Swiper: a new paradigm for efficient weighted distributed protocols

ANDREI TONKIKH, LTCI, Télécom Paris, Institut Polytechnique de Paris, France
LUCIANO FREITAS, LTCI, Télécom Paris, Institut Polytechnique de Paris, France

The majority of fault-tolerant distributed algorithms are designed assuming a *nominal* corruption model, in which at most a fraction f_n of parties can be corrupted by the adversary. However, due to the infamous Sybil attack, nominal models are not sufficient to express the trust assumptions in open (i.e., permissionless) settings. Instead, permissionless systems typically operate in a *weighted* model, where each participant is associated with a *weight* and the adversary can corrupt a set of parties holding at most a fraction f_w of total weight.

In this paper, we suggest a simple way to transform a large class of protocols designed for the nominal model into the weighted model. To this end, we formalize and solve three novel optimization problems, which we collectively call *the weight reduction problems*, that allow us to map large real weights into small integer weights while preserving the properties necessary for the correctness of the protocols. In all cases, we manage to keep the sum of the integer weights to be at most linear in the number of parties, resulting in extremely efficient protocols for the weighted model. Moreover, we demonstrate that, on weight distributions that emerge in practice, the sum of the integer weights tends to be far from the theoretical worst-case and, sometimes, even smaller than the number of participants.

While, for some protocols, our transformation requires an arbitrarily small reduction in resilience (i.e., $f_w = f_n - \epsilon$), surprisingly, for many important problems we manage to obtain weighted solutions with the same resilience ($f_w = f_n$) as nominal ones. Notable examples include erasure-coded distributed storage and broadcast protocols, verifiable secret sharing, and asynchronous consensus. Although there are ad-hoc weighted solutions to some of these problems, the protocols yielded by our transformations enjoy all the benefits of nominal solutions, including simplicity, efficiency, and a wider range of possible cryptographic assumptions.

Since the release of the first version of this paper online, a version of the weight reduction approach has been integrated into a major layer-1 blockchain system for implementing a randomness beacon.[1]

1 INTRODUCTION

1.1 Weighted distributed problems

Traditionally, distributed problems are studied in the egalitarian setting where n parties communicate over a network and any t of them can be faulty or corrupted by a malicious adversary. Different combinations of n and t are possible depending on the problem at hand, the types of failures (crash, omission, semi-honest, or malicious, also known as Byzantine), and the network model (typically, asynchronous, semi-synchronous, or synchronous). However, for most distributed protocols, t has to be smaller than a certain fraction of n. For example, most practical Byzantine fault-tolerant consensus protocols [21, 22] can operate for any $t < \frac{n}{3}$. We call such models *nominal* and use f_n to denote their *resilience*, i.e., a nominal protocol with resilience f_n operates correctly as long as less than $f_n n$ parties are corrupt, where n is the total number of participants.

However, this simple corruption model is not always sufficient to express the actual fault structure or trust assumptions of real systems. As a result, we see many practical blockchain protocols adopt a more general, *weighted* model, where each party is associated with a real *weight* that, intuitively, represents the number of "votes" this party has in the system. The assumption on the *number* of corrupt parties in this setting is replaced by the assumption that the *total weight* of the corrupt parties is smaller than a fraction f_w of the total weight of all participants. For example,

[1]We omit the name of the blockchain system to respect the double-blind policy of the conference.

Authors' addresses: Andrei Tonkikh, LTCI, Télécom Paris, Institut Polytechnique de Paris, France, tonkikh@telecom-paris.fr; Luciano Freitas, LTCI, Télécom Paris, Institut Polytechnique de Paris, France, lfreitas@telecom-paris.fr.

in permissionless systems, the weight can correspond to the amount of "stake" or computational resources a participant has invested in the system and, in the context of managed systems, to a function of the estimated failure probability.

There are two main reasons to adopt the weighted model in the context of blockchain systems. First and foremost, it protects the system from the infamous *Sybil attacks*, i.e., malicious users registering themselves multiple times in order to obtain multiple identities, thereby surpassing the resilience threshold f_n. Secondly, it is speculated that users with a greater amount of resources (monetary, computational, or otherwise) invested in the system, and consequently a higher weight, will be more committed to the system's stability and less likely to engage in malicious behavior.

1.2 Weighted voting and where it needs help

Perhaps, the most prevalent tool used for the design of distributed protocols is *quorum systems* [35, 45, 48]. Intuitively, to achieve fault tolerance, each "action" is confirmed by a sufficiently large set of participants (called a *quorum*). Then, if two actions are conflicting or somehow interdependent (e.g., writing and reading a file in a distributed storage system), then the parties in the intersection of the quorums are supposed to ensure consistency. Thus, many distributed protocols can be converted from the nominal to the weighted setting simply by changing the quorum system, i.e., instead of waiting for confirmations from a certain number of parties, waiting for a set of parties with the corresponding fraction of the total weight. We call this strategy *weighted voting* and it often allows translating protocols from the nominal to the weighted model while maintaining the same resilience (i.e., $f_w = f_n$) and, in some cases, with virtually no overhead.

However, weighted voting has two major downsides. First and foremost, many protocols rely on primitives beyond simple quorum systems and weighted voting is often not sufficient to translate these protocols to the weighted model. Notable examples include threshold cryptography [11, 31], secret sharing [17, 55], erasure and error-correcting codes [44], and numerous protocols that rely on these primitives.

Another example relevant to blockchain systems is Single Secret Leader Election protocols [12, 23, 24]. It illustrates that not all protocols that cannot be easily converted to the weighted model by applying weighted voting belong to the categories above and motivates the general approach taken in this paper.

The second drawback of weighted voting is that it requires a careful examination of the protocol in order to determine whether weighted voting is sufficient to convert it to the weighted model, as well as non-trivial modifications to the protocol implementation. It would be much nicer to have a "black-box" transformation that would take a protocol designed and implemented for the nominal model and output a protocol for the weighted model. In this paper, we offer both "open-box" and "black-box" transformations.

1.3 Our contribution

Our contribution to the fields of distributed computing and applied cryptography is twofold:

(1) We present a simple and efficient black-box transformation that can be applied to convert a wide range of protocols designed for the nominal model into the weighted model. Crucially, one can determine the applicability of our transformation simply by examining the *problem* that is being solved (e.g., Byzantine consensus) instead of the *protocol* itself (e.g., PBFT [22]) and it does not require modifications to the source code, only a slim wrapper around it. The price to pay for this transformation is an arbitrarily small decrease in resilience ($f_w = f_n - \epsilon$, where $\epsilon > 0$) and an increase in the communication and computation complexities proportional to $\frac{f_w}{\epsilon}$.

(2) Furthermore, by opening the black box and examining the internal structure of distributed protocols, we discover that by combining our transformation with weighted voting, in many cases, we can obtain weighted algorithms *without* the reduction in resilience ($f_w = f_n$) and with a minor or non-existent performance penalty.

distributed problem	**nominal solutions**	**weight-reduction problem**	f_w	f_n	**worst-case average comm. overhead**	**worst-case average comp. overhead**
		Derived Protocols				
Efficient Asynchronous State-Machine Replication	[27, 32, 41, 47, 57]	WR for RNG WQ for Broadcast	1/3	1/3	×1.33 for Broadcast ×1.33 for RNG	×3.56 for Broadcast ×1.33 for RNG
Structured Mempool	[27]	WQ for Broadcast	1/3	1/3	×1.33 for Broadcast	×3.56 for Broadcast
Validated Asynchronous Byzantine Agreement	[6, 18]	WR for RNG	1/3	1/3	×1.33 for RNG	×1.33 for RNG
Consensus with Checkpoints	[8]	WR for signing	1/3	1/3	×1.33 for signing	×1.33 for signing
		Useful Building Blocks				
Erasure-Coded Storage and Broadcast	[20, 37, 49, 50, 53, 59]	WQ	1/3	1/3	×1.33	×3.56
		WR (BB)	1/4	1/3	–	×3
Error-Corrected Broadcast	[29]	WQ	1/3	1/3	×1.33	×7.11
		WR (BB)	1/4	1/3	–	×3
Verifiable Secret Sharing	[55]	WR	1/3	1/3	×1.33	×1.33
Common Coin Blunt Threshold Signatures Blunt Threshold Encryption Blunt Threshold FHE	[19, 52] [11, 56, 58] [31] [14, 39]	WR	1/3	1/2	×1.33	×1.33
Tight Secret Sharing Tight Threshold Signatures Tight Threshold Encryption Tight Threshold FHE	Sec. A.2 (this paper)	WR	1/3	1/3	×1.33	×1.33
Linear BFT Consensus Chain-Quality SSLE	[60] [12]	WR (BB)	1/4	1/3	×2.67	×2.67

Table 1. Examples of suggested weighted distributed protocols with the upper bounds on communication and computation overhead compared to the nominal solutions with the same number of participants. See Appendices A and B and section 6 for details on how these numbers were obtained. In Section 7, we study real-world weight distributions and conclude that, in practice, the overhead should be much smaller. "WR" and "WQ" refer to weight-reduction problems defined in Section 2. "WR (BB)" refers to the black-box transformation described in Appendix A.3.

We summarize some examples of our techniques applied to a range of different protocols in Table 1. The last two columns of the table give the upper bound on the overhead of the obtained weighted protocols compared to their nominal counterparts executed with the same number of parties. Note, however, that, in many cases, the overhead applies only to specific parts of the protocol, which may not be the bottlenecks. Thus, further experimental studies may reveal that the real overhead is even lower or non-existent, even with worst-case weight distribution. Columns "f_w" and "f_n" specify the resilience of the obtained weighted protocols and the original nominal protocols, respectively. As was discussed before, in most cases, we manage to avoid sacrificing resilience (i.e., $f_w = f_n$).

Furthermore, the main building block of our constructions, the *weight reduction problems*, may be of separate interest and may have important applications beyond distributed protocols. It is, indeed, an interesting and somewhat counter-intuitive observation that large real weights can be efficiently reduced to small integer weights while preserving the key structural properties.

1.4 Empirical study

The performance of the weighted protocols constructed as suggested in this paper is sensitive to the distribution of weights of the participants. While we provide upper bounds and thus analyze our protocols for the worst weight distribution possible, it is interesting whether such weight distributions emerge in practice.

To study real-world weight distributions, we tested our weight reduction algorithms on the distribution of funds from multiple existing blockchain systems [7, 36, 43, 46] ranging in size from a hundred parties [2, 7] up to multiple tens of thousands [1, 46]. We perform an in-depth analysis in Section 7.

Roadmap

The paper is organized as follows: we formally define weight reduction problems in Section 2 and provide upper bounds for them in Section 3. We then proceed to present our solver called Swiper in Section 4. Section 5 briefly outlines some of the direct applications of these problems, with more detailed discussion postponed to Appendices A and B. Section 6 illustrates how to integrate weight reduction into larger protocols and how to combine it with weighted voting in order to maintain the same resilience as in nominal solutions. In Section 7 and appendix D, we discuss the results of the empirical study we performed on real-world weight distribution. We discuss related work in Section 8 and conclude the paper in Section 9.

2 WEIGHT REDUCTION PROBLEMS

In this section, we define the key building block to our construction, the *weight reduction problems*, which is a class of optimization problems that map (potentially, large) real weights $w_1, \dots, w_n \in \mathbb{R}_{\geq 0}$ to (ideally, small) integer weights $t_1, \dots, t_n \in \mathbb{Z}_{\geq 0}$ while preserving certain key properties. For convenience, we use the word *"tickets"* to denote the units of the assigned integer weights, i.e., if $t_1, \dots, t_n$ is the output of a weight reduction problem, we say that party i is assigned t_i *tickets.*

Notation

To avoid repetition, throughout the rest of the paper, we use the following notation:

(1) $[n] := \{1, 2, \dots, n\}$
(2) for any $S \subseteq [n]$: $w(S) := \sum_{i \in S} w_i$
(3) for any $S \subseteq [n]$: $t(S) := \sum_{i \in S} t_i$
(4) $W := w([n]) = \sum_{i=1}^{n} w_i$
(5) $T := t([n]) = \sum_{i=1}^{n} t_i$

2.1 Weight Restriction

The first weight reduction problem is *Weight Restriction* (or simply WR). It is parameterized by two numbers $\alpha_w, \alpha_n \in (0, 1)$ and requires the mapping to preserve the property that any subset of parties of weight less than α_w obtains less than α_n tickets. More formally:

Problem statement 1 (Weight Restriction)

Given $\alpha_w, \alpha_n \in (0, 1)$ and $w_1, \dots, w_n \in \mathbb{R}_{\geq 0}$ as input, find $t_1, \dots, t_n \in \mathbb{Z}_{\geq 0}$ such that $\sum_{i=1}^{n} t_i$ is minimized, subject to the following restrictions:

(1) $\forall S \subseteq [n]$ such that $w(S) < \alpha_w W$: $t(s) < \alpha_n T$
(2) $T \neq 0$

In Appendix A, we apply Weight Restriction to implement the black-box transformation announced in Section 1.3 as well as weighted versions of secret sharing and threshold cryptography with different access structures. In Section 3, we prove the following theorem:

Theorem 2.1 (WR upper bound). *For any $\alpha_w, \alpha_n \in (0,1)$ such that $\alpha_w < \alpha_n$: there exists a solution to the Weight Restriction problem with $T \leq \left\lceil \frac{\alpha_w(1-\alpha_w)}{\alpha_n - \alpha_w} n \right\rceil$*

To make sense of this expression note that: (1) it is proportional to n; (2) it is inversely proportional to the "gap" between α_w and α_n; (3) the numerator $\alpha_w(1-\alpha_w)$ is smaller than 1 and, in fact, never exceeds 1/4. For a fixed α_w, one can see $\alpha_w(1-\alpha_w)$ as the "constant" and $O\left(\frac{n}{\alpha_n - \alpha_w}\right)$ as the "complexity".

2.2 Weight Qualification

The next weight reduction problem we study is *Weight Qualification* (or simply WQ). It requires the mapping to preserve the property that any subset of parties of weight more than β_w obtains more than β_n tickets. In some sense, WQ is the opposite of the Weight Restriction problem discussed above. More formally:

Problem statement 2 (Weight Qualification)

Given $\beta_w, \beta_n \in (0,1)$ and $w_1, \ldots, w_n \in \mathbb{R}_{\geq 0}$ as input, find $t_1, \ldots, t_n \in \mathbb{Z}_{\geq 0}$ such that $\sum_{i=1}^{n} t_i$ is minimized, subject to the following restrictions:

(1) $\forall S \subseteq [n]$ such that $w(S) > \beta_w W$: $t(s) > \beta_n T$
(2) $T \neq 0$

In Appendix B, we show how to apply Weight Qualification to implement weighted versions of storage and broadcast protocols that rely on erasure and error-correcting codes for minimizing communication and storage complexity.

Interestingly, there exists a simple reduction between WR and WQ:

Theorem 2.2. *For any $\beta_w, \beta_n \in (0,1)$ and $w_1, \ldots, w_n \in \mathbb{R}_{\geq 0}$, the following problems are identical:*

(1) *$WQ(\beta_w, \beta_n, w_1, \ldots, w_n)$*
(2) *$WR(1-\beta_w, 1-\beta_n, w_1, \ldots, w_n)$*

Proof. Let us prove that any valid solution to $WR(1-\beta_w, 1-\beta_n, w_1, \ldots, w_n)$ is a valid solution to $WQ(\beta_w, \beta_n, w_1, \ldots, w_n)$. The inverse can be proven analogously. Indeed, if $\forall S \subseteq [n]$ such that $w(S) > \beta_w W$: $w([n] \setminus S) = W - w(S) < (1-\beta_w)W$. Hence, $t([n] \setminus S) < (1-\beta_n)T$ and $t(S) = T - t([n] \setminus S) > \beta_n T$. □

From Theorems 2.1 and 2.2, we obtain the following:

Corollary 2.3 (WQ upper bound). *For any $\beta_w, \beta_n \in (0,1)$ such that $\beta_n < \beta_w$: there exists a solution to the Weight Qualification problem with $T \leq \left\lceil \frac{\beta_w(1-\beta_w)}{\beta_w - \beta_n} n \right\rceil$*

2.3 Weight Separation

Finally, Weight Separation, in a sense, combines WR and WQ: it has parameters α and β and, intuitively, guarantees that any set of weight β gets more tickets than any set of weight α. Intuitively, it is similar to solving $WR(\alpha, \gamma)$ and $WQ(\beta, \gamma)$ for some unknown $\gamma \in (\alpha, \beta)$ *at the same time*, i.e., with the just a single ticket assignment.

Problem statement 3 (Weight Separation)

Given $\alpha, \beta \in (0,1)$ and $w_1, \dots, w_n \in \mathbb{R}_{\geq 0}$ as input, find $t_1, \dots, t_n \in \mathbb{Z}_{\geq 0}$ such that $\sum_{i=1}^{n} t_i$ is minimized, subject to the following restrictions:

(1) $\forall S_1, S_2 \subseteq [n]$ such that $w(S_1) < \alpha W$ and $w(S_2) > \beta W$: $t(S_1) < t(S_2)$
(2) $T \neq 0$

In this paper, we primarily focus on Weight Restriction and Weight Qualification as they are sufficient for most applications and, being less restrictive on the ticket assignment, permit more efficient solutions. Nevertheless, for completeness, we provide an upper bound on Weight Separation as well.

Theorem 2.4 (WS upper bound). *For any $\alpha, \beta \in (0,1)$ such that $\alpha < \beta$: there exists a solution to the Weight Separation problem with $T \leq \frac{(\alpha+\beta)(1-\alpha)}{\beta-\alpha} n$.*

Note that the numerator $(\alpha + \beta)(1 - \alpha)$ is always smaller than 1 for $0 < \alpha < \beta < 1$.

3 UPPER BOUNDS

In this section, we provide formal proofs for Theorems 2.1 and 2.4 and corollary 2.3.

3.1 Upper bounds on Weight Restriction and Weight Separation

Let us start with some auxiliary definitions. A *ticket assignment* t is a vector of n numbers: $t_1, \dots, t_n \in \mathbb{Z}_{\geq 0}$. With a slight abuse of notation, for a ticket assignment t and a set $S \subseteq [n]$, we use notation $t(S)$ to denote $\sum_{i \in S} t_i$. Let us say that a ticket assignment t is *viable* if $t([n]) \neq 0$ and $\forall S \subseteq [n]$: if $w(S) < \alpha_w W$, then $t(S) < \alpha_n t([n])$, that is if it satisfies the requirements of the Weight Restriction problem as defined in Section 2.

In this section, we formally prove Theorem 2.1 by constructing a viable ticket assignment $\hat{t}$ such that $\hat{t}([n]) \leq \left\lceil \frac{\alpha_w(1-\alpha_w)}{\alpha_n - \alpha_w} n \right\rceil$. As the starting point, we consider a family of ticket assignments parameterized by a single number $s > 0$: $t_{s,i} := \lfloor w_i s + \alpha_w \rfloor$.

Initial bound. As a warm-up and in order to later bootstrap our algorithm in Section 4, let us prove that, for $s := \frac{\alpha_n(1-\alpha_w)n}{(\alpha_n - \alpha_w)W}$, t_s is viable. This will yield our first upper bound on Weight Restriction that we will then further improve.

Let t denote t_s and let S be a subset of $[n]$ such that $w(S) < \alpha_w W$ and $t(S)$ is maximum among all subsets of weight less than $\alpha_w W$. We need to prove that $t(S) < \alpha_n t([n])$, which is equivalent to showing that $(1 - \alpha_n)t(S) < \alpha_n t(\overline{S})$, where $\overline{S} := [n] \setminus S$. Indeed:

$$t(S) < \alpha_n T \Leftrightarrow t(S) < \alpha_n(t(S) + t(\overline{S})) \Leftrightarrow (1 - \alpha_n)t(S) < \alpha_n t(\overline{S})$$

To this end, let us provide an upper bound on $t(S)$ and a lower bound on $t(\overline{S})$:

$$t(S) = \sum_{i \in S} t_i = \sum_{i \in S} \lfloor w_i s + \alpha_w \rfloor \leq \sum_{i \in S} (w_i s + \alpha_w) < \alpha_w W s + \alpha_w |S|$$

$$t(\overline{S}) = \sum_{i \notin S} \lfloor w_i s + \alpha_w \rfloor \geq \sum_{i \notin S} (w_i s + \alpha_w - 1) > (1 - \alpha_w) W s - (1 - \alpha_w)(n - |S|)$$

Now we can prove that t is viable. Notice that "$<$" changes to "$\leq$" because both the upper bound on $t(S)$ and the lower bound on $t(\overline{S})$ are strict:

$$\begin{aligned}
&(1-\alpha_n)t(S) < \alpha_n t(\overline{S}) \\
\Leftarrow\ &(1-\alpha_n)(\alpha_w W s + \alpha_w |S|) \leq \alpha_n((1-\alpha_w)Ws - (1-\alpha_w)(n-|S|)) \\
\Leftrightarrow\ &s \geq \frac{\alpha_n(1-\alpha_w)n}{(\alpha_n-\alpha_w)W} - \frac{1}{W}|S|
\end{aligned}$$

Since $|S| \geq 0$, this inequality is trivially satisfied for $s := \frac{\alpha_n(1-\alpha_w)n}{(\alpha_n-\alpha_w)W}$. Hence, t is viable. Intuitively, what we see here is that the bigger s is–the closer t_s approximates the input weights $w_1, \ldots, w_n$ and, for a sufficiently large s, regardless of the input weights, the approximation is guaranteed to be good enough to satisfy the Weight Restriction requirement.

Let us now provide an upper bound on the number of tickets resulting from such distribution:

$$t([n]) = \sum_{i\in[n]} \lfloor w_i s + \alpha_w \rfloor \leq \sum_{i\in[n]} (w_i s + \alpha_w) \leq Ws + n\alpha_w = \frac{\alpha_n(1-\alpha_w)}{\alpha_n - \alpha_w} n + n\alpha_w = \frac{\alpha_n - \alpha_w^2}{\alpha_n - \alpha_w} n$$

This upper bound is suboptimal and we will significantly improve it. Even here you can see that, assuming α_n and α_w are constants and $\alpha_w < \alpha_n$, the total number of tickets is at most $O(n)$. However, to achieve the results stated in Theorem 2.1, we need a slightly more elaborate approach as presented in the rest of this section.

Improved bound. Let s^* be a *locally minimal* viable value for s, i.e., a positive number such that t_{s^*} is viable, but $t_{s^*-\varepsilon}$ is not, for any sufficiently small ε. Since we already proved that viable values of s exist, it is easy to see that such s^* exists. Moreover, there must be some j such that $s^* w_j + \alpha_w$ is an integer. Indeed, if this does not hold, we would be able to slightly decrease s^* without changing the ticket assignment, which would contradict the assumption that s^* is a local minimum. Let $t^* := t_{s^*}$ and $J := \{j \in [n] \mid s^* w_i + \alpha_w \text{ is an integer}\}$. Let t' be a ticket assignment in which we take one ticket from each party in J, i.e.:

$$t'_i := \begin{cases} t^*_i - 1 & \text{if } i \in J \\ t^*_i & \text{otherwise} \end{cases}$$

Notice that t' is equal to $t_{s^*-\varepsilon}$ for a sufficiently small $\varepsilon > 0$.[2] Hence, by construction, t' is not viable. Now, let us consider a set of "intermediate" ticket assignments: we will be taking tickets from parties in J as long as the ticket assignment stays viable. We will end up with two ticket assignments: $\hat{t}$ and $\hat{\hat{t}}$ such that $\hat{t}$ is viable and $\hat{\hat{t}}$ is not, and $\hat{t}([n]) = \hat{\hat{t}}([n]) + 1$. All that is left is to prove that $\hat{t}([n]) \leq \left\lceil \frac{\alpha_w(1-\alpha_w)}{\alpha_n-\alpha_w} n \right\rceil$ or, equivalently, that $\hat{\hat{t}}([n]) \leq \left\lceil \frac{\alpha_w(1-\alpha_w)}{\alpha_n-\alpha_w} n \right\rceil - 1$.

Since $\hat{\hat{t}}$ is not viable, either $\hat{\hat{t}}([n]) = 0$ or there must exist a set $S \subseteq [n]$ such that $w(S) < \alpha_w W$ and $\hat{\hat{t}}(S) \geq \alpha_n \hat{\hat{t}}([n])$. As the former case is trivial, we will focus on the latter. Let us provide an upper bound on $\hat{\hat{t}}(S)$ and a lower bound on $\hat{\hat{t}}(\overline{S})$, where $\overline{S} := [n] \setminus S$. To this end, let us note that, for any $i \in [n]$, it holds that $\hat{\hat{t}}_i \geq w_i s^* + \alpha_w - 1$. Indeed, there are two cases to consider:

(1) if $\hat{\hat{t}}_i = t^*_i$, the inequality holds trivially as $\hat{\hat{t}}_i = t^*_i = \lfloor w_i s^* + \alpha_w \rfloor$;
(2) otherwise, $\hat{\hat{t}}_i = t^*_i - 1$. However, by construction, it means that $w_i s^* + \alpha_w$ is an integer and, thus $t^*_i = w_i s^* + \alpha_w$ and $\hat{\hat{t}}_i = w_i s^* + \alpha_w - 1$.

[2]Indeed, if we decrease s^* by any positive amount, each party in J will lose at least 1 ticket as they will step over the rounding threshold. However, it is also easy to see that ε can be made small enough so that no other party will lose a ticket and no party in J will lose more than one ticket.

Hence:

$$\hat{\hat{t}}(S) = \sum_{i \in S} \hat{\hat{t}}_i \leq \sum_{i \in S} t_i^* = \sum_{i \in S} \lfloor w_i s^* + \alpha_w \rfloor < \alpha_w W s^* + \alpha_w |S|$$
$$\hat{\hat{t}}(\overline{S}) = \sum_{i \notin S} \hat{\hat{t}}_i \geq \sum_{i \notin S} (w_i s^* + \alpha_w - 1) > (1 - \alpha_w) W s^* - (1 - \alpha_w)(n - |S|)$$

By construction, $\hat{\hat{t}}(S) \geq \alpha_n \hat{\hat{t}}([n])$ and $\hat{\hat{t}}([n]) = \hat{\hat{t}}(S) + \hat{\hat{t}}(\overline{S})$. Hence, $(1 - \alpha_n)\hat{\hat{t}}(S) \geq \alpha_n \hat{\hat{t}}(\overline{S})$. From this, we can derive an upper bound on s^*:

$$\begin{aligned}
&(1 - \alpha_n)\hat{\hat{t}}(S) \geq \alpha_n \hat{\hat{t}}(\overline{S}) \\
\Rightarrow\ &(1 - \alpha_n)(\alpha_w W s^* + \alpha_w |S|) > \alpha_n((1 - \alpha_w) W s^* - (1 - \alpha_w)(n - |S|)) \\
\Rightarrow\ &s^* < \frac{\alpha_n (1 - \alpha_w) n}{(\alpha_n - \alpha_w) W} - \frac{|S|}{W}
\end{aligned}$$

Finally, we can combine everything into an upper bound on $\hat{\hat{t}}([n])$:

$$\hat{\hat{t}}([n]) \leq \frac{\hat{\hat{t}}(S)}{\alpha_n} < \frac{\alpha_w}{\alpha_n}(W s^* + |S|) < \frac{\alpha_w}{\alpha_n}\left(\frac{\alpha_n (1 - \alpha_w) n}{\alpha_n - \alpha_w} - |S| + |S|\right) = \frac{\alpha_w (1 - \alpha_w)}{\alpha_n - \alpha_w} n$$

Since $\hat{\hat{t}}([n])$ is an integer and the inequality is strict, we can rewrite it as $\hat{\hat{t}}([n]) \leq \left\lceil \frac{\alpha_w(1-\alpha_w)}{\alpha_n - \alpha_w} n \right\rceil - 1$. As, by construction, $\hat{t}$ is viable and $\hat{t}([n]) = \hat{\hat{t}}([n]) + 1$, we found a viable ticket assignment with at most $\left\lceil \frac{\alpha_w(1-\alpha_w)}{\alpha_n - \alpha_w} n \right\rceil$ tickets, thus concluding the proof of Theorem 2.1 and corollary 2.3. □

3.2 Upper bound on Weight Separation

Let $\gamma := \frac{\alpha+\beta}{2}$. For Weight Separation, we analyze a family of ticket assignments of form $t_{s,i} := \lfloor w_i s + \gamma \rfloor$. Let us consider the case when the WS conditions are violated, i.e., there exist sets S_1 and S_w such that $w(S_1) < \alpha W$, $w(S_2) > \beta W$, and $t(S_1) \geq t(S_2)$. This means that at least one of two events happened: $t(S_1) \geq \gamma T$ or $t(S_2) < \gamma T$, or, equivalently, $t(\overline{S_2}) > (1 - \gamma)T$.

Let us first consider the case when $t(S_1) \geq \gamma T$. This can only happen when $s < \frac{\gamma(1-\gamma)n}{(\gamma-\alpha)W}$. The proof is done using the same set of techniques as in Section 3.1:

$$t(S_1) = \sum_{i \in S_1} t_i = \sum_{i \in S_1} \lfloor w_i s + \gamma \rfloor \leq \sum_{i \in S_1} (w_i s + \gamma) < \alpha W s + \gamma |S|$$
$$t(\overline{S_1}) = \sum_{i \notin S_1} \lfloor w_i s + \gamma \rfloor \leq \sum_{i \notin S_1} (w_i s + \gamma - 1) > \beta W s - (1 - \gamma)(n - |S|)$$
$$t(S_1) \geq \gamma T \Leftrightarrow (1 - \gamma) t(S_1) \geq \gamma t(\overline{S_1})$$
$$(1 - \gamma)(\alpha W s + \gamma |S|) > (1 - \gamma) t(S_1) \geq \gamma t(\overline{S_1}) > \gamma(\beta W s - (1 - \gamma)(n - |S|))$$
$$s < \frac{\gamma(1 - \gamma) n}{(\gamma - \alpha) W}$$

Analogously, in the case when $t(\overline{S_2}) > (1-\gamma)T$, we can prove that $s < \frac{(1-\gamma)(1-(1-\gamma))n}{((1-\gamma)-(1-\beta))} W = \frac{\gamma(1-\gamma)n}{\beta - \gamma}$. We specifically chose $\gamma = \frac{\alpha+\beta}{2}$ so that the two bounds coincide: $s < \frac{2\gamma(1-\gamma)n}{(\beta-\alpha)W}$. Hence, it is sufficient to select $s := \frac{\gamma(2-\alpha-\beta)n}{(\beta-\alpha)W}$ to guarantee that neither of the two events happens and $t(S_1) < \gamma T \leq t(S_2)$.

Let us now compute a bound on the total number of tickets:

$$T \le sW + \gamma n = \frac{(\alpha+\beta)(1-\alpha)}{\beta-\alpha} n$$

□

4 SWIPER: APPROXIMATE SOLVER FOR WEIGHT REDUCTION PROBLEMS

System	Total weight	# parties	# tickets using Swiper			
			$\alpha_w = 1/4$ $\alpha_n = 1/3$	$\alpha_w = 1/3$ $\alpha_n = 3/8$	$\alpha_w = 1/3$ $\alpha_n = 1/2$	$\alpha_w = 2/3$ $\alpha_n = 3/4$
			$\beta_w = 3/4$ $\beta_n = 2/3$	$\beta_w = 2/3$ $\beta_n = 5/8$	$\beta_w = 2/3$ $\beta_n = 1/2$	$\beta_w = 1/3$ $\beta_n = 1/4$
Aptos [2, 7]	8.4708×10^8	104	58	206	27	110
Tezos [4, 36]	6.7579×10^8	382	133	419	61	258
Filecoin [3, 43]	2.5242×10^{19}	3700	3106	8225	1535	4699
Algorand [1, 46]	9.7223×10^9	42 920	745	13449	291	6354

Table 2. Number of tickets allocated by the Swiper protocol on sample weight distributions.

To facilitate applications of weight reduction problems, we designed Swiper – a fast approximate solver for Weight Restriction and Weight Qualification. We are also planning on adding support for Weight Separation shortly. Swiper enjoys a number of desirable properties:

(1) **Robustness:** It always respects the upper bounds stated in Section 2. This means that even under a malicious distribution of weights, the number of assigned tickets will be within a known limit, linear in the number of parties.
(2) **Determinism:** Swiper is a deterministic protocol. Hence, assuming the initial weights are common knowledge, each party can run it locally and all parties will obtain the same result. This eliminates the need for executing a complex consensus protocol to agree on the ticket assignment.
(3) **Practical allocation efficiency:** As we study extensively in Section 7, Swiper performs really well on real-world weight distributions, allocating even fewer tickets that is predicted by the upper bounds. In Table 2, we summarize the number of tickets allocated by Swiper on the distribution of funds in four major blockchain systems [1, 2, 3, 4] with some example thresholds. Notice that, in many cases, the number of tickets is actually below the number of users. This happens partly due to the distributions being significantly skewed and a large number of users actually owning only a small fraction of the total funds.
(4) **Computational efficiency:** Assuming that the thresholds (α_w, α_n, β_w, and β_n) are constants, Swiper run-time is just $\tilde{O}(n^2)$. For especially large systems, there is also a quasilinear mode with runtime $\tilde{O}(n)$. In practice, it yields almost the same number of tickets as the full solver, but the upper bounds for this mode are not as good even though the total number of tickets is still provably at most linear in n. Moreover, Swiper is optimized to run fast in practice. Generating Table 2 takes only about 30 seconds in full mode and 15 seconds in quasilinear mode on a single core with a Python implementation. For low-latency applications, an implementation in a more performance-oriented programming language may be required.

To respect the double-blind policy of the conference, we provide the full code for Swiper in an anonymous GitHub repository.[3]

At its core, Swiper simply follows the structure of the proof in Section 3.1. However, making it efficient requires several important observations.

First and foremost, we observe that one can efficiently verify if a ticket assignment is viable (i.e., if it satisfies the Weight Restriction requirements) by solving the Knapsack problem [42] in time $O(n^2)$. While, in general, Knapsack is known to be NP-hard, in our specific case, it can be solved in time $O(Tn)$ by using *dynamic programming on profits* [42, Section 2.3]. Recall that in Section 3.1 we proved that by selecting $s := \frac{\alpha_n(1-\alpha_w)n}{(\alpha_n-\alpha_w)W}$ we will obtain a viable ticket assignment with at most $O(n)$ tickets. In Swiper, we use this bound to bootstrap the protocol and, hence, we only ever solve Knapsack with $T = O(n)$.

Second, we use binary search to find the local minima of s^* and k^* – the number of parties in J that should be rounded up in the construction of Section 3.1. Indeed, while finding globally minimal s that would still yield a viable ticket assignment may be difficult, we only need a local minimum, and, thus, binary search is sufficient. The same applies to k^*.

Finally, we use a well-known approximation algorithm for Knapsack in the binary search before using an actual Knapsack solver. Then, we use a special form of local-first binary search to precisely find the optimal values. Thanks to this optimization, in practice, we only need to run the Knapsack solver a few times, typically less than 5, instead of a logarithmic number. We completely omit this step in the aforementioned quasilinear mode.

5 DIRECT APPLICATION EXAMPLES

Besides studying the weight reduction problems themselves, the major contribution of this paper is in exploring applications of weight reduction in distributed computing and applied cryptography. Unfortunately, due to the space limitations, the detailed discussion of the direct applications of Weight Restriction and Weight Separation is delegated to Appendices A and B respectively. In this Section, we briefly outline the intuition behind some of them. We will then discuss how to combine them with weighted voting in Section 6.

5.1 Distributed random number generation

As a motivating example for Weight Restriction, consider the *Distributed Random Number Generation* problem. Typically, it needs to satisfy two properties:

- If *all* honest parties cooperate, they can generate the next random number;
- Unless *at least one* honest party wants to open the next random number, it remains completely unpredictable to the adversary.

We say that it has *blunt access structure* (defined formally in Appendix A.1).

Perhaps, the simplest way it can be achieved [52] is by having a trusted party generate the random number and pre-distribute it using secret sharing [55], such that each party gets a number t_i of shares and any subset of parties possessing at least $\lceil \alpha_n T \rceil$ shares (where $T = \sum_{i=1}^{n} t_i$) can reconstruct the secret, but no set of parties possessing less than this amount shares can learn anything about the secret.

Thus, by setting α_w to the resilience of the protocol ($\alpha_w := f_w$) and $\alpha_n \leq \frac{1}{2}$, we can guarantee that:

- Corrupt participants will receive less than $\alpha_n T$ shares and, hence, will not be able to reconstruct the random number unless some honest party also wants to open it;

[3]https://github.com/swiper-double-blind/swiper-double-blind-submission

- Honest participants will receive more than $(1 - \alpha_n)T \geq \alpha_n T\rceil$ shares and, hence, will be able to reconstruct the random number.

Practical randomness beacons [19, 54] operate similarly, but replace the trusted party with a distributed setup protocol and employ *unique threshold signatures* [11, 56] in order to be able to reuse the same secret multiple times.

5.2 Black-box transformation

Weight Restriction also allows us to transform an arbitrary nominal protocol to the weighted model with a slight reduction in resilience using what we call a "black-box transformation". Intuitively, given a nominal protocol with resilience f_n, one can set $\alpha_n := f_n$ and $\alpha_w := f_n - \epsilon$ for an arbitrary small ε. Then we can simulate a protocol with $T = \sum_{i\in[n]} t_i$ virtual participants, with each party i controlling t_i of them. By the definition of Weight Restriction, if the adversary is limited to corrupting parties of total weight $f_w := \alpha_w$, it will also control less than $\alpha_n = f_n$ virtual participants and, thus, will not be able to "break" the nominal protocol.

Assuming the initial weights are common knowledge, if all parties execute the same deterministic algorithm for solving Weight Restriction (e.g., Swiper), they will also all agree on the new virtual membership.

Notice, however, that this transformation may negatively affect any sort of "fairness" properties. Indeed, after the mapping, some party may observe their relative weight decrease. We discuss the issues of fairness in slightly more detail in Section 9.

5.3 Erasure and error-correcting coded storage and broadcast

The main application of Weight Qualification that we consider is erasure and error-correcting codes in storage and broadcast protocols. Indeed, what we typically want when applying such codes, is for any subset of honest parties of a certain weight (β_w) to be able to reconstruct the original encoded data if they collaborate. If we use tickets to determine how many fragments of the original data a party gets, then this requirement directly translates to them obtaining at least a fraction α_n of tickets, where α_n is the rate of the code. Notice that, in these applications, we are not interesting in restricting any set of parties from reconstructing the data as erasure codes are not designed neither they are typically used to provide secrecy on their own.

We discuss how to apply WQ to obtain weighted versions of the state-of-the-art nominal storage systems and broadcast protocols in much greater detail in Appendix B. Crucially, in order to move from the nominal to the weighted model, we just slightly reduce the rate of the code, thus adding a small communication overhead, but we can keep the resilience of the weighted protocol the same as of its nominal counterpart ($f_w = f_n$).

6 DERIVED APPLICATIONS

In this section, we discuss indirect applications of weight reduction problems that are obtained by using one or multiple building blocks discussed in Section 5 and appendices A and B. For all applications discussed here, we manage to avoid losing resilience despite applying weight reduction. In all cases, the majority of the protocol logic should be converted to the weighted model by applying weighted voting, as discussed in Section 1.2.

6.1 Asynchronous State Machine Replication

For asynchronous state machine replication protocols [27, 32, 41, 47, 57], we simply need to use a weighted communication-efficient broadcast protocol (discussed in Appendix B) and weighted distributed random number generation (discussed in Appendix A.1). The distributed number generation part can use a nominal protocol with threshold $\alpha_n = \frac{1}{2}$ and set $\alpha_w := \frac{1}{3}$, which is the resilience of the rest of the protocol. Thus, in some sense, we level the resilience of different parts of the protocol, without affecting the resilience of the composition.

6.2 Validated Asynchronous Byzantine Agreement

The same approach can be applied to generate randomness for Validated Asynchronous Byzantine Agreement (VABA) [6, 18].

These protocols also require tight threshold signatures. However, in practice, multi-signatures [11, 51] are usually applied instead as they have almost no overhead over threshold signatures on the system sizes where such protocols could be applied (below 1000 participants): it suffices to append the multi-signature with an array of n bits, indicating the set of parties that produced the signature. Then, along with the verification of the validity of the multi-signature itself, anyone can verify that the signers together hold sufficient weight.

Alternatively, one could apply the approach described in Appendix A.2 to implement tight weighted threshold signatures. However, it would lead to an increase in message complexity of the resulting protocol, which we want to avoid.

Finally, an ad-hoc weighted threshold signature scheme can be applied, such as the one recently proposed in [28]. Note that these signatures cannot be used for distributed randomness generation as they lack the necessary uniqueness property, and thus we still need to apply Swiper to obtain a complete protocol.

6.3 Consensus with Checkpoints

We can apply the same approach for checkpointing proof-of-stake consensus protocols [8], but this time for blunt threshold signatures (as discussed in Appendix A.1) instead of random number generation. If, for some reason, one wants to use a tight threshold signature, the approach described in Appendix A.2 can be applied at the cost of just 1 additional message delay per checkpoint.

Compared to ad-hoc solutions for weighted threshold signatures [28], we claim that our approach is more computationally efficient as it is basically as fast as the underlying nominal protocol. For example, 1 pairing to verify a BLS signature [11] compared to 13 pairings to verify a signature in [28]. Moreover, the weight reduction approach is more general and can support other types of threshold signatures, such as RSA [56] and Schnorr [58], the latter being particularly important in the context of checkpointing to Bitcoin [8].

7 ANALYZING WEIGHT RESTRICTION ON SAMPLE SYSTEMS

We performed two kinds of experiments on real blockchain data. In the first experiment, shown in the left column of Figure 1, we analyzed the influence of the choice of parameters α_w and α_n for the original data retrieved from the blockchains; the value of α_n was varied in the range $[0.1, 1]$, while the value of α_w was tested in the range $[0.1 \times \alpha_n, 0.9 \times \alpha_n]$. In the experiments showcased in the right column of Figure 1, we kept these parameters fixed and analyzed the influence of the number of parties in the metrics we tracked. In order to simulate having the same blockchain with different numbers of parties, we used the statistical technique known as bootstrapping. To this end, we performed 380 experiments sampling parties with replacement from the blockchain data and taking the average of the results.

In each experiment, we tracked the total number of tickets distributed, the maximum number of tickets held by a single party, and the number of parties that get at least one ticket (in the figures, we label them as the number of holders). In Figure 1, we show the results for the Tezos blockchain. The results for Algorand, Aptos, and Filecoin are available in Appendix D. The analysis of the results reveals the following information: the upper bound given in Section 2 is very pessimistic for

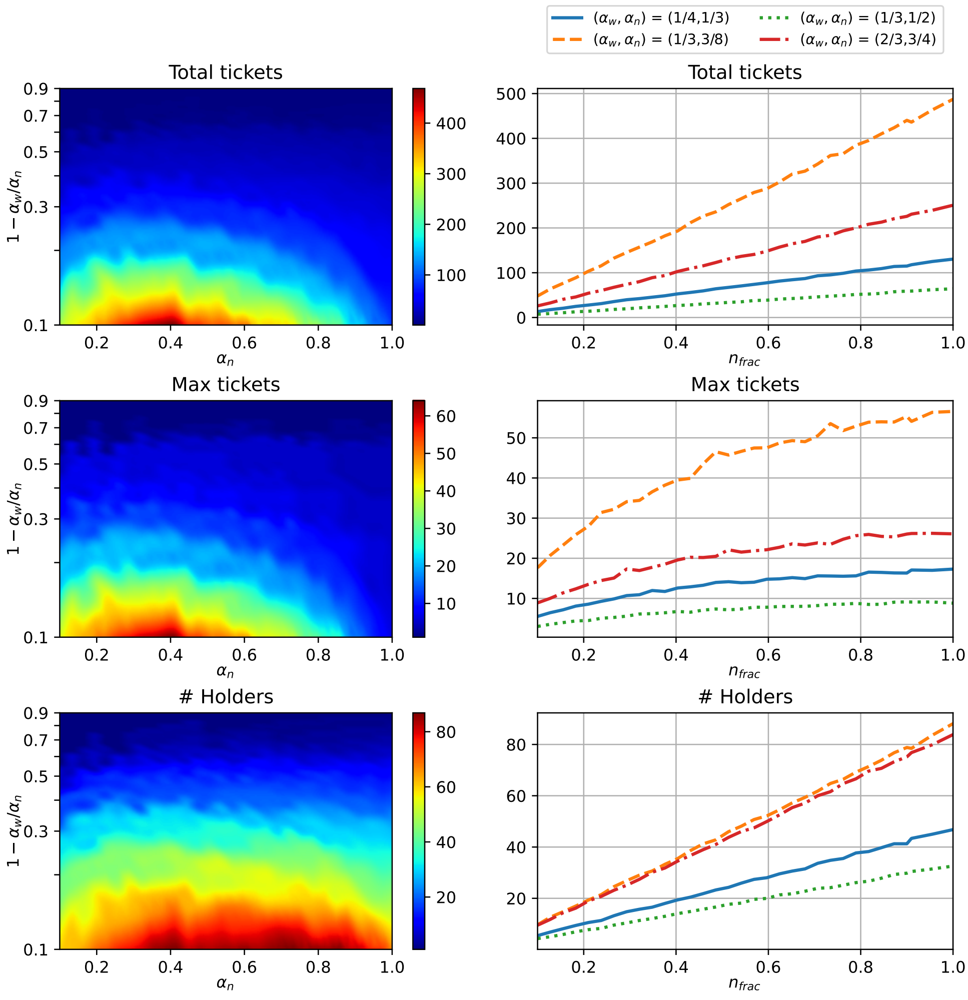

Fig. 1. Experiment results using Tezos (380 samples per data point)

weight distributions emerging in practice, with the total number of tickets rarely surpassing the number of parties for different values of α_n and α_w. The total number of tickets varies extremely close to a linear function on the number of parties, as well as the number of holders. The maximum number of tickets, on the other hand, seems to saturate when the number of parties in absolute terms surpasses the order of magnitude of 1000, remaining almost constant after that point.

8 RELATED WORK

Knapsack. The Knapsack problem and its variations hold huge importance in theoretical computer science and have numerous applications in both theory and practice. The weight reduction problems studied in this paper seem to be related to, or can even be seen as a variation of the famous Knapsack problem. For example, one can see Weight Restriction as the problem of constructing "worst possible" profits for a Knapsack instance given the weights and the capacity. We refer to [42] for a comprehensive survey on the topic.

Virtual users. The simplest solution for creating a weighted threshold cryptographic system is to simply have a user of weight w to become w virtual users and to give one key to each of them. Shamir's paper describing his secret sharing scheme [55] puts forward this solution. However, in practice, the total weight tends to be prohibitively large, and "quantizing" it requires solving weight reduction problems, which is the main subject of this paper.

Weighted voting. In [35], Gifford presents the idea of weighted voting for distributed storage systems. The paper suggests assigning weights to replicas according to the estimated failure probabilities and using weight-based quorums to store and retrieve data. We discuss the merits and limitations of this approach in Section 1.2. The goal of our paper is to complement the weighted voting approach and design a framework for implementing weighted distributed protocols that can benefit from solutions and primitives that are initially designed for the nominal model. In Appendices A and B and section 6, we discuss in detail how to combine weighted voting and weight reduction to obtain extremely efficient weighted protocols without sacrificing resilience.

Ad-hoc solutions. There is a large body of work studying ad-hoc weighted cryptographic protocols [9, 10, 25, 28, 33, 38]. Compared to these works, the weight reduction approach studied in this paper has a number of benefits, such as simplicity, efficiency, wider applicability, and a wider range of possible cryptographic assumptions. Moreover, in many cases, ad-hoc solutions can be combined with and benefit from weight reduction. In this paper, we also study other applications, non-cryptographic, applications, such as erasure and error-corrected distributed storage and broadcast protocols.

Similar work by Benhamouda, Halevi, and Stambler. A recent work [10] mentioned a similar idea of reducing real weights to integers to construct *ramp* secret sharing. This project has been started and the first versions of Swiper has been drafted before the online publication and without any knowledge of [10]. As the main focus of [10] is different, we believe that we do a much more in-depth exploration of this direction by studying different kinds of weight reduction problems and their applications beyond secret sharing, as well as providing much tighter bounds and implementing a solver that is not only linear in the worst case but also allocates very few tickets in empirical evaluations on real-world weight distributions.

9 CONCLUDING REMARKS AND FUTURE WORK DIRECTIONS

In this paper, we have presented a family of optimization problems called weight reduction that, to the best of our knowledge, has not been studied before. We provided practical protocols to find good, albeit not optimal, solutions to these problems. As we have shown, it allows us to obtain efficient implementations of many weighted distributed protocols.

We believe that weight reduction problems will play an important role in the future of blockchain systems as they become more sophisticated and the need for threshold cryptography as well as erasure coding and protocols like single secret leader election grows. At the time of writing, at least one major layer-1 blockchain has already integrated a version of Weight Separation for generating on-chain randomness.

In this paper, we attempted the first systematic study of this family of problems, but there are still many important questions being left for future research.

Fairness. Weight reduction naturally leads to slight deviations in relative weights of the participants. While in this paper we focused on *safety* and *liveness* properties and showed that they can still be preserved, we did not consider any kind of *fairness* properties. However, we believe that, somewhat counter-intuitively, some form of fairness can be preserved as well. To this end, we are considering two possible direction:

(1) **Expected fairness:** In addition to deterministically assigned tickets, we can allocate some small number of tickets randomly so that each party gets exactly the same fraction of tickets as its fraction of weight *in expectation*. We believe that it can be done while still preserving safety and liveness *deterministically*, i.e., even in the worst case, when all the "random" tickets are received by the adversary.
(2) **Integral fairness:** Similarly, one can imagine a *deterministic* protocol that provides fairness *over time*. In such a scheme, the ticket allocation will be updated periodically and each party will get exactly the right number of tickets *on average*, over a large enough period.

Incentives. One important aspect of proof-of-stake blockchains is the distribution of incentives, which should depend on the weight of each party. It is not immediately clear what is the right way to allocate incentives in a system where weight reduction is being applied.

Other applications. While we covered a wide range of applications in this paper, we believe that there must be others, including ones not related to distributed computing or applied cryptography.

Adversarial attacks. In this paper, we study the "worst case" weight distributions by providing the upper bounds and the "organic case" by studying the real-world weight distributions. However, in practice, under an adversarial attack, the weight distribution will be a hybrid one: the weights of honest parties will be organic, but the weights of the adversarial parties may be redistributed maliciously. It is an interesting avenue for future work to study how much an adversary can affect the number of tickets (and, thus, the performance of the system) by redistributing their weight in a malicious manner.

Complexity and more precise bounds. Finally, there are still many theoretical questions about these problems. Are there polynomial-time exact solutions? What are the lower bounds? Can we derive better upper bounds? Finally, what are some other interesting and useful weight reduction problems, apart from the three defined in this paper?

REFERENCES

[1] [n. d.]. Algorand stake distribution. https://algoexplorer.io/top-accounts. Accessed: 2023-03-28.
[2] [n. d.]. Aptos stake distribution. https://aptoscan.com/validators?ps=100&p=. Accessed: 2023-03-28.
[3] [n. d.]. Filecoin stake distribution. https://filfox.info/en/ranks/power. Accessed: 2023-03-28.
[4] [n. d.]. Tezos stake distribution. https://tezos.fish/leaderboard/all. Accessed: 2023-03-28.
[5] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. 2021. Reaching consensus for asynchronous distributed key generation. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*. 363–373.
[6] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. 2019. Asymptotically optimal validated asynchronous byzantine agreement. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. 337–346.
[7] Aptos. 2022. *White paper – The Aptos Blockchain: Safe, Scalable, and Upgradeable Web3 Infrastructure.* Technical Report. https://aptos.dev/aptos-white-paper/
[8] Sarah Azouvi and Marko Vukolić. 2022. Pikachu: Securing PoS Blockchains from Long-Range Attacks by Checkpointing into Bitcoin PoW using Taproot. In *Proceedings of the 2022 ACM Workshop on Developments in Consensus*. 53–65.
[9] Amos Beimel and Enav Weinreb. 2006. Monotone circuits for monotone weighted threshold functions. *Inform. Process. Lett.* 97, 1 (2006), 12–18.
[10] Fabrice Benhamouda, Shai Halevi, and Lev Stambler. 2022. Weighted Secret Sharing from Wiretap Channels. *Cryptology ePrint Archive* (2022).
[11] Alexandra Boldyreva. 2002. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *Public Key Cryptography — PKC 2003*, Yvo G. Desmedt (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 31–46.
[12] Dan Boneh, Saba Eskandarian, Lucjan Hanzlik, and Nicola Greco. 2020. Single secret leader election. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*. 12–24.
[13] Dan Boneh, Saba Eskandarian, Lucjan Hanzlik, and Nicola Greco. 2020. Single secret leader election. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*. 12–24.

[14] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter MR Rasmussen, and Amit Sahai. 2018. Threshold cryptosystems from threshold fully homomorphic encryption. In *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part I 38*. Springer, 565–596.

[15] Gabriel Bracha and Sam Toueg. 1985. Asynchronous consensus and broadcast protocols. *Journal of the ACM (JACM)* 32, 4 (1985), 824–840.

[16] Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. 2011. *Introduction to reliable and secure distributed programming*. Springer Science & Business Media.

[17] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. 2002. Asynchronous verifiable secret sharing and proactive cryptosystems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*. 88–97.

[18] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. 2001. Secure and efficient asynchronous broadcast protocols. In *Advances in Cryptology—CRYPTO 2001: 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19–23, 2001 Proceedings*. Springer, 524–541.

[19] Christian Cachin, Klaus Kursawe, and Victor Shoup. 2000. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography. In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*. 123–132.

[20] Christian Cachin and Stefano Tessaro. 2005. Asynchronous verifiable information dispersal. In *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*. IEEE, 191–201.

[21] Ran Canetti and Tal Rabin. 1993. Fast asynchronous Byzantine agreement with optimal resilience. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*. 42–51.

[22] Miguel Castro, Barbara Liskov, et al. 1999. Practical byzantine fault tolerance. In *OsDI*, Vol. 99. 173–186.

[23] Dario Catalano, Dario Fiore, and Emanuele Giunta. 2022. Adaptively secure single secret leader election from DDH. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*. 430–439.

[24] Dario Catalano, Dario Fiore, and Emanuele Giunta. 2023. Efficient and universally composable single secret leader election from pairings. In *Public-Key Cryptography–PKC 2023: 26th IACR International Conference on Practice and Theory of Public-Key Cryptography, Atlanta, GA, USA, May 7–10, 2023, Proceedings, Part I*. Springer, 471–499.

[25] Pyrros Chaidos and Aggelos Kiayias. 2021. Mithril: Stake-based threshold multisignatures. *Cryptology ePrint Archive* (2021).

[26] Benoît Colson, Patrice Marcotte, and Gilles Savard. 2007. An overview of bilevel optimization. *Annals of operations research* 153, 1 (2007), 235–256.

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

[28] Sourav Das, Philippe Camacho, Zhuolun Xiang, Javier Nieto, Benedikt Bunz, and Ling Ren. 2023. Threshold Signatures from Inner Product Argument: Succinct, Weighted, and Multi-threshold. *Cryptology ePrint Archive* (2023).

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

[30] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. 2022. Practical asynchronous distributed key generation. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2518–2534.

[31] Yvo Desmedt. 1993. Threshold cryptosystems. In *Advances in Cryptology—AUSCRYPT'92: Workshop on the Theory and Application of Cryptographic Techniques Gold Coast, Queensland, Australia, December 13–16, 1992 Proceedings 3*. Springer, 1–14.

[32] Sisi Duan, Michael K Reiter, and Haibin Zhang. 2018. BEAT: Asynchronous BFT made practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2028–2041.

[33] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. 2022. Cryptography with Weights: MPC, Encryption and Signatures. *Cryptology ePrint Archive* (2022).

[34] Giuliano Garrammone. 2013. On decoding complexity of reed-solomon codes on the packet erasure channel. *IEEE Communications Letters* 17, 4 (2013), 773–776.

[35] David K Gifford. 1979. Weighted voting for replicated data. In *Proceedings of the seventh ACM symposium on Operating systems principles*. 150–162.

[36] L.M Goodman. 2014. *White paper – Tezos: a self-amending crypto-ledger*. Technical Report. https://tezos.com/whitepaper.pdf

[37] James Hendricks, Gregory R Ganger, and Michael K Reiter. 2007. Verifying distributed erasure-coded data. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*. 139–146.

[38] Mitsuru Ito, Akira Saito, and Takao Nishizeki. 1989. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)* 72, 9 (1989), 56–64.

[39] Aayush Jain, Peter MR Rasmussen, and Amit Sahai. 2017. Threshold fully homomorphic encryption. *Cryptology ePrint Archive* (2017).
[40] Jørn Justesen. 1976. On the complexity of decoding Reed-Solomon codes (Corresp.). *IEEE transactions on information theory* 22, 2 (1976), 237–238.
[41] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. 2021. All you need is dag. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*. 165–175.
[42] H. Kellerer, U. Pferschy, and D. Pisinger. 2004. *Knapsack Problems*. Springer, Berlin, Germany.
[43] Protocol Labs. 2017. *White paper – Filecoin: A Decentralized Storage Network*. Technical Report. https://filecoin.io/filecoin.pdf
[44] Florence Jessie MacWilliams and Neil James Alexander Sloane. 1977. *The theory of error-correcting codes*. Vol. 16. Elsevier.
[45] Dahlia Malkhi and Michael Reiter. 1997. Byzantine quorum systems. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. 569–578.
[46] Silvio Micali. 2016. ALGORAND: The Efficient and Democratic Ledger. *CoRR* abs/1607.01341 (2016). arXiv:1607.01341 http://arxiv.org/abs/1607.01341
[47] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 31–42.
[48] Moni Naor and Avishai Wool. 1998. The load, capacity, and availability of quorum systems. *SIAM J. Comput.* 27, 2 (1998), 423–447.
[49] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H Vaidya, and Zhuolun Xiang. 2020. Improved Extension Protocols for Byzantine Broadcast and Agreement. In *34th International Symposium on Distributed Computing*.
[50] Kamilla Nazirkhanova, Joachim Neu, and David Tse. 2021. Information dispersal with provable retrievability for rollups. *arXiv preprint arXiv:2111.12323* (2021).
[51] Kazuo Ohta and Tatsuaki Okamoto. 1999. Multi-signature schemes secure against active insider attacks. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 82, 1 (1999), 21–31.
[52] Michael O Rabin. 1983. Randomized byzantine generals. In *24th annual symposium on foundations of computer science (sfcs 1983)*. IEEE, 403–409.
[53] Michael O Rabin. 1989. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM (JACM)* 36, 2 (1989), 335–348.
[54] Mayank Raikwar and Danilo Gligoroski. 2022. SoK: Decentralized Randomness Beacon Protocols. In *Information Security and Privacy: 27th Australasian Conference, ACISP 2022, Wollongong, NSW, Australia, November 28–30, 2022, Proceedings*. Springer, 420–446.
[55] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.
[56] Victor Shoup. 2000. Practical threshold signatures. In *Advances in Cryptology—EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings 19*. Springer, 207–220.
[57] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. 2022. Bullshark: Dag bft protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2705–2718.
[58] Douglas R. Stinson and Reto Strobl. 2001. Provably Secure Distributed Schnorr Signatures and a (t, n) Threshold Scheme for Implicit Certificates. In *Proceedings of the 6th Australasian Conference on Information Security and Privacy (ACISP '01)*. Springer-Verlag, Berlin, Heidelberg, 417–434.
[59] Lei Yang, Seo Jin Park, Mohammad Alizadeh, Sreeram Kannan, and David Tse. 2022. DispersedLedger: High-Throughput Byzantine Consensus on Variable Bandwidth Networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 493–512.
[60] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. 347–356.

A APPLICATIONS OF WEIGHT RESTRICTION

Many distributed protocols, especially in the Byzantine corruption model and the blockchain setting, rely on distributed cryptographic primitives such as secret sharing [17, 55] and threshold signatures [11, 31, 58]. Furthermore, threshold signatures form the basis of the most commonly used protocol [19] for the problem of distributed random number generation (also known as threshold coin tossing, random beacon, or common coin), which, in turn, has numerous applications of its own, including many consensus and state machine replication protocols [6, 18, 27, 41]. However, most threshold cryptosystems are based on the idea of splitting a secret key into a number of discrete pieces such that any t out of n pieces are sufficient to reconstruct the secret key or perform operations on it (e.g., create a threshold signature). Therefore, simple weighted voting, as described in Section 1.2, cannot be applied to such systems.

A.1 Blunt Secret Sharing and derivatives

In cryptography, certain actions have an associated access structure $\mathbb{A}$ which determines all sets of parties that are able to perform these actions once they collaborate. Traditional $(n, k+1)$-threshold systems can be seen as a particular access structure where $\mathbb{A} = \{P \subseteq [n] : |P| > k\}$. Analogously, a *weighted* threshold access structure can be defined as $\mathbb{A} = \{P \subseteq \Pi : \sum_{i \in P} w_i > \beta \sum_{i \in \Pi} w_i\}$.

We can also define the *adversary structure* $\mathbb{F} \subseteq 2^{\Pi}$, the set of all sets of parties that can be simultaneously corrupted at any given execution. Often, the adversary structure is also defined via a threshold, with a maximum corruptible weight fraction α, i.e., $\mathbb{F}_w(\alpha) = \{P \subset \Pi : \sum_{i \in P} w_i < \alpha \sum_{i \in \Pi} w_i\}$.

While threshold access structures are commonly studied in cryptography and are applied in numerous distributed protocols, in practice, as we discuss in Section 6, it is often sufficient if the access structure provides the following two properties:

- There exists at least one set entirely composed of honest parties that belongs to the access structure. This typically guarantees the *liveness properties* of the accompanying protocol.
- Any set containing only corrupt parties does not belong to the access structure, as this would break *safety properties.*

Hence, we define a *blunt access structure* as follows:

Definition A.1 (Blunt access structure). Given a set of parties Π and the adversary structure $\mathbb{F} \subseteq 2^{\Pi}$, $\mathbb{A}$ is a blunt access structure w.r.t. $\mathbb{F}$ if $(\forall F \in \mathbb{F} : F \notin \mathbb{A})$ and $(\exists A \in \mathbb{A} : A \cap F = \emptyset)$.

The following theorem shows that solving WR is sufficient to implement weighted cryptographic protocols with blunt access structure by reduction to their nominal counterparts.

Theorem A.2. *Given a set of parties, a nominal threshold access structure protocol $\mathcal{P}$ with threshold $k \leq n/2$, we obtain a blunt threshold access structure w.r.t. a weighted threshold adversarial structure $\mathbb{F}_w(\alpha)$, assuming $\alpha < \frac{k}{n}$, by solving Weight Restriction with parameters $\alpha_w = \alpha$ and $\alpha_n = \frac{k}{n}$. This is accomplished by instantiating $\mathcal{P}$ with $\hat{n} = T$ virtual users and allowing party i to control t_i of them.*[4]

Proof. By definition of WR, once it distributes T tickets, the number of tickets (and, hence, virtual users) allocated to the corrupt parties will be less than $\alpha_n T = \frac{k}{\hat{n}}$. Hence, no element of the adversary structure shall appear in the resulting access structure. In addition, honest participants will receive more than $(1 - \alpha_n)T \geq \alpha_n T = \frac{k}{\hat{n}}$ tickets (and, hence, virtual users), ensuring that there exists a set consisting of only honest parties in the access structure. □

Note that all participants must agree on how many virtual users are assigned to each party as nominal protocols typically assume that the membership is common knowledge. To this end, it is sufficient for all parties to run an agreed upon *deterministic* weight-restriction protocol (e.g., Swiper).

Among other things, this way, one can obtain weighted versions of verifiable secret sharing [55], distributed random number generation [19], threshold signatures [11], threshold encryption [31],

[4]Recall that t_i is the number of tickets assigned to party i and T is the total number of tickets assigned by the solution to the weight reduction problem (in this case, to WR). See Section 2 for details.

and threshold fully-homomorphic encryption [39], all with blunt access structures. In the next section, we discuss how to do it for other access structures.

A.2 Tight Secret Sharing and derivatives

Although a blunt access structure is sufficient for a large spectrum of applications, more restrictive access structures are sometimes necessary as well. Here, we present a straightforward approach that involves just one extra round of communication to transform a blunt access structure into a weighted threshold access structure.[5] This means that our construction can be readily utilized in any protocol that already uses threshold cryptography without requiring significant redesign efforts. We showcase the transformation in Algorithm 1 using the particular example of a generic secret sharing scheme with threshold α.

Algorithm 1 Blunt to weighted access structure for party i

```
1: $\{t_1, \dots, t_n\} \leftarrow WR(\{w_1, \dots, w_n\}, f_w, f_n)$
2: $T \leftarrow \sum_{i=1}^{n} t_i$
3: **Operation** SHARE$(m, \{w_1, \dots, w_n\})$:   // Executed by dealer.
4:     $\{s_1^1, \dots, s_1^{t_1}, \dots, s_n^1, \dots, s_n^{t_n}\} \leftarrow (\lceil f_n T \rceil, T)\text{-}share(m)$
5:     $\forall j \in [n]$ : send $\langle$SHARES : $s_i^1, \dots, s_i^{t_i}\rangle$ to party $i$
6: **Operation** RETRIEVE:  // Executed by the parties.
7:     Send $\langle$REQUEST$\rangle$ to all parties
8: **Upon** receiving $\langle$REQUEST$\rangle$ from party $j$:
9:     // $w_{req}$ is initially 0
10:    $w_{req} \leftarrow w_{req} + w_j$
11:    **if** $w_{req} \geq \alpha \sum_{i=1}^{n} w_i$ **then**
12:        send shares received from the dealer to all parties
13: **Upon** receiving $\lceil f_n T \rceil$ shares:
14:    Reconstruct message $m$
```

In order to obtain the weighted access structure, the first step is to compute how many shares need to be dealt to each party by solving the Weight Restriction problem. Using the nominal secret sharing operations, we generate shares for the input message by treating each ticket as a "virtual user", setting the total number of shares to the total number of tickets, and using the parameter f_n to set the reconstruction threshold. The dealer then sends to each party the number of shares determined by the solution to WR.

In the reconstruction phase, each party keeps track of the total weight of parties that have requested the reconstruction. Once the established threshold is surpassed, the participants transmit the shares of the secret that they previously received from the dealer. Finally, the secret is reconstructed once the threshold is surpassed.

The correctness of this transformation comes from the fact that before parties corresponding to at least a fraction α of the total weight request the reveal of the secret, the honest parties do not send their shares, prohibiting the reconstruction of the secret as the shares of corrupted parties are not sufficient, by design, to perform this action. However, once this threshold is surpassed, all honest parties send their shares, and the secret is eventually retrieved, as their shares are, once again, by design, sufficient to surpass the threshold of the nominal secret sharing scheme.

[5]In fact, this can be further generalized to arbitrary access structures.

Beyond simple secret sharing, one can obtain weighted versions of the same set of protocols as was discussed at the end of Appendix A.1, but with arbitrary access structures.

A.3 Black-Box transformation

The same approach of allocating a number of virtual users according to the number of tickets as described in Appendix A.1 can be applied to arbitrary distributed protocols. Intuitively, a distributed protocol $\mathcal{P}$ with resilience f_n (f_w) in the nominal (weighted) model must solve the stated problem iff less than $f_n n$ parties (parties with weightless than $f_w W$) are corrupted. Hence, by applying the "virtual users" approach, we can essentially emulate the nominal model in the weighted one as long as $f_w < f_n$. However, as we demonstrate later in this section on the example of the Single Secret Leader problem, this approach has its limitations with respect to what kinds of distributed problems can be solved with it.

We illustrate the black-box transformation with two examples, showing first an example where it works smoothly and then an example where we have to slightly relax the problem statement for it to be applicable.

Linear BFT consensus. One of the major contributions of the Hotstuff protocol [60] was to achieve linear communication complexity BFT consensus. This result was achieved by designing the communication of the protocol in a star pattern, where each participant only communicates with the leader. Thus, in order for the leader to demonstrate that its proposal was accepted by a quorum of replicas, the protocol uses threshold signatures which guarantee that a valid signature can only be generated by combining at least $n - f$ shares. This guarantees that incompatible values are never both validated by a quorum since they shall intersect in at least $f + 1$ replicas and at least one of them will be correct, which cannot happen since honest replicas do not vote for different values.

In this case, we cannot apply either of the construction Appendix A.1 as we need a tight access structure for the threshold signature, nor can we apply the construction of Appendix A.2 as it would make the communication complexity quadratic whereas the main goal of Hotstuff is to keep it linear. However, what we can do is simply apply the virtual users approach in a black-box manner: pick any threshold $f_w < f_n$, run a deterministic WR protocol, and determine how many virtual identities should each party assume.

Single Secret Leader Election. SSLE [13] is a distributed protocol that has as an objective to select one of the participants to be a leader with an additional constraint that only the elected party knows the result of the election. Then, once the leader is ready to make a proposal, it reveals itself and other participants can then correctly verify that the claiming leader was indeed elected by the protocol.

The original paper contains nominal solutions for the protocol relying on ThFHE [14] and on shuffling a list of commitments under the DDH assumption. The authors initially suggest that their protocols could support weights by replicating each party to match their weights. As already discussed, this would create a huge overhead in the protocol for systems with large total weight.

Interestingly, in the original protocol, it is required for the election to be *fair*, that is, for the probability of each party being elected to be uniform. One could think however of an alternative formulation to the protocol where *chain-quaility* is required instead, where we might specify that the fraction of blocks produced by corrupt parties should not surpass f_n when the adversary might control a fraction of the weights up to f_w. In this case, we can thus simply apply WR with parameters $\alpha_w = f_w, \alpha_n = f_n$, immediately guaranteeing such a notion.

Properties such as *fairness* are one of the limitations of our transformations since any property that is a function of the weight of the parties may not be preserved after the transformation is applied. We discuss fairness in slightly more detail and speculate about possible fixes to this issue in Section 9.

B APPLICATIONS OF WEIGHT QUALIFICATION

B.1 Erasure-Coded Storage and Broadcast

Erasure-coded storage systems [20, 37, 50, 53, 59], also known under the names of Information Dispersal Algorithms (IDA) [53] and Asynchronous Verifiable Information Dispersal (AVID) [20], are crucial to many systems for space and communication-efficient, secure, and fault-tolerant storage. Moreover, as demonstrated in [20], they can yield highly communication-efficient solutions to the very important problem of asynchronous Byzantine Reliable Broadcast [15, 16], a fundamental building block in distributed computing that, among other things, serves as the basis for many practical consensus [27, 32, 41, 47, 57], distributed key generation [5, 30], and mempool [27] protocols.

The challenge of applying these protocols in the weighted setting is that (k, m) erasure coding, by definition, converts the original data into m discrete *fragments* such that any k of them are sufficient to reconstruct the original information. Thus, each party will inevitably get to store an integer number of these fragments, and the smaller m is, the more efficient the encoding and reconstruction will be. Moreover, for the most commonly used codes–Reed Solomon–the original message must be of size at least $k \log m$ bits. Hence, using a large m may lead to increased communication as the message may have to be padded to reach this minimum size. As we illustrate in this section, determining the smallest "safe" number of fragments to give to each party is exactly the Weight Qualification problem defined in Section 2.

Let us consider the example of [20] as it is the first erasure-coded storage protocol tolerating Byzantine faults. We believe Weight Qualification can be applied analogously to other similar works.

This protocol operates in a model where any t out of n parties can be malicious or faulty, where $t < \frac{n}{3}$. In other words, it has the nominal fault threshold of $f_n = \frac{1}{3}$. The protocol encodes the data using $(t + 1, n)$ erasure coding, and the data is considered to be reliably stored once at least $2t + 1$ parties claim to have stored their respective fragments. The idea is that, even if t of them are faulty, the remaining $t + 1$ parties will be able to cooperate to recover the data.

In order to make a weighted version of this protocol, instead of waiting for confirmations from $2t + 1$ parties, one needs to wait for confirmations from a set of parties that together possess more than a fraction $2f_w$ of total weight, where $f_w = f_n = \frac{1}{3}$. A subset of weight less than f_w of these parties may be faulty. Hence, for the protocol to work, it is sufficient to guarantee that any subset of total weight more than $2f_w - f_w = f_w$ gets enough fragments to reconstruct the data. To this end, we can apply the WQ problem with the threshold $\beta_w = f_w$. We can set β_n to be an arbitrary number such that $0 < \beta_n < \beta_w$. Then, we can use $(\lceil \beta_n T \rceil, T)$ erasure coding, where T is the total number of tickets allocated by the WQ solution. Hence, whenever a set of weight more than $2f_w$ of parties claim to have stored their fragments, we will be able to reconstruct the data with the help of the correct participants in this set. As for the rest of the protocol, it can be converted to the weighted model simply by applying weighted voting, as was discussed in Section 1.2.

As a result, we manage to obtain a weighted protocol for erasure-coded verifiable storage with the same resilience as in the nominal protocol ($f_w = f_n = \frac{1}{3}$). The "price" we pay is using erasure coding with a smaller rate (β_n instead of f_w), i.e., storing data with a slightly increased level of redundancy. However, note that β_n can be set arbitrarily close to f_w, at the cost of more total tickets and, hence, more computation.

Example instantiations. The communication and storage complexity of these protocols depends linearly on the rate of the erasure code. Using Reed-Solomon with Berlekamp-Massey decoding algorithm, the decoding computation complexity [34] is $O(m^2 \cdot \frac{M}{rm}) = O(\frac{m}{r} \cdot M)$, where M is the size of the message (which we do not affect), r is the rate of the code (in our case, $r = \beta_n$), and m is the number of fragments (in our case, the number of tickets allocated by the solution to the WQ problem). For the sake of illustration, let us fix β_n to be $\frac{1}{4}$. Then, the rate of the code used in the weighted solution will be $\frac{4}{3}$ times smaller than in the nominal solution. For the number of fragments m, let us substitute the upper bound from Corollary 2.3 ($m \le \left\lceil \frac{\beta_w(1-\beta_w)}{\beta_w - \beta_n} n \right\rceil$). For $\beta_w = \frac{1}{3}$ and $\beta_n = \frac{1}{4}$, $m \le \frac{8}{3} n$. Hence, the overall slow-down compared to the nominal solution is $\frac{8}{3} \cdot \frac{4}{3} \approx 3.56$.

One can also consider using FFT-based decoding algorithms [40]. Since the complexity of the FFT-based decoding depends only polylogarithmically on the number of fragments m, one can select the rate of the code ($r = \beta_n$) to be much closer to β_w and, thus, minimize communication and storage overhead.

Some protocols [49] are designed for higher reconstruction thresholds, which allows them to be more communication- and storage-efficient compared to [20]. For these cases, we will need to set $\beta_w := \frac{2}{3}$. By setting $\beta_n := \frac{1}{2}$ and applying the upper bound from Corollary 2.3, we will obtain the same reduction of factor $\frac{4}{3}$ in rate and 2 times fewer tickets: $m \le \frac{1/3 \cdot 2/3}{2/3 - 1/2} n = \frac{4}{3}$. The computational overhead will be $\frac{4}{3} \cdot \frac{4}{3} \approx 1.78$.

B.2 Error-Corrected Broadcast

The exciting work of [29] illustrated how one can avoid the need for complicated cryptographic proofs in the construction of communication-efficient broadcast protocols by employing error-correcting codes, thus enabling a better communication complexity when a trusted setup is not available. The protocol of [29] can be used for the construction of communication-efficient Asynchronous Distributed Key Generation [5, 30] protocols.

Similarly to erasure codes, error-correcting codes convert the data into m discrete fragments, such that any k of them are sufficient to reconstruct the original information. However, they have the additional property that the data can be reconstructed even when some of the fragments input to the decoding procedure are invalid or corrupted. Reed-Solomon decoding allows correcting up to e errors when given $k + 2e$ fragments as input.

The protocol of [29] tolerates up to t failures in a system of $n \ge 3t + 1$ parties (for simplicity, we will consider the case $n = 3t + 1$). Its key contribution is the idea of *online error correction.* Put simply, the protocol first ensures that:

- Every honest party obtains a cryptographic hash of the data to be reconstructed;
- Every honest party obtains its chunk of the data.

Then, in order to reconstruct a message, an honest party solicits fragments from all other parties and repeatedly tries to reconstruct the original data using the Reed-Solomon decoding and verifies the hash of the output of the decoder against the expected value. As the protocol uses $k = t + 1$ and $m = n$, after hearing from all $2t + 1$ honest and $e \le t$ malicious parties, it will be possible to reconstruct the original data (as $2t + 1 + e \ge k + 2e$, for $k = t + 1$).

To convert this protocol into the weighted model, it is sufficient to make sure that all honest parties together possess enough fragments to correct all errors introduced by the corrupted parties. To this end, we will apply the WQ problem. We will set β_w to the fraction of weight owned by honest parties, i.e., $\beta_w := 1 - f_w = \frac{2}{3}$ (where f_w will be the resilience of the resulting weighted protocol, $f_w = f_n = \frac{1}{3}$). However, it is not immediately obvious how to set β_n to allow the above-mentioned property.

If we want to use error-correcting codes with rate r, we need to guarantee that the fraction of fragments received by the honest parties (which is at least β_n) is at least $r + e$, where e is the fraction of fragments received by the corrupted parties. However, since honest parties get at least the fraction β_n of all fragments, then $e \leq 1 - \beta_n$. Hence, we need to set β_n so that $\beta_n \geq r + (1 - \beta_n)$. We can simply set $\beta_n := \frac{r}{2} + \frac{1}{2}$ for arbitrary $r < \frac{1}{3}$.

Example instantiation. For the sake of an example, we can set $\beta_w := \frac{2}{3}$, $r := \frac{1}{4}$ and $\beta_n := \frac{5}{8}$. Then, using the bound from Corollary 2.3, the number of tickets will be at most $\frac{2/3 \cdot 1/3}{2/3 - 5/8} \cdot n \leq \frac{16}{3} n$.

As was discussed above, for erasure codes, we can either use the Berlekamp-Massey decoding algorithm or the FFT-based approaches. The same applies to error-correcting codes. As most practical implementations use the former, we will focus on it. In this case, the communication overhead will be $\frac{r_n}{r_w}$, where $r_n = \frac{1}{3}$ is the rate used in the nominal protocol and r_w is the rate used for the weighted protocol (in the example above, $r = \frac{1}{4}$). The computation overhead is $\frac{r_n}{r_w} \cdot \frac{T}{n}$, where T is the number of tickets allocated by the WQ solution (in the example above, $T \leq \frac{16}{3} n$ in the worst case). Hence, for the example parameters, the worst-case computational overhead is $\frac{4}{3} \cdot \frac{16}{3} \approx 7.11$.

C EXACT SOLUTION TO WR USING MIXED INTEGER PROGRAMMING

The way we formulate *WR* in section 2.1 can be directly translated into an instance of bi-level optimization problem [26]. In such problems, we define an *upper level* optimization problem which contains another (lower-level) optimization problem in its constraints, namely:

$$\begin{aligned}
&\text{minimize} \sum_{i=1}^{n} t_i \\
&\text{subject to} \sum_{i=1}^{n} x_i t_i < \alpha_n \sum_{i=1}^{n} t_i \\
&\qquad\qquad \text{maximize} \sum_{i=1}^{n} x_i t_i \\
&\qquad\qquad \text{subject to} \sum_{i=1}^{n} w_i x_i < \alpha_w \sum_{i=1}^{n} w_i \\
&\qquad\qquad \sum_{i=1}^{n} t_i \geq 1 \\
&\qquad\qquad x_i \in \{0, 1\}, t_i \in \{0, 1, 2, \dots\}
\end{aligned}$$

The following theorem will be useful for simplifying this formulation and others we shall build.

Theorem C.1. *Minimizing the total number of tickets that the adversary can obtain in Weight Restriction is equivalent to minimizing the total number of tickets.*

Proof. Let T_A be the maximum number of tickets that the adversary can obtain in a solution of WR that distributes T tickets in total. Then, $T = \left\lfloor \frac{T_A}{\alpha_n} \right\rfloor + 1$. This stems from the fact that the problem requires $T_A < \alpha_n T \implies T > T_A / \alpha_n$. The minimum integer that satisfies this constraint is given by the expression above. Because this is an increasing function, the theorem holds. □

Theorem C.1 allows us to reformulate WR as a minimax problem:

$$\begin{aligned}
&\min_t \max_x \sum_{i=1}^{n} x_i t_i \\
&\text{subject to} \sum_{j=1}^{n} x_i t_i < \alpha_n \sum_{i=1}^{n} t_i \\
&\qquad \sum_{i=1}^{n} w_i x_i < \alpha_w \sum_{i=1}^{n} w_i \\
&\qquad \sum_{i=1}^{n} t_i \geq 1 \\
&\qquad x_i \in \{0,1\}, k_i \in \{0,1,2,\dots\}
\end{aligned}$$

A common method for solving minimax problems in MIP is to minimize a new variable that is greater or equal to all the feasible options, which eliminates in our case the variable x, but introduces $O(2^n)$ constraints to the problem, as the following:

$$\begin{aligned}
&\text{minimize } K_A \\
&\text{subject to } K_A < \alpha_n \sum_{i=1}^{n} k_i \\
&\qquad \forall I \subseteq [n] \text{ s.t. } \sum_{i \in I} w_i < \alpha_w W : \sum_{i \in I} k[i] \leq K_A \\
&\qquad \sum_{i=1}^{n} k_i \geq 1 \\
&\qquad x_i \in \{0,1\}, K_A, k_i \in \{0,1,2,\dots\}
\end{aligned}$$

We can replace the exponential constraints on every subset of weight less than $f_w W$ by a constraint on the Knapsack solution, as it will bound all feasible solutions. In order to do so, we can hard-code a dynamic programming by profits solution to the Knapsack problem into the constraints. Unfortunately, the resulting MIP still has a lot, albeit only a polynomial number, of constraints and, thus, is prohibitively slow for inputs of size larger than a couple of dozens.

D EXPERIMENT RESULTS IN THE OTHER BLOCKCHAINS

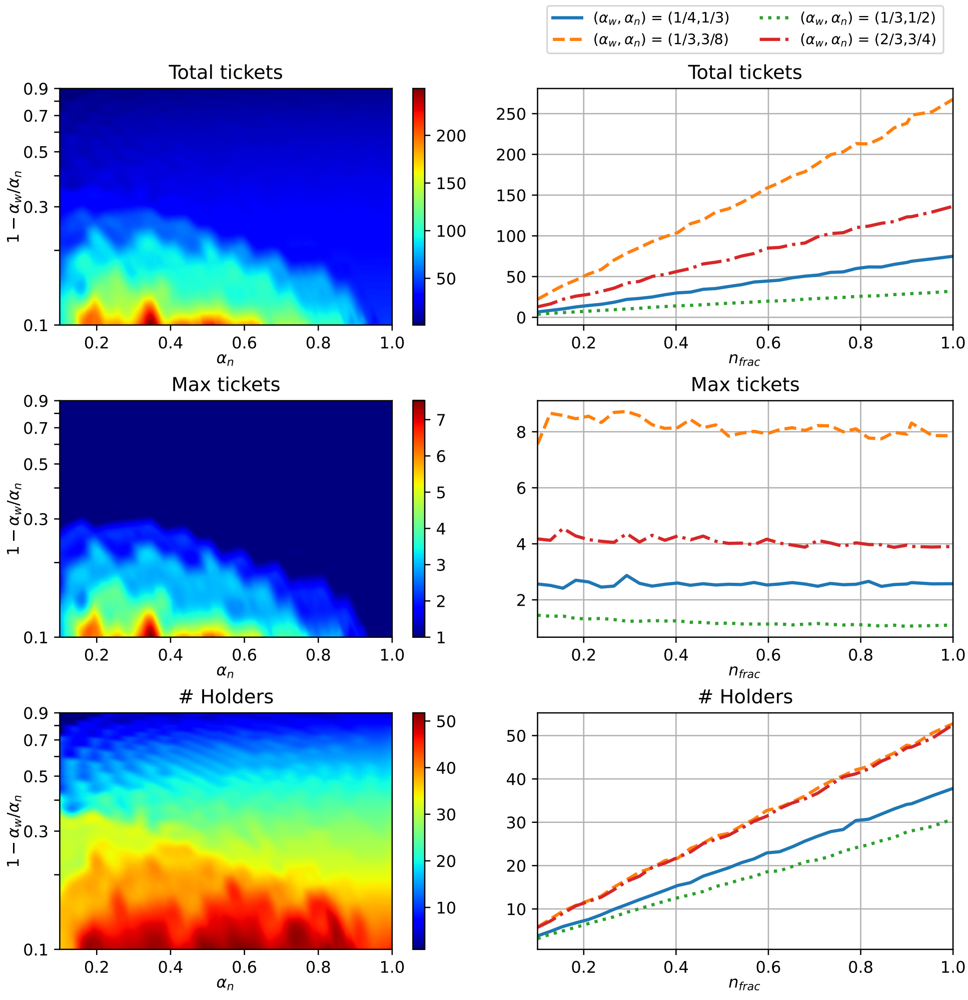

Fig. 2. Experiment results using Aptos (200 samples per data point)

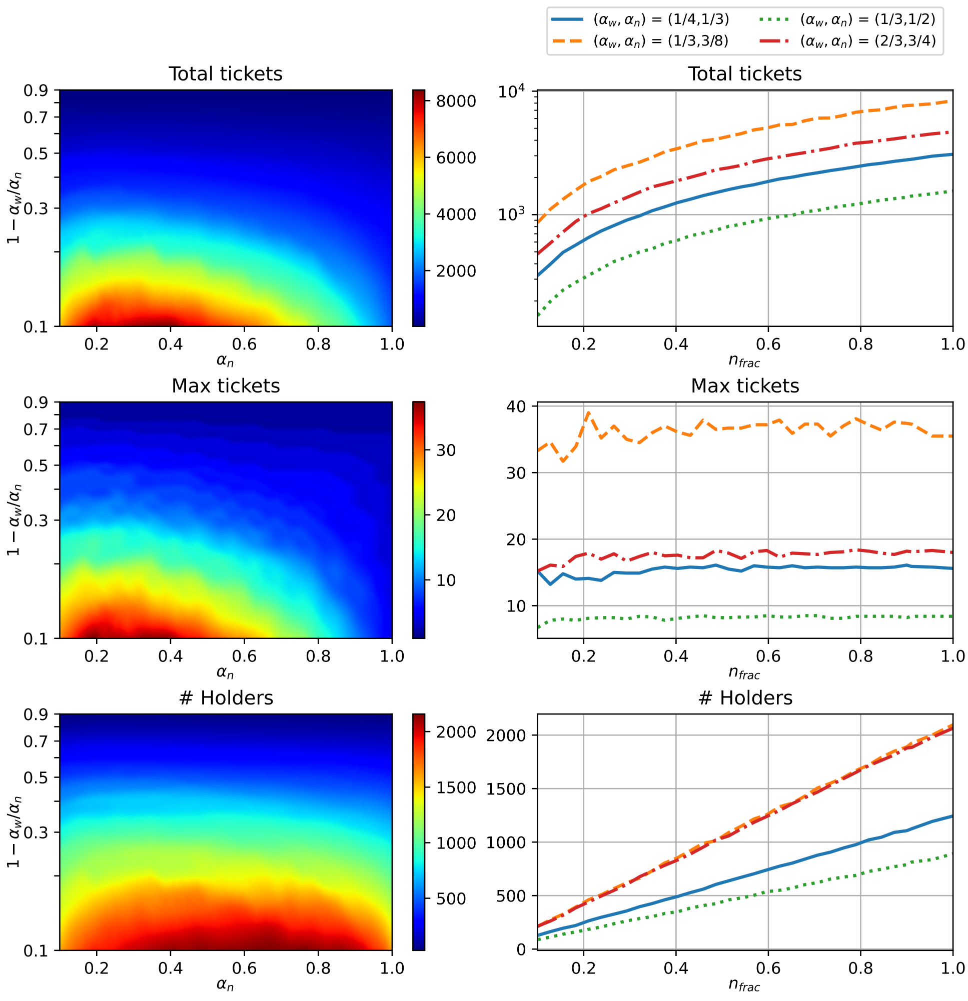

Fig. 3. Experiment results using Filecoin (30 samples per data point)

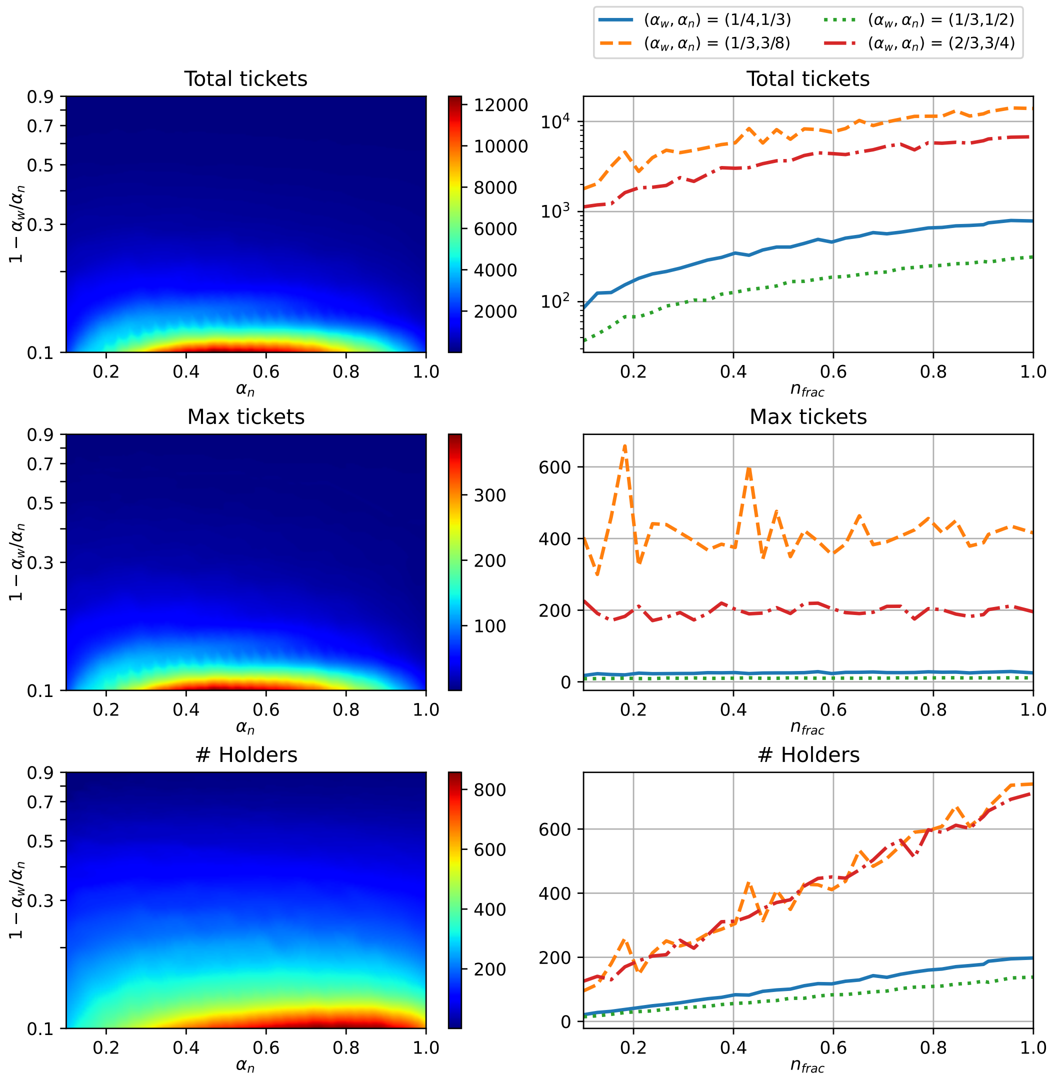

Fig. 4. Experiment results using Algorand (50 samples per data point)