Onions Got Puzzled: On the Challenges of Mitigating Denial-of-Service Problems in Tor Onion Services

Jinseo Lee, Hobin Kim, and Min Suk Kang
KAIST

Abstract

Denial-of-service (DoS) attacks present significant challenges for Tor onion services, where strict anonymity requirements render conventional mitigation strategies inapplicable. In response, the Tor community has recently revived the client puzzle idea in an official update to address real-world DoS attacks, leading to its adoption by several major onion services. In this paper, we uncover a critical vulnerability in the current puzzle system in Tor through a novel family of attacks, dubbed ONIONFLATION. The proposed attacks artificially inflate the required puzzle difficulty for all clients without causing noticeable congestion at the targeted service, rendering any existing onion service largely unusable at an attack cost of a couple of dollars per hour. Our ethical evaluation on the live Tor network demonstrates the impact of these attacks, which we have reported to the Tor Project and received acknowledgment. Our analysis reveals an undesirable trade-off in the client puzzle mechanism, which is the root cause of the discovered vulnerability, that forces the Tor onion system to choose between inflation resistance and congestion resistance, but not both. We offer practical guidance for Tor onion services aimed at balancing the mitigation of these attacks.

1 Introduction

The Tor network [16] stands as the leading tool for anonymous communication, ensuring the anonymity of both senders and receivers through its Tor onion services [45] (formerly known as hidden services). They are a vital component of the Tor network, playing a crucial role for whistle-blowers, journalists, and activists who rely on platforms like SecureDrop [41], GlobaLeaks [23], and Proton Mail [36] for secure and confidential information sharing.

While substantial research has focused on breaking the anonymity and confidentiality of Tor onion services, there has been a notable lack of attention to understanding and enhancing the availability of these services. This gap in research is critical, as several real-world incidents have highlighted the importance of guaranteeing the availability of Tor onion services, with documented denial-of-service (DoS) attacks repeatedly targeting these services over the years [7, 8, 13, 28, 29, 39]. Consequently, these attacks have prompted the Tor community to call for more robust defenses [13, 28, 58]. Without reliable availability, users are forced to resort to less secure communication methods, thereby compromising their anonymity and confidentiality.

Unfortunately, the unique system design of Tor onion services makes defending against DoS attacks particularly challenging. For example, in Tor onion services, the sender is anonymous, making it difficult to attribute the source of an attack and to rate-limit or block potentially malicious connection requests. Also, replicating services in a scalable manner is challenging for Tor onion services due to the receiver anonymity; see more discussion in Section 2.3.

Given the persistent risks of DoS attacks on Tor onion services and the technical difficulties in applying traditional DoS defenses, the Tor community officially recommended the adoption of *client puzzles* in August 2023 to mitigate specific types of DoS attacks, including introduction-flooding attacks [58]. The resurgence of this 2.5-decade-old defense mechanism — originally proposed by Juels and Brainard [27] in 1999 — is a response to this pressing need. This revival is met with both excitement, as client puzzles have yet to see large-scale adoption in commodity systems, and concern, given that their effectiveness at scale remains unproven. Nevertheless, the official proposal has been well-received by the Tor community, with several major Tor onion services including the CIA [10] and instances of SecureDrop [40] already implementing client puzzles to protect their services.

In this paper, however, we show that a critical denial-of-service vulnerability remains in the current client puzzle solution of Tor onion services. We demonstrate on the live Tor network that a malicious client with minimal computing resources (e.g., a handful of 4-core laptops) can significantly increase the puzzle difficulty for all clients of a target service, effectively preventing the majority of legitimate clients from connecting to the service even after 60 seconds (where the

median connection waiting time without attacks is about 10 seconds). The core idea of our attack, which we dub OnionFlation, is to *inflate the puzzle difficulty* suggested to all clients, thereby increasing the puzzle solving time until it becomes prohibitively expensive while minimizing the volume of attack traffic and required computing resources. The cost of successfully executing and maintaining the OnionFlation attack is only about a couple of dollars per hour, which is more than two orders of magnitude cheaper than the cost of the traditional introduction-flooding attack [8, 13, 28, 29]. We have reported the identified vulnerability to the Tor developers, and they have acknowledged the issue (stating they are "taking it very seriously") and are working on a fix.

As we design the OnionFlation attack and its mitigation, we identify several critical limitations due to the strong anonymity properties of the Tor onion services (such as the lack of timely, accurate, and individualized feedback to clients) that render large-scale operations of client puzzles challenging. Considering these limitations, we formulate the client puzzle operation model for Tor onion services and analyze its effectiveness. Consequently, we find that the inherent limitations lead to a fundamental *trade-off* in the design of client puzzles for Tor onion services. That is, the client puzzle mechanism in Tor onion services should choose between two undesirable outcomes, either making the service vulnerable to inflation attacks or congestion attacks.

With the understanding of the undesirable choice between inflation resistance and congestion resistance (but not both) in Tor onion services, we propose a simple, configurable difficulty update algorithm for client puzzles that strikes a balance between resisting inflation attacks and managing congestion effectively. Our evaluation of the proposed update algorithm shows that the service operator can easily make the system robust against inflation attacks (e.g., enforcing more than 80% of attack cost compared to the traditional introduction-flooding attacks) while moderately hedging against temporary congestion attacks (e.g., increasing the suggested puzzle difficulty proportionally to the attack intensity). Furthermore, adjusting the balance between inflation and congestion resistance can be performed easily with one parameter. This new algorithm is currently under review by the Tor security team.

We summarize our contributions as follows:

- We identify a new set of powerful denial-of-service attacks that inflate puzzle difficulties without causing significant congestion at Tor onion services. We conduct a real-world evaluation on the live Tor network and receive acknowledgment from the Tor Project.
- We conduct the first academic investigation into applying client puzzles within Tor onion services, highlighting the fundamental limitations and an undesirable trade-off inherent in using puzzles in this context.
- We propose a practical mitigation strategy balancing inflation resistance and effective congestion management.

2 Background

2.1 Tor and Onion Services

The Tor network is one of the most popular anonymity networks for Internet users, routing their communications through a series of volunteer-operated servers known as relays [16]. This process, which involves multiple layers of encryption, ensures that the original source of the data is concealed from both the destination and intermediary entities.

Tor onion services [45] (previously known as hidden services) extend the anonymity provided by the Tor network to both clients and servers, enabling a system where neither party knows the other's identity or location. This dual anonymity is achieved through a specialized addressing scheme that uses `.onion` addresses. Each onion service is associated with a unique `.onion` address, which serves as its identifier within the Tor network. When a client wishes to connect to an onion service, it uses this address to establish a connection, with the entire process being routed through Tor relays that further protect the identities of both the client and the service.

2.2 Introduction-Flooding Attacks against Tor Onion Services

Introduction points and rendezvous points. Central to the Tor onion service architecture's provision of mutual anonymity are rendezvous points (RPs). Both the client and the service independently establish Tor circuits to a common RP, where these circuits are stitched together, ensuring that neither party can directly trace the other's network location.

In the early designs of onion routing [24, 25], RPs were semi-static; they were chosen by the onion services, and clients selected these RPs from a public list advertised by the onion services. However, this static nature of RPs made them vulnerable to various attacks. To mitigate this, the concept of an introduction point (IP) was introduced [16], enabling the fully distributed selection of RPs. The client first selects an RP and establishes a circuit to it. Following this, the client sends an introduction request (or *intro-request*) to the onion service via an IP, asking the onion service to establish its circuit to the chosen RP. IPs, a known set of Tor relays, are responsible for forwarding these intro-requests from clients to onion services, preserving the anonymity of both parties.

Introduction-flooding attacks. Yet, a critical vulnerability remains within the introduction request handling logic and this has given rise to a class of attacks known as *introduction-flooding* (or intro-flooding) attacks, which specifically target the intro-request handling mechanism of onion services. In the wild, real attacks have been observed where adversaries flood the onion service with a large number of intro-requests [8, 13, 28, 29]. As a result of the attack, the targeted onion service is forced to establish an excessive number of Tor circuits to numerous RPs, leading to significant resource exhaustion.

The focus of this work is to address this threat model — namely, the introduction-flooding attack. Our scope is confined to the analysis and mitigation of DoS attacks against the *intro-request handling logic* within onion services. It is important to distinguish this from directly flooding the IPs themselves, which falls outside the scope of this paper.[1]

2.3 Challenges of Handling DoS against Onion Services

In this paper, we focus on client puzzle mechanisms for mitigating intro-flooding attacks as they represent the only officially implemented, practical solution currently available. However, other denial-of-service (DoS) mitigation strategies also deserve discussion. Here, we briefly examine three classes of DoS mitigation strategies and explain why the *anonymity-first* design of Tor makes them inapplicable to onion services. For a more comprehensive taxonomy of DoS mitigation strategies, readers are referred to the literature [30].

- *Source-based filtering is inapplicable.* When a target service is able to attribute the source of traffic, it can employ measures such as rate limiting, banning suspicious sources, prioritizing traffic, or redirecting traffic to a scrubbing service for further analysis. However, source-based filtering schemes require the target service to be able to *distinguish* between different sources of traffic. This is precisely the challenge faced by Tor onion services, which are designed to prevent the identification of traffic sources. The anonymity feature of Tor, by design, prevents the target service from distinguishing between different sources of traffic, rendering source-based filtering ineffective. As a result, the current Tor system is unable to attribute the source of intro-flooding attacks.

  Note that this limitation should not be confused with the existing rate-limiting mechanisms implemented at individual introduction points. These mechanisms limit the overall rate of introduction requests [47] without distinguishing between traffic sources.

- *Replication-based mitigation is inapplicable.* Another common DoS mitigation is the overprovision of the target service through replication. Replicating Tor onion services is, unfortunately, non-trivial. Due to the design of the Tor network, which prioritizes anonymity and decentralization, replicating an onion service would require not only additional onion service instances but also the distribution of these instances in a manner that maintains user anonymity. As a result, fast (e.g., within seconds to minutes) scale-out of introduction points along with the distribution of replicated services has been limited in the Tor network. For instance, Onionbalance [51] has been proposed as a solution for horizontal scaling of onion services; however, it has notable limitations as a DoS mitigation tool: (1) it may prevent clients from accessing any introduction points when the service is under a DoS attack [43,52] and (2) it imposes constraints on the maximum number of replications [48]. Worse still, its ability to scale rapidly enough to counter sudden surges in DoS attack traffic remains untested.

- *Mitigation through testing sources is inapplicable.* Some DoS mitigation strategies rely on challenge-response mechanisms to test client legitimacy before granting access to the service. CAPTCHA [54], for instance, distinguishes human users from automated bots to prevent bots from overwhelming the service. However, CAPTCHA is impractical for Tor onion services due to their sender anonymity design. A service can only present a CAPTCHA challenge to a client after establishing a rendezvous circuit, which occurs after the intro-request has been received (see Section 4.2.1 for a detailed discussion).

2.4 Return of the Puzzles

The idea of client puzzles is one of the oldest and most well-understood strategies in the context of DoS mitigation. Originally proposed by Dwork and Naor in 1992 [18] for spam mitigation, the idea of proof-of-work has been widely adopted across various systems to curb unwanted activities in computing. Proof-of-work is particularly effective in addressing the asymmetry between attackers and defenders: attackers must expend significant computational resources to generate attack traffic (e.g., spam emails), while defenders can easily verify the proof-of-work with minimal effort.

Client puzzles in DoS defense. In 1999, Juels and Brainard [27] first applied the idea of proof-of-work to DoS defense by introducing *client puzzles* as a means to mitigate general volumetric DoS attacks. In a client puzzle system, the server challenges each client to solve a cryptographic puzzle before granting access to the service. This imposes some (hopefully small) computational burden on legitimate clients but places a significant burden on attackers, who must solve many puzzles to sustain an attack, thereby countering the asymmetry that favors attackers in traditional DoS scenarios.

Client puzzles quickly gained attention within the research community and have been extensively studied in various contexts, including TCP/IP [11,33,55,56], TLS [15,34], and Internet key exchange [32], etc. The idea has also been integrated into other DoS mitigation systems, such as the capability-setup channel [35].

Why have client puzzles *not* been widely adopted? Despite their theoretical effectiveness, client puzzles have not seen widespread adoption over the past 2.5 decades in the real-world Internet applications. The reasons for this are complex and varied, but one clear drawback is the unjustifiable and non-negligible burden placed on legitimate clients.

While client puzzles effectively reduce the asymmetry between attackers and defenders, they place a computational

[1] In fact, directly flooding IPs does not pose a significant threat in practice because IPs are often well-provisioned and rate-limited.

burden on legitimate clients, who must solve puzzles to access the service, whereas the target incurs minimal cost. This leads to a poor user experience, as even a few seconds of additional delay can be frustrating in many modern Internet services, such as video streaming or web browsing [31]. In time-sensitive applications like online gaming, these delays can become prohibitive. This drawback makes client puzzles less attractive than other DoS mitigation strategies that place the burden on the service's infrastructure rather than its users.

Return of the puzzles for onion services: *Why now?* After years of relative dormancy, the idea of client puzzles has seen renewed interest and, for the first time, global-scale deployment. The Tor community has officially recommended the adoption of client puzzles to mitigate DoS attacks on Tor onion services [58], and many onion service operators have already implemented this strategy in practice.

The short answer to 'why now?' is that, unfortunately, there are no other practical alternatives. As discussed earlier, many traditional DoS mitigation techniques are impractical for Tor onion services, leaving client puzzles as the best available solution at the moment.

A more detailed explanation lies in the nature of Tor onion services, which are generally more tolerant of the delays introduced by client puzzles than other Internet services. For example, the average connection establishment time for Tor onion services is around 10 seconds (see our experiments in Section 3.1.4), significantly longer than the 2–3 seconds typical for clearnet services [9, 26]. Therefore, adding a few more seconds for solving a puzzle is considered a minor increase in the overall connection time for Tor onion services. Our real-world experiments, detailed in Section 3.1.4, confirm that the additional delay introduced by client puzzles is indeed marginal in Tor onion services, while still being effective in mitigating DoS attacks.

3 Vulnerability in Tor Onion Puzzles

The renewed interest in client puzzles for Tor onion services, however, we argue, has also brought about a new vulnerability. In this section, we present a novel class of attacks, which we refer to as OnionFlation, that exploits the client puzzle system in Tor onion services. We first describe the current client puzzle system in Tor onion services (Section 3.1), followed by the design of the OnionFlation attack and its evaluation on the live Tor network (Section 3.2).

3.1 Current Onion Puzzle Design

In this section, we overview the basic puzzle design (Section 3.1.1), the system's end-to-end operation (Section 3.1.2), the puzzle difficulty update algorithm (Section 3.1.3), and the system's effectiveness against intro-flooding attacks (Section 3.1.4). All information presented here regarding the current onion puzzle design has been derived from two primary sources: the specification [49] and the source code [50]. Despite our best efforts to ensure accuracy, the information may not be exhaustive. For additional or specific details, readers are encouraged to consult these references.

3.1.1 Basic Puzzle Design

Client puzzles are a proof-of-work mechanism used to mitigate DoS attacks by requiring clients to solve cryptographic challenges before receiving service [27] (see Section 2.4 for details). Each puzzle instance typically includes service-specific parameters and a seed that prevents precomputation attacks: once the seed changes, previously valid solutions become invalid. Upon receiving a puzzle, clients iteratively perform cryptographic operations to find a valid solution, whereas the server verifies proposed solutions with significantly less computational effort, ensuring efficient verification.

Building on these fundamental principles, several important improvements have been made to the puzzle design over the years. We summarize three characteristics that have been incorporated into the current Tor onion puzzle system [49].

- ASIC resistance: Application-Specific Integrated Circuits (ASICs) may provide unfair advantages to adversaries against benign clients with general-purpose hardware as they can be optimized for specific puzzle-solving tasks [37]. To counter this, onion puzzles use a custom hash function that randomizes its internal logic for each new seed, effectively mitigating fixed-hardware optimizations [49].
- Memory-requirements: Several client puzzles have been designed to be memory-hard, requiring substantial RAM operations to solve puzzles, thus reducing the performance disparity between devices [17]. Tor onion puzzles also employ this type of puzzle structure [49].
- Linear difficulty scaling: Unlike traditional cryptographic puzzles that often use exponential difficulty functions (e.g., leading zeros requirements [6, 15]), onion puzzles employ a linear difficulty scale. The computational workload, quantified as the number of expected hash operations, increases linearly with difficulty. This approach allows onion services to make finer-grained adjustments to difficulty [49].

3.1.2 End-to-End Client Puzzle Operation

Figure 1 illustrates the end-to-end operation model of the current onion puzzle system. A client fetches a descriptor from one of directory servers and extracts the seed, suggested puzzle difficulty, and service-specific identity string. The client solves a puzzle based on these parameters, and sends an introduction request to the service through an introduction point.

Upon receiving an intro-request, the service verifies the puzzle solution and enqueues the request into the priority queue, which sorts requests based on their puzzle difficulties. The service in parallel dequeues requests from the queue and

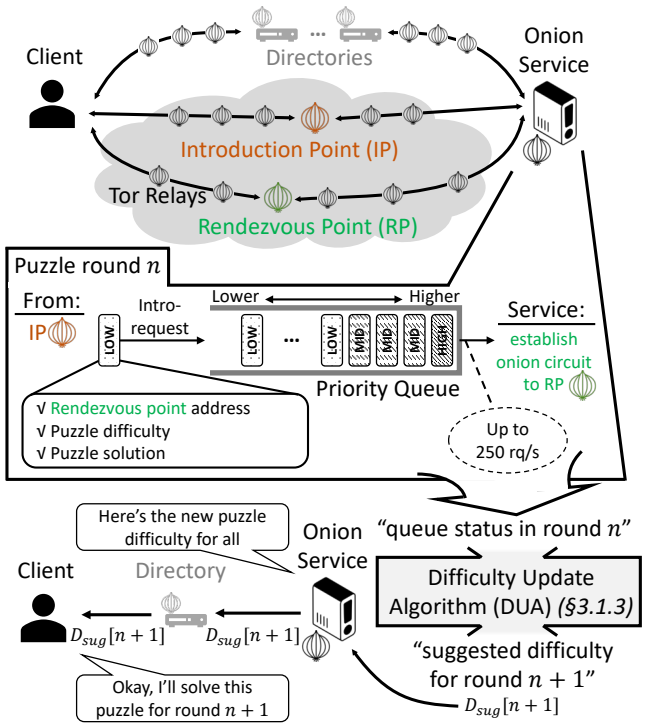


Figure 1: Overview of the client puzzle mechanism in Tor onion services.

establishes onion circuits to the rendezvous points specified in the requests. The rate at which the service dequeues requests is capped by 250 requests per second on average in the current implementation to curb the load on the Tor network.

Note that simply increasing this service rate (which is one of the most common DoS mitigation strategies in non-Tor systems) is not a viable option in the Tor network. Raising the dequeue rate (i.e., creating more rendezvous connections at the service) in response to a surge of incoming requests can significantly increase the load on the overall Tor network during intro-flooding attacks. Hence, the current onion puzzle system imposes a limit of 250 requests per second on average.

Server-centric feedback loop. While the Tor onion puzzle system allows clients to choose own puzzle difficulty at their discretion in theory, the default implementation simply follows the puzzle difficulty suggested by the onion service.[2] The most critical operation in the onion puzzle system, therefore, is how the service updates the suggested puzzle difficulty. The current onion puzzle system updates the suggested puzzle difficulty periodically at the end of each update round. The suggested difficulty is then advertised to clients through directory servers; see the bottom part of Figure 1.

[2]The effectiveness of the client-side difficulty adjustment mechanism is very limited in practice; thus, we exclude it from our main discussion. See Appendix A for a more in-depth discussion.

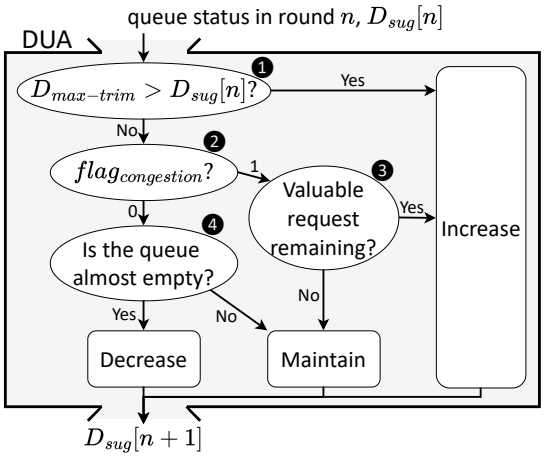


Figure 2: Illustration of the current difficulty update algorithm. For a detailed definition of ❸ and ❹, refer to line 4 and line 7 of Algorithm 1, respectively.

3.1.3 Difficulty Update Algorithm (DUA)

The heart of the puzzle difficulty update is at the Difficulty Update Algorithm (DUA). The onion service must set the suggested puzzle difficulty to a proper level to limit the impact of DoS attacks whenever necessary. The higher the suggested difficulty is set, the more costly it becomes for adversaries to generate attack traffic, thus reducing the amount of attack traffic from a rational adversary with limited resources.

Here, we describe the exact details of the DUA that has been implemented in the Tor onion service since its introduction in August 2023 [58]. The main design principle of the DUA is to adjust the suggested puzzle difficulty based on the status of the priority queue. Most fundamentally, it detects a *symptom of congestion* to efficiently update the suggested difficulty. The amount of each adjustment basically follows an additive increase and multiplicative decrease (AIMD) rule (which is, of course, inspired by the TCP congestion control algorithm [2, 12]); that is, the suggested difficulty is increased by one when a symptom is found and decreased to two-thirds when such symptoms are not detected. In addition to this, the DUA also implements a few other rules to quickly adapt to the changing environment by ramping up the suggested difficulty when noticing signs of severe congestion. For example, see ❶ in Figure 2 and the *increase* logic in Algorithm 1. Figure 2 illustrates the process as a simple flow chart, while Algorithm 1 provides the detailed steps. The service interprets the trimming of high-difficulty requests as a symptom of congestion and increases the suggested difficulty based on queue status.

3.1.4 Effectiveness against Intro-Flooding Attacks

The proposed official onion puzzle system has been widely adopted (e.g., CIA [10], instances of SecureDrop [40]) as of September 2024. To evaluate its effectiveness against

Algorithm 1 Current Difficulty Update Algorithm

state:

$D_{sug}[n]$: The suggested puzzle difficulty at round n.

$D_{max-trim}[n]$: The highest puzzle difficulty of trimmed requests during round n. Trimming can occur when requests that are too old (i.e., enqueued more than 15 seconds earlier) are dequeued, or when the queue reaches its capacity (i.e., 16,384 or 17,500 requests, depending on the configuration). For the latter, the service discards half of the queue.

$\sum D_{enqueued}[n]$: The sum of all puzzle difficulties of enqueued requests during round n.

$rend_{handled}[n]$: The number of handled requests during round n.

$flag_{congestion}$: A flag which is set if the priority queue is filled with a certain number of requests (i.e., 16 or 63 requests, depending on the configuration). It is unset at the start of each round.

decision:

at the end of each update round, the service

if $D_{max-trim}[n] > D_{sug}[n]$ **then**
    increases the suggested difficulty
else if $flag_{congestion}$ **then**
    if at least one request remains whose puzzle difficulty is $D_{sug}[n]$ or high **then**
        increases the suggested difficulty
    end if
else if the current number of requests in the queue is below a threshold (i.e., 16 or 63 requests, depending on the configuration) **then**
    decreases the suggested difficulty
else
    maintains the suggested difficulty
end if

increase:

$D_{sug}[n+1] \leftarrow \max(\frac{\sum D_{enqueued}[n]}{rend_{handled}[n]}, D_{sug}[n]+1)$

decrease:

$D_{sug}[n+1] \leftarrow \frac{2}{3} \times D_{sug}[n]$

introduction-flooding attacks, we conduct a real-world attack experiment on the live Tor network (see Section 7 for ethical considerations).

Experiment setup. We set up our own onion service on the live Tor network and send a large number of intro-requests for 24 hours.[3] Specifically, requests are sent at the maximum possible rate within the constraints of limited resources throughout the experiment. We dedicate one server-grade CPU to run the onion service and another server-grade CPU to launch the attack. To test varying attack intensities, we adjust the number of CPU cores used for the attacks. For this experiment, we utilize 30 cores, the highest number employed to implement OnionFlation attack strategies (see Section 3.2.2 for details). As a result, we achieve a peak rate of 505 requests per second in one round and an average rate of 47 requests per second. Note that 30 cores are sufficient to evaluate the effectiveness of onion puzzles against naive intro-flooding attacks,

[3] All Tor implementations used are based on Tor version 0.4.9.0-alpha-dev. The adversary's Tor client is modified to perform attacks.

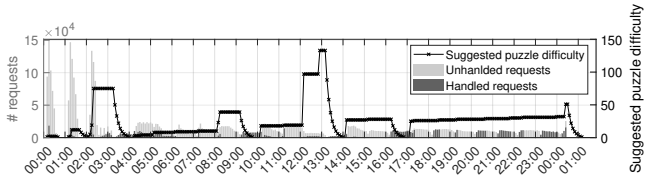


Figure 3: (Left Y-axis) Number of enqueued requests aggregated for each round; and (right Y-axis) changes in the suggested puzzle difficulty during intro-flooding attacks.

as an adversary with this level of resources can disrupt the onion service in the absence of onion puzzles (see Figure 4).

Figure 3 provides the changes in the suggested puzzle difficulty for one full day of the experiment, along with the number of requests that were either handled or unhandled throughout the attack for each round. The X-axis represents time, the main Y-axis (bars) indicates the number of requests aggregated for each update round, and the secondary Y-axis ($\times$) indicates the suggested puzzle difficulty. Although the number of requests sent initially fluctuated due to the experiment being conducted on the live network, it stabilized over time. This figure clearly demonstrates that the current DUA can maintain the number of requests at a manageable level (e.g., 20,000). In other words, the adversary succeeded in generating large enough traffic when the suggested puzzle difficulty was low, but began to fail as the suggested puzzle difficulty increased.

We observe that the current DUA mechanism often leads to non-negligible variations in difficulty levels, even when the server experiences similar traffic volumes. For instance, a similar number of requests were sent around 08:00 and 12:00 (approximately 8,800 and 10,150, respectively), yet the service managed to handle twice as many requests at 08:00 compared to 12:00, resulting in a much lower suggested puzzle difficulty. This disparity in handled requests is largely attributable to the behavior of the mass trimming mechanism (see $D_{max-trim}$ in Algorithm 1), which reacts differently to varying packet burst patterns. For further insights, we include two additional full-day experiment results in Appendix B.

Figure 4 illustrates the efficacy more directly. It provides cumulative distributions of legitimate clients' waiting times under intro-flooding attacks compared to normal circumstances. We measure the waiting time of a legitimate client using the Tor Browser one hundred times throughout each experiment. Unlike the service and attackers, we use a standard desktop to model the client. Given that the increase in waiting time due to the attack is much shorter when the onion puzzle system is activated, it is clear that the current onion puzzle system effectively mitigates intro-flooding attacks.

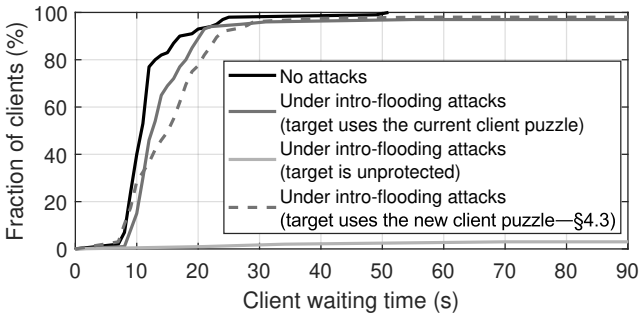
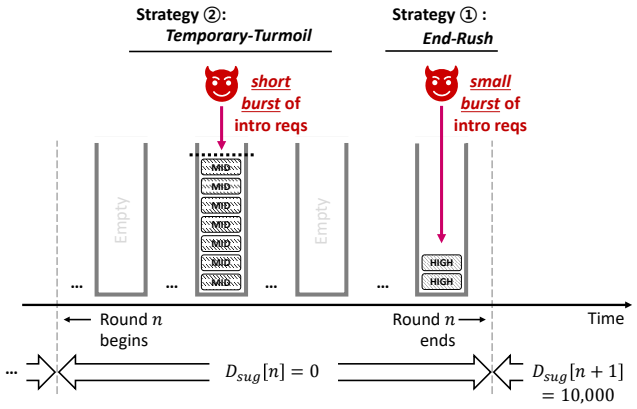
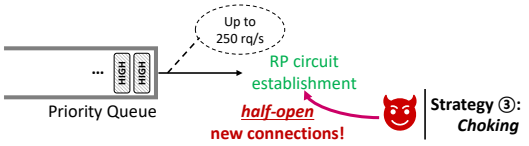


Figure 4: Cumulative distribution of legitimate clients' waiting time. Compared to normal circumstances, clients wait only a few seconds longer under intro-flooding attacks when the current puzzle is activated. By contrast, the waiting time mostly exceeds 90-second timeout when the puzzle is absent. The result with the new DUA (dashed line) is discussed later in Section 4.3.



(a) Priority queue filled with intro-requests over time during round n. The **end-rush attack (Strategy ①)** and **temporary-turmoil attack (Strategy ②)** are depicted.



(b) Onion service dequeuing intro-requests and establishing RP circuits. The **choking attack (Strategy ③)** is illustrated.

Figure 5: Three attack strategies of OnionFlation. The first two creates a false impression of congestion while the third actually reduces the service's capacity.

3.2 The OnionFlation Attack

We introduce a novel class of attacks, which we call OnionFlation, that aims to artificially inflate the puzzle difficulty the target onion service would suggest to its clients. While inflating the puzzle difficulty, the attack aims to minimize the attack traffic volume and computing resources required to solve puzzles, thus it barely causes congestion on the target.

3.2.1 Attack Strategies

By thoroughly inspecting the system, we designed two attack strategies to inflate the suggested puzzle difficulty, one to reduce the service capacity, and one to maintain the inflated difficulty, each exploiting different parts of the system. Figure 5 illustrates the first three attack strategies.

- **Strategy ①: *End-rush* attack strategy.** The goal of this strategy is to create a false impression that the priority queue has been occupied for most of the time on the target service. It exploits one particular mechanism in the current Tor onion puzzle system that attempts to capture a symptom of congestion: the service increases the suggested puzzle difficulty if, at the *end* of each update round, at least one unhandled request whose difficulty is at least the suggested difficulty remains (❸ in Figure 2). The adversary can induce this symptom by sending a small number of high-difficulty requests just before the next update; see how the adversary leaves requests unhandled at the end of the round by sending only a few intro-requests in Figure 5(a). This is feasible because timing information of the next update (e.g., when it will occur) can be easily obtained by frequently fetching the descriptors from the directory servers. Consequently, the adversary can effectively inflate the suggested difficulty by sending a minimal number of requests, without causing congestion on the service.
- **Strategy ②: *Temporary-turmoil* attack strategy.** The goal of this strategy is also to create a false impression of congestion on the target service, but with much easier puzzle solutions and more intro-requests than the end-rush strategy. It exploits one specific algorithm in the current Tor onion puzzle system that calculates the new suggested puzzle difficulty (the *increase* logic in Algorithm 1). Currently, when increasing the suggested difficulty, the larger value between $\frac{\sum D_{enqueued}}{rend_{handled}}$ and $D_{sug}+1$ is chosen. The issue is that the ratio can lead to a rapid increase in suggested difficulty due to its inherent discrepancy: while $\sum D_{enqueued}$ includes the puzzle difficulty of both handled and unhandled requests, $rend_{handled}$ counts only the number of requests that were actually handled. Therefore, if an adversary triggers the mass trimming of the queue (i.e., discarding half of the queue; see $D_{max-trim}[n]$ in Algorithm 1) by sending a sufficient number of requests (i.e., 16,384 or 17,500 requests, depending on the configuration), the suggested difficulty will be much higher than what the adversary actually solved; see how the number of requests in the queue changes during this attack in Figure 5(a).
- **Strategy ③: *Choking* attack strategy.** Unlike the previous two strategies, this one aims to actually reduce the service's

capacity, not just inflate the suggested difficulty. This strategy kicks in after the adversary has successfully inflated the suggested difficulty using the end-rush or temporary-turmoil attack strategies. When the suggested difficulty has been inflated beyond the client-side maximum difficulty (10,000; see Appendix A for details), the service is forced to *limit* the number of establishing connections to the rendezvous points to 16 in the current implementation. This operation is what our choking strategy exploits. Once the suggested difficulty has been inflated, the adversary periodically[4] sends some (e.g., 16 or slightly more) high-difficulty requests but deliberately leaves the corresponding rendezvous connections *half-open* (i.e., does not connect to the rendezvous points), making the service temporarily unavailable to handle benign (thus, low-difficulty) requests. This behavior is illustrated by the adversary half-opening new rendezvous connections in Figure 5(b).

- **Strategy ④: Maintenance strategy.** After successfully inflating the suggested puzzle difficulty, the adversary must maintain the inflated suggested difficulty to extend the effects. This can be easily achieved by sending several zero-difficulty requests (i.e., 16 or 63 requests, depending on the configuration), as the current system maintains the suggested puzzle difficulty based solely on the number of requests, regardless of their difficulty levels (❹ in Figure 2).

3.2.2 Evaluation on the Live Tor Network

We evaluate each strategy against our own onion service on the live Tor network, utilizing a similar experiment setup as described in Section 3.1.4, but with variations in experiment duration, attack strategies, and the number of CPU cores used for attacks. A detailed discussion on the ethical considerations of conducting these live network experiments is provided later in this paper (Section 7).

Experiment parameters. Regarding the specific parameters used in our experiments, we first perform analytical calculations to determine optimal values and then fine-tune them through iterative experimentation. This approach allows us to identify near-optimal parameters that are empirically validated to be effective in practice (see Appendix C for details).

***End-rush* attack strategy.** We show that the OnionFlation adversary successfully inflates the suggested difficulty from 0 to 27,636 with the end-rush strategy. First, the adversary sends a sufficient number of zero-difficulty requests (i.e., up to 439 requests per second) to trigger a suggested difficulty update, while fetching the service descriptor at 0.5-second intervals to infer the precise timing of the updates. Using this timing information, the adversary then solves puzzles with a difficulty of 11,011 and begins sending intro-requests about 15 seconds before an update begins. The volume of requests is set to be sufficient to trigger the flag $flag_{congestion}$ and leave at least one request pending at the end of the round (i.e., up to a maximum of 273 requests per second).

[4]Note that the onion service dynamically determines the timeout for circuit establishment at runtime, rather than using a fixed value.

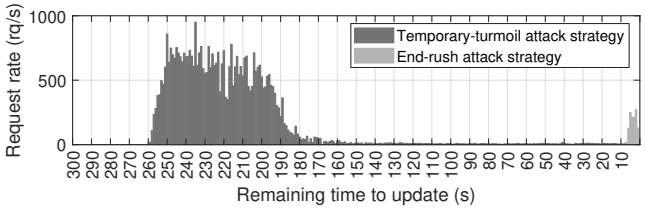


Figure 6: Number of enqueued intro-requests per second during a 5-minute update round. The two attack strategies require different amount of intro-requests at different times.

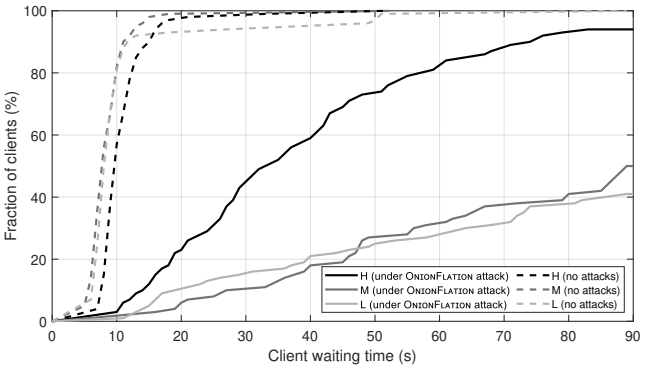


Figure 7: Cumulative distribution of legitimate clients' waiting time. H, M, and L refer to the performance categories of the devices, high-end, mid-range, and low-end, respectively. Regardless of device categories, the OnionFlation attack significantly increases clients' waiting time.

The attack process is well demonstrated in Figure 6, which shows the request rates during the update round. The X-axis represents the remaining time until the next update, while the Y-axis indicates the request rates. For the end-rush attack strategy, it is evident that a number of requests were enqueued just before the suggested difficulty update.

To better understand its impact, we measure the waiting time for legitimate clients using different devices with the Tor Browser one hundred times when the suggested difficulty is 27,636. Since the average time required to solve puzzles depends on the client's processor, we categorize the devices into three segments based on their computing power in CPU cycles: high-end (3+ GHz), mid-range (2-3 GHz), and low-end (0-2GHz). Typically, high-end devices include modern laptops, desktops, and flagship mobile devices, while older laptops and desktops or non-flagship mobile devices fall into mid-range or low-end categories. For the measurement, we use specific devices from each segment: a desktop with an Intel® Core™ i7-10700 Processor (maximum frequency of 4.7

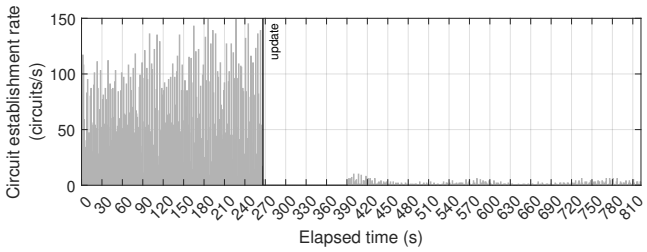


Figure 8: Rate of rendezvous circuit establishment by the onion service during the choking attack (i.e., Strategy ③). The suggested difficulty is updated at around 270 seconds.

GHz) for the high-end category, a Samsung Galaxy A24 (2.2 GHz) for the mid-range category, and a BlackBerry KEY2 LE (1.8 GHz) for the low-end category.

Figure 7 shows the CDF of clients' waiting time using devices from each device segment before and after a successful attack. First, even the high-end devices experience significant increase of waiting time, if not completely fail to make connections before the 90-second timeout. The average waiting time for the high-end devices goes beyond 30 seconds, which is well above the typical 10-second Tor onion service waiting time. Moreover, about 10% of the high-end devices experience timeouts, which is an extremely rare event under normal circumstances. Mid-range and low-end devices suffer even more from the OnionFlation attack, with their average waiting times exceeding 90 seconds, which is the hard timeout limit for the Tor Browser. For those that succeed in establishing connections (before the timeout), the waiting time is still significantly longer (about 50 seconds on average) than the normal waiting time. This clearly indicates that inflating the suggested difficulty causes significant delays across all device categories and leads to frequent timeouts for mid-range and low-end devices.

It is important to note that while this experiment clearly demonstrates the attack's impact (as evidenced by the significant increase in waiting time across all device categories), precise comparisons between device categories should be avoided due to the limited numbers and highly heterogeneous nature of the devices used in the study (i.e., distinct hardware and OS configurations). For example, the high-end device used in the study exhibits slightly longer waiting times under normal conditions than mid-range and low-end devices, which may not accurately represent the performance of all high-end devices. Similarly, the low-end device occasionally shows slightly shorter waiting times than the mid-range device, a result that should not be generalized to all low-end devices.

While Figure 7 illustrates the attack's impact in terms of waiting time, it does not provide insights into how quickly the attack takes effect or when the service begins to fail. For the temporal aspect of the attack, readers are referred to Figure 6, which shows that the attack becomes effective approximately 10 seconds after the arrival of the first attack packet.

In this experiment, the adversary is limited to using only 11 cores, and the total time taken to solve the puzzles was approximately 7,185 seconds. Based on AWS on-demand pricing [4], the cost of this attack strategy amounts to $1.2 ($6\ \textit{instances} \times \$0.09576\ /(\textit{hour} \times \textit{instance}) \times 7,185\ \textit{seconds}$).[5]

Considering that a naive introduction-flooding attack (i.e., sending high-difficulty requests at the maximum dequeue rate) to inflate and maintain the suggested difficulty would cost $478.9 per hour, the end-rush attack costs ***only 0.2%*** of that amount (see Appendix D for the estimation process).

***Temporary-turmoil* attack strategy.** We also show that the OnionFlation adversary succeeds in inflating the suggested difficulty from 0 to 12,770 by solving and submitting *only low-difficulty* (800) puzzles using the temporary-turmoil strategy. Unlike the end-rush strategy, this approach focuses on relatively easy puzzles while the number of puzzles solved was large enough to trigger massive queue trimming (i.e., up to maximum 950 requests per second). As a result, the service discards half of the queue during the attack, leading to a sharp increase in the suggested difficulty at the end of the round. This pattern is well shown in Figure 6, where a large number of requests are enqueued in the middle of the update round.

The impact of this strategy is equivalent to that of the end-rush attack. Since the client-side maximum puzzle difficulty is capped at 10,000 (see Appendix A for details), clients would solve puzzles at this maximum difficulty level rather than the inflated values of 12,770 or 27,636 observed in the experiments. We omit the experiment results for this strategy, as they are exactly the same as those for the end-rush strategy in Figure 7. Similar to the end-rush strategy, the attack becomes effective once the suggested difficulty is updated; e.g., around 260 seconds after the arrival of the first packet in Figure 6.

The adversary is allowed to use only 30 cores and solved the puzzles over about 6,930 seconds. Based on AWS on-demand pricing [4] again, the cost of this attack is $2.8, which is ***only 0.6%*** of the cost of the naive intro-flooding attack.

***Choking* attack strategy.** This attack strategy affects the onion service in two sequential phases: first, it reduces the service rate (using the end-rush or temporary-turmoil strategies), and then it consumes the reduced service capacity. Accordingly, we analyze these two effects separately.

To demonstrate the impact of service rate reduction, we measure the number of newly established rendezvous circuits with legitimate clients per second, both before and after inflating the suggested difficulty. Figure 8 shows the circuit establishment rates throughout the experiment. Before the suggested difficulty is inflated, more than 60 circuits are established per second on average. However, immediately following the inflation, the service completely stops dequeuing requests for over two minutes. Subsequently, the circuit establishment rate drops below 10 for most of the time and

[5]We verified that the per-core computing power of Amazon EC2 M7i-flex [3] is comparable to that of the processors used in our experiments.

frequently falls to zero. This clearly demonstrates the service rate reduction caused by the suggested difficulty inflation.

Next, we examine whether the adversary can efficiently consume up the reduced service capacity. By sending high-difficulty (10,001) requests and leaving the corresponding rendezvous connections half-open, the adversary makes the service primarily handle these requests, almost monopolizing (i.e., 85%) its capacity.

Based on these observations, we estimate the cost of an attack capable of *fully* consuming the reduced service capacity. Considering the number of hash operations required per second, we estimate that such an attack could be successful with 21 instances, which would ***cost $2.0 per hour*** according to AWS on-demand pricing [4]. For a detailed explanation of the estimation process, see Appendix D.

Maintenance strategy. We finally demonstrate that the OnionFlation adversary successfully maintains the inflated suggested difficulty by simply sending only zero-difficulty requests (a total of 184 requests). Given that the number of active cores is limited to one in this experiment, the cost of this strategy is ***only $0.1 per hour***, based on the same pricing [4].

4 Mitigation against Inflation Attacks

4.1 Tweaking Difficulty Update Algorithm

The best practical mitigation against inflation attacks would be to make use of the current difficulty update algorithm (DUA) only by tweaking its parameters to a certain extent.[6] However, surprisingly perhaps, no simple tweak of the current DUA seems to be effective against inflation attacks, as we will show in this section. Specifically, the current DUA seems to be overly sensitive to small bursts of intro-requests, rendering it difficult to mitigate inflation attacks by making simple tweaks.

To make a comprehensive evaluation of all possible tweaks, we revisit the current DUA logic, illustrated in Figure 2, and enumerate various simple adjustments. We list the proposed tweaks as follows:

- **[T1] Extreme thresholds for trimming and $flag_{congestion}$:** The service discards requests or sets $flag_{congestion}$ less frequently, only when the queue length exceeds an extreme threshold (e.g., update round duration $\times$ maximum dequeue rate). This modifies Condition ❶ and ❷ in Figure 2.
- **[T2] Randomized round duration:** The service updates the suggested puzzle difficulty at irregular intervals, making timing-based exploitation (i.e., end-rush attack strategy) more challenging. This tweak significantly weakens the adversary's ability to exploit Condition ❸.
- **[T3] Eased conditions for decreasing suggested puzzle difficulty:** The service reduces the suggested difficulty more frequently. Regardless of $flag_{congestion}$, the service decreases the suggested difficulty if either the queue is nearly empty (i.e., the current Condition ❹) or the average difficulty of remaining requests is below the suggested difficulty. This modifies Condition ❹.
- **[T4] New formula for calculating the suggested puzzle difficulty:** When increasing the suggested difficulty, the service sets it to $\max(\frac{\sum D_{enqueued}}{rend_{enqueued}}, D_{sug}+1)$ instead of $\max(\frac{\sum D_{enqueued}}{rend_{handled}}, D_{sug}+1)$, where $rend_{enqueued}$ is the number of enqueued requests. We include this tweak in our consideration despite introducing a new state, as it directly fixes the flaw that the temporary-turmoil strategy exploits.
- **[T5] Cap on suggested puzzle difficulty updates:** The service limits the increase in the suggested difficulty. If the previous suggested difficulty was zero, the maximum new suggested difficulty is capped at 8; otherwise, it is limited to twice the previous value. This adjustment ensures a more gradual increase in the suggested puzzle difficulty.

We assess the effectiveness of each adjustment by estimating the cost of attack strategies it impacts. All estimated costs are calculated based on the required number of hash operations and AWS on-demand pricing [4] (see Appendix D for details). Unfortunately, neither the above adjustments nor their combinations raise the costs enough to deter inflation attacks. Table 1 summarizes the evaluation results.

Table 1: Estimated attack costs in response to the proposed tweaks T1–T5. All the listed costs are one-time expenses, except for the hourly cost of the maintenance strategy (‡). We omit the estimated cost for T2 because it would effectively mitigate the end-rush attack strategy.

Tweak	Affected Condition	Affected Strategy	Estimated Cost
T1	Condition ❶, ❷	Strategy ①, ②	$2.6
T2	Condition ❸	Strategy ①	-
T3	Condition ❹	Strategy ④	$0.5‡
T4	-	Strategy ②	$10.0
T5	-	Strategy ①, ②	$1.9
T2-T5	Condition ❸, ❹	Strategy ①, ②, ④	$25.9, $0.5‡

First, the estimated inflation attack cost against the DUA with extreme thresholds is $2.6 ($14\ instances \times 0.09576\ USD/(hour \times instance) \times 6{,}930\ seconds$). This suggests that merely adjusting parameters is insufficient to mitigate the attack. Furthermore, these extreme values allow the adversary to conduct large-scale DoS attacks without raising the suggested difficulty. Therefore, we use this approach as a stand-alone tweak, without combining with other adjustments.

Randomizing round duration effectively mitigates the end-rush attack by making it challenging to obtain timing information. However, the temporary-turmoil attack remains effective.

Easing the conditions for lowering the suggested difficulty raises the maintenance strategy cost, but not significantly. The hourly cost would rise from $0.1 to only $0.5 (5 instances).

[6] Since an effective choking attack requires a successful inflation attack to precede it, choking attacks can be largely thwarted by mitigating inflation attacks. Therefore, we focus solely on mitigating inflation attacks.

Employing the new formula increases the cost of the temporary-turmoil strategy by lifting the required puzzle difficulty, but the increase is insignificant: the total estimated cost is only $10.0 (54 instances and 6,930 seconds).

Finally, capping suggested difficulty updates also results in an insufficient increase in the attack cost. Although the number of rounds required for the suggested difficulty to reach an extreme value increases from one to twelve, its impact on the overall cost is minimal. The estimated cost remains just $1.9 (247 instances and 286 seconds).

Even worse, combining these adjustments fails to effectively mitigate inflation attacks. Inflating the suggested difficulty above 10,000 by attacking the DUA with all adjustments applied, except for extreme thresholds, would cost $25.9 (3,401 instances and 286 seconds) as a one-time expense, with an additional cost of $0.5 per hour to maintain the inflated difficulty. Given that the cost of the naive introduction-flooding attack (i.e., sending high-difficulty requests at the maximum dequeue rate) is estimated at $478.9 per hour, this cost is still considered too low (**5.5%**).

4.2 Root Cause: Undesirable Trade-Off between Inflation and Congestion

Understanding that simple tweaks are not effective, we delve deeper into the root cause of the problem. The core of the problem is deeply intertwined with the fundamental limitations of Tor onion services, particularly when scaling client puzzles for large-scale operations.

4.2.1 Limitations in Tor Onion Services

Quick review of client puzzle *operation models*. To manage the client puzzle mechanism at a large scale, a feedback-based control system is essential for adjusting puzzle difficulty based on the system's overall status. Historically, since its inception in 1999 [27], client puzzle mechanisms have employed one of two types of feedback-based control models.

The first model, known as the *puzzle-auction model*, allows each client to independently adjust its puzzle difficulty based on the outcome of its last request. After an initial connection attempt, if the request is not handled or is delayed beyond a certain threshold, the client increases the puzzle difficulty (i.e., *its bid*). Each client separately applies these adjustments according to its own policy, considering the service's importance and its available computational resources, while the service prioritizes handling requests with higher bids. This model, originally proposed by Wang and Reiter in 2003 [55], has been shown to be robust against various attacks [32, 35].

The second model is the *challenge-response model*, where the server assigns a new puzzle challenge with a directed difficulty separately to each client upon receiving a request [6, 11, 15, 27, 32–34, 56]. The server determines the puzzle difficulty based on the overall system status. This model grants the server full control over the system, enabling it to swiftly increase puzzle difficulty in response to congestion. The downside, however, is that clients have no control over the difficulty, which may be problematic for those with limited computing resources, such as mobile devices.

Fundamental limitations. Unfortunately, neither of these models is well-suited for Tor onion services due to the unique characteristics of the Tor network. The first limitation is the *inability to provide instant feedback*. Unlike the public Internet, Tor operates on a much larger timescale. While a typical web client expects a response within a few seconds, Tor onion services are much slower and thus Tor clients are configured by default to wait up to 90 seconds for a response. Consequently, it may take up to 90 seconds for a client to determine that its request was not handled, making the puzzle-auction model too slow to adapt to the DoS attacks.

The second limitation is the *inability to provide individualized feedback* due to Tor's stringent anonymity requirements. To preserve client anonymity, the service cannot identify or directly communicate with clients before establishing a secure onion circuit, which includes several relays and a rendezvous point. When a service receives an initial intro-request, no secure circuit is available for sending feedback. This limitation renders the challenge-response model impractical for Tor onion services.

Therefore, the only possible operation model left for Tor onion services is to make *global announcements* to suggest the difficulty of the puzzle. Worse yet, such global announcements (from an onion service to all its potential clients) cannot be made quickly enough, unfortunately. This is because the announcements should be made through the Tor directories (due to the receiver anonymity) that are not designed to support real-time broadcast services. As a result, the service suggests the common difficulty of the puzzle to all clients at a slow pace with *infrequent announcements*. This is the *root cause* of difficulties in mitigating inflation attacks in Tor onion services, as we will discuss in the following.

4.2.2 Undesirable Trade-Off

Trade-off between inflation and congestion. The limitations of the current puzzle operation model, rooted in the design of Tor onion services, result in an undesirable trade-off concerning resistance to inflation attacks. Here, we define two types of attack goals: *inflation*, where the attacker seeks to increase the puzzle difficulty to an extent that most legitimate users cannot solve in a reasonable time, and *congestion*, where the attacker aims to use up significant portion of service resources, leading to service denial for legitimate users during the attack.

These goals differ in that inflation creates a false sense of flooding, while congestion directly impacts the service, denying access to legitimate users. Our findings indicate that no DUA mechanism for onion services can simultaneously resist both inflation and congestion. This undesirable trade-off

observed is that the more resistant a service is to inflation, the more vulnerable it becomes to congestion, and vice versa.

The current DUA of the Tor onion services is designed to be highly resistant to congestion attacks, which inherently makes it susceptible to inflation attacks. A short burst of intro-requests within a round prompts the DUA to react aggressively, significantly increasing puzzle difficulty for the next round. This strategy effectively mitigates congestion attacks by making subsequent congestion attacks more costly and thus smaller in scale. However, it also increases the risk of inflation attacks, as observed in the ONIONFLATION attack.

Conversely, consider a hypothetical DUA resistant to inflation attacks. Such a DUA would need to be nearly insensitive to small, short bursts of introduction requests within a round, avoiding significant increases in puzzle difficulty. While this approach would protect against inflation attacks, it would leave the service vulnerable to congestion attacks, as it would fail to escalate defenses against brief congestion attempts, allowing congestion attacks to succeed in subsequent rounds.

4.2.3 Security Analysis

To better analyze this trade-off, we devise an abstract model that focuses on high-level design choices while abstracting away low-level details. We present our model through a few important definitions and assumptions.

Definition 1. *An adversary A attacks onion services by sending a burst of intro-requests. The parameter α refers to the ratio of the attack period to the round length, and $D_{att}[t]$ indicates the difficulty of puzzles that A solves at round t.*

Definition 2. *A Tor onion service S handles intro-requests by dequeuing them from a priority queue, with a maximum rate of μ_{max} requests per second. S periodically broadcasts the suggested puzzle difficulty D_{sug}, based on the queue status during the last update period. The difficulty $D_{sug}[t]$ refers to the suggested puzzle difficulty at round t. The function $\beta(\alpha) = \frac{D_{sug}[t]}{D_{att}[t-1]}$ models the difficulty update algorithm (DUA). Where γ is the maximum adjusting parameter for the suggested difficulty, all $\beta(\alpha)$ must satisfy the following condition:*

$$0 \leq \beta(\alpha) = \frac{D_{sug}[t]}{D_{att}[t-1]} \leq \gamma$$

Definition 3. *The resistance to congestion attacks of a DUA is defined as $\int_0^1 \beta(\alpha)d\alpha$. If this value is close to γ, that DUA is classified as highly congestion-resistant. This implies that S increases the suggested difficulty even if α is close to 0. The function $\beta_{r-c}(\alpha)$ is the most congestion-resistant DUA:*

$$\beta_{r-c}(\alpha) = \begin{cases} 0 & (\alpha = 0) \\ \gamma & (\alpha > 0) \end{cases}$$

Definition 4. *The resistance to inflation attacks of a DUA is defined as $\int_0^1 \big(\gamma - \beta(\alpha)\big)d\alpha$. If this value is close to γ, that DUA is classified as highly inflation-resistant. This implies that S increases the suggested difficulty only if α is close to 1. The function $\beta_{r-i}(\alpha)$ is the most inflation-resistant DUA:*

$$\beta_{r-i}(\alpha) = \begin{cases} 0 & (\alpha < 1) \\ \gamma & (\alpha = 1) \end{cases}$$

Assumptions. We assume that adversary A sends requests at a rate of μ_{max} during attacks and adjusts attack intensity using varying α and D_{att} values. Additionally, we assume that all $\beta(\alpha)$ are monotonically increasing, starting at 0 when there is no attack ($\alpha = 0$) and reaching γ at maximum attack intensity ($\alpha = 1$), reflecting rational behavior by service S.

Using the above model, we now prove that a trade-off exists between resistance to congestion attacks and inflation attacks.

Lemma 1. *There exists a perfect negative correlation between the resistance to congestion attacks and the resistance to inflation attacks of a DUA.*

Proof. By definition, the sum of the resistance to congestion attacks and the resistance to inflation attacks is calculated as follows:

$$\int_0^1 \beta(\alpha)d\alpha + \int_0^1 \big(\gamma - \beta(\alpha)\big)d\alpha = \int_0^1 \gamma d\alpha = \gamma$$

Therefore, an increase in the resistance to congestion attacks must correspond to a decrease in the resistance to inflation attacks, and vice versa. □

Lemma 2. *A congestion-resistant DUA is vulnerable to inflation attacks.*

Proof. Let $\beta'_{r-c}(\alpha)$ be a congestion-resistant DUA. By definition, $\beta'_{r-c}(\alpha)$ should be close to $\beta_{r-c}(\alpha)$. Since all $\beta(\alpha)$ are monotonically increasing, we have $\beta'_{r-c}(\alpha) \approx \gamma$ for all small α sufficiently distant from 0. Therefore, adversary A can increase the suggested puzzle difficulty to an extreme value by sending only a short burst of intro-requests. □

Lemma 3. *An inflation-resistant DUA is vulnerable to congestion attacks.*

Proof. Let $\beta'_{r-i}(\alpha)$ be an inflation-resistant DUA. By definition, $\beta'_{r-i}(\alpha)$ should be close to $\beta_{r-i}(\alpha)$. Since all $\beta(\alpha)$ are monotonically increasing, we have $\beta'_{r-i}(\alpha) \approx 0$ for all large α sufficiently distant from 1. Therefore, adversary A can significantly consume the service resources without raising the suggested puzzle difficulty. This allows A to conduct successful congestion attacks continuously in subsequent rounds. □

Theorem 1. *No DUA can be simultaneously resistant to both congestion and inflation attacks.*

Proof. By Lemma 1, if a DUA is congestion-resistant, it cannot be inflation-resistant. Conversely, if a DUA is inflation-resistant, it cannot be congestion-resistant. By Lemma 2 and 3, a congestion-resistant DUA is vulnerable to inflation attacks, and an inflation-resistant DUA is vulnerable to congestion attacks. Therefore, no DUA can be resistant to both types of attacks at the same time. □

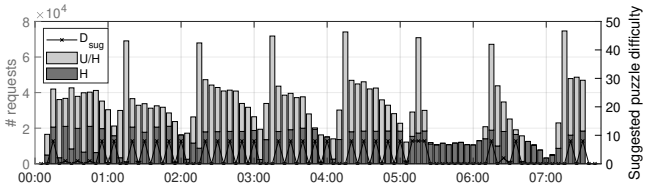


Figure 9: Number of enqueued requests aggregated for each round and changes in the suggested puzzle difficulty during intro-flooding attacks, with the new DUA employed. U/H denotes unhandled requests, and H indicates handled requests.

4.3 Finding a Balance

Recognizing the undesirable trade-off, we propose a new DUA that offers controllability to onion service operators. Our design aims to provide a simple yet effective control mechanism, enabling service operators to balance this trade-off. We first design a DUA that linearly adjusts the suggested difficulty based on attack intensity, then add a parameter to adjust its sensitivity for controlling the balance. It is important to note that we present this as one approach to mitigate the identified vulnerability, rather than as the optimal solution.

Our new DUA departs from the current approach by monitoring the overall dequeue rate rather than specific *symptoms of congestion* (e.g., trimming or remaining requests), enabling a more direct and consistent adjustment of puzzle difficulty. The next suggested puzzle difficulty is determined by the discrepancy between the target and (adjusted) actual dequeue rates, without requiring a decision. By default, the actual dequeue rate is used as-is for calculating the suggested difficulty, but it can be adjusted to control sensitivity. Algorithm 2 presents the detailed process. Note that the suggested puzzle difficulty is set to 8 instead of 1 at line 4 to accelerate the initial difficulty adjustment process.

The new DUA significantly raises the cost of inflating the suggested difficulty, making inflation attacks more expensive and less appealing to adversaries compared to the current system. For instance, inflating the suggested difficulty to 10,001 requires $383.1 per hour with the default parameters. If the service operator increases δ from 1 to 1.2 (i.e., reducing sensitivity), the cost rises to $459.7 per hour. Conversely, decreasing δ to 0.8 lowers the hourly cost to $306.5. These costs are 80.00%, 96.00%, and 66.67% of the cost of conducting the naive introduction-flooding attack, respectively.

Additionally, this algorithm escalates defenses against relatively short congestion attacks. For example, if the adversary attacks the service for only half of the round duration (i.e., $\alpha = 0.5$), the suggested difficulty would increase to $\frac{\mu_{max}}{\mu_{target}} \times \frac{D_{att}}{2}$ with the default δ. With $\delta = 1.2$, it is set to $\frac{\mu_{max}}{\mu_{target}} \times \frac{D_{att}}{2.4}$, while it becomes $\frac{\mu_{max}}{\mu_{target}} \times \frac{D_{att}}{1.6}$ if δ is 0.8.

To further evaluate our algorithm, we conduct a similar experiment to the one described in Section 3.1.4. Figure 9 presents the suggested puzzle difficulties and the number of enqueued requests aggregated for each round, during the intro-flooding attack against a service using our new DUA. Compared to the results with the current DUA (see Figure 3), it is evident that the new DUA is much more stable in adjusting the suggested puzzle difficulty.

Algorithm 2 New Difficulty Update Algorithm

parameter:

μ_{target}: The target dequeue rate that the service aims to maintain.

δ: The adjusting parameter, with a default value of 1.

state:

$D_{sug}[n]$: The suggested puzzle difficulty at round n.

$\sum D_{handled}[n]$: The sum of all puzzle difficulties of handled requests during round n.

$rend_{handled}[n]$: The number of handled requests during round n.

update:

at the end of each update round,

1: $\mu_{adjust}[n] \leftarrow \frac{rend_{handled}[n]}{T_{round} \times \delta}$ where T_{round} is round duration

2: $\overline{D_{handled}}[n] \leftarrow \frac{\sum D_{handled}[n]}{rend_{handled}[n]}$

3: **if** $\overline{D_{handled}}[n] = 0$ and $\mu_{adjust}[n] > \mu_{target}$ **then**

4: $\quad D_{sug}[n+1] \leftarrow 8$

5: **else**

6: $\quad D_{sug}[n+1] \leftarrow \frac{\mu_{adjust}[n]}{\mu_{target}} \times \overline{D_{handled}}[n]$

7: **end if**

While our DUA less aggressively increases the suggested puzzle difficulty, it maintains incoming request rates at a moderate level, as shown in Figure 4. The solid brown line and dotted brown line represent the cumulative distribution of legitimate clients' waiting times during attacks when using the original DUA and new DUA, respectively. The slight difference between these lines suggests that our DUA remains effective in managing congestion under intro-flooding attacks.

Finally, we evaluate the effectiveness of our new DUA in resisting the ONIONFLATION attack. The attack strategies are implemented with the same configuration as in Section 3.2.2, but with the onion service employing our new DUA instead of the original. Although all parameters remain unchanged, the adversary completely fails to inflate the suggested difficulty (staying at zero) when using the end-rush and temporary-turmoil attack strategies, thereby failing to execute the choking attack. The adversary also fails to maintain the suggested difficulty by sending zero-difficulty requests (i.e., implementing the maintenance strategy). These results align with our expectations, as the new DUA evaluates the average queue status rather than detecting isolated symptoms of congestion.

5 Related Work

We review several related studies and projects that fall within and extended scope of our work. We first discuss previous studies on DoS mitigation in Tor onion services, followed by a

review of client puzzles as a general DoS defense mechanism. Finally, we explore the broader context of mitigating DoS attacks in other anonymity networks.

DoS mitigation in Tor onion services. A few studies have explored strategies to mitigate intro-flooding attacks in Tor onion services. Döpmann et al. proposed a cryptographic token-based defense [19], providing flexibility in the choice and implementation of challenge types. However, their solution requires users to have successfully connected to the target service at least once, seriously limiting its effectiveness against general intro-flooding attacks; see Section 2.3 for a discussion on the challenges related to source testing. More recently, Arora and Garman introduced CenTor [5], a content delivery network designed for onion services. As one of the earliest replication-based DoS mitigation solutions, CenTor has the potential to spin up new instances of the target service dynamically, potentially limiting the impact of intro-flooding attacks. Yet, actual evaluation results on the effectiveness of CenTor against intro-flooding attacks have not been provided. Note also that both approaches are third-party proposals, unlike onion puzzles which are officially implemented and recommended by the Tor project.

Client puzzles for DoS mitigation. Since their introduction [27], various studies have explored the effectiveness of client puzzles as a DoS defense. We discuss two main aspects: security analysis and real-world deployment.

Existing research primarily focuses on proposing novel designs or implementations of client puzzles. Most studies [6, 11, 15, 27, 32–35, 55–57] address straightforward DoS attacks, where adversaries overwhelm the target by sending excessive requests, as demonstrated with the introduction-flooding attacks in Section 3.1.4, as well as other potential vulnerabilities of client puzzles (e.g., precomputation attacks, replay attacks, collision attacks, and dictionary attacks). Beyond these basic attacks, only brief discussions have been made on the mere possibility of more sophisticated attacks. For instance, Waters et al. [57] and Feng et al. [11] expressed concerns about the risk of eavesdropping on the puzzle solution; yet, this is trivially mitigated in Tor onion puzzles due to its encryption scheme. Nygren et al. pointed out that hostile servers may exploit legitimate clients to solve puzzles for other services [34] but this is not applicable to onion puzzles due to the service-specific string used in puzzle generation. Last, Parno et al. [35] briefly mentioned the potential for inflation attacks in their client puzzle design for general DoS mitigation but concluded that such attacks would be ineffective against general puzzle designs like theirs, which allow for individualized and timely feedback.

While having been discussed since its inception, client puzzles have not seen widespread adoption in real-world Internet applications until recently with the introduction of Tor onion puzzles. Outside of Tor, RFC 8019 [32] was officially accepted as a client puzzle based DoS defense mechanism for the Internet Key Exchange Protocol (IKE) 2 in 2016; however, no large-scale adoption has followed [42]. The lack of widespread adoption can be attributed to several operational challenges: (1) the puzzle construction and difficulty-setting process can be non-trivial in practice [1]; (2) non-negligible computational burden on legitimate clients may lead to undesirable delays. While Waters et al. [57] proposed puzzles that can be solved off-line to reduce delay, their applicability depends on the specific operation model, and they still impose a non-negligible burden on legitimate clients.

DoS mitigation in other anonymity networks. This work focuses exclusively on the Tor network for its discussion on DoS mitigation. Yet, several other anonymity networks [14, 22, 38, 44] may face similar challenges in addressing DoS attacks. To the best of the authors' knowledge, no existing anonymity network has implemented concrete DoS defenses similar to Tor's onion puzzles. Consider the Invisible Internet Project (I2P) [44] as an example. I2P, known for its hidden service-centric design (referred to as eepSites or I2P sites) [20], has not yet adopted DoS defenses comparable to Tor's onion puzzles. As a result, we are unable to directly compare or evaluate the OnionFlation attack and proposed client puzzle mechanism within the context of I2P or other anonymity networks.

Instead, we outline key considerations for effectively integrating robust client puzzles into I2P or other anonymity networks. Many anonymity networks share two fundamental limitations with Tor when adopting client puzzles (see Section 4.2.1): the inability to provide instant and individualized feedback. For example, I2P hidden services cannot provide instant feedback due to latency constraints [21] and are similarly unable to offer individualized feedback due to anonymity requirements. Consequently, many existing DoS mitigation mechanisms (e.g., rate limiting, over-provisioning) may not be directly applicable to them, leaving client puzzles as a promising alternative; see a similar discussion for Tor onion services in Section 2.3. When considering the deployment of client puzzles in I2P or other anonymity networks, our findings—namely, the undesirable trade-off and OnionFlation attack—are relevant and deserve careful examination.

6 Conclusion

Denial-of-service attacks, a persistent challenge for many services on today's open Internet, are even more difficult to address in the context of Tor onion services. The non-trivialities of handling DoS in Tor end up reviving the client puzzle idea, which has not been explored in this context until now. In this paper, we take a close look at this problem and identify a critical trade-off that must be considered when implementing client puzzles in these networks. Without a careful design that accounts for this trade-off, we argue that there is a risk of introducing a new and powerful type of attack, which we call the OnionFlation attack. We have responsibly disclosed this vulnerability to the Tor community.

Acknowledgments

This work is supported by the National Research Foundation of Korea (NRF) and the Ministry of Science and ICT (MSIT) under grant RS-2024-00464269.

7 Ethics Considerations

We conduct all experiments with careful ethical precautions to minimize any impact on the broader Tor network. Moreover, we have shared our attack scripts and the setup on the live Tor network with the Tor developers so that they can review any potential impact on the network while reproducing our results.

As a basic precaution, we set up our *own* onion service as the target and establish as many introduction points as possible (twenty) to distribute the attack traffic effectively. We then carefully review the impact of our experiments on other parts of the network (e.g., relays, directory servers) to ensure it is not significant.

First, in terms of bandwidth, our experiments can consume up to 3.7 Mbit per second in the worst case ($771\ \mathit{requests/second} \times 602\ \mathit{bytes/request} \times 8\ \mathit{bit/byte}$). With twenty introduction points, our bandwidth consumption per relay is 0.2 Mbit/s. Given that the minimum required bandwidth is 10 Mbit/s and the recommended bandwidth is 16 Mbit/s [46], our experiments consume only 1.9% of the minimum required bandwidth (1.2% of the recommended bandwidth) even in the worst case.

Next, since each experiment generates a different amount of traffic, we first calculate the maximum number of generated requests by summing the number of requests each experiment can produce. Assuming we conduct the same experiment three times, the total number of requests is 14,100,000. Given that the maximum traffic each request can generate is less than 7,000 bytes, the total traffic generated is less than 98.7 GB, with 4.9 GB per relay. Considering that the minimum required traffic is 100 GB/month and the recommended traffic is 2 TB/month [46], our experiments account for 4.9% of the minimum required traffic and 0.2% of the recommended traffic even in the worst case.

Regarding CPU and memory, it is important to note that the primary target of our experiments is not the relays but the onion service under our control. Since relays are merely parts of the paths to the service, the impact on the relays' CPU and memory is minimal. Specifically, given that relays should be able to handle at least 7,000 concurrent connections [46] and the maximum instantaneous connection rate in our experiments is 39 per relay, we only occupy 0.6% of the minimum required capacity. Even assuming each connection lasts for 5 seconds (our connections typically last less than 5 seconds), we still account for just 2.8% of the required capacity.

Responsible disclosure. To ensure the ethical disclosure of our findings, we reported the discovered vulnerability to the Tor developers in August 2024. We shared comprehensive details of the vulnerability, including test sets and experiment results, to assist the developers in identifying and resolving the issue. They acknowledged the issue and the team is currently evaluating the attack strategies and working on a fix. In particular, we have obtained permission to state that the Tor's security team is "taking it very seriously."

8 Open Science

Our research artifacts comprise a comprehensive set of seven modified versions of the Tor source code, eight attack scripts (including three Python scripts and five Bash scripts), and twenty-five miscellaneous files. Because these artifacts can be directly exploited to perform the described attacks on onion services, we have decided to share them exclusively with the core Tor developers rather than making them publicly available. We have provided the developers with access to our GitHub repository, and they subsequently imported our artifacts into the official Tor GitLab project as a private repository.

References

[1] Isra Mohamed Ali, Maurantonio Caprolu, and Roberto Di Pietro. Foundations, properties, and security applications of puzzles: A survey. *ACM Comput. Surv.*, 53(4), 2020.

[2] Mark Allman, Vern Paxson, and William Richard Stevens. TCP Congestion Control, 1999.

[3] Amazon Web Services. Amazon EC2 m7i and M7i-flex instances. https://aws.amazon.com/ec2/instance-types/m7i/, retrieved August 2024.

[4] Amazon Web Services. Amazon EC2 on-demand pricing. https://aws.amazon.com/ec2/pricing/on-demand/, retrieved August 2024.

[5] Arushi Arora and Christina Garman. Improving the performance and security of tor's onion services. *Proceedings of the 25th Privacy Enhancing Technologies Symposium*, 2025(1):531–552, 2025.

[6] Tuomas Aura, Pekka Nikander, and Jussipekka Leiwo. DOS-resistant authentication with client puzzles. In *Proceedings of the 8th Cambridge International Workshop on Security Protocols*, Security Protocols '00, pages 170–177, Berlin, Germany, 2000. Springer.

[7] Isabela Bagueros. Tor is slow right now. here is what is happening. https://blog.torproject.org/tor-network-ddos-attack/, February 2023.

[8] Thomas Brewster. Tor hidden services and drug markets are under attack, but help is on the way. https://www.forbes.com/sites/thomasbrewster/2015/04/01/tor-hidden-services-under-dos-attack/, April 2015.

[9] Michael Butkiewicz, Harsha V. Madhyastha, and Vyas Sekar. Understanding website complexity: measurements, metrics, and implications. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, page 313–328, New York, NY, USA, 2011. ACM.

[10] Central Intelligence Agency. Cia's latest layer: An onion site. https://www.cia.gov/stories/story/cias-latest-layer-an-onion-site/, 2019.

[11] Wu chang Feng, Edward Kaiser, Wu chi Feng, and Antoine Luu. Design and implementation of network puzzles. In *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, INFOCOM '05, pages 2372–2382, New York, NY, USA, 2005. IEEE.

[12] Dah-Ming Chiu and Raj Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17(1):1–14, 1989.

[13] Catalin Cimpanu. Tor project to fix bug used for ddos attacks on onion sites for years. https://www.zdnet.com/article/tor-project-to-fix-bug-used-for-ddos-attacks-on-onion-sites-for-years/, July 2019.

[14] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: design of a type iii anonymous remailer protocol. In *Proceedings of the 2003 Symposium on Security and Privacy*, S&P '03, pages 2–15, New York, NY, USA, 2003. IEEE.

[15] Drew Dean and Adam Stubblefield. Using client puzzles to protect TLS. In *Proceedings of the 10th USENIX Security Symposium*, USENIX Security '01, Berkeley, CA, USA, 2001. USENIX Association.

[16] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, USENIX Security '04, pages 303–320, Berkeley, CA, USA, 2004. USENIX Association.

[17] Cynthia Dwork, Andrew Goldberg, and Moni Naor. On memory-bound functions for fighting spam. In *Proceedings of the 23rd Annual International Cryptology Conference*, CRYPTO '03, pages 426–444, Berlin, Germany, 2003. Springer.

[18] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Proceedings of the 12th Annual International Cryptology Conference*, CRYPTO '92, pages 139–147, Berlin, Germany, 1992. Springer.

[19] Christoph Döpmann, Valentin Franck, and Florian Tschorsch. Onion pass: Token-based denial-of-service protection for tor onion services. In *Proceedings of the 2021 IFIP Networking Conference*, IFIP Networking '21, pages 1–9, New York, NY, USA, 2021. IEEE.

[20] Christoph Egger, Johannes Schlumberger, Christopher Kruegel, and Giovanni Vigna. Practical attacks against the i2p network. In *Proceedings of the 16th International Symposium on Research in Attacks, Intrusions and Defenses*, RAID '13, pages 432–451, Berlin, Germany, 2013. Springer.

[21] Mathias Ehlert. I2p usability vs. tor usability a bandwidth and latency comparison. Technical report, Humboldt University of Berlin, 2011.

[22] Michael J. Freedman and Robert Morris. Tarzan: a peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, page 193–206, New York, NY, USA, 2002. ACM.

[23] GlobaLeaks. GlobaLeaks - free and open-source whistleblowing software. https://www.globaleaks.org/, retrieved August 2024.

[24] Ian A. Goldberg. *A Pseudonymous Communications Infrastructure for the Internet*. PhD thesis, University of California at Berkeley, 2000.

[25] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding routing information. In *Proceedings of the First International Workshop on Information Hiding*, IH '96, pages 137–150, Berlin, Germany, 1996. Springer.

[26] HTTP Archive. Top 1,000,000: Loading speed. https://httparchive.org/reports/loading-speed?lens=top1m&start=2018_01_01&end=2024_01_01&view=list, retrieved August 2024.

[27] Ari Juels and John Brainard. Client puzzles: A cryptographic defense against connection depletion attacks. In *Proceedings of the Network and Distributed System Security Symposium 1999*, NDSS '99, pages 1–15, Reston, VA, USA, 1999. The Internet Society.

[28] George Kadianakis. How to stop the onion denial (of service). https://blog.torproject.org/stop-the-onion-denial/, August 2020.

[29] Neal Krawetz. Tor 0day: Crashing the tor network. https://www.hackerfactor.com/blog/index.php?/archives/897-Tor-0day-Crashing-the-Tor-Network.html, October 2020.

[30] Jelena Mirkovic and Peter Reiher. A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[31] Fiona Fui-Hoon Nah. A study on tolerable waiting time: how long are web users willing to wait? *Behaviour & Information Technology*, 23(3):153–163, 2004.

[32] Yoav Nir and Valery Smyslov. Protecting Internet Key Exchange Protocol Version 2 (IKEv2) Implementations from Distributed Denial-of-Service Attacks. RFC 8019, Internet Engineering Task Force, 2016.

[33] Mohammad A. Noureddine, Ahmed M. Fawaz, Amanda Hsu, Cody Guldner, Sameer Vijay, Tamer Başar, and William H. Sanders. Revisiting client puzzles for state exhaustion attacks resilience. In *Proceedings of the 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN '19, pages 617–629, New York, NY, USA, 2019. IEEE.

[34] Erik Nygren, Samuel Erb, Alex Biryukov, Dmitry Khovratovich, and Ari Juels. TLS Client Puzzles Extension. Internet-Draft draft-nygren-tls-client-puzzles-02, Internet Engineering Task Force, 2016. Expired.

[35] Bryan Parno, Dan Wendlandt, Elaine Shi, Adrian Perrig, Bruce Maggs, and Yih-Chun Hu. Portcullis: protecting connection setup from denial-of-capability attacks. In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '07, page 289–300, New York, NY, USA, 2007. ACM.

[36] Proton Mail. Proton mail: Get a private, secure, and encrypted email account. https://proton.me/mail, retrieved August 2024.

[37] Ling Ren and Srinivas Devadas. Bandwidth hard functions for asic resistance. In *Proceedings of the fifteenth Theory of Cryptography Conference*, TCC '17, pages 466–492, Cham, Switzerland, 2017. Springer.

[38] Marc Rennhard and Bernhard Plattner. Introducing morphmix: peer-to-peer based anonymous internet usage with collusion detection. In *Proceedings of the 2002 ACM Workshop on Privacy in the Electronic Society*, WPES '02, page 91–102, New York, NY, USA, 2002. ACM.

[39] Patrick Schleizer. Most onions down due to a denial of service attack on the tor network. https://forums.whonix.org/t/most-onions-down-due-to-a-denial-of-service-attack-on-the-tor-network/10979, January 2021.

[40] SecureDrop. Directory. https://securedrop.org/directory/, retrieved August 2024.

[41] SecureDrop. Share and accept documents securely. https://securedrop.org/, retrieved August 2024.

[42] Fabio Streun, Joel Wanner, and Adrian Perrig. Evaluating susceptibility of vpn implementations to dos attacks using adversarial testing. In *Proceedings of the Network and Distributed Systems Security Symposium 2022*, NDSS '22, pages 1–17, Reston, VA, USA, 2022. The Internet Society.

[43] Ceysun Sucu. Tor: Hidden service scaling. Master's thesis, University College London, 2015.

[44] The Invisible Internet Project. I2p anonymous network. https://geti2p.net/, retrieved December 2024.

[45] Tor Project. How do onion services work? https://community.torproject.org/onion-services/overview/, retrieved August 2024.

[46] Tor Project. Relay requirements. https://community.torproject.org/relay/relays-requirements/, retrieved August 2024.

[47] Tor Project. Denial-of-service defense extension (DOS_PARAMS). https://spec.torproject.org/rend-spec/introduction-protocol.html#EST_INTRO_DOS_EXT, retrieved December 2024.

[48] Tor Project. onionbalance/hs_v3/descriptor.py · main · the tor project / onion services / onionbalance · gitlab. https://gitlab.torproject.org/tpo/onion-services/onionbalance/-/blob/main/onionbalance/hs_v3/descriptor.py, retrieved December 2024.

[49] Tor Project. Proof of work for onion service introduction. https://spec.torproject.org/hspow-spec/index.html, retrieved December 2024.

[50] Tor Project. The tor project / core / tor · gitlab. https://gitlab.torproject.org/tpo/core/tor, retrieved December 2024.

[51] Tor Project. The tor project / onion services / onionbalance · gitlab. https://gitlab.torproject.org/tpo/onion-services/onionbalance, retrieved December 2024.

[52] Tor Project. Design - the onion services ecosystem. https://onionservices.torproject.org/apps/base/onionbalance/v2/design/#limitations, retrieved January 2025.

[53] Tor Project. Performance–tor metrics. https://metrics.torproject.org/onionperf-latencies.html, retrieved January 2025.

[54] Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. Captcha: Using hard ai problems for security. In *Proceedings of International Conference on the Theory and Applications of Cryptographic Techniques 2003*, EUROCRYPT '03, pages 294–311, Berlin, Germany, 2003. Springer.

[55] Xiaofeng Wang and Michael K. Reiter. Defending against denial-of-service attacks with puzzle auctions. In *Proceedings of the 2003 Symposium on Security and Privacy*, S&P '03, pages 78–92, New York, NY, USA, 2003. IEEE.

[56] Xiaofeng Wang and Michael K. Reiter. Mitigating bandwidth-exhaustion attacks using congestion puzzles. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, CCS '04, page 257–267, New York, NY, USA, 2004. ACM.

[57] Brent Waters, Ari Juels, J. Alex Halderman, and Edward W. Felten. New client puzzle outsourcing techniques for dos resistance. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, CCS '04, page 246–256, New York, NY, USA, 2004. ACM.

[58] Pavel Zoneff. Introducing proof-of-work defense for onion services. https://blog.torproject.org/introducing-proof-of-work-defense-for-onion-services/, August 2023.

A Client-Side Difficulty Adjustment

The current onion puzzle design heavily relies on the server-side difficulty update algorithm, as we discussed in Section 3.1.2. While it allows clients to adjust the difficulty of puzzles in theory, the current implementation rarely activates this in practice. Let us first present the detailed client-side puzzle difficulty adjustment algorithm in Algorithm 3.

Due to the lack of a direct communication channel between the client and service (because of the strict sender anonymity requirement), the client can only infer the failures of previous attempts based on timeouts. This process is inherently slow: since the client increases puzzle difficulty incrementally after each failure, it should experience several failures before reaching an appropriate difficulty level. Given that each failure can take a considerable amount of time (e.g., 90 seconds by default, though adjustable at runtime), the client-side difficulty adjustment is too slow to be practical.

Algorithm 3 Client-Side Difficulty Adjustment Algorithm

state:

D_{sug}: The suggested puzzle difficulty.

D_{client}: The difficulty of puzzles a client solves. It cannot be greater than $D_{client-max}$.

limit:

if $D_{sug} > D_{client-max}$ **then**
  $D_{client} \leftarrow D_{client-max}$
else
  $D_{client} \leftarrow D_{sug}$
end if

adjust:

while a rendezvous connection is not established or waiting time is less than an acceptable threshold **do**
  the client solves a puzzle with D_{client}
  if a rendezvous connection is established **then**
    break
  else if a certain amount of time elapses **then**
    $D_{client} \leftarrow \gamma \times D_{client}$ where γ is a constant
  end if
end while

Our experiments support this conclusion, as we barely observed such client-side difficulty adjustments, demonstrating the impracticality of the mechanism due to its slow feedback loop. Even worse, while we identified and addressed a few bugs in the current implementation that aggravated these delays, the mechanism remains too slow. This simple fix we made has been suggested to Tor developers, and all our experiments involving benign clients were conducted using the revised version.

B Additional Evaluation of Onion Puzzles' Effectiveness against Intro-Flooding Attacks

To assess the effectiveness of onion puzzles against intro-flooding attacks, we have conducted real-world attack experiments on the live Tor network in Section 3.1.4. This section presents two additional experiment results that complement our main discussion.

We repeated the experiment using the same parameters in two additional runs: once for 24 hours (Figure 10, top) and once for 27 hours (Figure 10, bottom). As before, the X-axis represents time, the main Y-axis (bars) indicates the number of requests aggregated for each update round, and the secondary Y-axis ($\times$) represents the suggested puzzle difficulty. Consistent with the main results, we observe that the adversary began to struggle in generating sufficient traffic as the suggested difficulty increased. Additionally, the patterns of puzzle difficulty changes exhibited similar fluctuations to some extent, aligning with the results presented in

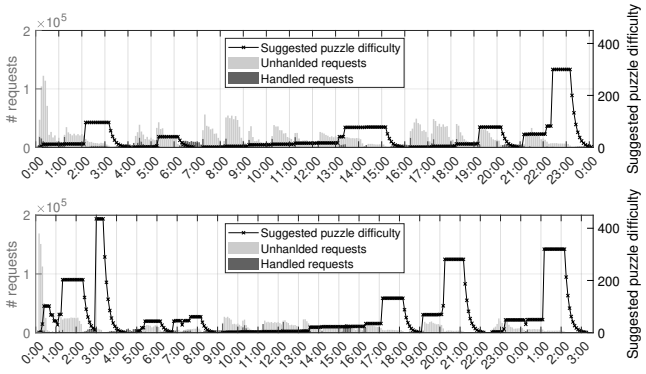


Figure 10: (Left Y-axis) Number of enqueued requests aggregated for each round; and (right Y-axis) changes in the suggested puzzle difficulty during intro-flooding attacks.

Section 3.1.4.

C Determination of Experiment Parameters

In Section 3.2.2, we evaluate our ONIONFLATION attack strategies on the live Tor network, targeting our own onion service. In this section, we briefly describe how several experiment parameters (e.g., puzzles difficulty, timing for initiating attacks, etc.) are determined.

First, we analytically model the onion puzzle mechanism and the attack process. This includes calculating the error in update timing inference for the end-rush strategy, the general latency of request transmission [53], and the average puzzle difficulty for each round. Throughout this process, we derive analytically optimal parameters that increase the suggested difficulty beyond 10,000 (the client-side maximum puzzle difficulty; see Appendix A) while minimizing the consumption of computing resources (i.e., the number of cores used). Next, we assess the effectiveness of our attack strategies with these calculated parameters and make incremental adjustments until they perform effectively in live experiments. These adjustments primarily involve fine-tuning puzzle difficulty and the number of cores allocated for the attacks. As a result, we identify parameter values that are both efficient and empirically proven to be effective in real-world experiments. While further optimizing our attacks through parameter adjustments may still be possible, we conclude that the current values are sufficiently efficient, particularly given the minimal cost of our attacks compared to naive intro-flooding attacks (0.2% and 0.6%; see Section 3.2.2).

D Cost Estimation Process

When estimating the cost of attacks based on the number of required hash operations, we assume that adversaries can scale their computing resources using cloud services. Specifically, we assume that the adversaries utilize Amazon EC2 M7i-flex [3] (M7i-flex.large) instances, as their per-core performance is comparable to that of the processors used in our experiments.

Although the time taken to solve puzzles has minimal impact on costs due to on-demand pricing [4], we assume that adversaries solve puzzles over the same duration as in our experiments (i.e., 7,185 seconds for Strategy ① and 6,930 seconds for Strategy ②), if possible.

The average number of hash operations needed to solve a puzzle of difficulty D is D itself, meaning adversaries must perform $\frac{R \times D}{T}$ hash operations per second to solve R requests of difficulty D within T seconds. Since each M7i-flex instance can perform approximately 500 hash operations per second, the required number of instances I can be calculated as

$$I = \lceil \frac{R \times D}{T} \times \frac{1}{500} \rceil.$$

Based on this number, the total cost C can be estimated as

$$C = I \times 0.09576 \times \frac{T}{3600}.$$

Similarly, the hourly cost C' can be estimated as

$$C' = I' \times 0.09576, \text{ where } I' = \lceil \frac{R \times D}{500} \rceil.$$