OPTIMIZING EXIT QUEUES FOR PROOF-OF-STAKE BLOCKCHAINS: A MECHANISM DESIGN APPROACH

MICHAEL NEUDER[A], MALLESH M PAI[B], AND MAX RESNICK[C]

ABSTRACT: Byzantine fault-tolerant consensus protocols have provable safety and liveness properties for static validator sets. In practice, however, the validator set changes over time, potentially eroding the protocol's security guarantees. For example, systems with accountable safety may lose some of that accountability over time as adversarial validators exit. As a result, protocols must rate limit entry and exit so that the validator set changes slowly enough to ensure security. Here, the system designer faces a fundamental trade-off: the harder it is to exit the system, the less attractive staking becomes; the easier it is to exit the system, the less secure the protocol will be.

This paper provides the first systematic study of exit queues for Proof-of-Stake blockchains. Given a collection of validator-set consistency constraints imposed by the protocol, the social planner's goal is to provide a constrained-optimal mechanism that minimizes disutility for the participants. We introduce the `MINSLACK` mechanism, a dynamic capacity first-come-first-served queue in which the amount of stake that can exit in a period depends on the number of previous exits and the consistency constraints. We show that `MINSLACK` is optimal when stakers are homogeneous in their need for liquidity. When stakers are heterogeneous, the optimal mechanism resembles a priority queue with dynamic capacity. However, this mechanism must reserve exit capacity for the future in case a staker with a more urgent need for liquidity arrives. We conclude with a survey of known consistency constraints and highlight the diversity of existing exit mechanisms.

[A] ETHEREUM FOUNDATION
[B] SPECIAL MECHANISMS GROUP AND RICE UNIVERSITY
[C] SPECIAL MECHANISMS GROUP

1. INTRODUCTION

In Proof-of-Stake networks, validators must stake tokens to participate in the consensus protocol. These staked tokens serve two purposes. First, they solve the problem of Sybil resistance: agents who operate two validators must procure twice as much stake as those who only manage one. Second, they allow the protocol to hold validators accountable for violating the predefined rules. A validator's stake is held in escrow and can be *slashed* if adversarial behavior is detected, providing *crypto-economic security* to the system.[1] Most modern blockchains (e.g., Ethereum and Solana) implement a version of Proof-of-Stake and the principles of staking have been extended beyond base-layer chains and into the smart contract layer (e.g., re-staking as popularized by EigenLayer).

The literature typically treats the set of stakers as static to establish positive results; however, in practice, staking protocols have a validator set that changes over time. New agents may arrive and wish to stake, while existing stakers may want to withdraw their tokens for use elsewhere (see, e.g., Chitra (2021)). But how should a stake-based protocol design this egress procedure?[2] There are two competing desiderata. The first is the security of the underlying protocol. For example, if a malicious validator can corrupt the service for personal gain but then withdraw their stake before the corruption is detected, the validator is immune to punishment, and the protocol is not secure. We describe these concerns in more detail in Section 6. The second is ensuring that validators can quickly enter and exit the system since delays decrease the utility of participation. Offering fast withdrawals also indirectly benefits the protocol since, *ceteris paribus*, a more rigid protocol must offer higher rewards in the form of emissions to compensate users for the decrease in their optionality. While these general principles are well understood, the design of the exit procedures in the context of blockchains has not yet attracted much formal attention.[3] As a consequence, the optimal queue designs we suggest in Sections 4 and 5 perform much better than those currently used in practice, which we survey in Section 6.

The first contribution of this paper is to formally define the designer's dilemma as a constrained optimization problem: minimizing the adverse effects of withdrawal delays while satisfying the protocol's safety constraints. In the setting where all validators have the same time sensitivity, we show that a stateful, first-come-first-served queue where the amount of stake withdrawn in each period depends on the history of previous periods is constrained optimal.

---

[1]See Deb, Raynor, and Kannan (2024) for an extended definition of crypto-economic security.

[2]Similar considerations apply to the design of deposit (ingress) procedures; we focus on withdrawals (egress) in the present paper.

[3]We defer a discussion of the related literature to Section 2.

However, even among honest validators, the desire to exit can be heterogeneous—for example, a capital-constrained validator might need to withdraw urgently to meet a margin call elsewhere. In this setting, a first-come-first-served queue is no longer optimal, because time-sensitive stakers incur a higher penalty from delay, so allowing them to pay to cut the line increases aggregate welfare. Therefore, the optimal mechanism processes transactions based on their willingness to pay for priority. Further, in some cases, the optimal mechanism reserves capacity for the future in case more time-sensitive agents arrive. We formally define this mechanism as the solution to a Markov Decision Process (MDP) and show that an appropriately defined dynamic Vickrey-Clark-Groves (VCG) mechanism can implement the efficient outcome.

We complement these results with a survey to connect our theoretical model to practice. First, we discuss the exit mechanisms in use today by popular blockchains. Our results suggest that some of these mechanisms are either (highly) sub-optimal or the designers believed the mechanism should satisfy additional constraints external to our model. Further, we should note that no protocol that we are aware of uses a payment-for-priority mechanism.[4] Combined with our theoretical results, this collection may be helpful for blockchains and staking protocols more generally to design or improve their exit procedures.

Organization. Section 2 presents the related literature. Section 3 defines the model and outlines the form of the security constraints of a staking system. Section 4 studies the common-value setting by defining `MINSLACK` and proving its optimality. Section 5 introduces heterogeneous values, formalizes the extended problem as an MDP, and presents numerical results quantifying the performance of various algorithms. Section 6 justifies the form of the constraints, presents the intricacies of the Ethereum design, and surveys other staking withdrawal procedures. Section 7 concludes.

2. LITERATURE REVIEW

Early in the development of Ethereum, Vitalik outlined concerns about a long-range attack on a Proof-of-Stake blockchain (Buterin, 2014). In particular, he described the potential "nothing-at-stake" attack, in which a malicious staker withdraws his `ETH` while building a competing fork starting from a historical epoch before he withdrew. This way, the staker cannot be punished for creating the fork because he has exited from the consensus mechanism. The solution, he argued, was *weak subjectivity*, where nodes locally store a subjectivity "checkpoint" block and ignore any messages from before that epoch (Ansgaonkar, 2020). Weak-subjectivity checkpoints prevent long-range attacks but require

---

[4]Priority payment is standard in other congested parts of blockchains, notably in the context of transaction fees, and a substantial literature explores the design of such fees, see, e.g., Roughgarden (2020); Huberman, Leshno, and Moallemi (2021).

the validator set to change slowly enough to reach a subjectivity checkpoint without a nothing-at-stake attack. The naïve approach simply delays *all* withdrawals for the weak-subjectivity period, guaranteeing the chain's safety. Buterin argued in Buterin (2018) that this imposed too strict a penalty on validators who wanted to withdraw under normal circumstances when there was no evidence of an attack, arguing for an exit queue model instead. The simple first-come-first-served (FCFS) exit queue proposed allowed validators to "withdraw very quickly in 'the normal case' but not during attacks". Buterin (2019a) elaborated on this point and fleshed out the arguments for an FCFS exit queue with a fixed number of exits per epoch (as a fraction of the total stake). Buterin (2019b) gave a formal case for why the consistency imposed by the exit queue was enough to safely last until the next weak-subjectivity checkpoint through an inductive argument. As detailed in Section 6, the FCFS exit queue has been used in Ethereum since April 2023, when the Shanghai/Capella hard fork enabled beacon chain validators to withdraw. More generally, Buterin was concerned with preserving the formal property known as *accountable safety* (Buterin and Griffith, 2017). Accountable safety guarantees that in the case of a safety violation, the fault is attributable to a subset of validators (because they must have signed conflicting attestations).

Lewis-Pye and Roughgarden (2024); Budish, Lewis-Pye, and Roughgarden (2024) formalized the economic limits of consensus mechanisms and showed that no partially synchronous protocol can fully implement slashing rules without bounding the resolution time of communication between honest nodes. Given some bound on this overhead, protocols could implement slashing against an attacker with $< 2/3$ of the total stake, a positive result that justifies using the weak-subjectivity period as a heuristic for preventing nothing-at-stake attacks. Neu, Tas, and Tse (2023) formalized the relationship between accountable safety and finality, while Asgaonkar, D'Amato, Saltini, Zanolini, and Zhang (2024) introduced a new confirmation rule for potentially improving the pre-finality guarantees for transactions in Ethereum. Kyne (2023) proposed allowing some withdrawals to be processed ahead of others by the nature of originating from a different source that required payment.

Systematic attention to the design of exit procedures in blockchains has been sparse; however, mechanism design has proved useful for blockchain designers in other contexts, particularly in designing transaction-fee mechanisms. The question here is similar: if there is a finite supply of block space and demand may exceed supply, how should the block space be allocated? Bitcoin used a simple 'pay-as-bid' mechanism, which was fruitfully studied using tools from queueing theory in Huberman, Leshno, and Moallemi (2021). Pay-as-bid mechanisms result in strategic bidding, contributing to poor user experience. In 2021, Ethereum adopted a dynamic reserve price mechanism, EIP-1559, which was comprehensively studied in the seminal Roughgarden (2020) (see also Roughgarden

(2021)). There has been recent interest in studying dynamic mechanism design in this setting— see e.g., Nisan (2023); Pai and Resnick (2023). In a different context, a few market design papers have studied the design of queues, mainly in organ transplantation–see, e.g., Leshno (2022) and Su and Zenios (2006).

3. MODEL

Time is discrete, and each period corresponds to a point during which a validator may request a withdrawal and be removed from the active set — for example, Ethereum processes withdrawals at epoch boundaries. Denote the set of possible validators, V, and at each time t, let $S(v,t) \in \{0,1\}$ denote whether validator v is currently staked or not.[5] Thus the total amount staked at period t is $\overline{S}(t) = \sum_v S(v,t)$.

At the end of each period, any validator may signal their desire to withdraw their stake by joining a waiting list $W(t)$. For now, we model every element of the waiting list as a tuple (v,t'), where v is the validator identity and t' is the period at which they initiated their withdraw. Note that we must have that $t' \leq t$, as any element in the waiting list must have joined in the past. Let $R(t)$ be the set of exit requests arriving in period t.[6]

An exit mechanism $\mathcal{M}$ in each period t, given a waiting list $W(t)$, selects a subset $P(t) \subseteq W(t)$ of withdrawal requests to *process*. We allow the exit mechanism's choices to depend on past choices. Formally, let us define a history of previous withdrawal requests as:

$$H(t) = (P(1), P(2) \ldots, P(t-1)),$$

and the set of all possible histories as $\mathcal{H}$. A mechanism then formally is:

$$\mathcal{M} : \mathcal{H} \times W(t) \mapsto \{0,1\}^{|W(t)|},$$

where the binary string is an indicator function for the withdrawals processed during each period. The system follows the rules of motion:

$$W(t) = W(t-1) \setminus P(t-1) \cup R(t),$$
$$P(t) = \mathcal{M}(H(t), W(t)),$$
$$H(t+1) = H(t) \cup P(t).$$

The stake distribution $S(\cdot, t)$ is then updated based on the exits and fresh entries. In words, $W(t)$ is the *waiting list* of withdraw requests at the *beginning* of period t, $P(t)$ is the subset of waiting to withdraw requests that are *processed* in period t. $H(t)$ is the *history* of processed requests up to and including period t.

---

[5]For simplicity, we assume that each validator has the same quantity of tokens staked, normalized to 1. This can easily be relaxed.

[6]Most blockchains also limit entry to have a stable validator set for consensus. In this paper, we focus on the design of exit queues and consider entry unrestricted.

The number of withdrawals allowed over various time horizons constrains the protocol designer. We model this as a finite set of constraints, each described by a tuple $(\delta, T) \in [0,1] \times \mathbb{N}$. A constraint of (δ, T) means that if, in any period t, the total stake is $\overline{S}(t)$, then the maximum number of withdrawals processed over the following T periods (from $t + 1$ thru $t + T$) is bounded above by $\delta \times \overline{S}(t)$. We take the constraints as given, motivating this construction in Section 6.1. Formally, the designer faces some k constraints given by $\mathcal{C} = \{(\delta_1, T_1), \dots, (\delta_k, T_k)\}$ and aims to maximize the utility of the validators withdrawing from the staking system. Calculating this utility depends on validators having differing values for exiting the system; we begin by examining the simplest case, where each validator has a common value.

4. HOMOGENEOUS VALUES

To begin our analysis, we consider the case where all agents have the same value for withdrawing or equivalently face the same economic penalty for each period between when they make a withdrawal request and when that request is fulfilled. In this case, the social planner cannot increase efficiency by reordering withdrawal requests, so efficiency demands that exit requests be processed as quickly as possible without violating the established constraints.

Given the constraint set $\mathcal{C}$, the following algorithm, which we call MINSLACK, greedily processes the maximum amount of withdrawals allowed within the bounds of the constraints. In other words, for every constraint (δ_i, T_i), calculate the difference between $\delta_i \overline{S}(t - T_i)$ and the amount withdrawn in periods $t - T_i$ thru $t - 1$.[7] This difference is the maximum number of withdrawals constraint i allows in period t given the previous history. It follows that the lowest of these slacks is the maximum amount that can withdrawn in this period without violating constraints. Since the protocol is indifferent about the withdrawal order, if there is more demand for withdrawing than the allowed quantity, a natural solution is to use the FCFS rule to tie-break. We present this algorithm as Algorithm 1.

Proving that this algorithm is feasible and optimal is straightforward: as designed, it processes the maximum amount allowed by the protocol constraints, but never more. Before presenting this result, we explain why such a queue design may be helpful. As we describe in Section 6.2, the relevant constraints are that a given fraction of stake cannot withdraw over an extended period (e.g., $\mathcal{O}(\text{weeks})$). Nevertheless, the actual queue implemented on Ethereum allows the withdrawal of at most eight validators per epoch (a value set in EIP-7514, Lion and Bieko 2023). In practice, validators must wait longer than

---

[7]For expositional simplicity, we elide over the difficulties caused by the fact that $\delta_i \overline{S}(\cdot)$ may not be a whole number. In what follows we implicitly assume that this is a whole number, alternately, we could allow for fractional withdrawals at the cost of significantly messier notation.

Algorithm 1 `MINSLACK`

1: **Input:** Constraints $\mathcal{C} = \{(\delta_1, T_1), \ldots, (\delta_k, T_k)\}$.
2: **Input:** Initial staking $S(\cdot, 0)$.
3: $\overline{S}(0) \leftarrow \sum_v S(v, 0)$.
4: **Initialize:** $H(0), W(0), P(0) \leftarrow$ `NULL`.
5: **Initialize:** $\overline{P}(0) = 0$.
6: **for** each period $t \geq 1$ **do**
7:    $W(t) \leftarrow W(t-1) \setminus P(t-1) \cup R(t)$.
8:    **for** each constraint $i \leq k$ **do**
9:       `SLACK`$_i \leftarrow \delta_i \overline{S}(t - T_i) - \sum_{\tau=t-T_i+1}^{t-1} \overline{P}(\tau)$.
10:    **end for**
11:    `MINSLACK` $\leftarrow \min\{$`SLACK`$_i : 1 \leq i \leq k\}$.
12:    $P(t) \leftarrow$ Largest prefix of W(t) such that total withdrawn $\leq$ `MINSLACK`
13:    $\overline{P}(t) \leftarrow$ Total withdrawn in $P(t)$
14:    $H(t+1) \leftarrow H(t) \cup P(t)$
15:    **Update:** $S(v, t)$ based on $P(t)$.
16: **end for**

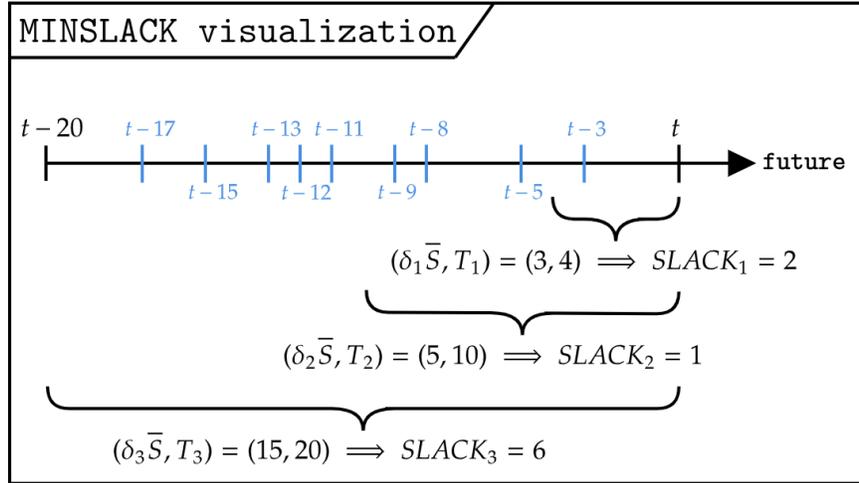---

FIGURE 1. A visual example of the calculation of `SLACK`$_i$ used in Algorithm 1 (`MINSLACK`). The example constraints $\mathcal{C} \implies \{(3, 4), (5, 10), (15, 20)\}$ are read as, e.g., $(3, 4) \implies$ "at most three withdrawals over the next four consecutive time steps." In the diagrammed example, the blue vertical lines represent the timestamps of processed withdrawals. With `SLACK`$_2 = 1$, the `MINSLACK` algorithm can process at most one withdrawal during the current period while still conforming to the constraints.

required during periods with higher-than-expected withdrawals. For example, in January 2024, the withdrawal queue on Ethereum rose to about 16,000 validators or about 5.5 days at peak due to Celsius bankruptcy proceedings.[8] However, there were about 900k

---

[8]See https://www.validatorqueue.com/ for historical data about the withdrawal queue.

total validators during this period, so processing these withdrawal requests immediately would not have violated the consistency constraints defined by the "weak-subjectivity period" (Ansgaonkar, 2020). With this motivation, we present a formal treatment of `MINSLACK`.

THEOREM 1. *Given any sequence of withdrawal requests $R(\cdot)$, let $P(\cdot)$ be the processed withdrawal requests and $\overline{P}(\cdot)$ be the resulting total amount withdrawn in each period by Algorithm 1. Then:*

    (1) ***Feasibility:*** *$P(\cdot)$ is feasible with respect to the protocol constraints.*

    (2) ***Optimality:*** *For any other feasible withdrawal decisions with total withdrawn in each period given by $\overline{P}'(\cdot)$, it must be the case that:*

$$\forall t \geq 1 : \sum_{\tau=1}^{t} \overline{P}'(\tau) \leq \sum_{\tau=1}^{t} \overline{P}(\tau). \tag{1}$$

PROOF. To show that the withdrawal resulting from `MINSLACK` is feasible, observe that in each period, the withdrawal amount is less than $\min\{\texttt{SLACK}_i : 1 \leq i \leq k\}$ so it necessarily satisfies all of the constraints. Since each withdrawal satisfies the constraints given the history, applying the algorithm always results in a history that is feasible by construction.

For optimality, consider for the sake of contradiction that there exists a feasible $\overline{P}'(\cdot)$ that violates (1). Let t be the earliest time such that:

$$\sum_{\tau=1}^{t} \overline{P}'(\tau) > \sum_{\tau=1}^{t} \overline{P}(\tau).$$

Since t is the earliest time to violate condition (1), we must have that for all $t' < t$,

$$\forall t' < t : \quad \sum_{\tau=1}^{t'} \overline{P}'(\tau) \leq \sum_{\tau=1}^{t'} \overline{P}(\tau). \tag{2}$$

Analogous to the algorithm, let us term $\texttt{SLACK}'_i(\cdot)$ as the maximum withdrawable amount in a given period given process $P'(\cdot)$, with $\texttt{MINSLACK}'(\cdot)$ defined as the smallest constraint $i \in 1, \ldots k$. Note that by feasibility, we must have the following:

$$\overline{P}'(t) \leq \texttt{MINSLACK}'(\cdot).$$

Conversely, we know that by construction (see Algorithm 1),

$$\overline{P}(t) = \texttt{MINSLACK}(t).$$

For the contradiction, it is sufficient to show that

$$\texttt{MINSLACK}'(t) - \texttt{MINSLACK}(t) \leq \sum_{\tau=1}^{t-1} \overline{P}(\tau) - \sum_{\tau=1}^{t-1} \overline{P}'(\tau). \tag{3}$$

In other words, we must show that the additional slack available at time t under P' relative to P is, at most, the difference between the amount withdrawn up to time $t-1$ under P than P'. Feasibility of the withdrawals under P' then contradicts the claim that $\sum_{\tau=1}^{t'} \overline{P}'(\tau) \leq \sum_{\tau=1}^{t'} \overline{P}(\tau)$. To see (3) it is sufficient to show that for each $1 \leq k$:

$$\text{SLACK}_i'(t) - \text{SLACK}(t) \leq \sum_{\tau=1}^{t-1} \overline{P}(\tau) - \sum_{\tau=1}^{t-1} \overline{P}'(\tau). \tag{4}$$

The left-hand side of (4) can be rewritten as:

$$
\begin{aligned}
\text{SLACK}_i'(t) - \text{SLACK}(t) &= \delta_i \overline{S}'(t - T_i) - \sum_{\tau=t-T_i+1}^{t-1} \overline{P}'(\tau) - \left(\delta_i \overline{S}(t-T_i) - \sum_{\tau=t-T_i+1}^{t-1} \overline{P}(\tau) \right) \\
&\leq \left(\sum_{\tau=1}^{t-T_i} \overline{P}(\tau) + \sum_{\tau=t-T_i+1}^{t-1} \overline{P}(\tau) \right) - \left(\sum_{\tau=1}^{t-T_i} \overline{P}'(\tau) - \sum_{\tau=t-T_i+1}^{t-1} \overline{P}'(\tau) \right) \\
&= \sum_{\tau=1}^{t-1} \overline{P}(\tau) - \sum_{\tau=1}^{t-1} \overline{P}'(\tau).
\end{aligned}
$$

where the penultimate inequality follows since $\delta_i \in [0,1]$, and we have (2). ∎

Thus, MINSLACK is optimal for the common value setting. Still, in reality, stakers may have disparate values for accessing their stake, motivating the need to explore how a withdrawal mechanism could account for heterogeneous values.

5. HETEROGENEOUS VALUES AND PAYING FOR PRIORITY

While Theorem 1 shows that Algorithm 1 provides an optimal solution for the case when all stakers have a homogeneous value for withdrawing, in reality, they may have different values for getting access to their staked assets. A staking pool, for example, might be withdrawing some validators gradually to rotate the cryptographic keys used for participating in consensus. In this case, the pool has a relatively low value for their withdrawal because the underlying reason to withdraw is not highly time-sensitive. On the other hand, a hedge fund trying to withdraw staked capital in time to meet a margin call to avoid a forced liquidation may have an extremely high value for the liquidity from the withdrawal processing.

Each validator looking to exit has a delay cost per unit time c. The net payoff of a validator of type c whose withdrawal occurs after a delay Δ for a price (bid) of b is:[9]

$$U(\Delta, b, c) = -c\Delta - b.$$

___

[9]This is a standard model in the context of transaction fees, where users face a similar trade-off between paying for inclusion and suffering a delay—see, e.g., Huberman, Leshno, and Moallemi (2021).

In other words, their utility is linear in time according to their per-period disutility of waiting, less the amount they pay. We consider this canonical linear form for simplicity. More generally, one can consider other forms for the utility, including the time-varying disutility of waiting, see, e.g., Bergemann and Välimäki (2010).

As described below, the efficient mechanism will be more complicated: efficiency requires agents to express their disutility of waiting in the mechanism and managing agents' incentives involve payments. As in the previous section, we will consider the planner's objective to be efficiency, which is defined formally below.

OBSERVATION 1. *When values are heterogeneous, Algorithm 1, `MINSLACK`, may not be efficient.*

Recall that in every period, `MINSLACK` greedily processes as many withdrawal requests as possible, given the constraints. However, there are unknown future withdrawal requests at the time of processing. With heterogeneous values, it is possible that highly time-sensitive stakers with a high disutility of waiting may arrive in future periods. Suppose the current withdrawal requests have very low time sensitivity (i.e., very low c). In that case, the optimal behavior could be to withhold processing withdrawals in this period and reserve this capacity for the future. Intuitively, an efficient withdrawal mechanism must balance between processing withdrawals now while reserving some slack for hypothetical future withdrawals.

5.1. *Efficient withdrawals under heterogeneity*

This section describes a withdrawal algorithm based on the Vickrey-Clarke-Groves (VCG) mechanism. VCG in such dynamic settings is not novel — see Parkes and Singh (2003) or Lavi and Nisan (2000); it generalizes the second-price sealed-bid auction in static settings and has two desirable properties, namely, (i) it is incentive compatible for each agent to report their cost, c, and (ii) the mechanism is constrained-efficient.

As is standard in mechanism design, we first describe the efficient allocation rule, i.e., the optimal rule for a planner, in a setting where *the planner observes the delay costs of stakers as they arrive*. Then, we describe payment rules that make it *incentive compatible* for stakers to report their values truthfully.

Since this is a dynamic setting, as alluded to above, the mechanism must have a forecast of future arrivals to decide whether to process withdrawals or to reserve withdrawal slots for future arrivals. In this section, therefore, we assume that there is a known stochastic process behind the withdrawals. The number of withdrawal requests in each period is randomly distributed (for example, this may be the Poisson distribution with known parameter λ). Each withdrawal request has a type that is an i.i.d. draw according to a known probability distribution on $\mathbb{R}_+$.

5.1.1. *The Efficient Allocation Rule.* For now, suppose each agent truthfully states, at the time of joining the waiting list, their private cost, c. Each element of the waiting list is now a 4-tuple (v, s, t', c). Modulo this change, however, the system can be described just as in Section 3.

Given that some set of withdrawal requests $P(t)$ is processed in period t from a waiting list of $W(t)$, the system collects a penalty (net disutility) of:

$$\texttt{Penalty} = \sum_{(v,s,t',c) \in W(t) \setminus P(t)} sc.$$

In other words, the planner in period t collects a penalty equal to the disutility cost of every staker in the waiting list whose withdrawal is not processed. As before, the planner faces some constraints $\mathcal{C}$ on exits. The system aims to minimize expected discounted penalties over feasible exit plans, where $\rho \in [0, 1]$ is the planner's discount rate.

This is a dynamic program where the state of the problem at the beginning of period t is $(S(t), W(t), H(t-1))$. We can recursively define the value function of the planner as follows:

$$V(S(t), W(t), H(t-1)) \equiv \tag{5}$$

$$\min_{P(t)} \left(\sum_{(v,s,t',c) \in W(t) \setminus P(t)} sc + \delta \mathbb{E}[V(S(t+1), W(t) \setminus P(t) \right.$$

$$\left. \cup R(t+1), H(t) \cup P(t)] \right),$$

$$\text{s.t. } P(t) \subseteq W(t),$$

$$P(t) \text{ feasible wrt } \mathcal{C}.$$

Here, expectations are taken over the next period withdrawal requests $R(t+1)$: both the number of withdrawal requests and the corresponding waiting disutility is unknown at period t.

This framing is an infinite horizon Markov Decision Problem (MDP). Given the previous history, there is a maximum number of feasible withdrawals in every period. For any withdrawal processed from the waiting list, it is intuitive that the planner will remove the ones with the highest disutility of waiting first. However, as described above, the marginal value of holding onto a withdrawal slot can exceed the penalty of making a current staker on the list wait an extra period. Of course, the precise details depend on the arrival process and the system's current state. The algorithm is described in Program 5.

11

Algorithm 2 `OPTIMAL`

1: ... {same as `MINSLACK`}
2: **for** each period $t \geq 1$ **do**
3:    $W(t) \leftarrow W(t-1) \setminus P(t-1) \cup R(t)$.
4:    P(t) ← Solution of Program 5
5:    $\overline{P}(t) \leftarrow$ Total withdrawn in P(t)
6:    H(t+1) ← H(t) ∪ P(t)
7:    **Update:** S(v,t) based on P(t), E(t).
8: **end for**

`OPTIMAL` is nearly identical to `MINSLACK`; it only replaces the process for calculating the set of withdrawals to process in a current period, $P(t)$ (shown in brown text in Algorithm 2). The optimization problem in Program 5 must be solved to determine the policy of how many withdrawals to process at each period.

Algorithm 3 `PRIO-MINSLACK`

1: ... {same as `MINSLACK`}
2: Sort W(t) in decreasing order of waiting disutility.
3: **for** each constraint $i \leq k$ **do**
4:    $\text{SLACK}_i \leftarrow \delta_i \overline{S}(t - T_i) - \sum_{\tau = t - T_i + 1}^{t-1} \overline{P}(\tau)$.
5: **end for**
6: $\text{MINSLACK} \leftarrow \min\{\text{SLACK}_i : 1 \leq i \leq k\}$.
7: ... {same as `MINSLACK`}

Another candidate withdrawal algorithm, which we refer to as `PRIO-MINSLACK`, modifies `MINSLACK` to process withdrawals in order of priority fees. Algorithm 3 represents this one-line change in blue text.

5.1.2. *Pricing Rule.* So far, we have described the problem as an optimization problem where the planner *knows* the disutility from waiting suffered by the stakers in the queue. These are private, and there must be an incentive for stakers to report truthfully. Achieving this is straightforward (albeit computationally inefficient): every staker withdrawn in a period t should pay the expected delay costs imposed on the system by their presence. Existing theorems (see Parkes and Singh (2003), Bergemann and Välimäki (2010)) show that such a pricing rule results in a Bayes-Nash equilibrium, where each buyer reports their values truthfully.

5.1.3. *Optimal policy.* If new withdrawal arrival and value distributions are known, we can calculate the optimal withdrawal policy by solving the resulting MDP associated with Program 5.
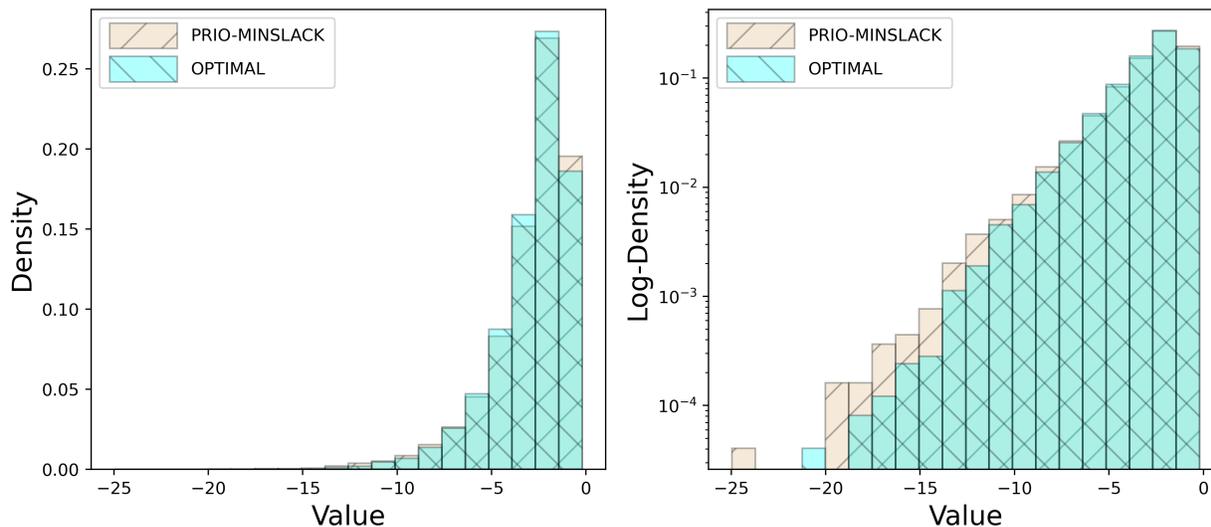
Overlaid Histograms of Policy Values

FIGURE 2. Performance comparison of `PRIO-MINSLACK` and `OPTIMAL` over 10,000 samples calculating the discounted reward following each policy from the initial state $s_0 = [0,0,0,0,0,0]$ for 350 steps with a discount factor of 0.9. The density of each histogram shows the probability a given trial ends in that range of values. When examining the raw density, the performance seems comparable, but the Log-Density plot demonstrates that the long tail performance of `PRIO-MINSLACK` is significantly worse than `OPTIMAL`. Intuitively, `PRIO-MINSLACK` is more of a "gambler" – the algorithm takes big risks by greedily processing as fast as possible. These risks are rewarded in the median case but occasionally have large disutilities by burning the capacity on low-value withdrawals. See Table 1 for more numerical comparisons between the two algorithms under different parameterizations.

A tractable instantiation. Consider the withdrawal problem with a single constraint of $(t_0, \bar{S}\delta_0) = (5,5)$ (no more than five withdrawals are allowed over five time periods).[10] Let the number of new withdrawals per period be distributed as $Y \sim \{0,1,5\}$ *w.p.* $\{0.5, 0.4, 0.1\}$ and the value of these distributions be distributed as $X \sim \{1, 10\}$ *w.p.* $\{0.9, 0.1\}$. We need only two values to represent the state of pending withdrawals, $W(t)$. Let w_ℓ and w_h denote the number of pending "low" ($c = 1$) and "high" ($c = 10$) withdrawals, respectively. Further, let $h_{-1}, h_{-2}, h_{-3}, h_{-4}$ denote the history of withdrawals processed (called $H(t-1)$ above) in each of the last four periods (with a $(5,5)$ constraint, this is the extent of the history that we must consider when deciding what withdrawals to process in this period). This leads the definition of each state

$$s = [w_\ell, w_h, h_{-1}, h_{-2}, h_{-3}, h_{-4}] \in S.$$

---

[10]For our numerical exercises, for simplicity, we model the constraints as corresponding to an absolute number of validators that can withdraw over some window of periods.

Algorithm	Arrival dist.	Value dist.	Discount	Performance
OPTIMAL PRIO-MINSLACK			0.85	-2.374 -2.413
OPTIMAL PRIO-MINSLACK	$Y \sim [0,1,5]$ $w.p.\ [0.5, 0.4, 0.1]$	$X \sim [1,10]$ $w.p.\ [0.9, 0.1]$	0.9	-2.933 -2.982
OPTIMAL PRIO-MINSLACK			0.95	-3.964 -3.999
OPTIMAL PRIO-MINSLACK		$X \sim [1,5]$ $w.p.\ [0.9, 0.1]$		-2.428 -2.422
OPTIMAL PRIO-MINSLACK	$Y \sim [0,1,5]$ $w.p.\ [0.5, 0.4, 0.1]$	$X \sim [1,10]$ $w.p.\ [0.9, 0.1]$	0.9	-2.959 -3.005
OPTIMAL PRIO-MINSLACK		$X \sim [1,20]$ $w.p.\ [0.9, 0.1]$		-3.902 -4.151
OPTIMAL PRIO-MINSLACK	$Y \sim [0,1,2]$ $w.p.\ [0.4, 0.4, 0.2]$			-1.637 -1.638
OPTIMAL PRIO-MINSLACK	$Y \sim [0,1,5]$ $w.p.\ [0.5, 0.4, 0.1]$	$X \sim [1,10]$ $w.p.\ [0.9, 0.1]$	0.9	-2.925 -2.969
OPTIMAL PRIO-MINSLACK	$Y \sim [0,1,10]$ $w.p.\ [0.6, 0.35, 0.05]$			-3.610 -3.620

TABLE 1. Performance over 10,000 simulations for OPTIMAL and PRIO-MINSLACK under different configurations of arrival distributions (Y), value distributions (X), and discount factors. The performance metric is the discounted value of the rewards starting in the initial state $[0,0,0,0,0,0]$; higher values (smaller disutility) are better. **Top three pairs:** *varying discount factors.* We use $n = 225, 350, 700$ for simulation steps for the discount factors of $\gamma = 0.85, 0.90, 0.95$ respectively (each selected so that the end of the trial has a weighting of $\approx 10^{-16}$). **Middle three pairs:** *varying the value distribution.* **Bottom three pairs:** *varying the arrival distribution.*

The action space in this MDP is $A = \{0, 1, 2, 3, 4, 5\}$, where the action a_i is legal if $\sum h_j + a_i \leq 5$. To limit the size of the action space, we only consider states where $w_\ell, w_h < 10$. Even with this extremely reduced setup, there are still $|A| \times |S| \times |S| = 6 \cdot 15246^2 = 1394643096$ probabilities and rewards to encode. Nevertheless, this is feasible since the transition and reward matrices are sparse.

Using value iteration, we numerically solve for the optimal policy, which determines, "given a state, how many withdrawals should we process during this period." We now compare the performance of OPTIMAL (Algorithm 2 (under an assumed discount factor of

0.9)) and `PRIO-MINSLACK` (Algorithm 3).[11] Recall that `PRIO-MINSLACK` is a much simpler heuristic, where it looks at the history and takes action $a_i = 5 - \sum h_j$. While this works well generally, there are situations where it is "overly aggressive" and can result in large disutitlities. For example, consider the state.

$$[10, 0, 0, 0, 0, 0] \implies 10 \text{ pending lows}, 0 \text{ pending highs, empty history.}$$

In this situation, `PRIO-MINSLACK` observes that it can process five low withdrawals immediately and does so ($a_i = 5$). The optimal policy, however, chooses $a_i = 3$ instead. By processing five withdrawals in a single period, `PRIO-MINSLACK` forces a state where no more withdrawals are possible for the following four periods. Using the available capacity, the mechanism runs the risk of a high withdrawal arriving and needing to wait, resulting in a large disutility. The optimal algorithm is "more cautious" by reserving two withdrawal slots for the future, protecting for the possibility that a high-value withdrawal comes in the following few periods. Figure 2 shows a performance comparison of `PRIO-MINSLACK` and `OPTIMAL`. There are ten states (i.e., configurations of the current queue and history of withdrawals) in which the action dictated by the optimal policy differs from `PRIO-MINSLACK` by two (e.g., optimal processes two fewer withdrawals than `PRIO-MINSLACK`) and 338 states in which the optimal action differs from `PRIO-MINSLACK` by one. Table 1 compares the performance of `OPTIMAL` and `PRIO-MINSLACK` under a few variations of (i) arrival distributions, (ii) value distributions, and (iii) discount factors from simulating the two policies.

5.2. *Practical considerations for the heterogeneous value setting*

The previous section outlines dynamic VCG, the optimal withdrawal mechanism given known stationary arrival and value distributions. In practice, the social planner may not know these distributions, and further, the expected number of withdrawals or urgency of the demand for liquidity could change over time. Beyond this, implementing dynamic VCG would require solving the dynamic program outlined in Program 5 and holding funds in escrow to execute the VCG payment rule—both of which seem possible on paper but present significant engineering challenges.

`PRIO-MINSLACK` is much simpler to implement, but may suffer under value heterogeneity because it is too eager to process withdrawals. The problem arises when `PRIO-MINSLACK` receives a burst of low-value withdrawals, in which case it consumes all the available capacity on the low-priority withdrawals and leaves potentially higher-value incoming withdrawals pending longer. These bursts induce a natural question: can we modify

---

[11]Table 1 considers other discount factors, arrival processes, and distributions of value.

Algorithm 4 α-`MINSLACK`

1: ... {same as `MINSLACK`}
2: Sort W(t) in decreasing order of waiting disutility.
3: **for** each constraint $i \leq k$ **do**
4:     `SLACK`$_i \leftarrow \delta_i \overline{S}(t - T_i) - \sum_{\tau=t-T_i+1}^{t-1} \overline{P}(\tau)$.
5: **end for**
6: `MINSLACK` $\leftarrow \min\{\text{SLACK}_i : 1 \leq i \leq k\}$.
7: P(t) $\leftarrow$ Largest prefix of W(t) such that total withdrawn $\leq \alpha \cdot$ `MINSLACK`
8: ... {same as `MINSLACK`}

`PRIO-MINSLACK` to be slightly more conservative with its remaining capacity while preserving its simplicity? One solution is to modify `PRIO-MINSLACK` to consume only an $\alpha \in (0, 1]$ proportion[12] of the `SLACK` available at each period.

Algorithm 4, which we call α-`MINSLACK`, makes the one-line modification (shown in red) to `PRIO-MINSLACK` by scaling the amount of processed withdrawals by α. By tuning α, we can make α-`MINSLACK` more or less aggressive in how much withdrawal capacity it uses now versus saving. At $\alpha = 1$, we reduce to the "maximally aggressive" version (`PRIO-MINSLACK`). In contrast, as $\alpha \to 0$, α-`MINSLACK` becomes increasingly conservative. The outcome here is that the slack continues to build up across the constraints, and you end up processing at the rate $\alpha \cdot \delta_i / T_i$ per-unit time, where $(\delta_i, T_i) \in \mathcal{C}$ s.t., $\delta_i / T_i = \min_j \delta_j / T_j$. In other words, process at a constant rate proportional to the "most restrictive" constraint in $\mathcal{C}$. More moderate values, e.g., $\alpha = 0.5$, present a more balanced version of α-`MINSLACK` where the algorithm functions on the heuristic of "using half of the remaining slack at each period."

While α-`MINSLACK` is not necessarily optimal, it does eliminate the need for arrival distribution knowledge, which the optimal mechanisms rely on. Further, its simplicity makes it much more feasible for an actual production system. We justify this statement by numerically comparing the performance of various mechanisms under different withdrawal distributions. Note that we can expand the set of value distributions compared to the optimal analysis because we are no longer constructing the entire state space of the MDP.

Table 2 compares the performance of four different algorithms across a constant arrival distribution and under three different value distributions. These results demonstrate that the `PRIO-` and α- versions of `MINSLACK` far outperform either the `CONSTANT` mechanism or regular `MINSLACK` (which serves as an FCFS-queue rather than a priority queue based on the value of the withdrawal). These results motivate that, under some distributions, α-`MINSLACK` may be preferable to `PRIO-MINSLACK`. Further work could be done to study adaptive algorithms that aim to learn the optimal value of α in an on-line fashion. Again,

---

[12]The interval is left-open because $\alpha = 0$ implies no withdrawals are ever processed.

Algorithm	Arrival dist.	Value dist.	Performance
CONSTANT (1) MINSLACK PRIO-MINSLACK α-MINSLACK ($\alpha = 0.9$)		$X \sim \text{Uniform}(0,1)$	-5.768 -5.464 -2.019 -2.002
CONSTANT (1) MINSLACK PRIO-MINSLACK α-MINSLACK ($\alpha = 0.9$)	$Y \sim [0,1,5]$ $w.p.\ [0.5, 0.4, 0.1]$	$X \sim \text{Exp}(0.1)$	-12.249 -11.648 -2.951 -2.986
CONSTANT (1) MINSLACK PRIO-MINSLACK α-MINSLACK ($\alpha = 0.9$)		$X \sim \text{Pareto}(2,5)$	-114.913 -109.354 -67.687 -63.070

TABLE 2. Numerical results for algorithm performance under a fixed withdrawal arrival distribution and three different value distributions. The performance metric measures the average disutility over the withdrawals and thus should be minimized (to maximize the utility). We calculate the mean disutility over ten independent samples of 10,000 steps each, with the first 1,000 steps of each sample discarded to allow the system to settle into a steady state. The single constraint was set as $(\delta, T) = (5,5)$: "a maximum of five withdrawals may be processed over five periods." CONSTANT processes one withdrawal per time step. MINSLACK and PRIO-MINSLACK follow the descriptions in Algorithms 1 and 3 respectively. For α-MINSLACK (Algorithm 4), we use $\alpha = 0.9$, which produces the following mapping for calculating how much slack to consume in a given time slot $[0,1,2,3,4,5] \mapsto [0,1,2,3,4,4]$. We can describe this simply as: "If the slack is exactly 5, use only four (reserving one for a potentially high-value arrival). If the slack is less than 5, use it entirely." The arrival distribution mimics occasional bursts of withdrawal requests while maintaining an expected value $E[Y] = 0.9$, less than the average capacity of one derived by the $(5,5)$ constraint. The withdrawal values were sampled from Uniform, Exponential, and Pareto distributions to demonstrate that under some conditions, α-MINSLACK can outperform PRIO-MINSLACK. In all cases, CONSTANT and MINSLACK perform far worse than the PRIO- and α- variants.

these heuristic rules for determining the withdrawal policy of the staking system are far more straightforward to construct and implement than the optimal versions described in Section 5.1.3.

6. THEORY AND PRACTICE

Why limit withdrawals in the first place? A thought experiment. Assume that withdrawals are not limited. An attacker, Eve, accumulates 1/3 of the total stake in the PoS mechanism and invests heavily in networking infrastructure. Eve contacts Alice to inquire about buying a Tesla Cybertruck©. Alice, who is feeling both cyber- and cypherpunk enough to accept `ETH` for the transaction, sees `txn 0xcb` on Etherscan as finalized, giving her confidence to hand the (car) keys to Eve. From Alice's perspective, the settlement assurance of 1/3 of all staked `ETH` (> 33 billion USD as of May 2024) is more than sufficient economic security for her transaction. However, Eve using her networking prowess, Eve had tricked the honest validators into finalizing two conflicting blocks, one which included `txn 0xcb` and another that didn't by partitioning the honest validators into two separate p2p groups and sharing conflicting attestations with each group. If withdrawals are not limited, she can fully withdraw her stake from both chains by the time honest validators reconnect (once Eve's network-level attack ends) and try to slash her. Alice has no Telsa Cybertruck© nor the `ETH` originally sent in `txn 0xcb`.

In light of this, blockchains place limits on withdrawals. However, as described below, there is substantial variation in the limits placed and the withdrawal procedure, with little systematic study.

6.1. *Accountable Safety and Limiting Withdrawals*

We begin with the following simple observation.

OBSERVATION 2. *The accountable safety of a finalized block **decreases** as time passes because the stake participating in the finalization of the block can withdraw from the system.*

This (rather counter-intuitive) fact means protocol designers must decide: "How quickly should validators be able to withdraw their stake from the system?" Let $\mathcal{D}$ denote the 'maximum-tolerable decay' in the accountable safety of a finalized block. For example, if $\mathcal{D} = 1/6$, then a finalized block may have accountable safety (in terms of proportion of the total stake that is slashable in case the transaction history changes) of $1/3 - \mathcal{D} = 1/6$. The security decay modifies the statement to, "any transaction in a finalized block will have accountable safety of at least 1/6 of all stake." This remains incomplete because over a sufficiently long time horizon, with withdrawals enabled, more than $\mathcal{D}$ stake may be removed from the system. Thus, we define an amount of time, denoted δ, over which the stake withdrawn must not exceed $\mathcal{D}$. This period can serve multiple purposes. One such usage is the weak-subjectivity period (Buterin, 2014), where the delay is an upper bound on the communication delay between all honest parties in the partially-synchronous protocol; this value is $\mathcal{O}(weeks)$ to account for the natural overhead incurred when social
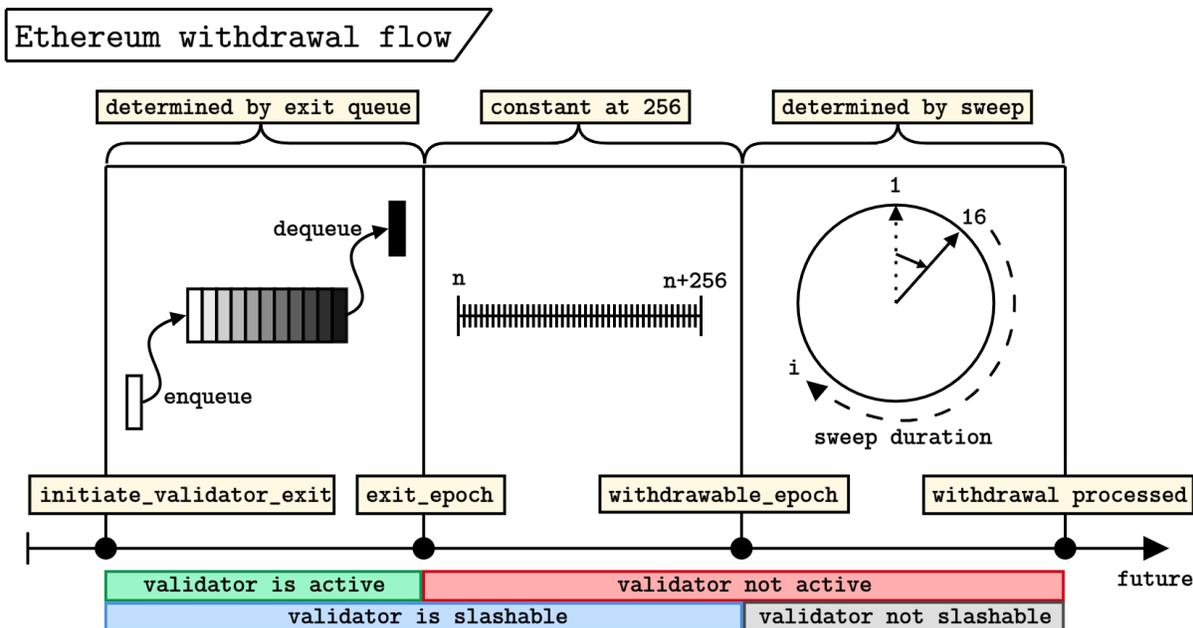
FIGURE 3. The withdrawal flow for Ethereum validators. Each phase has differing lengths and validator properties. The **top row** of tan labels demonstrate what determines the length of each phase. The **middle row** of tan labels annotate the timeline of events as described in the Capella-Specifications (2023). The **bottom row** of colored labels indicate the activity and slashability of the validator over time.

coordination is required to come to consensus.[13] Other constraints might be over much shorter time horizons, e.g., $\mathcal{O}(minutes)$, to ensure a bound on the rate at which the economic security of a block changes in short windows. Thus, the accountable safety of a Proof-of-Stake mechanism parameterized by $\mathcal{D}$ and δ is "any block finalized more than δ time ago is immutable (only social consensus could reverse it), and any block finalized within the past δ time has accountable safety of at least $1/3 - \mathcal{D}$."

6.2. *Ethereum*

Withdrawals in Ethereum Proof-of-Stake were fully activated in the Shanghai/Capella Hardfork[14] on April 12, 2023. While the full withdrawal process is quite involved, we dig into the details to demonstrate how much engineering can shape the withdrawal mechanisms in use today. Figure 3 demonstrates the full flow of an Ethereum withdrawal, which is split into three distinct phases.

---

[13]By ignoring any blocks published prior to the weak-subjectivity checkpoint, validators can also eliminate the risk of long-range attacks (in practice, validators treat their latest finalized block as a 'genesis' or irreversible block by simply rejecting any block that conflicts with it).

[14]https://ethereum.org/en/history/#shapella

Phase 1: Exit queue – When an Ethereum validator wants to withdraw their 32 `ETH` from the consensus mechanism, they trigger a "Voluntary Exit" (Beacon-Chain-Specifications, 2020). This process sets the validator's `exit_epoch` based on the rate-limited first-come-first-served exit queue; during each epoch, at most $\min(4, \lfloor \# \text{ validators}/2^{16} \rfloor)$ are processed (Beacon-Chain-Specifications, 2020) (though this was changed in EIP-7514 to cap the churn limit at 8 validators per-epoch Lion and Bieko (2023), making the new function $\max(8, \min(4, \lfloor \# \text{ validators}/2^{16} \rfloor)))$. The `CHURN_LIMIT_QUOTIENT` $= 2^{16}$ was selected[15] according to the rough heuristic that it should take approximately one month for 10% of the stake to exit (or equivalently, about 100 days for 33% of the stake to exit) Buterin (2020). For the entire time a validator is in the exit queue (this phase), they are both 'active' (meaning they must continue performing their consensus duties) and 'slashable' (meaning their stake is still accountable for their behavior). Keeping the validator active while in the exit queue minimizes the economic cost of a very long exit queue because they continue earning rewards (Buterin, 2019a).

Phase 2: Withdrawalability delay – Once the validator's `exit_epoch` has passed, they incur a constant delay of 256 epochs (27 hours) before their `withdrawable_epoch` (Beacon-Chain-Specifications, 2020). This fixed delay is a significant safety buffer to provide ample time for the protocol to include any slashing proof on chain. During this time, the validator is no longer active (and thus not earning any rewards), but they remain slashable (to avoid committing a slashing violation immediately before the withdrawal). The enforcement of this delay ensures that, even if the exit queue is empty, there is a period where the validator's stake is still accountable for their actions.

Phase 3: Validator sweep – Once past the validator's `withdrawable_epoch`, the function `is_fully_withdrawable_validator` returns `true` indicating that the withdrawal delay has passed and the validator is no longer slashable (Capella-Specifications, 2023). The last delay comes from the amount of time it takes for the actual withdrawal requests to send the `ETH` to the corresponding withdrawal address. All withdrawals are processed by looping through the validator set in order of validator index. This validator "sweep" can only process 16 withdrawals per block, corresponding to 8.8 days to iterate through the entire validator set (thus a 4.4-day additional delay on average).[16] This 8.8-day delay is present regardless of the length of the exit queues because "full" withdrawals (where a validator wants to leave the consensus layer altogether) are inter-mixed with "partial" withdrawals (where a small amount of validator rewards are involuntarily swept from each validator). While the original specification implemented the queues directly into

---

[15]Powers of two common for specification constants due to their compact binary representation.

[16]https://www.validatorqueue.com/

the protocol, Potuz (2022) changes this only to store the validator index and perform the sweep by mixing partial and full withdrawals.

This withdrawal mechanism is quite complex; the minimum time to exit the system is 27 hours. Due to the validator sweep, if the validator doesn't strategically time their exit request, the withdrawal will take over 5.5 days on average to fully clear, even if the exit queue is empty. This complexity highlights how engineering decisions can inform the exit queue mechanism design. Beyond Ethereum, there are many additional staking systems, though their withdrawal mechanisms are much simpler and thus presented in Table 3 & Table 4.

6.3. *Other Proof-of-Stake Blockchains*

Table 3 compares several other blockchain protocols and how they handle withdrawals. Ethereum is the only protocol that implements a dynamic queue, and in this regard, Ethereum takes on additional complexity to improve the efficiency of the withdrawal mechanism. Cosmos, Polygon, and Polkadot each implement the simple, fixed-duration withdrawal mechanism with delays of 21, 2, and 28 days, respectively. This mechanism is simple and easy to reason about. Still, it is much less efficient because each withdrawal takes the maximal amount of time regardless of the history of the mechanism (Birch, 2020; Polygon-Knowledge-Layer, 2023; Polkadot-Validator-Guide, 2024). Solana, Cardano, and Avalanche do not have in-protocol slashing, so staking serves only as an anti-Sybil mechanism in their systems; the stake can exit the system without a rate-limiting step and not change their security model (Solana-Documentation, 2023; Cardano-Undelegation, 2023; Avalanche-Documentation, 2024).

6.4. *Other applications of staking*

Beyond other blockchains, some applications have implemented staking and slashing mechanisms at the application layer of Ethereum. Table 4 performs the same high-level analysis of two such mechanisms.

EigenLayer and Chainlink use stake for slightly different purposes than the chains outlined in Table 3. EigenLayer creates a platform for buying and selling "economic security"; services built on EigenLayer (called "Actively Validated Services" or 'AVSs') purchase this security by incentivizing capital to delegate to an operator running their service. Because EigenLayer encumbers capital with additional slashing conditions, it also enforces a protocol-wide escrow period for stake removal. It is worth noting that the Ethereum withdrawal period can occur concurrently with the EigenLayer escrow period (EigenLayer-Documentation, 2023). Further, services buying security from EigenLayer can impose further constraints on the capital allocated to their system. Chainlink, on the

Protocol	Staking purpose	Withdrawal mechanism	One-line analysis
Ethereum	Consensus safety	Rate-limited FCFS queue with minimum duration.	Aims to be fast in the average case, but partial withdrawals induce high-variance delay.
Cosmos	Consensus safety	Fixed 21-day unbonding period.	Simple but inefficient.
Solana	Sybil resistance	All deactivations happen at epoch boundaries. A maximum of 25% of stake can deactivate at any given epoch boundary.	With no slashing, stake does not provide accountable safety to the protocol. Limiting withdrawals ensures the entire stake cannot exit in a single epoch.
Cardano	Sybil resistance	Immediate withdrawals.	With no slashing, stake does not provide accountable safety to the protocol. Withdrawals are immediately processed.
Polygon	Consensus safety	Fixed ≈ 40 hour unbonding period.	Simple but inefficient. It benefits from the fact that, as a sidechain, state updates are posted to Ethereum and are thus immutable – allowing for a relatively shorter fixed duration.
Polkadot	Consensus safety	Fixed 28-day unbonding period.	Simple but inefficient.
Avalanche	Sybil resistance	Validators dictate the duration of their staking before becoming active. The minimum duration is two weeks. After time has elapsed, the stake is immediately withdrawn.	With no slashing, stake does not provide accountable safety to the protocol. Withdrawals are immediately processed.

TABLE 3. Comparing staking and withdrawal mechanisms across L1 protocols and sidechains.

Protocol	Staking purpose	Withdrawal mechanism	One-line analysis
EigenLayer	Economic security guarantees	Fixed 7-day escrow period for all `ETH`-denominated withdrawals. Staked `EIGEN` has a fixed 24-day escrow period.	Withdrawals need to be limited because EigenLayer introduces new slashing conditions. Native restaked `ETH` may be withdrawn from the beacon chain during the EigenLayer escrow period. Each AVS could add its rate limiting in addition to the system-wide minimums.
Chainlink	Oracle safety	Fixed 28-day cool-down period before `LINK` is claimable.	Staking provides safety and availability conditions for data feeds. Withdrawals are rate-limited to ensure slashing has time to take place.

TABLE 4. Comparing staking and withdrawal mechanisms between Eigen-Layer and Chainlink, two app-layer protocols with slashing.

other hand, uses stake to provide security for the data feeds supplied by their oracle network (Breidenbach, Cachin, Chan, Coventry, Ellis, Juels, Koushanfar, Miller, Magauran, Moroz, et al., 2021). This stake may be slashed for "less objective" faults (e.g., slashing for being offline and not providing a price feed), which was recently dubbed "inter-subjective slashing" in EigenLabs (2024) and may grow to play a significant role in the future designs of slashing protocols.

6.5. *Liquid staking & restaking tokens*

Liquid staking tokens (LSTs) make a design trade-off when choosing how much of the capital in their system to deploy into consensus mechanisms. If they deploy too much of it, the withdrawals will be rate-limited by the underlying protocol, leading to a more capital-efficient protocol at the cost of a worse UX (slower withdrawals). Keeping some liquidity available for immediate redemption improves the UX, but any capital in that

state is not cash-flowing. LSTs are fully collateralized and thus do not face insolvency risk, but holders face the duration risk of holding the LST for however long the withdrawal takes. Liquid restaking tokens (LRTs) have a more complex design space, where they must balance withdrawals against various underlying protocols and services. Their withdrawal mechanisms are plagued by the nature of various protocol rewards denominated in different tokens and emissions rates. Hart and Einhorn (2024) explores some design trade-offs, including a market for withdrawals. Overall, this design space is extensive and out-of-scope for the modeling of this paper, but it presents an exciting avenue for future research.

7. CONCLUSION

System designers of staking and restaking protocols face a fundamental trade-off between the security and utility. Based on the mechanisms we surveyed in Section 6, the mechanisms currently in production maximally flexible given the rigidity they claim to require. In other words, nobody seems to be on the production-possibilities frontier of egress mechanisms in practice; we acknowledge that the practical engineering constraints, e.g., as described in the design of Ethereum's withdrawal mechanism in Section 6.2, may play a significant role in the decision making of existing protocols.

By formalizing this trade-off as a constrained optimization problem over mechanisms, we aim to improve the state of withdrawal systems more broadly. For blockchain designers, we distill our results into three pieces of advice. First, suppose your consistency constraints are over a longer time horizon than a single epoch. In that case, a queue with dynamic capacity can significantly reduce average wait times without sacrificing security — MINSLACK (Algorithm 1) is a simple example of maximally processing the rate of withdrawals given a set of constraints. Second, if you believe that participants in the system may have heterogeneous disutility from waiting in the exit queue, their welfare would be improved by implementing a priority queue – PRIO-MINSLACK (Algorithm 3) can quickly decrease the overall disutility. Third, if you think that the time-sensitivity or arrival process of future withdrawal requests is particularly fat-tailed, be sure to reserve some capacity in the system to allow the processing of highly time-sensitive withdrawals during periods of congestion — α-MINSLACK (Algorithm 4) is an example of this reservation.

We point to a few intriguing directions in terms of future work. Firstly, several empirical questions have been raised by this study. Assessing the actual staker surplus lost from sub-optimal queue designs would be helpful. The protocol may care about this staker surplus because reducing the staker disutility may lessen the emissions needed to incentivize token holders to stake in the first place. Further study on the heterogeneity in time preferences among stakers would help determine whether pay-for-priority systems are worth considering. Lastly, validator utility functions that are non-linear (e.g., a validator

who needs their withdrawal within the next week but doesn't care when) may lead to different design considerations and optimal withdrawal mechanisms.

On the theoretical side, note that some of the pay-for-priority systems we have proposed serve as benchmarks and are unlikely to be implementable in practice (e.g., the dynamic programming-based efficient allocation in Algorithm 2, which is both computationally difficult and requires knowledge of the distribution of withdrawal requests). Other mechanisms (e.g., PRIO-MINSLACK, Algorithm 3) are feasible as a pay-your-bid mechanism — reminiscent of the Bitcoin (and Ethereum before EIP-1559) transaction-fee mechanisms. Similar concerns faced in those contexts, users having to choose an appropriate bid, may apply in withdrawal mechansims too. The natural question is whether designs with better user experience, analogous to EIP-1559, exist in this setting.

References

ANSGAONKAR, A. (2020): "Weak Subjectivity in Ethereum 2.0," *URL: https://notes.ethereum.org/@adiasg/weak-subjectvity-eth2*.

ASGAONKAR, A., F. D'AMATO, R. SALTINI, L. ZANOLINI, AND C. ZHANG (2024): "A Confirmation Rule for the Ethereum Consensus Protocol," *arXiv preprint arXiv:2405.00549*.

AVALANCHE-DOCUMENTATION (2024): "How to Stake on Avalanche," *URL: https://docs.avax.network/nodes/validate/how-to-stake*.

BEACON-CHAIN-SPECIFICATIONS (2020): "Phase 0 – The Beacon Chain," *URL: https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/beacon-chain.md*.

BERGEMANN, D., AND J. VÄLIMÄKI (2010): "The dynamic pivot mechanism," *Econometrica*, 78(2), 771–789.

BIRCH, G. (2020): "The staking module," *URL: https://github.com/gavinly/CosmosParametersWiki/blob/master/Staking.md#1-unbondingtime*.

BREIDENBACH, L., C. CACHIN, B. CHAN, A. COVENTRY, S. ELLIS, A. JUELS, F. KOUSHANFAR, A. MILLER, B. MAGAURAN, D. MOROZ, ET AL. (2021): "Chainlink 2.0: Next steps in the evolution of decentralized oracle networks," .

BUDISH, E., A. LEWIS-PYE, AND T. ROUGHGARDEN (2024): "The Economic Limits of Permissionless Consensus," *arXiv e-prints*, pp. arXiv–2405.

BUTERIN, V. (2014): "Proof of Stake: How I Learned to Love Weak Subjectivity. 2014," *URL: https://blog. ethereum. org/2014/11/25/proof-stake-learned-love-weak-subjectivity*.

——— (2018): "Suggested average-case improvements to reduce capital costs of being a Casper validator," *URL: https://ethresear.ch/t/suggested-average-case-improvements-to-reduce-capital-costs-of-being-a-casper-validator/3844*.

——— (2019a): "Rate-limiting entry/exits, not withdrawals," *URL: https://ethresear.ch/t/rate-limiting-entry-exits-not-withdrawals/4942*.

——— (2019b): "Weak subjectivity under the exit queue model," *URL: https://ethresear.ch/t/weak-subjectivity-under-the-exit-queue-model/5187*.

——— (2020): "Vitalik's Annotated Ethereum 2.0 Spec," *URL: https://github.com/ethereum/annotated-spec/blob/master/phase0/beacon-chain.md*.

BUTERIN, V., AND V. GRIFFITH (2017): "Casper the friendly finality gadget," *arXiv preprint arXiv:1710.09437*.

CAPELLA-SPECIFICATIONS (2023): "Capella – The Beacon Chain," *URL: https://github.com/ethereum/consensus-specs/blob/dev/specs/capella/beacon-chain.md*.

CARDANO-UNDELEGATION (2023): "Cardano (ADA) Undelegation Period," *URL: https://p2p.org/faq/en/articles/5253938-cardano-ada-undelegation-period*.

CHITRA, T. (2021): "Competitive equilibria between staking and on-chain lending," *CryptoEconomic Systems (CES)*.

DEB, S., R. RAYNOR, AND S. KANNAN (2024): "STAKESURE: Proof of Stake Mechanisms with Strong Cryptoeconomic Safety," .

EIGENLABS (2024): "EIGEN: The Universal Intersubjective Work Token," *URL: https://github.com/Layr-Labs/whitepaper/blob/master/EIGEN_Token_Whitepaper.pdf*.

EIGENLAYER-DOCUMENTATION (2023): "Escrow Period (Withdrawal Delay)," *URL: https://docs.eigenlayer.xyz/eigenlayer/restaking-guides/restaking-user-guide/#escrow-period-withdrawal-delay*.

HART, S., AND M. EINHORN (2024): "Building a liquid restaking token from first principles," *URL: https://timewave.computer/liquid-restaking-token*.

HUBERMAN, G., J. D. LESHNO, AND C. MOALLEMI (2021): "Monopoly without a monopolist: An economic analysis of the bitcoin payment system," *The Review of Economic Studies*, 88(6), 3011–3040.

KYNE, O. (2023): "EIP-7002: Execution layer triggerable exits," *URL: https://ethereum-magicians.org/t/eip-7002-execution-layer-triggerable-exits/14195/6*.

LAVI, R., AND N. NISAN (2000): "Competitive analysis of incentive compatible on-line auctions," in *Proceedings of the 2nd ACM Conference on Electronic Commerce*, pp. 233–241.

LESHNO, J. D. (2022): "Dynamic matching in overloaded waiting lists," *American Economic Review*, 112(12), 3876–3910.

LEWIS-PYE, A., AND T. ROUGHGARDEN (2024): "Permissionless Consensus," .

LION, d., AND T. BIEKO (2023): "EIP-7514: Add Max Epoch Churn Limit," *URL: https://eips.ethereum.org/EIPS/eip-7514*.

NEU, J., E. N. TAS, AND D. TSE (2023): "Short paper: Accountable safety implies finality," *Cryptology ePrint Archive*.

NISAN, N. (2023): "Serial Monopoly on Blockchains," .

PAI, M., AND M. RESNICK (2023): "Dynamic Transaction Fee Mechanism Design," *arXiv preprint*.

PARKES, D. C., AND S. SINGH (2003): "An MDP-based approach to online mechanism design," *Advances in neural information processing systems*, 16.

POLKADOT-VALIDATOR-GUIDE (2024): "Run a Validator (Polkadot)," *URL: https://wiki.polkadot.network/docs/maintain-guides-how-to-validate-polkadot*.

POLYGON-KNOWLEDGE-LAYER (2023): "How to delegate," *URL: https://docs.polygon.technology/pos/how-to/delegate/#unbond-from-a-validator*.

POTUZ (2022): "Withdrawals without queues," *URL: https://github.com/ethereum/consensus-specs/pull/3068*.

ROUGHGARDEN, T. (2020): "Transaction fee mechanism design for the Ethereum blockchain: An economic analysis of EIP-1559," *arXiv preprint arXiv:2012.00854*.

——— (2021): "Transaction fee mechanism design," *ACM SIGecom Exchanges*, 19(1), 52–55.

SOLANA-DOCUMENTATION (2023): "Delegation Timing Considerations," *URL: https://solana.com/staking#delegation-timing-considerations*.

SU, X., AND S. A. ZENIOS (2006): "Recipient choice can address the efficiency-equity trade-off in kidney transplantation: A mechanism design model," *Management science*, 52(11), 1647–1660.