T$)
    onReceiveTx($T$)
    respond($j$, isStronglyPreferred($T$))
```

Fig. 10: Avalanche: voting and decision primitives.



$\langle \mathbf{c_{T_1}}, d(T_1)\rangle = \langle \mathbf{1}, 6\rangle$
$\mathcal{P}_{T_1}$
$\mathcal{P}_{T_2} = \mathcal{P}_{T_3}$
$\mathcal{P}_{T_9} = \mathcal{P}_{T_6} = \mathcal{P}_{T_7}$
$T_1$
$\langle \mathbf{1}, 5\rangle$
$T_2$
$T_3$
$\langle \mathbf{0}, 0\rangle$
$\langle \mathbf{1}, 2\rangle$
$T_4$
$T_5$
$\langle \mathbf{1}, 3\rangle$
$T_7$
$T_6$
$\langle \mathbf{0}, 0\rangle$
$T_8$
$T_9$
$\langle \mathbf{0}, 0\rangle$
$\langle \mathbf{1}, 1\rangle$
$\langle \mathbf{1}, 1\rangle$

Fig. 11: Example of $\langle$chit, confidence$\rangle$ values. Darker boxes indicate transactions with higher confidence values. At most one transaction in each shaded region will be accepted.

and $T_j$ are conflicting, then they belong to the same conflict set, i.e. $\mathcal{P}_{T_i} = \mathcal{P}_{T_j}$. It's worth noting this relation may sound counter-intuitive: conflicting transitions have the *equivalence* relation, because they are equivocations spending the *same* funds.

We write $T' \leftarrow T$ if $T$ has a parent edge to transaction $T'$, The "$\overset{*}{\leftarrow}$"-relation is its reflexive transitive closure, indicating a path from $T$ to $T'$. DAGs built by different nodes are guaranteed to be compatible, though at any one time, the two nodes may not have a complete view of all vertices in the system. Specifically, if $T' \leftarrow T$, then every node in the system that has $T$ will also have $T'$ and the same relation $T' \leftarrow T$; and conversely, if $T' \not\leftarrow T$, then no nodes will end up with $T' \leftarrow T$.

Each node $u$ can compute a confidence value, $d_u(T)$, from the progeny as follows:

$$d_u(T) = \sum_{T' \in \mathcal{T}_u, T \overset{*}{\leftarrow} T'} c_{uT'}$$

where $c_{uT'}$ stands for the chit value of $T'$ for node $u$. Each transaction initially has a chit value of 0 before the node gets the query results. If the node collects a threshold of $\alpha$ yes-votes after the query, the value $c_{uT'}$ is set to 1, otherwise remains 0 forever. Therefore, a chit value reflects the result from the one-time query of its associated transaction and becomes immutable afterwards, while $d(T)$ can increase as the DAG grows by collecting more chits in its progeny. Because $c_T \in \{0, 1\}$, confidence values are monotonic.

In addition, node $u$ maintains its own local list of known nodes $\mathcal{N}_u \subseteq \mathcal{N}$ that comprise the system. For simplicity, we assume for now $\mathcal{N}_u = \mathcal{N}$, and elide subscript $u$ in contexts without ambiguity.

Each node implements an event-driven state machine, centered around a query that serves both to solicit votes on each transaction and to notify other nodes of the existence of newly discovered transactions. In particular, when node $u$ discovers a transaction $T$ through a query, it starts a one-time query process by sampling $k$ random peers and sending a message to them, after $T$ is delivered via onReceiveTx.

Node $u$ answers a query by checking whether each $T'$ such that $T' \overset{*}{\leftarrow} T$ is currently preferred among competing transactions $\forall T'' \in \mathcal{P}_{T'}$. If every single ancestor $T'$ fulfills this criterion, the transaction is said to be *strongly preferred*, and receives a yes-vote (1). A failure of this criterion at any $T'$ yields a no-vote (0). When $u$ accumulates $k$ responses, it checks whether there are $\alpha$ yes-votes for $T$, and if so grants the chit (chit value $c_T := 1$) for $T$. The above process will yield a labeling of the DAG with a chit value and associated confidence for each transaction $T$.

Figure 11 illustrates a sample DAG built by Avalanche. Similar to Snowball, sampling in Avalanche will create a positive feedback for the preference of a single transaction in its conflict set. For example, because $T_2$ has larger confidence than $T_3$, its descendants are more likely collect chits in the future compared to $T_3$.

Similar to Bitcoin, Avalanche leaves determining the acceptance point of a transaction to the application. An application supplies an isAccepted predicate that can take into account the value at risk in the transaction and the chances of a decision being reverted to determine when to decide.

Committing a transaction can be performed through a *safe early commitment*. For virtuous transactions, $T$ is accepted when it is the only transaction in its conflict set and has a confidence greater than threshold $\beta_1$. As in Snowball, $T$ can also be accepted after a $\beta_2$ number of consecutive successful queries. If a virtuous transaction fails to get accepted due to a problem with parents, it could be accepted if reissued with different parents. Figure 8 shows how Avalanche performs parent selection and entangles transactions. Because transactions that consume and generate the same UTXO do not conflict with each other, any transaction can be reissued with different parents.

Figure 9 illustrates the protocol main loop executed by each node. In each iteration, the node attempts to select a transaction $T$ that has not yet been queried. If no such transaction exists, the loop will stall until a new transaction is added to $\mathcal{T}$. It then selects $k$ peers and queries those peers. If more than $\alpha$ of those peers return a positive response, the chit value is set to 1. After that, it updates the preferred transaction of each conflict set of the transactions in its ancestry. Next, $T$ is added to the set $\mathcal{Q}$ so it will never be queried again by the node. The code that selects additional peers if some of the $k$ peers are unresponsive is omitted for simplicity.

Figure 10 shows what happens when a node receives a query for transaction $T$ from peer $j$. First it adds $T$ to $\mathcal{T}$,

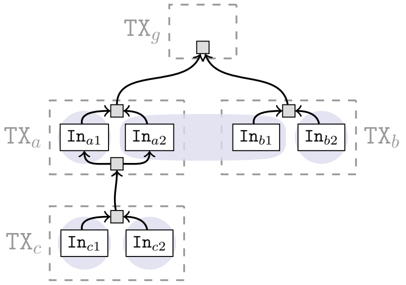


Fig. 12: The underlying logical DAG structure used by Avalanche. The tiny squares with shades are dummy vertices which just help form the DAG topology for the purpose of clarity, and can be replaced by direct edges. The rounded gray regions are the conflict sets.

unless it already has it. Then it determines if $T$ is currently strongly preferred. If so, the node returns a positive response to peer $j$. Otherwise, it returns a negative response. Notice that in the pseudocode, we assume when a node knows $T$, it also recursively knows the entire ancestry of $T$. This can be achieved by postponing the delivery of $T$ until its entire ancestry is recursively fetched. In practice, an additional gossip process that disseminates transactions is used in parallel, but is not shown in pseudocode for simplicity.

### *C. Multi-Input UTXO Transactions*

In addition to the DAG structure in Avalanche, an *unspent transaction output* (UTXO) [43] graph that captures spending dependency is used to realize the ledger for the payment system. To avoid ambiguity, we denote the transactions that encode the data for money transfer *transactions*, while we call the transactions ($T \in \mathcal{T}$) in Avalanche's DAG *vertices*.

We inherit the transaction and address mechanisms from Bitcoin. At their simplest, transactions consist of multiple inputs and outputs, with corresponding redeem scripts. Addresses are identified by the hash of their public keys, and signatures are generated by corresponding private keys. The full scripting language is used to ensure that a redeem script is authenticated to spend a UTXO. UTXOs are fully consumed by a valid transaction, and may generate new UTXOs spendable by named recipients. Multi-input transactions consume multiple UTXOs, and in Avalanche, may appear in multiple conflict sets. To account for these correctly, we represent *transaction-input* pairs (e.g. In$_{a1}$) as Avalanche vertices. The conflict relation of transaction-input pairs are transitive because of each pair only spends one unspent output. Then, we use the conjunction of isAccepted for all inputs of a transaction to ensure that no transaction will be accepted unless all its inputs are accepted (Figure 12). In other words, a transaction is accepted only if all its transaction-input pairs are accepted in their respective Snowball conflict sets. Following this idea, we finally implement the DAG of transaction-input pairs such that multiple transactions can be batched together per query.

#### *a) Optimizations*

We implement some optimizations to help the system scale. First, we use *lazy updates* to the DAG, because the recursive definition for confidence may otherwise require a costly DAG traversal. We maintain the current $d(T)$ value for each active vertex on the DAG, and update it only when a descendant vertex gets a chit. Since the search path can be pruned at accepted vertices, the cost for an update is constant if the rejected vertices have limited number of descendants and the undecided region of the DAG stays at constant size. Second, the conflict set could be very large in practice, because a rogue client can generate a large volume of conflicting transactions. Instead of keeping a container data structure for each conflict set, we create a mapping from each UTXO to the preferred transaction that stands as the representative for the entire conflict set. This enables a node to quickly determine future conflicts, and the appropriate response to queries. Finally, we speed up the query process by terminating early as soon as the $\alpha$ threshold is met, without waiting for $k$ responses.

#### *b) DAG*

Compared to Snowball, Avalanche introduces a DAG structure that entangles the fate of unrelated conflict sets, each of which is a single-decree instance. This entanglement embodies a tension: attaching a virtuous transaction to undecided parents helps propel transactions towards a decision, while it puts transactions at risk of suffering liveness failures when parents turn out to be rogue. We can resolve this tension and provide a liveness guarantee with the aid of two mechanisms.

First we adopt an adaptive parent selection strategy, where transactions are attached at the live edge of the DAG, and are retried with new parents closer to the genesis vertex. This procedure is guaranteed to terminate with uncontested, decided parents, ensuring that a transaction cannot suffer liveness failure due to contested, rogue transactions. A secondary mechanism ensures that virtuous transactions with decided ancestry will receive sufficient chits. Correct nodes examine the DAG for virtuous transactions that lack sufficient progeny and emit nop transactions to help increase their confidence. With these two mechanisms in place, it is easy to see that, at worst, Avalanche will degenerate into separate instances of Snowball, and thus provide the same liveness guarantee for virtuous transactions.

Unlike other cryptocurrencies [48] that use graph vertices directly as votes, Avalanche only uses DAG for the purpose of batching queries in the underlying Snowball instances. Because confidence is built by collected chits, and not by just the presence of a vertex, simply flooding the network with vertices attached to the rejected side of a subgraph will not subvert the protocol.

### *D. Communication Complexity*

Let the DAG induced by Avalanche have an expected branching factor of $p$, corresponding to the width of the DAG, and determined by the parent selection algorithm. Given the $\beta_1$ and $\beta_2$ decision threshold, a transaction that has just reached the point of decision will have an associated progeny $\mathcal{Y}$. Let $m$ be the expected depth of $\mathcal{Y}$. If we were to let the Avalanche network make progress and then freeze the DAG at a depth $y$, then it will have roughly $py$ vertices/transactions, of which $p(y-m)$ are decided in expectation. Only $pm$ recent transactions would lack the progeny required for a decision. For each node, each query requires $k$ samples, and therefore the total message cost per transaction is in expectation $(pky)/(p(y-m)) = ky/(y-m)$. Since $m$ is a constant determined by the undecided region of the DAG as the system constantly makes progress, message complexity per node is $O(k)$, while the total complexity is $O(kn)$.

## VI. Evaluation

### A. Setup

We conduct our experiments on Amazon EC2 by running from hundreds (125) to thousands (2000) of virtual machine instances. We use `c5.large` instances, each of which simulates an individual node. AWS provides bandwidth of up to 2 Gbps, though the Avalanche protocol utilizes at most around 100 Mbps.

Our implementation supports two versions of transactions: one is the customized UTXO format, while the other uses the code directly from Bitcoin 0.16. Both supported formats use secp256k1 crypto library from bitcoin and provide the same address format for wallets. All experiments use the customized format except for the geo-replication, where results for both are given.

We simulate a constant flow of new transactions from users by creating separate client processes, each of which maintains separated wallets, generates transactions with new recipient addresses and sends the requests to Avalanche nodes. We use several such client processes to max out the capacity of our system. The number of recipients for each transaction is tuned to achieve average transaction sizes of around 250 bytes (1–2 inputs/outputs per transaction on average and a stable UTXO size), the current average transaction size of Bitcoin. To utilize the network efficiently, we batch up to 40 transactions during a query, but maintain confidence values at individual transaction granularity.

All reported metrics reflect end-to-end measurements taken from the perspective of all clients. That is, clients examine the total number of confirmed transactions per second for throughput, and, for each transaction, subtract the initiation timestamp from the confirmation timestamp for latency. Each throughput experiment is repeated for 5 times and standard deviation is indicated in each figure. As for security parameters, we pick $k = 10$, $\alpha = 0.8$, $\beta_1 = 11$, $\beta_2 = 150$, which yields an MTTF of ˜$10^{24}$ years.

### B. Throughput

We first measure the throughput of the system by saturating it with transactions and examining the rate at which transactions are confirmed in the steady state. For this experiment, we first run Avalanche on 125 nodes with 10 client processes, each of which maintains 400 outstanding transactions at any given time.

As shown by the first group of bars in Figure 13, the system achieves 6851 transactions per second (tps) for a batch size of 20 and above 7002 tps for a batch size of 40. Our system is saturated by a small batch size comparing to other blockchains with known performance: Bitcoin batches several thousands of transactions per block, Algorand [27] uses 2–10 Mbyte blocks, i.e., 8.4–41.9K tx/batch and Conflux [38] uses 4 Mbyte blocks, i.e., 16.8K tx/batch. These systems are relatively slow in making a single decision, and thus require a very large batch (block) size for better performance. Achieving high throughput with small batch size implies low latency, as we will show later.

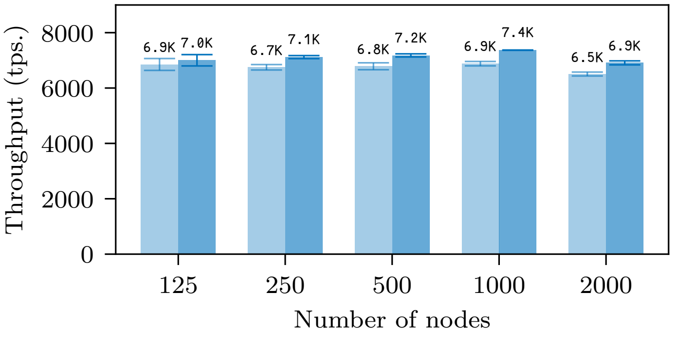


Fig. 13: Throughput vs. network size. Each pair of bars is produced with batch size of 20 and 40, from left to right.

### C. Scalability

To examine how the system scales in terms of the number of nodes participating in Avalanche consensus, we run experiments with identical settings and vary the number of nodes from 125 up to 2000.

Figure 13 shows that overall throughput degrades about 1.34% to 6909 tps when the network grows by a factor of 16 to $n = 2000$. This degradation is minor compared to the fluctuation in performance of repeated runs. Note that the x-axis is logarithmic.

Avalanche acquires its scalability from three sources: first, maintaining a partial order that captures only the spending relations allows for more concurrency than a classical BFT replicated log that linearizes all transactions; second, the lack of a leader naturally avoids bottlenecks; finally, the number of messages each node has to handle per decision is $O(k)$ and does not grow as the network scales up.

### D. Cryptography Bottleneck

We next examine where bottlenecks lie in our current implementation. The purple bar on the right of each group

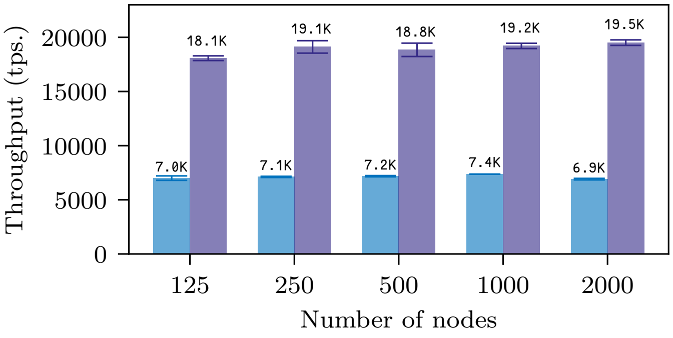


Fig. 14: Throughput for batch size of 40, with (left) and without (right) signature verification.

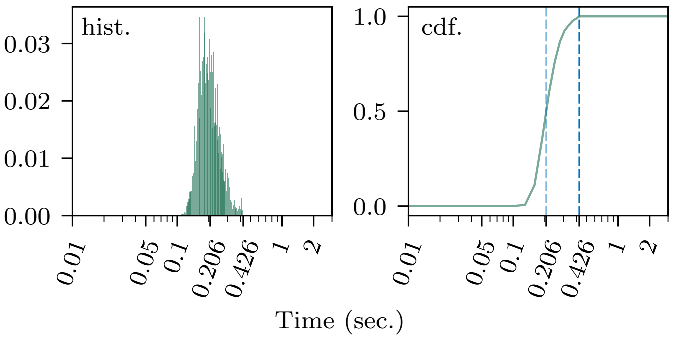


Fig. 15: Transaction latency distribution for $n = 2000$. The x-axis is the transaction latency in log-scaled seconds, while the y-axis is the portion of transactions that fall into the confirmation time (normalized to 1). Histogram of all transaction latencies for a client is shown on the left with 100 bins, while its CDF is on the right.

in Figure 14 shows the throughput of Avalanche with signature verification disabled. Throughputs get approximately 2.6x higher, compared to the blue bar on the left. This reveals that cryptographic verification overhead is the current bottleneck of our system implementation. This bottleneck can be addressed by offloading transaction verification to a GPU. Even without such optimization, 7K tps is far in excess of extant blockchains.

### *E. Latency*

The latency of a transaction is the time spent from the moment of its submission until it is confirmed as accepted. Figure 15 tallies the latency distribution histogram using the same setup as for the throughput measurements with 2000 nodes. The x-axis is the time in seconds while the y-axis is the portion of transactions that are finalized within the corresponding time period. This figure also outlines the Cumulative Distribution Function (CDF) by accumulating the number of finalized transaction over time.

This experiment shows that most transactions are confirmed within approximately 0.3 seconds. The most common latencies are around 206 ms and variance is low, indicating that nodes converge on the final value as a group around the same time. The second vertical line shows the maximum latency we observe, which is around 0.4 seconds.

Figure 16 shows transaction latencies for different numbers of nodes. The horizontal edges of boxes represent minimum, first quartile, median, third quartile and maximum latency respectively, from bottom to top. Crucially, the experimental data show that median latency is more-or-less independent of network size.

### *F. Misbehaving Clients*

We next examine how rogue transactions issued by misbehaving clients that double spend unspent outputs can affect latency for virtuous transactions created by honest clients. We adopt a strategy to simulate misbehaving clients where a fraction (from 0% to 25%) of the pending transactions conflict with some existing ones. The client processes achieve this by designating some double spending transaction flows among all simulated pending transactions and sending the conflicting transactions to different nodes. We use the same setup with $n = 1000$ as in the previous experiments, and only measure throughput and latency of confirmed transactions.

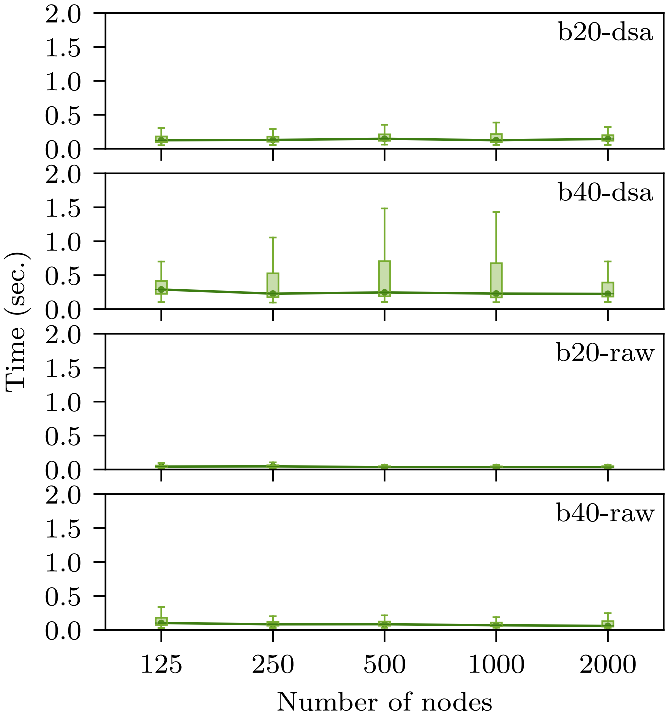


Fig. 16: Transaction latency vs. network size. “b” indicates batch size and “raw” is the run without signature verification.

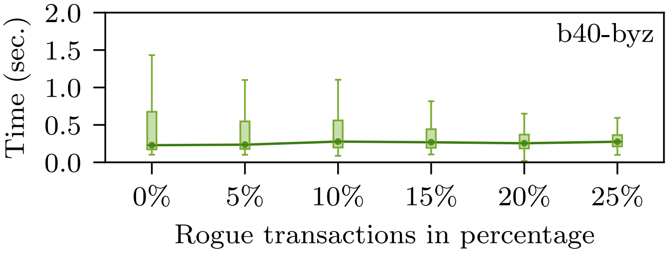


Fig. 17: Latency vs. ratio of rogue transactions.

Avalanche’s latency is only slightly affected by misbehaving clients, as shown in Figure 17. Surprisingly, maximum latencies drop slightly when the percentage of rogue transactions increases. This behavior occurs because, with the introduction of rogue transactions, the overall *effective* throughput is reduced and thus alleviates system load. This is confirmed by Figure 18, which shows that throughput (of virtuous transactions) decreases with the ratio of rogue transactions. Further, the reduction in throughput appears proportional to the number of misbehaving clients, that is, there is no leverage provided to the attackers.

### *G. Geo-replication*

Next experiment shows the system in an emulated geo-replicated scenario, patterned after the same scenario in prior work [27]. We selected 20 major cities that appear to be near substantial numbers of reachable Bitcoin nodes, according to [9]. The cities cover North America, Europe, West Asia, East Asia, Oceania, and also cover the top 10 countries with

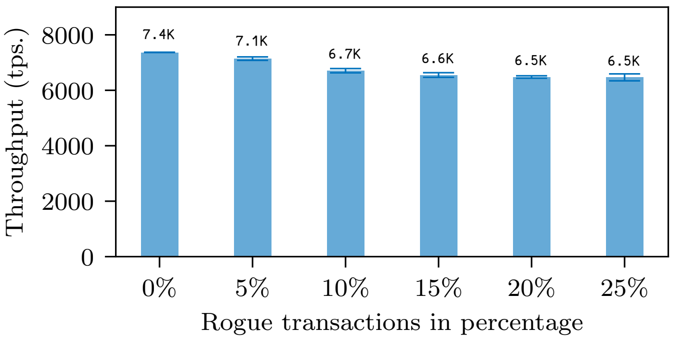


Fig. 18: Throughput vs. ratio of rogue transactions.

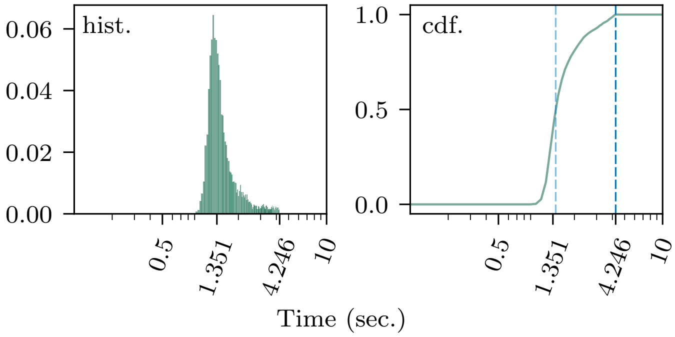


Fig. 19: Latency histogram/CDF for $n = 2000$ in 20 cities.

the highest number of reachable nodes. We use the latency and jittering matrix crawled from [58] and emulate network packet latency in the Linux kernel using `tc` and `netem`. 2000 nodes are distributed evenly to each city, with no additional network latency emulated between nodes within the same city. Like Algorand's evaluation, we also cap our bandwidth per process to 20 Mbps to simulate internet-scale settings where there are many commodity network links. We assign a client process to each city, maintaining 400 outstanding transactions per city at any moment.

In this scenario, Avalanche achieves an average throughput of 3401 tps, with a standard deviation of 39 tps. As shown in Figure 19, the median transaction latency is 1.35 seconds, with a maximum latency of 4.25 seconds. We also support native Bitcoin code for transactions; in this case, the throughput is 3530 tps, with $\sigma = 92$ tps.

### *H. Comparison to Other Systems*

Though there are seemingly abundant blockchain or cryptocurrency protocols, most of them only present a sketch of their protocols and do not offer practical implementation or evaluation results. Moreover, among those who do provide results, most are not evaluated in realistic, large-scale (hundreds to thousands of full nodes participating in consensus) settings.

Therefore, we choose Algorand and Conflux for our comparison. Algorand, Conflux, and Avalanche are all fundamentally different in their design. Algorand's committee-scale consensus algorithm falls into the classical BFT consensus category, and Conflux extends Nakamoto consensus by a DAG structure to facilitate higher throughput, while Avalanche belongs to a new protocol family based on metastability. Additionally, we use Bitcoin [43] as a baseline.

Both Algorand and Avalanche evaluations use a decision network of size 2000 on EC2. Our evaluation picked shared `c5.large` instances, while Algorand used `m4.2xlarge`. These two platforms are very similar except for a slight CPU clock speed edge for `c5.large`, which goes largely unused because our process only consumes 30% in these experiments. The security parameters chosen in our experiments guarantee a safety violation probability below $10^{-9}$ in the presence of 20% Byzantine nodes, while Algorand's evaluation guarantees a violation probability below $5 \times 10^{-9}$ with 20% Byzantine nodes.

Neither Algorand nor Conflux evaluations take into account the overhead of cryptographic verification. Their evaluations use blocks that carry megabytes of dummy data and present the throughput in MB/hour or GB/hour unit. So we use the average size of a Bitcoin transaction, 250 bytes, to derive their throughputs. In contrast, our experiments carry real transactions and fully take all cryptographic overhead into account.

The throughput is 3-7 tps for Bitcoin, 874 tps for Algorand (with 10 Mbyte blocks), 3355 tps for Conflux (in the paper it claims 3.84x Algorand's throughput under the same settings).

In contrast, Avalanche achieves over 3400 tps consistently on up to 2000 nodes without committee or proof-of-work. As for latency, a transaction is confirmed after 10–60 minutes in Bitcoin, around 50 seconds in Algorand, 7.6–13.8 minutes in Conflux, and 1.35 seconds in Avalanche.

Avalanche performs much better than Algorand in both throughput and latency because Algorand uses a verifiable random function to elect committees, and maintains a totally-ordered log while Avalanche establishes only a partial order. Algorand is leader-based and performs consensus by committee, while Avalanche is leader-less.

Avalanche has similar throughput to Conflux, but its latency is 337–613x better. Conflux also uses a DAG structure to amortize the cost for consensus and increase the throughput, however, it is still rooted in Nakamoto consensus (PoW), making it unable to have instant confirmation compared to Avalanche.

In a blockchain system, one can usually improve throughput at the cost of latency through batching. The real bottleneck of the performance is the number of decisions the system can make per second, and this is fundamentally limited by either Byzantine Agreement ($\text{BA}^*$) in Algorand and Nakamoto consensus in Conflux.

## VII. Related Work

Bitcoin [43] is a cryptocurrency that uses a blockchain based on proof-of-work (PoW) to maintain a ledger of UTXO transactions. While techniques based on proof-of-work [4], [23], and even cryptocurrencies with minting based on proof-of-work [49], [57], have been explored before, Bitcoin was the first to incorporate PoW into its consensus process. Unlike more traditional BFT protocols, Bitcoin has a probabilistic safety guarantee and assumes honest majority computational power rather than a known membership, which in turn has enabled an internet-scale permissionless protocol. While permissionless and resilient to adversaries, Bitcoin suffers from low throughput (˜3 tps) and high latency (˜5.6 hours for a network with 20% Byzantine presence and $2^{-32}$ security guarantee). Furthermore, PoW requires a substantial amount of computational power that is consumed only for the purpose of maintaining safety.

Countless cryptocurrencies use PoW [4], [23] to maintain a distributed ledger. Like Bitcoin, they suffer from inherent scalability bottlenecks. Several proposals for protocols exist that try to better utilize the effort made by PoW. Bitcoin-NG [24] and the permissionless version of Thunderella [46] use Nakamoto-like consensus to elect a leader that dictates writing of the replicated log for a relatively long time so as to provide higher throughput. Moreover, Thunderella provides an optimistic bound that, with 3/4 honest computational power and an honest elected leader, allows transactions to be confirmed rapidly. ByzCoin [35] periodically selects a small set of participants and then runs a PBFT-like protocol within the selected nodes.

Protocols based on Byzantine agreement [37], [47] typically make use of quorums and require precise knowledge of membership. PBFT [13], a well-known representative, requires a quadratic number of message exchanges in order to reach agreement. The Q/U protocol [2] and HQ replication [16] use a quorum-based approach to optimize for contention-free cases of operation to achieve consensus in only a single round of communication. However, although these protocols improve on performance, they degrade very poorly under contention. Zyzzyva [36] couples BFT with speculative execution to improve the failure-free operation case. Past work in permissioned BFT systems typically requires at least $3f+1$ replicas. CheapBFT [32] leverages trusted hardware components to construct a protocol that uses $f+1$ replicas.

Other work attempts to introduce new protocols under redefinitions and relaxations of the BFT model. Large-scale BFT [50] modifies PBFT to allow for arbitrary choice of number of replicas and failure threshold, providing a probabilistic guarantee of liveness for some failure ratio but protecting safety with high probability. In another form of relaxation. Zeno [52] introduces a BFT state machine replication protocol that trades consistency for high availability. More specifically, Zeno guarantees eventual consistency rather than linearizability, meaning that participants can be inconsistent but eventually agree once the network stabilizes. By providing an even weaker consistency guarantee, namely fork-join-causal consistency, Depot [40] describes a protocol that guarantees safety under $2f+1$ replicas.

NOW [28] uses sub-quorums to drive smaller instances of consensus. The insight of this paper is that small, logarithmic-sized quorums can be extracted from a potentially large set of nodes in the network, allowing smaller instances of consensus protocols to be run in parallel.

Snow White [18] and Ouroboros [34] are some of the earliest provably secure PoS protocols. Ouroboros uses a secure multiparty coin-flipping protocol to produce randomness for leader election. The follow-up protocol, Ouroboros Praos [19] provides safety in the presence of fully adaptive adversaries.

HoneyBadger [42] provides good liveness in a network with heterogeneous latencies. Tendermint [10], [11] rotates the leader for each block and has been demonstrated with as many as 64 nodes. Ripple [51] has low latency by utilizing collectively-trusted sub-networks in a large network. The Ripple company provides a slow-changing default list of trusted nodes, which renders the system essentially centralized. In the synchronous and authenticated setting, the protocol in [3] achieves constant-3-round commit in expectation, at the cost of quadratic message complexity. Stellar [41] uses Federated Byzantine Agreement in which *quorum slices* enable heterogeneous trust for different nodes. Safety is guaranteed when transactions can be transitively connected by trusted quorum slices. Algorand [27] uses a verifiable random function to select a committee of nodes that participate in a novel Byzantine consensus protocol.

Some protocols use a Directed Acyclic Graph (DAG) structure instead of a linear chain to achieve consensus [5], [8], [53]–[55]. Instead of choosing the longest chain as in Bitcoin, GHOST [54] uses a more efficient chain selection rule that allows transactions not on the main chain to be taken into consideration, increasing efficiency. SPECTRE [53] uses transactions on the DAG to vote recursively with PoW to achieve consensus, followed up by PHANTOM [55] that achieves a linear order among all blocks. Like PHANTOM, Conflux also finalizes a linear order of transactions by PoW in a DAG structure, with better resistance to liveness attack [38]. Similar to Thunderella, Meshcash [8] combines a slow PoW-based protocol with a fast consensus protocol that allows a high block rate regardless of network latency, offering fast confirmation time. Hashgraph [5] is a leader-less protocol that builds a DAG via randomized gossip. It requires full membership knowledge at all times, and it is a PBFT-variant that requires quadratic messages in expectation.

## VIII. Conclusion

This paper introduced a novel family of consensus protocols, coupled with the appropriate mathematical tools for analyzing them. These protocols are highly efficient and robust, combining the best features of classical and Nakamoto consensus. They scale well, achieve high throughput and quick finality, work without precise membership knowledge, and degrade gracefully under catastrophic adversarial attacks.

There is much work to do to improve this line of research. One such improvement could be the introduction of an adversarial network scheduler. Another improvement would be to characterize the system's guarantees under an adversary whose powers are realistically limited, whereupon performance would improve even further. Finally, more sophisticated initialization mechanisms would bear fruitful in improving liveness of multi-value consensus. Overall, we hope that the protocols and analysis techniques presented here add to the arsenal of the distributed system developers and provide a foundation for new lightweight and scalable mechanisms.

## References

[1] Crypto-currency market capitalizations. https://coinmarketcap.com. Accessed: 2017-02.

[2] Abd-El-Malek, M., Ganger, G. R., Goodson, G. R., Reiter, M. K., and Wylie, J. J. Fault-scalable byzantine fault-tolerant services. In *ACM SIGOPS Operating Systems Review* (2005), vol. 39, ACM, pp. 59–74.

[3] Abraham, I., Devadas, S., Dolev, D., Nayak, K., and Ren, L. Efficient synchronous byzantine consensus. *arXiv preprint arXiv:1704.02397* (2017).

[4] Aspnes, J., Jackson, C., and Krishnamurthy, A. Exposing computationally-challenged byzantine impostors. Tech. rep., Technical Report YALEU/DCS/TR-1332, Yale University Department of Computer Science, 2005.

[5] Baird, L. Hashgraph consensus: fair, fast, byzantine fault tolerance. Tech. rep., Swirlds Tech Report, 2016.

[6] Banerjee, S., Chatterjee, A., and Shakkottai, S. Epidemic thresholds with external agents. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications* (2014), IEEE, pp. 2202–2210.

[7] Ben-Or, M. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing* (1983), ACM, pp. 27–30.

[8] Bentov, I., Hubácek, P., Moran, T., and Nadler, A. Tortoise and Hares Consensus: the Meshcash framework for incentive-compatible, scalable cryptocurrencies. *IACR Cryptology ePrint Archive 2017* (2017), 300.

[9] Bitnodes. Global Bitcoin nodes distribution. https://bitnodes.earn.com/. Accessed: 2018-04.

[10] Buchman, E. *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, 2016.

[11] Buchman, E., Kwon, J., and Milosevic, Z. The latest gossip on bft consensus, 2018.

[12] Burrows, M. The chubby lock service for loosely-coupled distributed systems. In *7th Symposium on Operating Systems Design and Implementation (OSDI'06), November 6-8, Seattle, WA, USA* (2006), pp. 335–350.

[13] Castro, M., and Liskov, B. Practical byzantine fault tolerance. In *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999* (1999), pp. 173–186.

[14] Central Intelligence Agency. The world factbook. https://www.cia.gov/library/publications/the-world-factbook/geos/da.html. Accessed: 2018-04.

[15] Chvátal, V. The tail of the hypergeometric distribution. *Discrete Mathematics 25*, 3 (1979), 285–287.

[16] Cowling, J., Myers, D., Liskov, B., Rodrigues, R., and Shrira, L. Hq replication: A hybrid quorum protocol for byzantine fault tolerance. In *Proceedings of the 7th symposium on Operating systems design and implementation* (2006), USENIX Association, pp. 177–190.

[17] Croman, K., Decker, C., Eyal, I., Gencer, A. E., Juels, A., Kosba, A. E., Miller, A., Saxena, P., Shi, E., Sirer, E. G., Song, D., and Wattenhofer, R. On scaling decentralized blockchains - (a position paper). In *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers* (2016), pp. 106–125.

[18] Daian, P., Pass, R., and Shi, E. Snow white: Provably secure proofs of stake. Cryptology ePrint Archive, Report 2016/919, 2016. https://eprint.iacr.org/2016/919.

[19] David, B., Gazi, P., Kiayias, A., and Russell, A. Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II* (2018), pp. 66–98.

[20] Digiconomist. Bitcoin energy consumption index. https://digiconomist.net/bitcoin-energy-consumption. Accessed: 2018-04.

[21] Douceur, J. R. The sybil attack. In *International Workshop on Peer-to-Peer Systems* (2002), Springer, pp. 251–260.

[22] Draief, M., Ganesh, A., and Massoulié, L. Thresholds for virus spread on networks. In *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools* (2006), ACM, p. 51.

[23] Dwork, C., and Naor, M. Pricing via processing or combatting junk mail. In *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings* (1992), pp. 139–147.

[24] Eyal, I., Gencer, A. E., Sirer, E. G., and van Renesse, R. Bitcoin-NG: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2016, Santa Clara, CA, USA, March 16-18, 2016* (2016), pp. 45–59.

[25] Ganesh, A., Massoulié, L., and Towsley, D. The effect of network topology on the spread of epidemics. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.* (2005), vol. 2, IEEE, pp. 1455–1466.

[26] Garay, J. A., Kiayias, A., and Leonardos, N. The Bitcoin Backbone Protocol: Analysis and applications. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II* (2015), pp. 281–310.

[27] Gilad, Y., Hemo, R., Micali, S., Vlachos, G., and Zeldovich, N. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017* (2017), pp. 51–68.

[28] Guerraoui, R., Huc, F., and Kermarrec, A.-M. Highly dynamic distributed computing with byzantine failures. In *Proceedings of the 2013 ACM symposium on Principles of distributed computing* (2013), ACM, pp. 176–183.

[29] Hoeffding, W. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association 58*, 301 (1963), 13–30.

[30] Hunt, P., Konar, M., Junqueira, F. P., and Reed, B. Zookeeper: Wait-free coordination for internet-scale systems. In *2010 USENIX Annual Technical Conference, Boston, MA, USA, June 23-25, 2010* (2010).

[31] Johansen, H. D., van Renesse, R., Vigfusson, Y., and Johansen, D. Fireflies: A secure and scalable membership and gossip service. *ACM Trans. Comput. Syst. 33*, 2 (2015), 5:1–5:32.

[32] Kapitza, R., Behl, J., Cachin, C., Distler, T., Kuhnle, S., Mohammadi, S. V., Schröder-Preikschat, W., and Stengel, K. Cheapbft: resource-efficient byzantine fault tolerance. In *Proceedings of the 7th ACM european conference on Computer Systems* (2012), ACM, pp. 295–308.

[33] Keeling, M. J., and Rohani, P. *Modeling infectious diseases in humans and animals*. Princeton University Press, 2011.

[34] Kiayias, A., Russell, A., David, B., and Oliynykov, R. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I* (2017), pp. 357–388.

[35] Kokoris-Kogias, E., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., and Ford, B. Enhancing Bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.* (2016), pp. 279–296.

[36] Kotla, R., Alvisi, L., Dahlin, M., Clement, A., and Wong, E. L. Zyzzyva: Speculative byzantine fault tolerance. *ACM Trans. Comput. Syst. 27*, 4 (2009), 7:1–7:39.

[37] Lamport, L., Shostak, R. E., and Pease, M. C. The byzantine generals problem. *ACM Trans. Program. Lang. Syst. 4*, 3 (1982), 382–401.

[38] Li, C., Li, P., Xu, W., Long, F., and Yao, A. C. Scaling nakamoto consensus to thousands of transactions per second. *CoRR abs/1805.03870* (2018).

[39] Liggett, T. M., et al. Stochastic models of interacting systems. *The Annals of Probability 25*, 1 (1997), 1–29.

[40] Mahajan, P., Setty, S., Lee, S., Clement, A., Alvisi, L., Dahlin, M., and Walfish, M. Depot: Cloud storage with minimal trust. *ACM Transactions on Computer Systems (TOCS) 29*, 4 (2011), 12.

[41] Mazieres, D. The Stellar consensus protocol: A federated model for internet-level consensus. *Stellar Development Foundation* (2015).

[42] Miller, A., Xia, Y., Croman, K., Shi, E., and Song, D. The Honey Badger of BFT protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016* (2016), pp. 31–42.

[43] Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system, 2008.

[44] Pass, R., Seeman, L., and Shelat, A. Analysis of the blockchain protocol in asynchronous networks. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the*

*Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II* (2017), pp. 643–673.

[45] PASS, R., AND SHI, E. Fruitchains: A fair blockchain. *IACR Cryptology ePrint Archive 2016* (2016), 916.

[46] PASS, R., AND SHI, E. Thunderella: Blockchains with optimistic instant confirmation. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II* (2018), pp. 3–33.

[47] PEASE, M. C., SHOSTAK, R. E., AND LAMPORT, L. Reaching agreement in the presence of faults. *J. ACM 27*, 2 (1980), 228–234.

[48] POPOV, S. The tangle. https://www.iota.org/research/academic-papers. Accessed: 2018-04.

[49] RIVEST, R., AND SHAMIR, A. Payword and micromint: Two simple micropayment schemes. In *Security protocols* (1997), Springer, pp. 69–87.

[50] RODRIGUES, R., KOUZNETSOV, P., AND BHATTACHARJEE, B. Large-scale byzantine fault tolerance: Safe but not always live. In *Proceedings of the 3rd Workshop on Hot Topics in System Dependability* (2007).

[51] SCHWARTZ, D., YOUNGS, N., BRITTO, A., ET AL. The Ripple protocol consensus algorithm. *Ripple Labs Inc White Paper 5* (2014).

[52] SINGH, A., FONSECA, P., KUZNETSOV, P., RODRIGUES, R., AND MANIATIS, P. Zeno: Eventually consistent byzantine-fault tolerance. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2009, April 22-24, 2009, Boston, MA, USA* (2009), pp. 169–184.

[53] SOMPOLINSKY, Y., LEWENBERG, Y., AND ZOHAR, A. SPECTRE: A fast and scalable cryptocurrency protocol. *IACR Cryptology ePrint Archive 2016* (2016), 1159.

[54] SOMPOLINSKY, Y., AND ZOHAR, A. Secure high-rate transaction processing in Bitcoin. In *Financial Cryptography and Data Security, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers* (2015), pp. 507–527.

[55] SOMPOLINSKY, Y., AND ZOHAR, A. PHANTOM: A scalable blockdag protocol. *IACR Cryptology ePrint Archive 2018* (2018), 104.

[56] TAN, W. On the absorption probabilities and absorption times of finite homogeneous birth-death processes. *Biometrics* (1976), 745–752.

[57] VISHNUMURTHY, V., CHANDRAKUMAR, S., AND SIRER, E. G. Karma: A secure economic framework for peer-to-peer resource sharing. In *Workshop on Economics of Peer-to-peer Systems* (2003), vol. 35.

[58] WONDERNETWORK. Global ping statistics: Ping times between wondernetwork servers. https://wondernetwork.com/pings. Accessed: 2018-04.

# APPENDIX A
# ANALYSIS

In this appendix, we provide an analysis of Slush, Snowflake and Snowball.

## A. Preliminaries

We assume the network model as discussed in Section II. We let R ("red") and B ("blue") represent two generic conflicting choices. Without loss of generality, we focus our attention on counts of B, i.e. the total number of nodes that prefer blue.

### *a) Hypergeometric Distribution*

Each network query of $k$ peers corresponds to a sample without replacement out of a network of $n$ nodes, also referred to as a hypergeometric sample. We let the random variable $\mathcal{H}(\mathcal{N}, x, k) \to \{0, \ldots, k\}$ denote the resulting counts of B in the sample (unless otherwise stated), where $x$ is the total count of B in the population. The probability that the query achieves the required threshold of $\alpha$ or more votes is given by:

$$P(\mathcal{H}(\mathcal{N}, x, k) \geq \alpha) = \sum_{j=\alpha}^{k} \binom{x}{j} \binom{n-x}{k-j} / \binom{n}{k} \tag{2}$$

For ease of notation, we overload $\mathcal{H}(*)$ by implicitly referring to $P(\mathcal{H}(\mathcal{N}, x, k) \geq \alpha)$ as $\mathcal{H}(\mathcal{N}, x, k, \alpha)$.

### *b) Tail Bounds On Hypergeometric Distribution*

We can reduce some of the complexity in Equation 2 by introducing a bound on the hypergeometric distribution induced by $\mathcal{H}^k_{\mathcal{N},x}$. Let $p = x/n$ be the ratio of support for B in the population. The expectation of $\mathcal{H}(\mathcal{N}, x, k)$ is exactly $kp$. Then, the probability that $\mathcal{H}(\mathcal{N}, x, k)$ will deviate from the mean by more than some small constant $\psi$ is given by the Hoeffding tail bound [29], as follows,

$$\begin{aligned} P(\mathcal{H}(\mathcal{C}, x, k) \leq (p - \psi)k) &\leq e^{-k\mathcal{D}(p-\psi,p)} \\ &\leq e^{-2(p-\psi)^2 k} \end{aligned} \tag{3}$$

where $\mathcal{D}(p - \psi, p)$ is the Kullback-Leibler divergence, measured as

$$\mathcal{D}(a, b) = a \log \frac{a}{b} + (1-a) \log \frac{1-a}{1-b} \tag{4}$$

### *c) Concentration of Sub-Martingales*

Let $\{X_{\{t \geq 0\}}\}$ be a sub-martingale and $|X_t - X_{t-1}| < c_t$ almost surely. Then, for all positive reals $\psi$ and all positive integers $t$,

$$P(X_t \geq X_0 + \psi) \leq e^{-\psi^2/2\sum_{i=1}^{t} c_t^2} \tag{5}$$

## B. Slush

Slush operates in a non-Byzantine setting; that is, $f = 0, c = n$. In this section, we will characterize the irreversibility properties of Slush (which appear in Snowflake and Snowball), as well as the precise converge rate distribution. The distribution of of both safety and liveness of Slush translate well to the Byzantine setting.

The procedural version of Slush in Figure 4 made use of a parameter $m$, the number of rounds that a node executes Slush queries. What we ultimately want to extract is the total number of rounds $\phi$ that the scheduler will need to execute in order to guarantee that the entire network is the same color, whp.

We analyze the system mainly using a continuous time process. Let $\{X_{\{t \geq 0\}}\}$ be a CTMC. The state space $\mathcal{S}$ of the stochastic process is a condensed version of the full configuration space, where each state $\{0, \ldots, n\}$ represents the total number of blue nodes in the system.

Let $\mathcal{F}_{X_s}$ be the filtration, or the history pertaining to the process, up to time $s$. This process is Markovian and time-homogeneous, conforming to

$$P\{X_t = j | \mathcal{F}_{X_s}\} = P\{X_t = j | X_s\} = P\{X_t = j | X_0\}$$

Throughout the paper, we use $Q \equiv (q_{ij}, i, j \in \mathcal{S})$ notation to refer to the infinitesimal generator of the process, where death $(i \to i-1)$ and birth $(i \to i+1)$ rates of configuration transitions are denoted via $\mu_i$ and $\lambda_i$ ($\lambda_i$ is distinct from the clock parameter $\lambda$, and will be clear from context). These rates are

$$\begin{cases} \mu_i = i \ \mathcal{H}(\mathcal{N}, c-i, k, \alpha), & \text{for } i \to i-1 \\ \lambda_i = (c-i) \ \mathcal{H}(\mathcal{N}, i, k, \alpha), & \text{for } j \to i+1 \end{cases}$$

for $1 \leq i \leq c-1$, and where $i = 0$ and $i = c$ are absorbing. Let $p_{ij}(t)$ refer to the probability of transitioning from state $i$

to $j$ at time $t$. We always assume that

$$p_{ij}(t) = \begin{cases} \lambda_i t + o(t), & \text{for } j = i+1 \\ \mu_i t + o(t), & \text{for } j = i-1 \\ 1 - (\lambda_i + \mu_i)t + o(t), & \text{for } j = i \\ o(t), & \text{otherwise} \end{cases}$$

where all $o(t)$ are uniform in $i$.

*a) Irreversibility*

In Section IV, we discussed the loose Chvatal bound which provided intuitive understanding into the strong irreversibility dynamics of our core subsampling mechanism. In particular, once the network drifts to some majority value, it tends to revert back with only an exponentially small probability. We compute the closed-form expression for reversibility, and show that it is exponentially small.

**Theorem 2.** *Let $\xi_\delta$ be the probability of absorption into the all-red state ($s_0$), starting from a drift of $\delta$ (i.e. $\delta$ drift away from $n/2$). Then, assuming $\delta > 1$,*

$$\xi_\delta = 1 - \frac{\sum_{l=1}^{\delta} \prod_{i=1}^{l-1} \mu_i^2 \prod_{j=l}^{n-l} \lambda_j}{2 \sum_{l=1}^{n/2} \prod_{i=1}^{l-1} \mu_i^2 \prod_{j=l}^{n-l} \mu_j} \tag{6}$$

*and*

$$\begin{aligned} \frac{\xi_\delta - \xi_{\delta+1}}{\xi_{\delta+1} - \xi_{\delta+2}} = \sqcap_{\delta+1} = \frac{\lambda_{\delta+1}}{\mu_{\delta+1}} \\ \approx \frac{n - \delta - 1 \sum_{j=\alpha}^{k} \frac{(n-\delta-1)^k (\delta+1)^{k-j}}{n^{2k-j}}}{\delta + 1 \sum_{j=\alpha}^{k} \frac{(\delta+1)^k (n-\delta-1)^{k-j}}{n^{2k-j}}} \end{aligned} \tag{7}$$

*where from now on we refer to $\sqcap_{\delta+1}$ as the drift of the process.*

*Proof.* Our results are derived based on constructions from Tan [56]. We construct a sub-matrix of $Q$, denoted $B$, as shown in Figure 20. Let $W_1' = (\mu_1, 0, \ldots, 0)$, $W_{n-1}' = (0, \ldots, 0, \lambda_{n-1})$. Then, we can express $Q$ as

$$Q = \begin{bmatrix} 0 & \ldots & 0 \\ W_1 & B & W_{n-1} \\ 0 & \ldots & 0 \end{bmatrix}$$

As a reminder, the stationary distribution can be found via $\lim_{t\to\infty} P(t) = e^{Qt}$, where we have

$$e^{Qt} = \sum_{i=0}^{\infty} \frac{t^i}{i!} Q^i = \sum_{i=0}^{\infty} \frac{t^i}{i!} \begin{bmatrix} 0 & \ldots & 0 \\ B^{i-1} W_1 & B^i & B^{i-1} W_{n-1} \\ 0 & \ldots & 0 \end{bmatrix}$$

As Tan (eq. 2.3) shows, we have

$$\xi(t) = B^{-1} \left[ \sum_{i=0}^{\infty} B^i - \mathbb{I}_{n-1} \right] W_1$$

Since we want the ultimate probabilities, we have that

$$\xi = \lim_{t\to\infty} \xi(t) = -B^{-1} W_1$$

We can explicitly compute $\xi_\delta$ in terms of our rates $\mu_i$ and $\lambda_i$, getting

$$\xi_\delta = \frac{\sum_{l=1}^{n-\delta} \prod_{i=1}^{n-l} \mu_i \prod_{j=n-l+1}^{n-1} \lambda_j}{\sum_{l=1}^{n} \prod_{i=1}^{n-l} \mu_i \prod_{j=n-l+1}^{n-1} \lambda_j}$$

However, we note that $u_i = \lambda_{n-i}$. Algebraic manipulation from this observation leads to the two equations in the theorem. This expression is strictly lower than the Chvatal bounds used in Section IV. □

Using the construction for the absorption (and (ir)reversibility) probabilities as discussed previously, a natural follow up computation is in regards to *mean convergence time*. Let $T_z(t) = \inf\{t \geq 0 : X_t = \{0, n\} | X_0 = z\}$, and let $\tau_z = \mathbb{E}[T_z(t)]$. $\tau_z$ is the mean time to reach either absorbing state, starting from state $z$, which corresponds to the mean convergence time. The next theorem characterizes this distribution.

**Theorem 3.** *Let $\tau_z$ be the expected time to convergence, starting from state $z > n/2$, to any of the two converging states in the network (all-red or all-blue). Then,*

$$\tau_z = \frac{\sum_{d=1}^{n-1} x(d) y(d)}{2 \sum_{l=1}^{n/2} \prod_{i=1}^{l-1} \mu_i^2 \prod_{j=l}^{n-l} \mu_j} \tag{8}$$

*where $x(d)$ and $y(d)$ are*

$$\begin{aligned} x(d) &= \sum_{l=1}^{\min(z,d)} \prod_{i=1}^{l-1} \mu_i \prod_{j=l}^{d-1} \lambda_j \\ y(d) &= \sum_{l=1}^{n-d-\max(z-d,0)} \prod_{i=d+1}^{n-l} \mu_i \prod_{j=n-l+1}^{n-1} \lambda_j \end{aligned} \tag{9}$$

*Proof.* Following the calculations from before, $-B^{-1}$ at row $z$ provides the number of traversals to each other state starting from $z$. Calculating their sum, we have our result. The above equation is the full expression of the matrix row sum. □

Theorem 3 leads to the next lemma that captures property P2, under the assumption that at the beginning of the protocol, one proposal has at least $\alpha$ support in the network.

**Lemma 4.** *Slush reaches an absorbing state in finite time almost surely.*

*Proof.* Starting from any non-absorbing, transient state, there is a non-zero probability of being absorbed. Additionally, since termination is finite and everywhere differentiable, Theorem 3 also implies that the probability of termination of any network configuration where a proposal has $\geq \alpha$ support in bounded time $t_{max}$ is strictly positive. □

### C. Snowflake

In Snowflake, the sampled set of nodes includes Byzantine nodes. We introduce the decision function $\mathcal{D}(*)$, which is

$$B = \begin{bmatrix} -(\lambda_1+\mu_1) & \lambda_1 & 0 & \cdots & \cdots & 0 \\ \mu_2 & -(\lambda_2+\mu_2) & \lambda_2 & 0 & \cdots & 0 \\ 0 & \mu_3 & -(\lambda_3+\mu_3) & \lambda_3 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \mu_{n-3} & -(\lambda_{n-2}+\mu_{n-2}) & \lambda_{n-3} & 0 \\ \vdots & \dots & 0 & \mu_{n-1} & -(\lambda_{n-2}+\mu_{n-2}) & \lambda_{n-2} \\ 0 & \dots & 0 & 0 & \mu_{n-1} & -(\lambda_{n-1}+\mu_{n-1}) \end{bmatrix}$$

Fig. 20: Matrix $B$.

constructed by having each node also keep track of the total number of consecutive times it has sampled a majority of the same color ($\beta$). Finally, we introduce a function called $\mathcal{A}(\mathcal{S}_t)$, the adversarial strategy, that takes as parameters the entire configuration of the network at time $t$, as well as the next set of nodes chosen by the scheduler to execute, and as a side-effect, modifies the set of nodes $\mathcal{B}$ to some arbitrary configuration of colors.

In order for our prior framework to apply to Snowflake, we must deal with a key subtlety. Unlike in Slush, where it is clear that once the network has reached one of the converging states and therefore may not revert back, this no longer applies to Snowflake, since any adversary $f \geq \alpha$ has strictly positive probability of reverting the system, albeit this probability may be infinitesimally small. The CTMC is flexible enough to deal with a system where there is only one absorbing state, but the long-term behavior of the system is no longer meaningful since, after an infinite amount of time, the system is guaranteed to revert, violating safety. We could trivially bound the amount of time, and show safety using this bounded time assumption by simply characterizing the distribution of $e^{tQ}$, where $Q$ is the generator. However, we can make the following observation: if the probability of going from state $c$ (all-blue) to $c-1$ is exponentially small, then it will take the attacker exponential time (in expectation; note, this is a lower bound, and in reality it will take much longer) to succeed in reverting the system. Hence, we can assume that once all correct nodes are the same color, the attack from the adversary will terminate since it is impractical to continue an attack. In fact, under reasonably bounded timeframes, the variational distance between the exact approach and the approximation is very small. We leave details to the accompanying paper, but we briefly discuss how analysis proceeds for Snowflake.

As stated in Section IV, the way to analyze the adversary using the same construction as in Slush is to condition reversibility on the first node $u$ deciding on blue, which can happen at any state (as specified by $\mathcal{D}(*)$). At that point, the adversarial strategy collapses to a single function, which is to continually vote for red. The probabilities of reversibility, for all states $\{1, \dots, c-1\}$ must encode the probability that additional blue nodes commit, and the single function of the adversary. The birth and death rates are transformed as follows:

$$\begin{cases} \mu_i = & i(1-\mathbb{I}[\mathcal{D}(*, i, \mathbb{B})]) \ \mathcal{H}(\mathcal{N}, c-i+f, k, \alpha) \\ \lambda_i = & (c-i)(1-\mathbb{I}[\mathcal{D}(*, c-i, \mathbb{R})]) \ \mathcal{H}(\mathcal{N}, i, k, \alpha) \end{cases}$$

From here on, the analysis is the same as in Slush. Under various $k$ and $\beta$, we can find the minimal $\alpha$ that provides the system strong irreversibility properties.

The next lemma captures P3, and the proof follows from central limit theorem.

**Lemma 5.** *If $f < \mathcal{O}(\sqrt{n})$, and $\alpha = \lfloor k/2 \rfloor + 1$, then Snowflake terminates in $\mathcal{O}(\log n)$ rounds with high probability.*

*Proof.* The results follows from central limit theorem, wherein for $\alpha = \lfloor k/2 \rfloor + 1$, the expected bias in the network after sampling will be $\mathcal{O}(\sqrt{n})$. An adversary smaller than this bias will be unable to keep the network in a fully-bivalent state for more than a constant number of rounds. The logarithmic factor remains from the mixing time lower bound. □

### D. Snowball

We make the following observation: if the confidences between red and blue are equal, then the adversary has the same identical leverage in the irreversibility of the system as in Snowflake, regardless of network configuration. In fact, Snowflake can be viewed as Snowball but where drifts in confidences never exceed one. The same analysis applies to Snowball as in Snowflake, with the additional requirement of bounding the long-term behavior of the confidences in the network. To that end, analysis follows using martingale concentration inequalities, in particular the one introduced in Equation 5. Snowball can be viewed as a two-urn system, where each urn is a sub-martingale. The guarantees that can be extracted hereon are that the confidences of the majority committed value (in our frame of reference is always blue), grow always more than those of the minority value, with high probability, drifting away as $t \rightarrow t_{max}$.

### E. Safe Early Commitment

As we reasoned previously, each conflict set in Avalanche can be viewed as an instance of Snowball, where each progeny instance iteratively votes for the entire path of the ancestry. This feature provides various benefits; however, it also can lead to some virtuous transactions that depend on a rogue transaction to suffer the fate of the latter. In particular, rogue transactions can interject in-between virtuous transactions and reduce the ability of the virtuous transactions to ever reach the required ISACCEPTED predicate. As a thought experiment, suppose that a transaction $T_i$ names a set of parent transactions that are all decided, as per local view. If $T_i$ is sampled over a large enough set of successful queries without discovering any conflicts, then, since by assumption the entire ancestry of $T_i$ is decided, it must be the case (probabilistically) that we have achieved irreversibility.

To then statistically measure the assuredness that $T_i$ has been accepted by a large percentage of correct nodes without any conflicts, we make use of a one-way birth process, where a birth occurs when a new correct node discovers the conflict of $T_i$. Necessarily, deaths cannot exist in this model, because a conflicting transaction cannot be unseen once a correct node discovers it. Our births are as follows:

$$\lambda_i = \frac{c-i}{c}\left(1-\frac{\binom{n-i}{k}}{\binom{n}{k}}\right) \tag{10}$$

Solving for the expected time to reach the final birth state provides a lower bound to the $\beta_1$ parameter in the ISACCEPTED fast-decision branch. The table below shows an example of the analysis for $n = 2000$, $\alpha = 0.8$, and various $k$, where $\varepsilon \ll 10^{-9}$, and where $\beta$ is the minimum required value before deciding. Overall, a very small number of iterations are sufficient for the safe early commitment predicate. This supports the choice of $\beta$ in our evaluation.

| $k$ | 10 | 20 | 30 | 40 |
|---|---|---|---|---|
| $\beta$ | 10.87625 | 10.50125 | 10.37625 | 10.25125 |

### F. Churn and View Updates

Any realistic system needs to accommodate the departure and arrival of nodes. We now demonstrate that Avalanche nodes can admit a well-characterized amount of churn, by showing how to pick parameters such that Avalanche nodes can differ in their view of the network and still safely make decisions.

Consider a network whose operation is divided into epochs of length $\tau$, and a view update from epoch $t$ to $t+1$ during which $\gamma$ nodes join the network and $\bar{\gamma}$ nodes depart. Under our static construction, the state space $\mathcal{S}_t$ of the network had a key parameter $\Delta^t$ at time $t$, induced by $c^t, f^t, n^t$ and the chosen security parameters. This can, at worst, impact the network by adding $\gamma$ nodes of color B, and remove $\bar{\gamma}$ nodes of color R. At time $t+1$, $n^{t+1} = n^t + \gamma - \bar{\gamma}$, while $f^{t+1}$ and $c^{t+1}$ will be modified by an amount $\leq \gamma - \bar{\gamma}$, and thus induce a new $\Delta^{t+1}$ for the chosen security parameters. This new $\Delta^{t+1}$ has to be chosen such that the probability of reversibility from state $c^{t+1}/2 + \Delta^{t+1} - \gamma$ is $\leq \varepsilon$, which ensures that the system will converge under the previous pessimal assumptions. The system designer can easily do this by picking an upper bound on $\gamma, \bar{\gamma}$.

The final step in assuring the correctness of a view change is to account for a mix of nodes that straddle the $\tau$ boundary. We would like the network to avoid an unsafe state no matter which nodes are using the old and the new views. The easiest way to do this is to determine $\Delta^t$ and $\Delta^{t+1}$ for desired bounds on $\gamma, \bar{\gamma}$, and then to use the conservative value $\Delta^{t+1}$ during epoch $t$. In essence, this ensures that no commitments are made in configuration $\mathcal{S}_t$ unless they conservatively fulfill the safety criteria in state space $\mathcal{S}_{t+1}$. As a result, there is no possibility of a node deciding red at time $t$, the network going through an epoch change and finding itself to the left of the new irreversibility state $\Delta^{t+1}$.

This approach trades off some of the feasibility space, to add the ability to accommodate $\gamma, \bar{\gamma}$ node churn per epoch. Overall, if $\tau$ is in excess of the time required for a decision (on the order of minutes to hours), and nodes are loosely synchronized, they can add or drop up to $\gamma, \bar{\gamma}$ nodes in each epoch using the conservative process described above. We leave the precise method of entering and exiting the network by staking and unstaking to a subsequent paper, and instead rely on a membership oracle that acts as a sequencer and $\gamma$-rate-limiter, using technologies like Fireflies [31].