A Multi-Objective Optimization Framework for the COOL Intelligence Tier

January 2026

Abstract

We introduce a responsive multichain distributed ledger architecture that autonomously modifies network topology in reaction to evolving network circumstances. The framework incorporates bid submissions from applications and infrastructure operators, predictive analytics-driven forecasting of near-term and extended deviations, and the deployment of transient (temporary) execution environments generated as needed. Central to our methodology is a multi-objective optimization engine that orchestrates the grouping of applications, the designation of infrastructure operators, and the configuration of execution environment parameters. This engine simultaneously optimizes across numerous facets of network performance—encompassing aggregate fees, infrastructure operator returns, extended network vitality, and system resilience—facilitating a systematic balance between operational efficiency and stability. The architecture provides a basis for expandable, responsive, and economically viable multichain distributed ledger systems.

Important Notice: This research document is an internal working paper presently in DRAFT status. Multiple enhancements and revisions are under active development, and will be incorporated in the document prior to public dissemination.

1 Introduction

Distributed ledgers, while delivering transparency and security, frequently encounter inflexibility in their architecture and functioning. This inflexibility can generate inefficiency as well as diminished capacity to adapt effectively to fluctuating workloads, diverse application requirements, or infrastructure operator unpredictability. This absence of responsiveness can impede expandability, diminish performance, and constrain expressiveness in decentralized application deployment. Additionally, an extra advantage derived from responsiveness is enhanced user experience from applications, as it renders the multichain structure transparent to the end user.

In this research, we introduce a systematic methodology for constructing a responsive distributed ledger network: one where a network can autonomously reconfigure itself in reaction to evolving demands, infrastructure operator capacities, and takes into consideration projected future network conditions. The objectives of the system are: (1) to be responsive; (2) to be resource-efficient; (3) to be strategy-resistant.

Structurally, we examine a multichain distributed ledger, comprising a primary chain – designated as coreChain – and multiple subsidiary chains, designated as execChains, which can be conceptualized as rollups; together with the nodes of the optimization tier, designated as opChain validators. Specifically, we adopt a multi-tier methodology, where the coreChain functions as a settlement layer that completes transactions and hosts execution environments (execChains).

The foundation of our approach is a *multi-objective optimization engine* that synthesizes three fundamental components:

1. **Bid submissions from applications and infrastructure operators**, specifying resource demands for applications and resource availability for infrastructure operators, and economic preferences;
2. **Governance parameters**, employed primarily to establish how to reconcile the varying requirements of different participants;
3. **Predictive analytics-driven forecasting**, projecting near-term and extended network dynamics;
4. **An execution environment clustering and designation mechanism**, through which applications and infrastructure operators are assembled into transient execution environments with consistent characteristics such as fee-to-gas ratios and common operational parameters.

The optimization target is a multi-objective function that synthesizes the following elements: application benefit, infrastructure operator benefit, and system benefit. Application benefit reflects gas assigned to the application, reduction of inactivity and volatility, weighted according to application preferences. Infrastructure operator benefit primarily captures anticipated returns and reduction of computational expenses that pertain to implementing opChain configuration modifications. System benefit examines the aggregate fee volume flowing in the system, the aggregate stake in the system, together with supplementary criteria such as the variety of application categories.

1.1 Document Structure

The organization of the document is as follows:

- We commence with an *informal summary* of the suggested methodology, elaborating the input elements (application/infrastructure operator specifications, predictive analytics forecasts) and how they are integrated by the optimization engine.
- We proceed to introduce the *formal mathematical framework*, describing the optimization problem and its elements in depth.
- We subsequently provide a *theoretical examination* of the solution, establishing the pertinent objectives – specifically, we examine the general *network vitality*, which is constructed from application benefit, infrastructure operator benefit, and system benefit (which may incorporate certain extended objectives for the improvement of the network) – and addressing properties such as computational feasibility, incentive alignment, and compatibility.
- We conclude with a characterization of *prospective challenges* and potential extensions not addressed in the present framework.

2 Related Work

We position our research within several pertinent areas of related investigation: multichain distributed ledger structures; machine learning for decentralized systems, reflecting our application of predictive frameworks to enhance system responsiveness; and multi-objective optimization – particularly in the context of resource distribution (as infrastructure operators are fundamentally network resources to be distributed to applications), which constitutes the methodological basis of our methodology.

Multichain Distributed Ledger Structures. We introduce an architecture for an optimization-oriented multichain distributed ledger. Indeed, numerous distributed ledger systems seek to enhance expandability and modularity through multichain architectures. Below we examine several of these systems, and note that, to our knowledge, our architecture is the initial design that introduces *transient, optimization-guided execution environments* that adapt to workload dynamics, directed by application and infrastructure operator specifications.

- Internet Computer (ICP) accommodates smart contracts (canisters) on dynamic subnets, with management handled by the Network Nervous System (NNS). While subnets are reconfigurable, this is accomplished manually, and the framework lacks an optimization module, which is central to our methodology; moreover, applications in ICP are not treated as individually identifiable entities as they are in our architecture.
- Polkadot employs a relay chain (analogous, in our terminology, to the coreChain) to coordinate security for parachains (execChains, in our terminology), which are incorporated through governance or auctions. While it supports multichain execution, parachains are persistent and lack dynamic designation.
- Cosmos facilitates interoperability between sovereign chains utilizing the IBC protocol. Chain creation and validator designation are static and application-specific.

- Ethereum rollups offload execution to L2s, improving throughput. However, rollups are project-specific and do not support system-level coordination or application clustering.

Multi-Objective Optimization and Resource Distribution. The foundation of our architecture is a multi-objective optimization framework aiming at balancing competing trade-offs. As such, we adapt the multi-objective optimization paradigm to jointly optimize application and infrastructure operator outcomes in our context. Furthermore, as the infrastructure operators are essentially network resources to be distributed to applications (through transient execution environments), our research relates to resource distribution mechanisms. In particular, as different infrastructure operators possess different computational capabilities and economic characteristics), we note the mechanism of *Dominant Resource Fairness (DRF)*, which distributes multiple resource types equitably in distributed environments – note that we do not need to suffer the social welfare decrease that DRF suffers in favor of egalitarian fairness, as we consider a repeated, dynamic setting.

Finally, to ensure fairness across optimal solutions, our architecture incorporates randomized selection and compensation. This is inspired by work on ex-ante fairness and lottery-based distribution, as well as mechanisms for fair division in discrete settings.

Strategy-Resistant Mechanism Design and Incentive Alignment. A core objective of our architecture is to incentivize truthful declarations from applications and infrastructure operators—specifically regarding gas usage, fee preferences, and capabilities. This aligns with the literature on *strategy-resistant mechanism design*, which aims to ensure that agents cannot benefit by misreporting private information. Classical results in mechanism design provide a foundation for designing truthful and efficient distribution mechanisms in multi-agent systems. In particular, iterative combinatorial auctions and Vickrey–Clarke–Groves (VCG) mechanisms are known to elicit truthful bids under certain assumptions.

Responsive Decentralized Systems and Machine Learning. Our architecture uses machine learning (ML) to improve robustness and responsiveness, serving two core roles: (1) *predictive modeling* of infrastructure operator availability, application gas needs, and performance volatility; and (2) *optimization augmentation*, including imputing missing data, adjusting optimization weights (reflecting evaluations of the importance of certain aspects to different agents), and refining benefit estimates. Prior work has explored such use for (non-distributed ledger) self-healing infrastructures and for improving, e.g., scheduling, pricing, and load prediction; as well as for databases. A general, theoretical study of self-responsive systems also exists.

Responsive Distributed Ledgers. The general concept of a responsive distributed ledger has been suggested and studied. Some of the ideas are heuristic in nature; while other rely on machine learning techniques or concentrate on execution parameters of the distributed ledgers. To the best of our knowledge, no prior work has been done on the responsive clustering and assignment of applications to transient execution environments based on global optimization with bidding inputs and machine learning estimations.

3 Informal Description

At the heart of our methodology is a mathematical multi-objective optimization architecture that dynamically adapts the distributed ledger network to changing conditions. (Technically, assume that the framework is being solved once in an Epoch – An Epoch is a fixed-length time interval—can be measured in time or as a set number of blocks—during which the optimization framework is solved.) This architecture is designed to integrate several sources of input, balance competing goals, and generate system configurations that improve overall network performance while minimizing disruption.

The optimization framework draws on three primary inputs:

5. **Application and infrastructure operator bid submissions:** Applications (apps) and infrastructure operators (validators) submit declarations that describe their needs and capabilities. For apps, this includes parameters such as their desired gas usage, required stake, and maximum acceptable fee-to-gas ratio. Apps may also indicate other requirements, such as compatibility with specific virtual machines. Infrastructure operators report their available resources, including computational power, stake, and supported features, as well as preferences like minimum fee to gas they accept to operate with. Importantly, these declarations are treated as bids that inform the optimization, and misreporting is disincentivized through penalty mechanisms (and following the properties of the optimization used, as discussed below).
6. **Predictive analytics forecasting:** Complementing the declared bids, a machine learning (ML) module provides predictions that help estimate both near-term and long-term network dynamics. This includes forecasting potential deviations in gas usage, infrastructure operator availability, and latency patterns, as well as filling in missing or unspecified inputs (e.g., determining app-individual weights for apps that are not supplying them). These predictions improve the robustness of the optimization by accounting for uncertainty and variability in the system, towards a network that can adapt proactively rather than reactively.

Remark 2. *The roles of the ML:*

- The optimization value can be fine-tuned by ML; e.g., if the ML model predicts (based on data analysis, user signals, etc.) that a certain type of apps—say, gaming apps—are more important for network vitality (e.g., will provide more fees), and if allowed by governance, then the optimization objective can be adjusted to assign greater weight to their benefit.
- It can be used as a recommendation system for apps without history (i.e., filling-out a bid for an app that does not provide it, based on the bids of other, similar apps that did provide it); as well as for apps that do provide history – but the specific values can be corrected based on ML predictions. (E.g., expected gas need for specific app, based on the relationship between historical app gas needs of that app and the gas declarations of the app; or, expected gas availability by a specific infrastructure operator that do have history, given its past gas capability declarations and actual demonstrated gas capabilities.)

7. **Current network state:** The optimization also takes into account the existing configuration of the network, using it as an anchor to measure and limit the instability introduced by changes. For example, if reassigning apps to different execution environments would cause disruption or downtime, the optimization incorporates this as a cost and seeks to minimize it (as an additional objective in the multi-objective optimization). Additionally, the execution cost of implementing configuration changes is taken into account in the optimization.

As there are multiple parties involved, and several desiderata, the optimization itself is naturally formulated as a multi-objective problem that balances the relevant, partially-competing goals – generally speaking, agent-specific weights are governed by the agents; while the overall balancing between the needs of the different agent types is controlled by network governance. These include maximizing overall network vitality (for example, by increasing the total fees generated and improving infrastructure operator yields), ensuring fair (for apps and infrastructure operators alike, taking into account these different stakeholders) and efficient allocation of computational resources to apps, and minimizing service disruptions or reconfiguration costs (to maximize service predictability and stability). The relative importance of these objectives can be set using default system weights or customized according to app-specific preferences.

Remark 3. *Note that infrastructure operator yield is in relative terms (percentage) and not in absolute terms (i.e., multiplied by infrastructure operator stake) and we can also in principle add inflationary rewards in the form of minted rewards as well as non-linear (in stake) reward.*

The output of the optimization is a detailed assignment of apps and infrastructure operators to transient execution environments, grouping participants into clusters with compatible fee-to-gas ratios and operational requirements. These execution environments are created on demand, allowing the network to expand or contract dynamically in response to workload fluctuations. The optimization can be computed directly using mathematical solvers such as Gurobi, which can process the full set of constraints and objectives.

In summary, the system combines economic inputs, predictive analytics, and optimization techniques to create a responsive and responsive multichain distributed ledger structure capable of meeting the demands of diverse and evolving workloads.

3.1 Architectural Building Blocks

We mention the architectural needs of the system; essentially, for implementation, we need the following building blocks:

- A way for apps to declare their needs and preferences.
- A way for infrastructure operators to declare their capabilities and preferences.
- A runtime analysis done by opChain validators (this is an offchain computation that can be verified onchain), including providing ML predictions on gas slack; estimations on app discomfort resulting from configuration changes; and other long-term predictions on network vitality. (Generally speaking, the offchain computation can be done using machine learning predictions based on time-based data analysis; and the onchain verification using ZK proofs – zero-knowledge proofs, that allow validators to prove that

a computation was done correctly without revealing the underlying data; this is left for future research.)

- An optimization (written below) that sends to the coreChain assignment commands as well as controls adjustments to the payments apps are making/receiving (i.e., to technically accommodate fining of apps and infrastructure operators for wrong declarations as well as compensations – in particular, apps pay base fee for the ML operations; should have collateral to cover its fee2gas and gas commitments; and perhaps be receiving/paying for compensations due to the picking of a solution uniformly at random).
- opChain validators solve the optimization by running a solver (e.g., Gurobi; if needed, doing incremental local search based on the solution of the last Epoch – a thorough discussion of how to solve the optimization more efficiently is given below). Note that, the optimization is to be computed off-chain with on-chain verification, similarly to the ML part.
- In typical distributed ledger systems, applications are not explicitly identified by the runtime environment; smart contracts operate in isolation without a native association to higher-level app identities. However, our architecture requires a more structured approach to enable accurate monitoring and optimization. Specifically, we assume two identification mechanisms: (1) all smart contracts associated with an application can be distinguished, allowing the system to aggregate resource usage and performance metrics at the app level; and (2) applications can be, in principle, roughly classified into types based on their code. This classification is not a clear must-have but useful for guiding the optimization process, as it enables more tailored resource allocation strategies and the use of predictive models that are specific to application categories.

4 Formal Model

We now present the formal mathematical framework underlying the responsive multichain distributed ledger architecture. The framework formulates a *multi-objective optimization problem* that balances several partially competing goals:

- **Application benefit** – captures the net benefit to an application, based on effective gas allocation and penalties for downtime and instability, weighted according to application preferences.
- **Infrastructure operator benefit** – reflects the expected economic return to an infrastructure operator, penalized by downtime caused by configuration change and by incurred computational cost for the execution of such configuration changes.
- **System benefit** – represents the overall performance of the system, combining global objectives such as fee throughput (fee times gas), total stake in the system, and application-type diversity.

These objectives are aggregated using weights that are defined by the system (most naturally, controlled by network governance). We now define the formal framework.

Remark 4. *Throughout, we consider time in timesteps of a single Epoch. While the specific decision does not bear mathematical significance, it is crucial to be consistent.*

4.1 Simplified Core Model

We define a simplified version of the responsive multichain distributed ledger system as a multi-objective assignment and resource allocation problem. This core model captures the essential mechanics: assigning applications and infrastructure operators to transient execution environments, provisioning gas and stake, computing utilities, and optimizing global objectives. Each execution environment is implicitly defined by the applications and infrastructure operators assigned to it.

4.1.1 Notation

We use the following notation:

- *APP*: set of applications, each denoted by app,
- *OP*: set of infrastructure operators, each denoted by op,
- *CHAIN*: set of transient execution environments, each denoted by chain.

4.1.2 Decision Variables

We have the following decision variables:

- $x_{app,chain} \in \{0, 1\}$: application app is assigned to execution environment chain,
- $y_{op,chain} \in \{0, 1\}$: infrastructure operator op is assigned to execution environment chain.

Each application and infrastructure operator must be assigned to at most one execution environment:

$\Sigma_{chain \in CHAIN} x_{app,chain} \leq 1 \quad \forall app \in APP, \qquad \Sigma_{chain \in CHAIN} y_{op,chain} \leq 1 \quad \forall op \in OP.$

Remark 5. *Note that applications and infrastructure operators are not required to be assigned to any execution environment; they may remain unassigned if doing so improves global benefit or is necessary for satisfying constraints.*

4.1.3 Inputs

Application Inputs (for each app ∈ APP):

- $gas_{app} > 0$: gas demand,
- $stake_{app} > 0$: required stake,
- $fee2gas_{app} > 0$: maximum acceptable fee-per-gas rate.

Infrastructure Operator Inputs (for each op ∈ OP):

- $gas_{op} > 0$: available gas capacity,
- $stake_{op} > 0$: available stake,
- $fee2gas_{op} > 0$: minimum acceptable fee-per-gas rate.

4.1.4 Derived Execution Environment Properties

For each chain ∈ CHAIN:

$APP_{chain} = \{app \mid x_{app,chain} = 1\}$,

$OP_{chain} = \{op \mid y_{op,chain} = 1\}$,

$Demand_{chain} = \Sigma_{app} x_{app,chain} \cdot gas_{app}$,

$Supply_{chain} = \Sigma_{op} y_{op,chain} \cdot gas_{op}$,

$Gas_{chain} = \min(Demand_{chain}, Supply_{chain})$,

$Stake_{chain} = \Sigma_{op} y_{op,chain} \cdot stake_{op}$,

$fee2gas_{chain} = \min\{fee2gas_{app} \mid x_{app,chain} = 1\}$.

4.1.5 Feasibility Constraints

Fee Compatibility:

$y_{op,chain} = 1 \Rightarrow fee2gas_{op} \leq fee2gas_{chain}$

Stake Feasibility:

$x_{app,chain} = 1 \Rightarrow Stake_{chain} \geq stake_{app}$

Remark 6. *Our current framework sets the minimum stake for an execution environment as the maximum of the individual stake requirements of the apps assigned to it. This implicitly assumes that the probability of multiple apps being attacked simultaneously is negligible. If such*

correlation exists, a higher stake—e.g., a multiplicative factor over the maximum—may be needed to reflect joint risk. Put differently, apps specify their stake needs independently, without considering co-location effects or the presence of other apps on the same execution environment. A more complete framework could account for inter-app risk, but we leave this extension to future work (and discuss it briefly in Section 8).

4.1.6 Utility Definitions

Application Utility:

$$\text{AppBaseUtil}_{app} = (\text{Gas}_{chain} / \text{Demand}_{chain}) \cdot \text{gas}_{app} \quad \text{if } \Sigma_{chain}\, x_{app,chain} > 0, \text{ else } 0$$

Remark 7. *The application utility function assumes that when total demand on an execution environment exceeds its gas supply, gas is allocated proportionally to each app's requested amount. While this is a natural and fair baseline—ensuring that all apps are treated equally relative to their demand—other allocation methods could be considered, but we leave this for future research.*

Infrastructure Operator Utility:

$$\text{Fee}_{chain} = \text{fee2gas}_{chain} \cdot \text{Gas}_{chain}$$

$$\text{OpFinalUtil}_{op} = (1 / \text{Stake}_{op}) \cdot \Sigma_{chain}\, y_{op,chain} \cdot (\text{Fee}_{chain} / \Sigma_{op'}\, y_{op',chain})$$

Remark 8. *In this formulation, we indeed split the total fee on an execution environment equally among all infrastructure operators assigned to that execution environment. While this simplification is suitable for our framework, real-world systems may include differentiated roles among infrastructure operators (e.g., block leaders) who may receive a larger share of the rewards (and this may also be correlated to their stake). Incorporating such role-based reward structures may affect infrastructure operator incentives and benefit, but is beyond the scope of the current framework.*

System Utility:

$$\text{SysUtil} = \Sigma_{chain}\, \text{Fee}_{chain}$$

4.1.7 Objective Function

$$\max \quad \lambda_{app} \cdot \Sigma_{app}\, \text{AppFinalUtil}_{app} + \lambda_{op} \cdot \Sigma_{op}\, \text{OpFinalUtil}_{op} + \lambda_{sys} \cdot \text{SysUtil}$$

where $\lambda_{app}, \lambda_{op}, \lambda_{sys} \geq 0$ are tunable weights (for normalization, summing up to one).

4.2 Roadmap for the Full Model

The simplified core model defines the essential structure of the responsive multichain distributed ledger system: applications and infrastructure operators are assigned to transient execution environments, which are then constructed to provision gas and stake, compute benefit, and optimize objectives.

To support a richer, more realistic deployment of the system, we modularly introduce additional layers of complexity. Each module builds on the core without disrupting its structure, allowing

for gradual implementation and independent analysis. The following roadmap outlines these extensions:

8. **Gas Overprovisioning (Slack Parameter γ)** – To mitigate the effects of estimation errors in gas demand declarations, we introduce a slack factor $\gamma > 1$, allowing execution environments to compute more gas than strictly demanded by the assigned applications. This softens under-utilization risks and improves robustness to demand misreporting (note that we penalize only for needing less gas than asking for, not for needing less than needing plus the overprovisioning).
9. **Capability Compatibility** – Applications may require (or explicitly avoid) certain operational capabilities (e.g., computational capabilities, support for EVM/WASM, privacy), and infrastructure operators may or may not support them. This module introduces compatibility declarations on both sides and imposes constraints on execution environment composition to ensure logical feasibility.
10. **Epoch-to-Epoch Stability** – To account for the cost of reconfiguration and volatility, this module tracks prior epoch assignments and utilities. It introduces penalties for applications experiencing downtime (e.g., when reassigned to a new execution environment), applications experiencing degradation in benefit (taking into account also their changing declarations), and instability as it affects infrastructure operators, in the sense that infrastructure operators are either switching execution environments or remaining on execution environments that receive new apps.
11. **Application-Type Diversity** – System vitality may depend on maintaining a diverse portfolio of application types (indeed, 'type' may refer to both domains and sizes, although – for concreteness – below we concentrate on domains). This module introduces a target distribution over app types and penalizes deviation from it in the system benefit, using an $\ell 1$ distance between the realized and target diversity vectors (other distances are possible of course, however $\ell 1$ is a natural first choice).

After describing these extensions, we describe how the full multi-objective optimization is implemented, including possible derived compensations for applications.

5 Theoretical Analysis

We now discuss the key theoretical properties of our architecture, focusing on incentive guarantees, fairness, and efficiency. We explain these intuitively and highlight why they are important for the system.

5.1 Incentive Properties (Strategy-resistance)

A major goal of the system is to make sure that apps and infrastructure operators are incentivized to declare their true needs and preferences (both in times where supply is greater than demand and vice versa). In particular:

- **Regarding declared gas needs**
  - If an app declares a too-high gas need then it risks being fined.
  - If an app declares a too-low gas need then it risks being assigned to a less adequate execution environment.

(The same holds for infrastructure operator declarations.)

- **Regarding declared fee to gas**
  - If an app declares a higher fee-to-gas bid than it is really willing to pay, it may end up assigned to an execution environment where it cannot afford the cost.
  - If an app declares a lower fee-to-gas bid than it can actually pay, it risks being excluded from the system entirely, as the optimizer will prioritize other apps.

This means that the safest and most beneficial strategy for apps is to report their true preferences — we call this property *strategy-resistance*.

Remark 18. *A possible collusion point involves an app and an infrastructure operator (or several thereof) coordinating to inflate the app's gas usage—e.g., by having the infrastructure operator prioritize transactions for the app to consume more gas than necessary. This is technically feasible since applications are distinguishable and gas usage can be attributed on-chain. However, such behavior can be detected post-hoc via on-chain analytics and penalized by governance or automated fines.*

5.2 App Clustering and Fee Stratification

We explore the expected emergent behavior of the system with regard to how apps are grouped into transient execution environments—particularly, the number of apps per execution environment and the variance in their declared fee-to-gas bids. We argue, intuitively, that the optimization will naturally lead to a stratified 'tier system' of execution environments with relatively homogeneous app bids, and illustrate the underlying dynamics through reasoning and examples. (Simulating the system to get a more formal conclusion is discussed below.)

Incentive Forces Behind Clustering. The number of apps per execution environment is shaped by competing forces within the optimization. On one hand, there is positive pressure to cluster apps together: assigning more apps to an execution environment increases the total fees the execution environment generates, and allows more apps to be served – in particular, as each

infrastructure operator is assigned only to a single execution environment, and as apps need certain lower bound on the total stake in an execution environment, the number of infrastructure operators (and the needs of the apps) essentially upper bounds the total number of execution environments. On the other hand, there is negative pressure from the fact that each execution environment's effective fee-to-gas ratio is determined by the lowest bid among its apps; thus, including low-bidding apps reduces the benefit the system (network) receives from higher-bidding apps in comparison to assigning higher bidding apps to other, higher-fee execution environments (and the infrastructure operator yield). While the optimizer can tolerate small variations in declared fees within an execution environment—particularly when doing so improves infrastructure operator yield or reduces disruption—it is penalized heavily for large variance. As a result, we expect the optimization to produce a tiered execution environment structure: clusters of apps with similar fee-to-gas bids, each group occupying its own execution environment. High-paying apps will tend to form small, exclusive execution environments (as, naturally, they are willing to pay more to get better service); lower-paying apps may form larger execution environments. (Note that, while it is possible for apps and infrastructure operators to strategize – above we discussed the informal strategy-resistance of the optimization, at least when collusion is not considered.)

5.3 Social Welfare and Congestion

It is useful to consider the possible reasons for congestion:

- Infrastructure Operator/Apps were wrong/lying
- Supply is less than demand

For the first reason: the ML predictions and the fining of lying infrastructure operators/apps gives a remedy (at least against wrong declarations; the detection of lies – or simply wrong declarations – is another challenge). For the second reason, the global aim at maximizing also infrastructure operator benefit may give incentive for new infrastructure operators (improving the demand/supply ratio, and also improving the stake and computational power of infrastructure operators); and the compensation over random solutions gives equality even when supply is lagging behind demand (the congestion itself cannot be avoided immediately).

5.4 Fairness Through Randomization

When there are multiple equally good (optimal) solutions, the system selects one at random. To make sure this random choice does not unfairly benefit or hurt any app, the system adjusts the app's payments using the formula:

$$\text{adjustment}_i = \text{util}_i - E[\text{util}_i],$$

where util_i is the app's benefit in the selected solution, and $E[\text{util}_i]$ is its average benefit across all optimal solutions. This ensures that apps are treated fairly in expectation. (It is a form of ex-ante fairness.)

We now show that the total compensation across all apps is zero:

$$\Sigma_i \text{comp}_i = \Sigma_i (E[\text{util}_i] - \text{util}_i) = E[\Sigma_i \text{util}_i] - \Sigma_i \text{util}_i = U^* - U^* = 0.$$

This confirms that, while individual apps may receive more or less benefit in a particular optimal solution compared to their expected benefit, the total 'gain' equals the total 'loss', preserving fairness in expectation.

5.5 Computational Efficiency

The mathematical optimization problem we solve can naturally be solved using standard mathematical programming solvers, such as Gurobi. Such solvers are efficient, however when the size of the program (e.g., the number of variables) become too large, they may be too slow. Consider the following, back-of-the-envelope computation.

Rough estimation For having a rough estimate, we can estimate the computational feasibility of our mathematical program by estimating the number of variables. Specifically, recalling that n denotes the number of infrastructure operators, ℓ the number of applications, and $m \leq \ell$ the number of execution environments; we have that the total number of variables in the program is $O((\ell + n) \cdot m)$, which is quadratic in the input size (and similarly for the number of constraints). As a rough benchmark, with hundreds of infrastructure operators and apps, modern solvers can produce an optimal solution in reasonable time. However, for reasonably-sized instances – e.g., with 10,000 infrastructure operators and 1000 apps, we get millions of variables, which in itself may be too demanding for standard solvers.

Luckily, however, the optimization problem we are considering is essentially an assignment problem – and such that each app and each infrastructure operator is assigned to (at most) a single execution environment; thus, the optimization matrix is quite sparse, which reduces the time complexity it takes to produce an optimal solution. Furthermore, to increase the sparsity of the program, we can add constraint (as constants) ensuring that, e.g., an app that needs EVM will not be able to be assigned (the variable assignment will not be possible; explicitly setting such variables to 0) to an execution environment with infrastructure operators without EVM support.

Furthermore, note, that in large-scale deployments of our architecture, we expect that – at stable times; i.e., not during build up period and not right after major market events – only a small fraction of the system state—such as a few infrastructure operators or apps—will change between successive Epochs. As such, solving the optimization problem from scratch in each round may be unnecessary and inefficient.

Below we discuss some solution approaches to speed up the computation.

Solution approaches First, we discuss classical optimization techniques. There are several established techniques that exploit temporal locality to improve the computational performance. Note, however, that-to guarantee the fairness via randomization and compensation as discussed above-our architecture also requires selecting an optimal solution uniformly at random, which limits the applicability of certain heuristic approaches out-of-the-box.

First, most solvers support warm starting, where the previous optimal solution—if still feasible under the new input—is supplied to the solver to initialize its internal state. Second, we can use incremental reoptimization: the approach is a kind of local search, in which we constrain the new solution to be within a bounded distance from the previous one (i.e., in each iteration we gradually increase the permissible local change; this is useful if, at a certain distance, the

disturbance cost and computational cost is so high so that the optimizition can be sure that no optimal solution is to be found in a larger distance).

Third, certain general algorithm engineering techniques may prove to be useful here as well; in particular: (1) we can use parallelization – e.g., Gurobi supports it, but the effectiveness of this is hard to estimate before-hand, as it depends on the structure of the program; (2) fine-tuning the solver (also based on learning).

Lastly, note that our optimization architecture naturally enables the creation of a market in which external actors can propose candidate solutions. Since the feasibility of any proposed solution—disregarding the uniform random sampling requirement—can be efficiently verified, this allows for an open, permissionless environment where various heuristics or solver implementations may compete. Such a market could incentivize the development of fast, specialized solvers or innovative approximation techniques, provided they yield valid configurations.

6 A Simulation Plan

While our work has focused primarily on theoretical modeling and analysis, simulations are crucial for validating the practical applicability of the proposed architecture. Below, we outline a simulation plan, beyond the scope of the current paper, but important for future investigation. We describe the plan according to three primary simulation goals and methodologies.

6.1 Simulation Goals

The concrete goals of the simulation is to answer these questions:

- **Runtime scaling:** How does the optimization runtime scale with the number of apps and infrastructure operators? With different configurations? Optionally, also **Algorithm engineering:** How much speedup is achieved using warm starts, local search over solving from scratch, parallelization, solver optimization and other algorithm engineering techniques?
- **Solution topology:** How many execution environments are typically formed, how many apps are assigned per execution environment? Also, **Fee clustering:** What is the variance of fee-to-gas bids within execution environments? Does the optimization naturally group similar bids?
- **Strategic behavior:** Do apps benefit from misreporting gas needs or fee-to-gas bids? Does the system discourage such behavior over time? How long does it take for the system to stabilize after external changes?

6.2 Simulation Work Plan

We describe a staged simulation work plan that mirrors the modular architecture of our formal model. Each stage builds incrementally, adding complexity in a controlled manner and enabling empirical evaluation of specific design elements. The plan consists of five stages:

Stage 1: Core Simulation Loop (Minimal Model).

- Implement a basic end-to-end simulation pipeline: Generate random instances with uniformly sampled application and infrastructure operator parameters (e.g., gas, stake, fee). Solve the core optimization model using a standard solver (e.g., Gurobi). Log assignment outputs, utilities, and solver runtime.
- Run preliminary scaling experiments to assess computational feasibility (of the core optimization model) with increasing instance sizes. We wish to reach something like thousands of apps and thousands of infrastructure operators. And we have in mind something like distributions that represent real-life capabilities of, say, 100K tps.

Stage 2: Modular Simulation Infrastructure.

- Refactor simulation code to support modular inclusion of the model extensions: Gas overprovisioning (slack factor γ), Capability compatibility constraints, Epoch-to-epoch penalties (downtime, reconfiguration, degradation), Application-type diversity objectives, Multi-objective optimization and weights.

- Generate random instances and run preliminary scaling experiments with the complete model. (If needed, consider also solver optimization to improve running time.)

Stage 3: Extended Instance Distributions.

- Expand the instance generator to support a wider variety of structural patterns and realistic assumptions; in particular, include support for: Correlated application and infrastructure operator parameters (e.g., stake vs. gas needs), Clustered applications (e.g., by declared capabilities or types), Heterogeneous bid distributions, Capability sparsity and diversity scenarios, Epoch-linked historical data to support degradation, stability modules, and parameters (such as the ML slack; most probably for future work, though).
- The goal is to evaluate the model under diverse, more representative input regimes. Practically, standard distributions may help (e.g., not only uniform but also normal and exponential); ideally, some more data-based distributions can be considered.

Stage 4: Full-Model Static Simulations.

- Analyze solver performance under full-model complexity and compare solver configurations (e.g., warm starts, heuristics, parallelization).
- Evaluate solution topology for the complete model under all distributions; in particular, implement and analyze the following structural measures: Solution topology: How many execution environments are formed? What is the distribution of the number of applications per execution environment? How are infrastructure operators distributed? Fee clustering: What is the variance of fee-to-gas bids within execution environments? Does the optimization naturally group similar bids? Efficiency loss from single-fee constraint: What is the welfare loss compared to a hypothetical baseline allowing per-app fees? Randomness impact: How much benefit variance is introduced by selecting uniformly among optimal solutions? Related, how much compensation is typically used? To what extent instability occurs? How it is distributed across apps? How much assigned infrastructure operator benefit is not utilized, across execution environments? What is the distribution of effective waste? Fairness towards infrastructure operators and fairness towards apps – e.g., plot the Gini index of utilities for apps and infrastructure operators.

Stage 5: Multi-Epoch Strategic Dynamics.

- Extend simulation to multiple epochs, allowing agent behavior to evolve: Applications and infrastructure operators update bids/preferences based on observed utilities, observed infrastructure operator utilization, and other measures of supply vs. demand. Simple behavior rules (e.g., reinforcement learning, myopic best response). Dynamic entry and exit of agents over time (e.g., when infrastructure operators/apps get too bad of a service – this includes the relation of fairness properties and the extent of infrastructure operator/apps churn).
- Investigate: Convergence patterns, Bid inflation or other emergent behavior, Robustness of fairness and incentive properties under repeated play (this also could/should relate to apps/infrastructure operators churn, as it relates to modeling dynamic population of apps/infrastructure operators), Strategy-resistant aspects.

7 Mathematical Program

For completeness (and for helping the simulation), we write the full mathematical program (in a modular way). We start with the basic model.

7.1 Mathematical Program of the Basic Model

Constants: APP: set of applications

OP: set of infrastructure operators

CHAIN: set of transient execution environments

$gas\_app > 0 \quad \forall app \in APP$

$stake\_app > 0 \quad \forall app \in APP$

$fee2gas\_app > 0 \quad \forall app \in APP$

$gas\_op > 0 \quad \forall op \in OP$

$stake\_op > 0 \quad \forall op \in OP$

$fee2gas\_op > 0 \quad \forall op \in OP$

$\lambda\_app, \lambda\_op, \lambda\_sys \geq 0, \quad \lambda\_app + \lambda\_op + \lambda\_sys = 1$

Decision Variables: $x\_app,chain \in \{0, 1\} \quad \forall app \in APP, chain \in CHAIN$

$y\_op,chain \in \{0, 1\} \quad \forall op \in OP, chain \in CHAIN$

Derived Quantities:

$Demand\_chain = \Sigma\_app \in APP \; x\_app,chain \cdot gas\_app \quad \forall chain \in CHAIN$

$Supply\_chain = \Sigma\_op \in OP \; y\_op,chain \cdot gas\_op \quad \forall chain \in CHAIN$

$Gas\_chain = \min(Demand\_chain, Supply\_chain) \quad \forall chain \in CHAIN$

$Stake\_chain = \Sigma\_op \in OP \; y\_op,chain \cdot stake\_op \quad \forall chain \in CHAIN$

$Fee2gas\_chain = \min\{fee2gas\_app \mid x\_app,chain = 1\} \quad \forall chain \in CHAIN$

$Fee\_chain = Fee2gas\_chain \cdot Gas\_chain \quad \forall chain \in CHAIN$

$AppBaseUtil\_app = \Sigma\_chain \in CHAIN \; x\_app,chain \cdot (Gas\_chain / Demand\_chain) \cdot gas\_app$

$OpBaseUtil\_op = (1 / stake\_op) \cdot \Sigma\_chain \; y\_op,chain \cdot (Fee\_chain / \Sigma\_op' \; y\_op',chain)$

$SysUtil = \Sigma\_chain \in CHAIN \; Fee\_chain$

Constraints:

$\Sigma\_chain \in CHAIN\ x\_app,chain \leq 1 \quad \forall app \in APP$

$\Sigma\_chain \in CHAIN\ y\_op,chain \leq 1 \quad \forall op \in OP$

$y\_op,chain = 1 \Rightarrow fee2gas\_op \leq Fee2gas\_chain \quad \forall op, chain$

$x\_app,chain = 1 \Rightarrow Stake\_chain \geq stake\_app \quad \forall app, chain$

Objective:

$\max \quad \lambda\_app \cdot \Sigma\_app \in APP\ AppBaseUtil\_app + \lambda\_op \cdot \Sigma\_op \in OP\ OpBaseUtil\_op + \lambda\_sys \cdot SysUtil$

8 Limitations and Generalizations

We discuss a set of practical considerations, limitations of the current formulation, and possible generalizations. While the preceding sections present a streamlined model for clarity and analytical tractability, realistic deployment is likely to require addressing additional factors and relaxing certain assumptions. The discussion below identifies aspects that are either omitted from the formal optimization or simplified in the current design, and outlines directions for extending the architecture to better align with real-world requirements.

8.1 Supporting Opting-Out

While the system is designed to manage applications through optimization for improved efficiency and fairness, it does not mandate participation. Specifically, applications may choose to *opt out* of the optimization architecture. Opt-out apps are excluded from the assignment and fee-adjustment mechanisms, and are assumed to bring their own execution resources (e.g., their own infrastructure operators), thus remaining unmanaged by the system.

To account for their continued use of shared infrastructure (most concretely, the ML analysis done by the architecture), and their impact of their opting-out on the global network vitality, such apps are charged an opt-out tax. This tax can be implemented in multiple ways: a simple base opt-out tax that reflects infrastructure usage (such as the ML used), a marginal-cost opt-out tax based on the reduction in overall system benefit caused by their non-participation (akin to VCG pricing), or a blend of the two. The marginal-cost approach may require re-solving or approximating the optimization without the opt-out app, and thus introduces additional complexity. The structure of this tax—whether fixed, dynamic, or hybrid—may ultimately be determined via governance.

Importantly, the architecture incentivizes participation by offering system-level advantages to managed apps. These include improved individual benefit through ML-driven optimization, and better service due to placement in well-provisioned execution environments with sufficient infrastructure operator stake and computational power. Thus, for most applications—except potentially very large ones with bespoke needs—joining the global optimization is both rational and beneficial.

8.2 Controller-Free Applications

The architecture is designed to accommodate not only applications with identifiable economic owners and logical controllers, but also *controller-free applications*—that is, decentralized applications that operate as public infrastructure without a central profit-seeking entity. These may include common-good protocols (like fire-and-forget utility programs/tools), community-maintained services, or application-oriented such as management applications of the distributed ledger ecosystem.

In our architecture, controller-free applications can be supported in two complementary ways:

- **Default declarations via ML or policy:** Since such applications cannot submit bids directly, the system can generate proxy declarations using historical usage data, machine learning predictions, or governance-defined templates. These declarations allow the

optimization model to integrate controller-free apps without sacrificing the integrity of resource allocation or fairness mechanisms.

- **Cost recovery through user payments:** Because there is no dedicated party responsible for covering costs (e.g., to cover the reparation costs), users interacting with controller-free apps may incur a per-transaction charge. This user-pays model ensures sustainability without violating decentralization. (However, indeed, at some cost to user experience – in the sense of users have to decide what value to put, similarly to the user experience in, e.g., Ethereum maximum gas; a more detailed discussion on possible variants, e.g., where a controller-free-app has some debt to be paid by future users, is beyond the scope of the current discussion.)

Remark 24. *Indeed, applications with a financial owner may attempt to falsely present themselves as controller-free in order to avoid committing collateral or circumvent bidding penalties. This behavior can be mitigated through governance oversight (say, a dedicated committee) and as such apps have smaller impact on how they are assigned; also, we can, e.g., disallow such apps from using authentication and authorization code to limit their commercial viability as well as use ML techniques to identify such false declarations (using standard cyber technologies).*

Overall, the system treats controller-free applications as first-class citizens, offering explicit support for their integration and sustainability within a principled optimization architecture. This expands the benefit and inclusiveness of the multichain network beyond purely revenue-driven actors.

8.3 Capability Requirements

Capabilities refer to specific features that an execution environment or execution environment may support. Applications may have different requirements with respect to these capabilities: they may require a capability, forbid it (e.g., due to performance or complexity concerns), or express indifference. Examples of capabilities include: Execution environment: EVM or WASM. An application typically uses one and not both. Privacy support: Some applications require privacy, others require public transparency. Data availability model: Whether the data is stored on-chain or through an external mechanism.

Each application specifies its capability requirements using a ternary vector. Formally, we model each capability by a value $\text{cap} \in \{-1, 0, 1\}$, where: $\text{cap} = 1$: the application requires capability, $\text{cap} = 0$: the application forbids capability, $\text{cap} = -1$: the application is indifferent to capability.

(Note that, while app specify ternary values as described above, infrastructure operators do the same, with the semantics of: 1 meaning support the capability; 0 does not support; −1 may support or not depending on context.)

This representation enables compact expression of requirements while preserving optimization flexibility.

9 Outlook

We have described a architecture for a responsive multichain distributed ledger using a multi-objective optimization framework. We have formally described the requirements from the architecture, its specific implementation, and theoretically analyzed its properties.

To conclude, this work provides a structured optimization-based architecture for dynamically allocating apps and infrastructure operators across multiple transient execution environments, while taking into account utility, fairness, and operational feasibility. By combining concrete system constraints with flexible objectives informed by apps and infrastructure operator bidding as well as machine learning, the model enables informed network optimization in dynamic distributed ledger environments.

Overall, the architecture provides:

- Incentive alignment (apps and infrastructure operators have no reason to lie).
- Efficient clustering by bid levels.
- Support for multiple capabilities and flexible bidding.
- Fair treatment across random solutions.
- Options for scaling up with incremental methods.

These features make the system both practical and robust.

We hope this architecture serves as a foundation for both practical deployment and continued theoretical investigation.

References

[1] Rajeev Arora, Anoop Kumar, Arpita Soni, and Aniruddh Tiwari. AI-driven self-healing cloud systems: Enhancing reliability and reducing downtime through event-driven automation. Preprints, page 2024081860, 2024.

[2] Dimitris Bertsimas and John N. Tsitsiklis. Introduction to Linear Optimization. Athena Scientific, 1997.

[3] Vitalik Buterin. An incomplete guide to rollups, 2020. https://vitalik.ca/general/2021/01/05/rollup.html.

[4] Jeeta Ann Chacko, Ruben Mayer, and Hans-Arno Jacobsen. How to optimize my blockchain? A multi-level recommendation approach. Proceedings of the ACM on Management of Data, 1(1):1–27, 2023.

[5] Jeeta Ann Chacko, Ruben Mayer, and Hans-Arno Jacobsen. A comprehensive outlook for analyzing and enhancing the performance of blockchain platforms. Proceedings of the VLDB Endowment. ISSN, 2150:8097, 2024.

[6] Ming Chen, Uzair Khan, Chen-Feng Liu, Mérouane Debbah, Mérouane Debbah. Machine learning for wireless networks with artificial intelligence: A tutorial on neural networks. In IEEE Transactions on Cognitive Communications and Networking, volume 4, pages 46–61, 2018.

[7] Edward H. Clarke. Multipart pricing of public goods. Public Choice, 11(1):17–33, 1971.

[8] Kalyanmoy Deb. Multi-Objective Optimization using Evolutionary Algorithms. Wiley, 2001.

[9] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In NSDI, 2011.

[10] Arpita Ghosh and David C. Parkes. Anonymous randomized allocations. In International Joint Conference on Artificial Intelligence (IJCAI), 2010.

[11] Theodore Groves. Incentives in teams. Econometrica, 41(4):617–631, 1973.

[12] Mahimna Kelkar, Soubhik Deb, and Sreeram Kannan. Order-fair consensus in the permissionless setting. In Proceedings of the 9th ACM on ASIA Public-Key Cryptography Workshop, pages 3–14, 2022.

[13] Jon Kleinberg, Manish Raghavan, and Sendhil Mullainathan. Fairness in decision-making: Ex-ante vs. ex-post. In ACM Conference on Economics and Computation (EC), 2018.

[14] Jan Kossmann. Self-driving: From general purpose to specialized dbmss. In PhD@ VLDB, 2018.

[15] Jan Kossmann and Rainer Schlosser. Self-driving database systems: a conceptual approach. Distributed and Parallel Databases, 38:795–817, 2020.

[16] Jae Kwon and Ethan Buchman. Cosmos: A network of distributed ledgers, 2019. https://github.com/cosmos/cosmos/blob/master/WHITEPAPER.md.

[17] Mingxuan Li, Yazhe Wang, Shuai Ma, Chao Liu, Dongdong Huo, Yu Wang, and Zhen Xu. Auto-tuning with reinforcement learning for permissioned blockchain systems. Proceedings of the VLDB Endowment, 16(5):1000–1012, 2023.

[18] Conor McMenamin. Sok: Cross-domain mev. arXiv preprint arXiv:2308.04159, 2023.

[19] Hervé Moulin. Incremental cost sharing: Characterization by coalition strategy-resistance. Social Choice and Welfare, 16(2):279–320, 1999.

[20] Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani, editors. Algorithmic Game Theory. Cambridge University Press, 2007.

[21] David C. Parkes. Iterative combinatorial auctions: Achieving economic and computational efficiency. In Proceedings of the 17th National Conference on Artificial Intelligence (AAAI), 2001.

[22] SD Prestwich and S Verachi. Constructive vs perturbative local search for general integer linear programming. In Proceedings of the Fifth International Workshop on Local Search Techniques in Constraint Satisfaction (LSCS), 2008.

[23] Ted Ralphs and Menal Güzelsoy. Duality and warm starting in integer programming. In The proceedings of the 2006 NSF design, service, and manufacturing grantees and research conference, 2006.

[24] Ted Ralphs, Yuji Shinano, Timo Berthold, and Thorsten Koch. Parallel solvers for mixed integer linear optimization. In Handbook of parallel constraint reasoning, pages 283–336. Springer, 2018.

[25] Tim Roughgarden. Transaction fee mechanism design for the ethereum blockchain: An economic analysis of eip-1559. arXiv preprint arXiv:2012.00854, 2020.

[26] Erel Segal-Halevi and Ariel D. Procaccia. Fair allocation of indivisible goods: Improvements and limitations. In International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2020.

[27] Danny Weyns, M Usman Iftikhar, Didac Gil De La Iglesia, and Tanvir Ahmad. A survey of formal methods in self-adaptive systems. In Proceedings of the fifth international c* conference on computer science and software engineering, pages 67–79, 2012.

[28] Dominic Williams. The internet computer for geeks, 2021. https://internetcomputer.org/whitepaper.pdf.

[29] Terence Wong, Markus Wagner, and Christoph Treude. Self-adaptive systems: A systematic literature review across categories and domains. Information and Software Technology, 148:106934, 2022.

[30] Gavin Wood. Polkadot: Vision for a heterogeneous multi-chain framework, 2016. https://polkadot.com/papers/Polkadot-whitepaper.pdf.

[31] Chenyuan Wu, Bhavana Mehta, Mohammad Javad Amiri, Ryan Marcus, and Boon Thau Loo. Adachain: A learned adaptive blockchain. arXiv preprint arXiv:2211.01580, 2022.

[32] Yiguang Zhang, Junxiong Lin, Zhihui Lu, Qiang Duan, and Shih-Chia Huang. Pbrl-tchain: A performance-enhanced permissioned blockchain for time-critical applications based on reinforcement learning. Future Generation Computer Systems, 154:301–313, 2024.