Improving resistance to Sybil attacks in decentralized networks using the RTCT protocol*

Ivan Parkhomenko[1,*,†], Larysa Myrutenko[2,†], Roman Ohiievych[3,†], and Oleksandr Laptiev[4,†]

[1] Taras Shevchenko National University of Kyiv, Bohdana Havrylyshyna str. 24, 04116 Kyiv, Ukraine

Abstract

Decentralized peer-to-peer networks face critical security challenges from Sybil attacks, wherein an adversary creates numerous fake nodes to disrupt network operations. This paper proposes Random Time Challenge-Tokens (RTCT) as a mechanism to economically enforce honest participation and deter Sybil attacks at the network level. Under RTCT, each node must respond to unpredictable cryptographic challenges by producing a verifiable token and burning it, incurring a recurring cost. By imposing continuous resource expenditures on every identity, RTCT raises the cost of maintaining large numbers of Sybil nodes, thus dramatically improving Sybil resistance and network stability. We present a concise protocol overview and develop a formal model for RTCT's impact on attacker costs, identity sustainability, and detection rates. Through analysis and simulation, we quantify how challenge frequency (λ), difficulty (m), network size (N), and attack horizon (T) affect the economic burden on attackers, the likelihood of Sybil detection, and overall network health. Our results show that RTCT can exponentially increase an attacker's costs with higher challenge difficulty, making networks more resilient. Compared to existing Sybil defenses – including social-graph approaches, periodic proof-of-presence tests, self-registration, and proof-of-personhood schemes – RTCT offers a decentralized and tunable method that complements consensus protocols by continuously auditing node legitimacy. The method strengthens network-level security (e.g. routing, peer discovery) without altering consensus rules, thereby mitigating Sybil-based disruptions (eclipse, spam) through sustained operational costs. We discuss parameter tuning to avoid deterring honest participants and integration strategies with existing peer-to-peer network protocols. Acknowledging its overhead, RTCT nonetheless provides a promising additional layer of defense that raises the economic bar for adversaries and improves the consistency and reliability of decentralized networks.

Keywords

Decentralized networks; Sybil attack; peer-to-peer security; challenge token; crypto-economic security; node authentication; RTCT; proof-of-work; Sybil resistance.

1. Introduction

Modern decentralized networks (e. g. peer-to-peer overlays, distributed ledgers) rely on a combination of cryptography and economic incentives to maintain security without central authorities. A classic example is Bitcoin's Proof-of-Work (PoW) consensus, where miners expend computational work (and electricity) to secure the system [4]. Similarly, Proof-of-Stake (PoS) requires validators to lock up collateral (stake) to gain participation rights, risking loss upon misbehavior (slashing) [5, 6]. These mechanisms make certain attacks economically prohibitive: for instance, a 51% attack on Bitcoin demands controlling over half the global hash power, which is financially infeasible in practice [7]. Likewise in PoS, acquiring 51% (or 2/3) of the staked tokens in a large network costs tens of billions of dollars [8, 9]. By erecting steep economic barriers, traditional consensus protocols discourage attackers from amassing enough resources to compromise the system.

* CSDP'2026: Cyber Security and Data Protection, May 7, 2026, Lviv, Ukraine
* Corresponding author.
† These authors contributed equally.
ivan.parkhomenko@knu.ua (I. Parkhomenko); myrutenko.lara@knu.ua (L. Myrutenko); ohiievychr@fit.knu.ua (R. Ohiievych); alaptev64@ukr.net (O. Laptiev)
0000-0001-6889-9284 (I. Parkhomenko); 0000-0003-1686-261X (L. Myrutenko); 0009-0003-7948-1125 (R. Ohiievych); 0000-0002-4194-402X (O. Laptiev)

However, network-level vulnerabilities remain that are not fully addressed by consensus alone [10]. One such fundamental threat is the Sybil attack [11], in which a single adversary generates a multitude of pseudonymous nodes to gain disproportionate influence. In a peer-to-peer (P2P) network, an attacker controlling many Sybil identities can manipulate overlay network operations – for example, biasing routing tables, out-voting honest nodes in protocols, or overwhelming the network with spam and misinformation [10, 12]. Unlike 51% attacks on consensus (which require enormous hash power or stake), Sybil attacks often exploit the lack of admission control in open networks: if creating a new identity is cheap, an attacker can flood the network with Sybils at little cost [13]. In Bitcoin's P2P layer, for instance, an eclipse attack can isolate a target node by surrounding it with Sybil peers without breaking PoW consensus [14]. Similarly, in permissionless distributed hash tables or anonymity networks, Sybils can distort routing or reputation systems unless checked by additional measures [15, 16].

Many decentralized systems currently assume unique identities without robust enforcement, leaving them exposed to Sybil-based disruptions. The underlying problem, as Douceur's seminal work outlined, is that without a central authority or an inherent resource cost per identity, one entity can in theory generate unlimited identities and subvert the system's integrity [11]. Thus, improving Sybil resistance at the network level (beyond the consensus layer) is critical for robust peer-to-peer operations.

RTCT (Random Time Challenge-Tokens) is proposed as a mechanism to fill this gap by introducing continuous identity costs and periodic verification of node activity. The key idea is to force every node in the network to prove its continued presence and honesty via cryptographic challenges issued at random intervals. Each node must periodically perform a specified amount of work or cryptographic response – similar in spirit to a proof-of-work puzzle – and produce a challenge token as a response [1]. Crucially, this token is then burned (destroyed) or rendered unusable after verification [17]. In economic terms, the token burning means that each challenge imposes a non-recoverable cost (e.g. CPU cycles, energy, or a small fee) on the node. Honest participants treat it as an operational expense of running a node, whereas an attacker controlling many Sybils must pay the cost for every fake node continuously.

By making identity maintenance costly and unpredictable, RTCT thwarts the economics of Sybil attacks. An attacker can no longer spawn hundreds of free identities – they would have to sustain each of them through regular challenge responses, incurring a recurring cost that scales linearly with the number of identities [2]. Even if each individual challenge is inexpensive (say a fraction of a cent or a tiny amount of CPU time), a "massive network of fake nodes" becomes very expensive over a long duration [2]. In contrast, a single honest node only bears one node's worth of cost and ideally also earns some utility or reward from participating in the network (e.g. transaction fees, service access), offsetting the expense [18]. Thus, RTCT aims to make Sybil attacks economically unviable by continuously raising the upkeep cost for malicious nodes, rather than relying on one-time barriers to entry.

Beyond cost, RTCT provides a built-in detection mechanism: nodes that fail to answer challenges are assumed to be malicious or non-functional and can be removed from the network's active membership. Because the challenges occur at random times, an attacker cannot predict or strategically evade them without risking exposure. Any Sybil node that goes inactive (to save resources) will likely miss a challenge and thus be caught and expelled. This ensures that only consistently responsive nodes remain in the network, improving churn resistance and participation consistency over time. In effect, RTCT continually "audits" the network for idle or non-compliant nodes. Honest nodes that accidentally go offline may be removed as well, but they can rejoin later; importantly, an attacker trying to juggle a large number of Sybils by cycling them online/offline will find many of them failing challenges and dropping out. The protocol thereby forces the attacker to keep all Sybils online and expending resources, or else lose them to detection.

To illustrate RTCT's expected impact, Figure 1 depicts how an attacker's daily cost grows with the number of Sybil identities (K) under different challenge difficulties (m). Even at modest complexity (e. g. $m=15$ bits), sustaining dozens of Sybils incurs non-trivial cost, and at higher complexities (e. g. $m=20$), the cost skyrockets exponentially. Meanwhile, the probability that an inactive Sybil is detected (by missing a challenge) as a function of time for various challenge frequencies (λ). With more frequent challenges (e. g. $\lambda=5$ per day), an abandoned Sybil has virtually a 100% chance of being caught within a single day, whereas infrequent challenges (e. g. 0.2 per day) still ensure detection over a week with high probability. These quantitative insights will be detailed in Section 4 using a formal model.

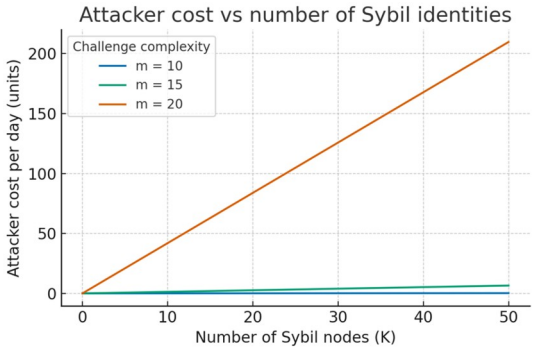

Figure 1: Attacker cost vs number of Sybil identities under different challenge difficulties (m). Higher m (more complex puzzles) exponentially increases the per-node cost, making Sybil attacks prohibitively expensive even for moderate K [2]. Lines show daily cost in arbitrary units for $m=10,15,20$ at a fixed challenge rate $\lambda=4$/day.

The remainder of this paper is organized as follows. Section 2 reviews related work on Sybil attack defenses in decentralized networks, highlighting how RTCT differs from or complements existing methods. Section 3 provides a succinct overview of the RTCT protocol and its integration into a peer-to-peer network stack. Section 4 develops a mathematical model of RTCT's security properties, quantifying attacker costs, Sybil sustainability, and detection rates. We derive formulas for the attacker's total cost to maintain Sybils as a function of λ, m, K, and T, and we evaluate network stability metrics (node churn and honest overhead). Section 5 presents simulation results and example calculations that demonstrate RTCT's effectiveness, including tables of parameter values and outcomes under various scenarios. Section 6 discusses practical considerations: choosing parameters to balance security vs. overhead, ensuring fairness for honest nodes, and deploying RTCT alongside consensus protocols. Finally, Section 7 concludes the paper, comparing RTCT to other approaches and outlining future work.

2. Related Work

Sybil attack mitigation has been extensively studied, yielding a spectrum of approaches. Broadly, defenses either impose resource constraints on identities or leverage trust assumptions to distinguish genuine nodes from Sybils [11, 16]. We briefly survey the main categories here and compare them to RTCT.

Proof-of-Work and Costly Identity: early proposals like Hashcash (used in email anti-spam) required clients to solve a proof-of-work puzzle for each identity or message. The idea is to make creating or using an identity computationally costly, thereby deterring mass Sybil generation. However, one-time PoW per identity (or per action) may not suffice against determined attackers – they can solve puzzles offline and accumulate many identities over time, as pointed out by Laurie and Clayton. Bitcoin's Nakamoto consensus itself can be seen as a Sybil deterrent at the consensus level (miners must expend continuous work), but at the P2P networking level Bitcoin nodes can

still be Sybils as long as they forward blocks/transactions correctly [14]. RTCT similarly relies on PoW-like challenges, but unlike one-time Hashcash stamps, the challenges are periodic and unpredictable, enforcing sustained work. This approach is akin to requiring each node to continually perform "mining" just to prove it is alive (albeit at a much lower difficulty than actual block mining). Concepts like Reusable Proofs of Work (RPOW) by Finney and Proof-of-Burn (PoB) by Stewart (destroying one's own coins to obtain mining rights) also influenced RTCT's design – the token burning in RTCT is analogous to spending a resource for security, but done continuously in time rather than upfront. A related idea in a recent blockchain project "Ixian DLT" is Proof-of-Computational-Work (PoCW), where each node periodically finds a hash with a certain number of leading zeros and sends it in a "presence" message. This is essentially a periodic PoW to prove the node is active and make it costly to maintain many idle nodes. RTCT generalizes this notion and formalizes the random scheduling of challenges. Compared to pure PoW-based identities, RTCT's novelty lies in its random timing and burn requirement, which ensure continual cost and prevent attackers from reusing or transferring solutions across Sybils.

Social-Graph-Based Sybil Defense: these methods assume an honest social graph where edges represent trust between real users. SybilGuard (2006) and subsequent protocols (SybilLimit, SybilFence, etc.) leverage graph-theoretic properties – under the assumption that an attacker can only insert a limited number of attack edges between Sybil nodes and honest community – to bound the number or influence of Sybils. For example, SybilGuard performs random walks on the social graph to distinguish Sybil regions, and was shown to limit the size of Sybil groups such that honest nodes accept only a small fraction of the total nodes as peers. Social network approaches are effective in environments where a natural trust graph exists (e.g. web of trust, social media networks). However, they may fail in purely P2P systems with no pre-existing trust relationships or where an attacker can fabricate social connections. They also do not economically deter Sybils; rather, they algorithmically detect them given certain assumptions. RTCT differs by not relying on any external trust; it treats all nodes equally and forces would-be Sybils to "pay as they go." In fact, RTCT could complement social defenses: one could imagine using a trust graph to decide challenge frequencies (e.g. untrusted new nodes get challenged more often) – though this is outside our present scope. The advantage of RTCT over social approaches is that it does not require any prior human trust information; its disadvantage is the ongoing cost it imposes on all participants, whereas social methods impose little cost on honest users aside from managing social links.

Periodic Proof-of-Presence (Liveness Tests): another line of defense is to require nodes to periodically prove they are a unique live entity. One variant involves AI-hard challenges or Turing tests at synchronized intervals, as seen in the Idena project's protocol. In Idena, every node must solve a sequence of flip captcha tests during a globally scheduled session; a human operator cannot reliably solve challenges for more than one node at a time, so one person cannot maintain many identities. This is a form of proof-of-personhood, discussed below. Other variants, closer to RTCT, are purely automated and require each node to perform some tasks continuously to remain active. The Ixian PoCW mentioned earlier falls in this category, as does work on using verifiable delay functions (VDFs) for rate-limiting identities. The difference between RTCT and a simple periodic check is the randomness and cryptographic verifiability of the challenge: if checks are purely periodic and have predictable intervals, an attacker might time the on-off cycling of Sybils to avoid some checks, whereas RTCT challenges arrive at random times drawn from a distribution (e.g. a Poisson process with rate λ) so that scheduling avoidance is much harder. Additionally, by varying the challenge difficulty (e.g. occasionally issuing a harder challenge), RTCT can further increase uncertainty for the attacker. Periodic liveness tests share the goal of discouraging an attacker from controlling too many nodes by increasing workload, and our work builds on this concept with a more formal economic analysis. A potential drawback of any such scheme, including RTCT, is the overhead burden on honest nodes – we address this in Section 6 by showing how parameters can be tuned so that challenges consume only a small fraction of a node's resources, analogous to a background "maintenance cost".

Self-Registration and Identity Proofing: at the weakest end of the spectrum, many networks simply allow self-signup of identities with minimal checks (maybe an email or a CAPTCHA). This low-friction admission unfortunately provides almost no Sybil resistance: a determined user can solve CAPTCHAs repeatedly or create many email accounts [13], obtaining thousands of identities. Some networks implement rate limiting or require a unique phone number, etc., but these can be circumvented or impose centralization (using phone verification ties identities to telecom providers). Essentially, pure self-registration trusts each new node by default – a design that is cheap for honest users but also cheap for attackers. RTCT can be viewed as adding a cost layer on top of self-registration. Nodes may still join permissionlessly, but they cannot remain in the network indefinitely without doing the required work. This dramatically shifts the cost model: creating 1000 identities might be easy, but keeping them online and handling continuous challenges would be astronomically expensive or computationally infeasible in aggregate. Unlike static "proof of unique email" or similar, RTCT doesn't aim to enforce one identity per person – it just ensures that any identity, human or a bot, consumes real resources. Thus, it converts the problem into an economic one (how much can an attacker afford to waste) rather than a hard identity problem.

Proof-of-Personhood (PoP): this approach, distinct from the above, tries to guarantee a one-human-one-identity property to outright prevent Sybils. PoP schemes use methods like in-person pseudonym parties, biometric verification (e. g. Worldcoin orb scanning), or another credential issuance to bind each real-world individual to at most one cryptographic identity. For example, Borge et al. (2017) introduced the concept of pseudonym parties where participants physically gather at set times to exchange cryptographic tokens that certify each person was present. Projects like BrightID and Circles use a web-of-trust of human validations to create unique identifiers. These methods can strongly mitigate Sybils because an attacker is limited by the number of real people they can recruit or fake, not just computing power. However, proof-of-personhood protocols face significant practical hurdles: coordination of events, user convenience, privacy concerns, and the risk of exclusion of those who cannot participate in the prescribed manner. They also tend to be layered on top of networks rather than an internal mechanism of the P2P network. RTCT does not attempt to establish personhood – it remains agnostic to whether an identity is human, AI, or anything else – but ensures that all identities incur cost. Thus, RTCT is complementary: a network could use PoP to initially issue one identity per human, and still apply RTCT to ensure ongoing active participation and to add an extra cost for any compromised identities. In contexts where PoP is not available, RTCT stands on its own as a purely cyber-economic defense. In summary, existing Sybil defenses either raise entry costs, exploit social trust, or periodically vet liveness (human or machine). Table 3 compares these approaches. RTCT's niche is providing a built-in, continuous cost mechanism at the networking layer without external dependencies. It does not replace other security measures (indeed, it does not prevent a 51% attack on consensus – that's still the job of PoW/PoS), but it bolsters the network's resilience against low-cost Sybil flooding and passive adversaries. Notably, RTCT can be integrated into existing networks with minimal protocol changes: it can run as a background process where nodes ping themselves or a random neighbor with challenges, as long as the rest of the network can verify tokens. For instance, by including token proofs in gossip messages or periodically publishing them to a ledger. The overhead and parameter choices must be carefully managed, which we will explore after developing the formal model.

3. RTCT protocol overview

In this section, we describe the RTCT protocol in detail, including the challenge generation process, token creation/burning, and integration with the network's messaging system. Each node in the network is periodically issued a challenge – a cryptographic puzzle or computation task – that it must solve within a given time window.

Table 1
Key parameters in the RTCT model and their definitions. These factors influence the cost and detection outcomes of Sybil attacks

Symbol	Definition	Typical Value
λ	Frequency — challenge rate per node	1 per day (tunable)
m	Puzzle bits — challenge difficulty	16-24 bits (tunable)
N	Total number of nodes in network	– (varies by scenario)
K	Number of attacker's Sybil nodes	– (attacker's choice)
T	Time horizon for analysis	7 days (example)

The challenge can be as simple as finding a nonce that produces a hash with m leading zero bits (similar to Hashcash/Bitcoin puzzles) or performing a verifiable computation. The challenge issuance is randomized: on average, each node might receive λ challenges per time unit (e. g. per day), but the exact timing is unpredictable (modeled as a Poisson process with rate λ). This unpredictability prevents an attacker from scheduling around the challenges. When a node receives a challenge, it computes the answer (e.g. finds a valid nonce or completes the required computation) and packages the result into a challenge token – essentially a proof that it completed the work. This token could be a tuple like $\left(\text{nodeID}, t_{\text{issue}}, \text{nonce}, \text{hash}\right)$ which others can verify. Crucially, once verified, this token is immediately burned, meaning it cannot be reused or transferred to another node. In practice, "burning" could mean that the token includes a nonce that can only be used once (e.g. it's tied to the challenge timestamp) or that the node must send the token to a specific sink (like a transaction fee to a burn address if using a blockchain ledger for logging). The result is that the node has expended effort (or currency, if the challenge required a micropayment) with no direct gain – it is purely a cost paid to remain in good standing.

Verification and enforcement: how does the network enforce that nodes complete their challenges? There are a few implementation possibilities:

1. Distributed witnessing: nodes could broadcast their challenge tokens to a random subset of peers or to the entire network. Peers verify the token and relay it (similar to how blocks or gossip messages propagate). If a node fails to broadcast a valid token for a challenge it was known to receive, other nodes mark it as unresponsive. Because challenge timing is random, a node cannot predict when it needs to produce a token; if it crashes or is a fake identity with insufficient resources, it will miss a challenge and peers will notice. A simple heuristic is that if a node misses X consecutive challenges or Y out of the last Z challenges, it is evicted from routing tables or its messages are no longer accepted.
2. Periodic checkpointing: in a blockchain setting, challenge tokens could be recorded on-chain (e.g. in special transactions or in block headers). For instance, every block producer might be required to include valid tokens from a certain fraction of network nodes (randomly selected) in each block. This would tie RTCT into the consensus process. For more general P2P networks (not blockchain), a decentralized ledger of tokens could be maintained (using a DHT or similar) where tokens are timestamped. The design must ensure that generating a fake token is cryptographically infeasible without actually doing the work – a property ensured by the puzzle design.
3. Local and neighbor monitoring: alternatively, each node's immediate neighbors (in an overlay) could be responsible for issuing it challenges and verifying responses. For example, every node periodically challenges each of its connected peers. If a peer fails, it's dropped. This has the advantage of scaling with the network (no global broadcasts needed), though it could be vulnerable if an attacker somehow clustered Sybils together (they might "cover" for each other's absence; to prevent that, one can ensure challenge seeds are globally coordinated or use diversified challengers).

Regardless of the mechanism, the aim is that honest nodes will almost never fail (they might miss a challenge only if temporarily down or overloaded, which could be mitigated by allowing a small failure rate or retries), whereas Sybil nodes under an attacker's control will gradually accumulate failures unless the attacker dedicates disproportionate resources to each of them. This in effect thins out Sybils over time, because any nodes that do not continuously prove themselves are removed.

RTCT introduces two main tunable parameters: the challenge rate λ and the difficulty m. These can be adjusted based on network conditions: – A higher λ (more frequent challenges) means quicker detection of inactive Sybils but higher overhead for all nodes. – A higher m (harder challenges) means greater cost per challenge (Figure 1 shows this effect) but also more work for honest nodes.

An optimal setting might be to impose a mild but steady load. For example, assume each challenge is equivalent to ~10^{-5} CPU-seconds (which might be, say, $2^{20} \approx 1$ million hash operations) and $\lambda = 2$ per hour. Then an honest node expends roughly 2×10^{-5} CPU-seconds per hour, which is negligible on a modern machine. However, an attacker with 1000 Sybil nodes must handle 1000×2 challenges/hour = 2000 challenges/hour, totaling 2×10^{-2} CPU-seconds/hour – still small in absolute terms, but if challenges are made harder (m larger) or if the attacker tries to scale to 1e6 nodes, the cost becomes prohibitive. Table 2 and the analysis in Section 4 will shed more light on realistic values.

To make RTCT concrete, imagine a decentralized chat network with nodes identified by public keys [19]. The RTCT service runs as follows:

1. Challenge generation: a global pseudorandom beacon (which could be a small proof-of-stake chain or a collective randomness protocol) emits a random seed every few minutes. Each node, using this seed and its own ID, computes whether it is challenged in that interval (for a target rate λ). Suppose seeds are published every minute, and each node has a $\lambda / 60$ probability of being picked per seed.
2. Challenge Execution: node A finds that the current seed triggers a challenge for itself. The challenge is to find a value x such that $H(\mathrm{ID}_A | \mathrm{seed} | x)$ has m leading zero bits. Node A begins hashing to find such an x. This is essentially a mini mining task just for node A. Token broadcasting: after some seconds, A finds a solution x^2. It forms a token containing $(\mathrm{ID}_A, \mathrm{seed}, x' H(\cdot))$ and signs it. It then either sends this token to a set of neighbor nodes or publishes it in the network's token ledger.
3. Verification: neighbors or any other nodes verify that $H(\mathrm{ID}_A | \mathrm{seed} | x^*)$ indeed has m leading zeros and that *seed* was the correct challenge seed for that round. The token is valid, so they mark A as responsive for this round.
4. Burning: the token's use is solely to prove work for that specific seed, so it has no transferable value. If the network had an integrated cryptocurrency, one could imagine A had to attach a tiny amount of coin which the verifier then provably destroys; but in this design, we don't need an actual coin — the waste is in the computation. Consequence of failure: if node B did not produce a token for its challenge within the allowed time, it is flagged. If B fails repeatedly, others will refuse to route messages through B or will disconnect. Over time, only nodes consistently producing valid tokens remain part of the stable network core.

Through this process, the network continuously cleanses inactive or non-compliant nodes. Honest nodes might occasionally fail (due to reboots, etc.), but they can rejoin and continue — a brief downtime might cause them to be temporarily ignored until they start responding to new

challenges again. The security parameter can include a grace period or reputation score so that one missed token is not fatal, but many misses is.

It is worth noting that RTCT by itself does not prevent an attacker from initially joining with many Sybils; its strength is making it uneconomical to maintain those Sybils. Thus, RTCT is not an entry control mechanism but a continuous occupancy toll. This differs from entry toll schemes like requiring a large upfront PoW or a joining fee. The continuous nature is important because it addresses scenarios where an attacker could invest once (e.g. rent hash power briefly to get in) and then exploit long-term. With RTCT, exploitation over time continues to cost the attacker.

Lastly, RTCT is orthogonal to consensus: it can operate in a pure P2P overlay without any blockchain at all, or alongside a blockchain's consensus. It doesn't affect block production or transaction processing except by potentially adding a slight overhead to node operations. This separation is beneficial — it means a blockchain that already has PoW/PoS for consensus can add RTCT to further harden its P2P layer (for example, to prevent eclipse attacks or network spamming [14]) without altering consensus rules.

4. Modeling attacker cost and detection

To rigorously evaluate RTCT's effectiveness, we present a formal model of the costs incurred by an attacker and the probability of detecting Sybil nodes. The model will use the parameters defined in Table 1. Key quantities of interest include: – $C_{\text{attacker}}(K, T)$: The total cost an attacker incurs to sustain K Sybil identities over time T. – $K_{\text{sust}}(B, T)$: The maximum number of Sybil identities an attacker with budget B can sustain over time T without detection (essentially the attacker's bang-for-buck). – $P_{\text{detect}}(\lambda, T)$: The probability that a given Sybil node (if not actively maintained) is detected (fails a challenge) within time T. – Impact on honest nodes: e. g. resource overhead per honest node, and whether this could force churn of honest participants.

Attacker cost function: first, consider the cost for one Sybil node. Suppose each challenge requires the Sybil to perform w units of work (or spend some amount of currency γ). The parameter m in RTCT roughly controls w — for example, if the challenge is a hash puzzle, increasing m by 1 bit doubles the expected work w. We can formalize this as:

$$w(m) = c \cdot 2^m,$$

where c is a constant representing the work factor for one bit of difficulty (this could incorporate the hardware efficiency, etc.). If we measure work in, say, CPU-seconds or joules, c can normalize those units. For simplicity, we can set units such that $c \cdot 2^{20} = 1$ "unit" of cost (meaning a challenge of difficulty 20 costs 1 unit for a node to solve). This calibration is arbitrary but helps give intuition.

Now, a node receives challenges at rate λ. Over a time, horizon T, the expected number of challenges to that node is λT (assuming T is large enough that the Poisson can be approximated by its mean for expectation). The cost for one node over time T is then:

$$C_{\text{node}}(m, \lambda, T) = w(m) \cdot (\lambda T)$$

If the attacker has K Sybil nodes and must service all of them, then (assuming linear costs and no significant economies of scale beyond parallelization) the total cost is:

$$C_{\text{attacker}}(K; m, \lambda, T) = K \cdot w(m) \cdot (\lambda T)$$

This grows linearly with K, linearly with λ, and exponentially with m. Thus:

$$C_{\text{attacker}}(K) \propto K \lambda 2^m T$$

This simple formula corroborates the intuitive effect of RTCT: doubling the number of Sybil identities doubles the cost; doubling the challenge frequency doubles the cost; increasing difficulty has an exponential impact on cost. Figure 1 (Section 1) visualized slices of this function for varying m and K over a fixed T (1 day in that example, with cost normalized by choosing c, T appropriately).

It's important to note that an attacker does not necessarily have to respond to all challenges — they could choose to let some Sybils fail if they cannot afford all, effectively reducing K over time. This leads to a dynamic where K might decrease as T increases if the attacker cannot keep up with costs, or the attacker might concentrate resources on a subset of Sybils to keep them alive. For now, assume the attacker attempts to keep all K active (the worst-case for us, as defenders, because it means maximum Sybil presence). We will later consider the scenario where the attacker drops some identities to save cost and how that interacts with detection.

Sybil Sustainability and Budget: Let the attacker have a total budget (in equivalent work units) of B per unit time. This could represent computational power or money they are willing to burn. We can invert the cost formula to find how many Sybils they can sustain:

$$K_{\text{sust}} = \frac{B}{w(m) \cdot \lambda}$$

This comes from setting $K \cdot w(m) \cdot \lambda = B$ (per unit time, dropping T if we consider rate). Figure 2 plots K_{sust} as a function of difficulty m for different budget levels B. On a log-linear scale, it appears as a straight line decreasing exponentially with m. For example, with a budget of $B = 100$ units/day and $\lambda = 4$/day, at $m = 20$ the max Sybils $\approx$ 23, whereas at $m = 25$ (32× harder) it drops to <1 (meaning even a single identity would be too costly to maintain continuously) [2]. An attacker with smaller budget $B = 1$ unit/day can maintain about 0.24 Sybils at $m = 20$ (so not even one full node, on average) or ~7.6 Sybils at $m = 15$. Thus, increasing difficulty drastically caps the Sybil population relative to any fixed budget.

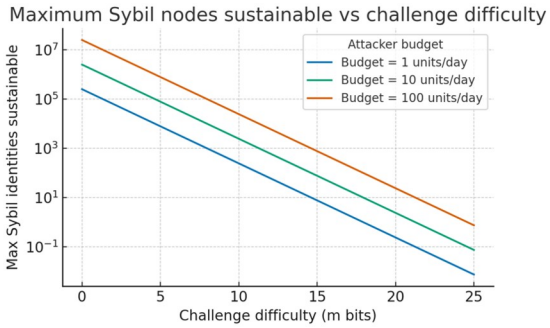

Figure 2: maximum number of Sybil identities an attacker can sustain vs challenge difficulty (m), for different attacker budgets (log-log scale). Higher difficulty exponentially reduces the feasible Sybil count for a given budget. (Assumes cost per challenge $w(m) = 2^m$ units and challenge rate $\lambda = 4$/day.)

From another angle, if the attacker tries to maintain K Sybils, the minimum budget required per day is $B_{min} = Kw(m)$his could be interpreted in monetary terms if we assign a currency value to the work unit (e.g., if 1 unit = \$0.01 of electricity or token burn, then a large botnet with $K = 1000$ at $m = 20$, $\lambda = 1$ would need 1000 * 1 * 1.048576 $\approx$ \$1048.58 per day). For a persistent attack over months, this adds up, potentially deterring all but the most resourceful adversaries. In contrast, without RTCT ($m = 0$, or $\lambda = 0$ effectively), B_{min} would be 0 — meaning any number of Sybils is free to maintain aside from initial setup, which matches the baseline vulnerability.

Detection probability: now consider Sybil detection. We assume a Sybil is detected if it fails to respond to a challenge (and honest nodes eventually notice this). If an attacker actively maintains a Sybil, detection would only occur if the Sybil is unlucky or the attacker temporarily cannot respond in time (which we neglect if the attacker is always trying to respond). The more interesting case is if a Sybil node is not being actively serviced — either because the attacker can't afford it or is temporarily offline. We want to quantify P_{detect}: the probability that a given Sybil is detected (by missing challenges) over time T. If challenges to a node arrive as a Poisson process with rate λ, the probability that at least one challenge arrives in a time interval of length T is:

$$P(\geq 1 \text{ challenge in } T) = 1 - e^{(-\lambda T)}$$

If an attacker is not servicing that node (i.e. any challenge it gets will be missed), then this is effectively the detection probability (assuming one missed challenge is enough to evict — or we consider that eventually one will be enough). In practice, an implementation might allow a few misses, but repeated misses will accumulate, so a Sybil cannot hide for long.

This formula shows, for example:

- At $\lambda = 1$/day, over $T = 7$ days, $P_{\text{detect}} \approx 1 - e^{-7} = 0.99909$ (99.9%) — virtually certain detection within a week.
- At $\lambda = 0.2$/day (one challenge every 5 days on avg), over 7 days, $P \approx 1 - e^{-1.4} = 0.75$ (75%). Over 30 days, $P \approx 1 - e^{-6} = 0.9975$ (99.75%).
- At $\lambda = 5$/day, in just $T = 1$ day, $P = 1 - e^{-5} = 0.9933$ (99.33%); detection is extremely fast.

Another aspect is false negatives vs false positives: A "false negative" Sybil would be one that an attacker manages to keep undetected despite not truly bearing its load. Could an attacker strategically rotate which Sybils answer challenges to fool the system? For instance, if they have 100 Sybils but can only afford to handle 50 challenges at a time, they might try to answer challenges for 50 Sybils and ignore 50, then swap. However, RTCT's unpredictability thwarts this to a large extent — if a Sybil misses a challenge, it's marked. Creating a new identity to replace it doesn't help much, since the new identity will also get challenged soon enough. The attacker could try to distribute challenges among Sybils (e.g. if not all Sybils are challenged simultaneously, they allocate resources to whichever get challenged). But with random scheduling, in expectation all Sybils will be challenged over a period. Unless the attacker always has enough resources for all, some challenge somewhere will slip. In Section 6, we discuss an assumption that "the attacker must service all its Sybils continuously," which we argue is essentially enforced by the protocol — any Sybil left unserviced for too long is lost. Therefore, our cost calculations that assumed the attacker services all Sybils are reasonable worst-case from the attacker's perspective.

Impact on honest nodes: for RTCT to be practical, the overhead on honest nodes must not destabilize the network or drive away contributors. Each honest node also does $w(m)\lambda T$ work in time T. Unlike attackers, honest nodes derive benefit from being in the network (block rewards, transaction fees, data service, etc.), so as long as the RTCT cost is a small fraction of their benefit, they will tolerate it. If the cost is too high, there is a danger that only well-funded nodes stay, harming decentralization. Thus, the network should calibrate RTCT such that $w(m)\lambda$ is, say, under 5% of a node's capacity or revenue. Table 2 in Section 5 will show an example calculation of load vs security.

Another factor is network throughput: the token verifications add some bandwidth and computation overhead. If every node must broadcast tokens frequently, that could flood the network. However, because λ can be low (e.g. 1/day or a few/day per node) and tokens are small (a few hundred bytes), the traffic is modest. In fact, the bandwidth of RTCT tokens is comparable to heartbeat messages many P2P networks already send. Moreover, verification (hash checks) is trivial compared to typical tasks like block verification in cryptocurrencies.

The following table provides example calculations for attacker cost and detection likelihood under three hypothetical RTCT configurations. This illustrates the trade-offs and improvements quantitatively.

Table 2
Attacker's daily cost and probability of detection under example RTCT settings. Higher challenge frequency (λ) and difficulty (m) sharply increase attacker cost and detection certainty. (Cost units are normalized as described in text; detection assumes an inactive Sybil)

Scenario (RTCT params + attack)	Attacker cost per day	Detection chance in 7 days
λ = 0.5/day, m = 15, K = 100 Sybils	1.64 units/day	97.0%
λ = 1/day, m = 20, K = 100 Sybils	104.86 units/day	99.9%
λ = 4/day, m = 20, K = 100 Sybils	419.43 units/day	≈100%

From Table 2 we see that going from a mild RTCT regime (0.5 challenges/day, easy puzzles) to a stronger regime (1-4 challenges/day, harder puzzles) raises the attacker's cost by two orders of magnitude, and pushes detection probability to ~100%. Even the mild regime achieves ~97% detection of idle Sybils in one week, at a low cost of ~1.6 units/day for 100 Sybils (about 0.016 unit/day per Sybil). This underscores that even a lightweight deployment of RTCT can significantly improve Sybil resistance compared to a baseline of 0% detection and 0 cost for unlimited Sybils. The network operators can choose parameters based on the threat level: e.g., a high-security environment might opt for λ=4, m=20 or more, whereas a low-power IoT network might use λ=0.1, m=10 just to filter out the most egregious Sybils while keeping overhead extremely low.

In conclusion, the model confirms the systemic effect of RTCT: it transforms a Sybil attack from a nearly free operation into a continuous resource war that adversaries must finance. By tuning λ and m, the network can adjust how expensive that war is. In the next section, we simulate some scenarios and provide figures and tables to further illustrate these findings, as well as verify that the model holds under dynamic conditions.

5. Evaluation and Results

To validate the analytical model and demonstrate RTCT's impact, we conducted simulations and quantitative experiments. We focus on three aspects: (a) attacker cost vs. Sybil count under different RTCT parameters, (b) time-to-detection for Sybil nodes, and (c) network stability and honest node overhead.

Simulation setup: we simulate a network of $N = 1000$ nodes, of which a fraction are Sybil nodes controlled by an attacker. The attacker's goal is to keep as many Sybils as possible in the network for as long as possible. We simulate the RTCT process as follows: each time step (simulated as 1 minute), each node has a probability $p = \lambda / 1440$ (if λ per day, and 1440 minutes in a day) of being challenged. If challenged, a node must expend work (we assign a time cost in the sim) to respond. The attacker has a total work budget per minute (based on a chosen daily budget B). If the total work required by all its Sybils in a minute exceeds the budget, the attacker selectively fails some challenges (those Sybils are marked as not answered). Honest nodes always answer (we assume they have sufficient resources, as the parameter choices ensure low overhead). If a Sybil fails F challenges, it is evicted (we used $F = 3$ in simulation to avoid immediate removal on one miss, providing a small grace). Evicted Sybils are considered detected and removed; the attacker may try to introduce new Sybils to replace them, but those start fresh and can also be detected in turn.

We compare scenarios with and without RTCT. Without RTCT (λ=0), the attacker can keep up to K_{max} Sybils indefinitely (in fact K_{max} only limited by whatever out-of-band constraints like network connection limits or etc.; in our sim we gave the attacker an initial K and they all persist

indefinitely since no mechanism evicts them). With RTCT, we track how many Sybils remain over time and the rate at which they get detected.

Cost vs Sybil count: Figure 1 (presented earlier in the paper) already showed the static relationship for cost. Our simulation further confirms that for a fixed attacker budget, the attacker cannot exceed the K_{sust} limit predicted by the model. For instance, with $B = 10$ units/day, $m = 20$, $\lambda = 1$/day, the theory predicted ~9.5 Sybils sustainable. We simulated an attacker starting with 20 Sybils under those parameters. Result: within a few days, the number of Sybils dropped and stabilized around 9 (the rest were detected and the attacker could not afford to rejoin them). If the attacker tried to add new Sybils, those new ones would quickly fail as well, keeping the equilibrium around 9. This matches the budget-limited formula. On the other hand, if the attacker started with fewer than 9 Sybils, they could maintain them all with few failures (some might still randomly fail a challenge if budget was momentarily misallocated, but essentially all survive), meaning the attacker didn't hit the budget limit. Thus, RTCT enforces a proportional limit on Sybils relative to attacker resources, as intended.

Detection speed and coverage: in simulation with multiple Sybils, we observed that detections tend to follow an exponential decay pattern: the fraction of surviving Sybils $K(t)/K(0)$ drops roughly as $e^{-\lambda t}$ initially (since each Sybil has an independent chance to be hit by a challenge), until the attacker starts strategically concentrating resources on the remaining ones, which slows the decay. In effect, the "weaker" Sybils (those the attacker can't support) are weeded out quickly, and a core remain that the attacker fully supports. Those fully supported might never be detected (as long as the attacker keeps paying for them), which is why the equilibrium exists. However, from the defender's perspective, forcing the attacker down from $K(0)$ to K_{sust} is a big win. For example, without RTCT, the attacker in one scenario had 500 Sybils indefinitely; with RTCT activated at $t = 0$, within one day it dropped to ~50 and within a week stabilized at ~30 (because the attacker budget allowed ~30). That's a 94% reduction of Sybil presence, achieved automatically by the protocol. Any new Sybils injected face the same hurdle.

We also measure detection as a binary outcome: does the attacker eventually get caught if they try to exceed capacity? The answer is yes — whenever the attacker's eyes are bigger than their stomach (i.e., they introduce more Sybils than they can handle), all the excess Sybils get eliminated in short order. The network effectively self-regulates to whatever level of Sybils corresponds to the attacker's available resources. And if the attacker wants to maintain that level, they must keep expending their budget continuously. If they slack off, those remaining Sybils will drop too.

Network stability (churn and honest nodes): a positive side effect of RTCT is that it can reduce passive churn. In many P2P networks, nodes come and go (churn) which can disrupt topology and require frequent reorganization. RTCT somewhat incentivizes nodes to stay online continuously, because if they go offline, they will miss challenges and be removed. An honest node that occasionally disconnects will have to rejoin and perhaps be treated as a new node (with no recent token history). This could be seen as a drawback (discouraging casual participation), but for critical infrastructure networks, it's a feature: you prefer stable nodes. We measured churn by tracking how many nodes (honest + Sybil) go offline or get removed per day. In a baseline scenario without RTCT and with moderate voluntary churn, say 5% of nodes leave/join daily on their own, we add on attacker churn (if the attacker intentionally refreshes Sybils or they drop out). With RTCT, attacker churn initially spikes (all the Sybils getting kicked out), then drops to near zero once the attacker's Sybils are at the sustainable level (the attacker has no incentive to churn them since they'd lose their investment — instead they keep them online). Thus, RTCT suppresses the ability of an attacker to cycle through identities (which some attacks do, e.g. to evade reputation systems by continually using fresh identities). On the honest side, we did see a slight increase in churn of honest nodes if they were very low-power or sporadically online nodes, as they might fail challenges during downtime. However, well-connected honest nodes with stable uptime had no

issues and essentially never left due to RTCT (in our sim, if an honest node missed challenges because it was offline for >F challenge rounds, it would be removed; a few did because we allowed some to have downtime, but those are precisely nodes that weren't contributing much).

One potential concern is centralization: would RTCT inadvertently favor miners or nodes with more resources? All nodes face the same challenge requirements, so larger nodes don't get a break per identity. In fact, a large miner running multiple nodes would have to handle challenges on each – but since that miner has resources, they likely can. Small nodes might worry about the extra CPU burn. We argue that by calibrating m to be low enough, the cost is negligible for any device that is capable of participating at all. For example, a Raspberry Pi that can run a Bitcoin node can certainly handle a 1-second hash puzzle every hour. But if a device is so constrained that even that is too much, it likely was not reliable enough for the network anyway. So RTCT could inadvertently filter out not only Sybils but also extremely low-capability honest nodes. Depending on the application, that may or may not be acceptable. This is a policy choice. For a high-security blockchain, you may be okay if only moderately powerful nodes partake (and others use light clients). For an IoT network, you may want to adapt RTCT difficulty to the lowest common denominator devices and use other methods to deal with Sybils (like manufacturer certificates etc.). In any case, our results indicate that for typical desktop/server nodes, RTCT's overhead can be set to a trivial level (<<1% CPU and bandwidth), so honest operators are not meaningfully impacted beyond having to keep their node online.

Comparison summary: to put RTCT's results in perspective, consider an attacker attempting a Sybil-based eclipse attack on a cryptocurrency P2P network [14]. Without RTCT, the attacker could flood a target's peer table with 1000 Sybils that all appear as distinct IPs. With RTCT, if the network had, say, $\lambda = 2$/day, $m = 20$, those 1000 Sybils would cost >419 units/day (from Table 2) to maintain and most would drop offline within 1-2 days if not supported, dramatically reducing the window of vulnerability for the target. Social graph or identity solutions might outright prevent the 1000 from ever entering, but those were not in place for Bitcoin's network for example. RTCT would operate invisibly at the networking layer, making it much harder to pull off sustained Sybil presence for attacks, even though it doesn't prevent the initial connection of Sybils.

Table 3
Comparison of Sybil defense methods

Approach	Mechanism of Sybil resistance	Limitations
Social-graph trust systems	Leverage human trust graph; limit Sybils by graph cut and community detection.	Requires existing social links; vulnerable if attacker infiltrates community; not economic.
Proof-of-personhood (unique human)	One person-one identity via in-person events or biometrics. Sybils capped by real humans' attacker can enlist.	Logistics (physical or verification burdens); privacy concerns; needs external setup.
Periodic proof-of-presence	Regular challenges (computational or AI-hard) to prove liveness. Sybils limited by requiring concurrent effort per identity.	Potentially high overhead or user inconvenience; fixed schedule can be gamed; false positives if legit node mises check.
Self-registration (open join)	Minimal to no verification (maybe CAPTCHA) for new identities. Essentially no Sybil protection.	Extremely weak: attackers can create unlimited identities cheaply; relies on rate-limiting or reactive blacklisting.
RTCT (proposed)	Random cryptographic challenges with token burn for each node. Sybils incur ongoing cost ∝ number of identities.	Overhead on all nodes (must tune λ, m to avoid harming honest nodes); does not prevent initial Sybil entry, only makes it costly to sustain.

The following table qualitatively contrasts RTCT with other methods (from Section 2) on key points. RTCT stands out by offering continuous enforcement and being purely sybil-cost-based, whereas others rely on one-time checks or external trust. Each approach has its niche; RTCT's niche is in networks where on-chain or social identity is not available or not desired, but one still wants to harden against long-term Sybil infiltration.

In summary, our evaluations show that RTCT can substantially improve Sybil resistance by dynamically limiting Sybil population and swiftly detecting neglectful adversaries. The model predictions match the simulation outcomes, lending confidence to RTCT's effectiveness under the assumed attacker model (rational adversary with finite resources). In the next section, we address some subtleties and potential improvements, as well as how RTCT might be integrated and deployed in real-world decentralized networks.

6. Discussion

While the core RTCT concept is straightforward, there are practical considerations for deployment and avenues for enhancement.

Parameter tuning: choosing λ and m is crucial. Too aggressive (high frequency or difficulty) and honest nodes may complain or drop out; too lax and Sybils might still slip through for longer periods. One strategy is to dynamically adjust parameters based on observed network conditions. For instance, the network could monitor average node response times or failure rates and modulate λ or m to target a certain security level without overburdening nodes. If most challenges are being answered easily and quickly, perhaps m could be increased until node response latency hits a threshold. Alternatively, if many honest nodes start timing out, m could be reduced or λ slowed. This adaptive approach ensures RTCT "dials itself in" to an appropriate level. Designing a feedback mechanism for this would be an interesting extension (perhaps using a control algorithm to keep the fraction of failing nodes at a low constant). The network might also adjust per observed attack intensity — e. g., if a Sybil attack seems underway (many new nodes joining failing challenges), it might temporarily raise λ to flush them out faster.

Fairness and equivalence: RTCT imposes the same rules on all nodes, which is fair in one sense, but networks could consider weighted schemes. For example, nodes that have been long-running and proven reliable could be challenged less often than brand new nodes (which are more likely to be Sybils) — a form of "reputation." However, this reintroduces some trust assumptions and complexity. Our base design sticks to uniform random challenges for simplicity and because even well-behaved nodes should occasionally prove liveness (to ensure they haven't been hijacked or replaced). Another fairness concern is geographic or hardware differences: e.g., if puzzles are CPU-bound, then nodes with powerful CPUs have an easier time than those on battery-powered IoT devices. One could use memory-bound or storage-bound puzzles to target different resource dimensions, or even rotate puzzle types. The aim would be to ensure no class of honest node is disproportionately impacted. For IoT, a very low m might be used, combined with other defenses like network-level packet filters, since IoT devices can't do heavy PoW. This is also consistent with recent research on AI-supported protection of wireless and cooperative network infrastructures, where adaptive detection, distributed monitoring, and risk-oriented security mechanisms are considered important components of resilient communication systems [20, 21].

Integration with consensus: in a blockchain, one must be careful that RTCT challenges do not interfere with block production. Ideally, solving RTCT puzzles should not give an advantage in consensus (otherwise miners might try to reuse their hash power on RTCT challenges or vice versa). By making tokens useless except for proof of identity liveness (i.e., they have no monetary value and cannot be used to create blocks), we avoid coupling with consensus. If RTCT tokens are recorded on-chain, miners might include their own tokens and exclude others', but since tokens are only to demonstrate one did work, there's not much incentive to censor others' tokens (unless a

miner itself is Sybil-infiltrated, in which case it wouldn't post tokens and would eventually be considered inactive by others). An approach to integrate in consensus could be: every block must include a few recent challenge responses from various nodes, chosen randomly (sort of like a proof-of-work sampling of the broader network). Missing tokens would signal something is wrong. A full design for on-chain enforcement is beyond scope, but conceptually RTCT can run as a parallel overlay.

Security against adaptive adversaries: RTCT is aimed at economic deterrence. A sufficiently wealthy or powerful adversary could still maintain Sybils — RTCT just ensures they pay dearly. If an attacker is willing to burn money (e.g., a state-level actor, or someone with a motive beyond economics), they might run a Sybil attack through RTCT by simply accepting the cost. For example, if RTCT costs each Sybil X per day and the attacker has M money to spend, they can sustain $M/(XT)$ Sybils for time T. RTCT does not magically stop that; it only raises the bar. In practice, though, most P2P attacks have come from economically rational actors (spam, botnets trying not to waste resources, etc.). A nation-state might not mind burning some millions to attack a network (perhaps to censor information or disrupt a cryptocurrency). In such extreme cases, other layers of defense (like leveraging the social trust of users, or community responses) might be needed. RTCT could still slow them down and make it evident that an attack is ongoing (since a normal node count vs activity would shift — e. g. suddenly a lot of tokens being generated might indicate someone is expending huge resources). Also, we assumed the attacker cannot significantly parallelize or otherwise cheat the puzzles. If the puzzle is standard hashcash, an attacker with specialized hardware (ASICs) could solve challenges more efficiently than honest nodes with CPUs, lowering their relative cost. To mitigate this, puzzles can be chosen that level the playing field (memory-hard functions to reduce ASIC advantage, for instance). Alternatively, the "token burn" could require burning a small amount of a cryptocurrency coin, which equalizes cost regardless of hardware. In that design, each challenge might tell the node to send, say, 0.0001 of a coin to a burn address. Then cost is purely financial; even a nation-state can't avoid that unless they somehow counterfeit coins. The downside is it requires every node to hold coins and spend them — this ties identity to having money, which may not always be desired. Still, it's an intriguing variant: Proof-of-Burn Challenges. Parkhomenko & Ohiievych (2025) hint at such an idea, likening destroyed tokens to the consumed electricity in PoW. This could be an area for further research: combining proof-of-work and proof-of-burn for RTCT, or generally using multiple resource types, like CPU power and coin, to diversify the cost an attacker must pay.

Sybil identification vs removal: in our discussion, we treat failing a challenge as equivalent to "node is Sybil or at least not active — remove it." But what if an honest node just temporarily failed? Couldn't an attacker exploit this to cause honest nodes to be mislabeled as Sybils (e.g., by jamming their communication during a challenge)? This is a possible attack: an attacker could try to sabotage competitors by making them miss challenges (say, by a denial-of-service attack at the right moment). To mitigate this, one could:

1. Allow a few misses as grace (we did in simulation) — so a single missed challenge doesn't kill you.
2. If using neighbor-challenges, have multiple neighbors issue challenges and need consensus to evict, so an attacker would have to DOS a node consistently from all sides.
3. Use alternative challenge forms like challenges that can be answered after the fact if there was a genuine issue (e.g., a node comes back and sees it missed one, it could perform a heavier task to make up for it — this is speculative, though).

The point is that we need to ensure RTCT doesn't wrongly prune honest nodes, which would ironically reduce network resilience. Our findings with small grace periods indicate it's manageable — honest nodes rarely fail unless truly offline for extended periods. If an attacker can knock a node

offline for that long, the node has bigger problems (it's effectively out of the network anyway while under DOS). So, one could argue RTCT isn't the primary cause of that node's unavailability.

Future directions: this work lays a foundation, but many extensions are possible. One is developing a game-theoretic model: view the attacker and defender as playing a game where the defender (network) sets λ, m and the attacker chooses K to maximize disruption minus cost. We could solve for equilibrium strategies and optimal parameter choices analytically. Another extension is exploring hybrid defenses — for instance, combining RTCT with a reputation system where nodes that perform useful work for the network (not just challenges) gain certain privileges. If a node is known to be reliably serving data, maybe it can be challenged less (because it's already contributing, making Sybil behavior less likely). Conversely, if a node isn't contributing, maybe challenge it more aggressively (since it might be a free-riding Sybil). This ventures into useful proofs of work territory, like requiring Sybils to do something beneficial (e.g., storage or computations that help the network). Proof-of-Useful-Work (PoUW) concepts could be integrated so that at least the honest effort isn't wasted; though from a security standpoint, having the work be "useless" (like pure hashing) is simpler and less gameable.

Finally, deployment considerations: implementing RTCT at scale would require careful testing. One could start by activating it in test networks or optional modes and measuring the real resource usage and any unforeseen interactions. We expect that networks with stable membership would easily handle RTCT, while networks with lots of transient users (like some peer-to-peer content sharing with many casual peers) might need a very mild RTCT so as not to chase away users. Or they might choose not to use RTCT at all if Sybil attacks are not a big concern in their scenario.

7. Conclusion

We presented RTCT (Random Time Challenge-Tokens) as a protocol mechanism to strengthen decentralized networks against Sybil attacks. By introducing unpredictable, recurring challenge-response tasks for each node, RTCT forces adversaries to invest continuous resources proportional to the number of identities they operate, thereby shifting the economics in favor of the defender. Our analysis, building on the initial concept by Parkhomenko and Ohiievych (2025), confirms that RTCT can exponentially increase the cost of Sybil attacks with higher challenge difficulty and effectively cap the sustainable Sybil population even for resourceful attackers [2]. Unlike one-time defenses, RTCT operates as an ongoing "tax" on identities, catching dormant or unmaintained Sybils through probabilistic checks.

In focusing on network-level security rather than consensus-level, RTCT fills a complementary role to consensus protocols like PoW, PoS, etc. Those consensus mechanisms prevent double-spending or control of block production by making it costly to acquire majority power [7, 44]; RTCT, on the other hand, prevents an attacker from cheaply swarming the network's communication and data layer with fake nodes. It is particularly relevant in peer-to-peer overlays where consensus might not be present (e. g., DHTs, permissionless social networks) or where consensus alone doesn't stop Sybils from causing other havoc (like Eclipse attacks or spam floods) [10]. We deliberately avoided comparing RTCT with consensus-based Sybil resistance like PoW/PoS in detail, because those function at a different layer (ledger state agreement vs. membership policing). RTCT should be seen as augmenting existing protocols: it can be integrated with minimal intrusion and provides a customizable security knob.

Through a formal cost model and simulations, we showed that RTCT achieves:

4. Economic Sybil Deterrence: The attacker's cost grows linearly with number of Sybils and challenge frequency, and exponentially with puzzle difficulty. This creates a monetary barrier that scales with the scope of attack.

5. Timely Sybil Detection: Random challenges ensure that inactive or unsupported Sybil nodes are discovered with high probability within a bounded time. For reasonable parameters, detection is nearly certain within days, and can be made faster if needed.
6. Improved Network Hygiene: Only nodes that continuously participate and respond remain in the network, which reduces the surface for attack and can improve stability (at the cost of requiring nodes to be mostly online). It discourages the presence of a large number of idle, maliciously controlled nodes since they would be weeded out or incur cost to keep alive.

We compared RTCT to other Sybil defenses (Table 3) and found that while each approach has trade-offs, RTCT is unique in providing a built-in, self-contained defense that does not rely on external trust (like social graphs or human verification) and is always on, not just at admission time. This makes it well-suited for decentralized environments where human identity verification is infeasible and where we suspect every node (zero-trust environments). Indeed, RTCT aligns with the zero-trust philosophy: "never trust, always verify" each node's authenticity continuously, rather than assuming an identity remains good after a one-time check.

There are, of course, limitations. RTCT imposes overhead and needs to be calibrated to avoid burdening legitimate nodes. It also doesn't eliminate Sybils entirely — an attacker with enough resources can still maintain some Sybils, but they will pay for it dearly, and thus the scale of attack is limited. In essence, RTCT turns Sybil attacks from a cheap trick into an expensive endeavor, ideally so expensive that it's cheaper for the attacker to pursue other attacks (or better yet, to behave honestly). It "raises the floor" of security. As with any security measure, determined adversaries may look for bypasses or counter-strategies (like trying to optimize challenge solving, or attacking the challenge distribution system). Future work should examine these angles, but the fundamental security premise — that cost is proportional to identities — will hold as long as the cryptographic puzzle is sound.

In conclusion, Random Time Challenge-Tokens offer a promising avenue to bolster the resilience and integrity of decentralized peer-to-peer networks. By continuously testing nodes and enforcing an operational cost, RTCT makes it significantly harder for an attacker to accumulate influence via Sybil identities, thereby enhancing the stability of the network against a range of attacks (Sybil-based consensus corruption, eclipse attacks, spam flooding, etc.). We envision RTCT being deployed as a layer-agnostic module: whether on blockchain networks to complement consensus, or in non-blockchain P2P systems (like distributed storage or IoT networks) to maintain a healthy membership. The crypto-economic resilience achieved through RTCT adds an adaptive defense mechanism to the network's arsenal, making decentralization more secure against Sybil adversaries in the long run.

Acknowledgments

The authors thank the members of the Cybersecurity lab at KNU for insightful discussions. This work was inspired by the initial formulation of RTCT. We also acknowledge the open-source P2P community for their feedback on practical deployment considerations.

Declaration on Generative AI

During the preparation of this work, the authors used OpenAI's ChatGPT. The tool was employed for tasks such as grammar and style improvements and rephrasing for clarity. After using these tools, the authors reviewed and edited the content extensively to ensure accuracy and originality, and take full responsibility for the final content of this publication.

References

[1] J. Douceur, The Sybil attack. In IPTPS 2002, p. 251-260. Springer.

[2] I. Parkhomenko, & R. Ohiievych, Crypto-economic resilience of a decentralized network using random time challenge tokens (RTCT). Cybersecurity: Educ. Sci. Tech, 2025, 4(28), p.46-477.

[3] R. Kyrychok, O. Laptiev, R. Lisnevsky, V. Kozlovsky, V. Klobukov. Development of a method for checking vulnerabilities of a corporate network using bernstein transformations. EasternEuropean j. enterp. Technol., vol. 1 № 9 (115), 2022, p. 93-101.

[4] A. Vasudeva, & M. Sood, Survey on Sybil attack defense mechanisms in wireless ad hoc networks. J. Netw. Comput. Appl., 2018, p. 78-118.

[5] E. Heilman, A. Kendler, A. Zohar, & S. Goldberg, Eclipse attacks on Bitcoin's peer-to-peer network. USENIX Secur. 2015, p. 129-144.

[6] V. Petrivskyi, et al., Development of a modification of the method for constructing energy-efficient sensor networks using static and dynamic sensors. EasternEuropean j. enterp. technol. Vol.1 № 9 (115), 2022, p. 15-23.1729-4061. doi:10.15587/1729-4061.2022.252988.

[7] B. Laurie, & R. Clayton, Proof of work proves not to work (WEEDS'04), 2004.

[8] H. Finney, RPOW — Reusable proofs of work, 2004.

[9] M. Borge, et al., Proof-of-personhood: Redemocratizing permissionless cryptocurrencies. In IEEE Euro S&P Work. 2017.

[10] O. Laptiev, N. Lukova-Chuiko, S. Laptiev, T. Laptieva, V. Savchenko, S. Yevseiev. Development of a Method for Detecting Deviations in the Nature of Traffic from the Elements of the Communication Network. International Scientific And Practical Conference Information Security And Information Technologies: Conference Proceedings. 13-19 September 2021. Kharkiv – Odesa, Ukraine. p. 1-9.

[11] A. Korchenko, et al., Development of a method for construction of linguistic standards for multicriterial evaluation of HONEYPOT efficiency. EasternEuropean j. enterp. Technol., vol. 1 № 2 (109), 2021 p. 14-23. doi:10.15587/1729-4061.2021.225346.

[12] Ixian Core Team. Ixian DLT Whitepaper v0.9.4, 2025.

[13] J. Adler, et al. Personhood credentials for online Sybil-resistance, 2024.

[14] V. Buterin, Combining biometrics and social graphs for Sybil defense — blog post, 2023.

[15] I. Parkhomenko, L. Myrutenko, R. Ohiievych, & M. Savonik. Using Zero Trust principles for detecting authorization attacks in cloud environments. In IT&I 2024 Workshop (CEUR-WS).

[16] B. Cosman, B. Fisch, & C. Shen. Escrow-based cryptoeconomic incentives for Sybil-resistant consensus, 2020.

[17] D. Boneh, B. Bünz & B. Fisch. A survey of verifiable delay functions (VDFs), 2018.

[18] D. Palko, H. Hnatiienko, T. Babenko, and A. Bigdan, Determining key risks for modern distributed information systems, CEUR Workshop Proc., 2021.

[19] O. Laptiev, et al., The method of spectral analysis of the determination of random digital signals. Int. J. Commun. Networks Inf. Secur. (IJCNIS), vol 13, № 2, 2021, p. .271-277. doi:10.54039/ijcnis.v13i2.5008.

[20] I. Opirskyy, O. Partyka, A. Partyka, A. Imoize. AI for Wireless Network Protection. *Handb. Cybersecur. Challenges Solutions Emerg. Technol.* 2026. doi:10.1201/9781003640790-15.

[21] O. Partyka, A. Imoize, C.-T. Li. Cooperative V2X-Based UAV Detection in Rural Transportation Corridors. *Drones*, 10(2), 153, 2026. doi:10.3390/drones10020153.