NinjaDoH: A Censorship-Resistant Moving Target DoH Server Using Hyperscalers and IPNS

Scott Seidenberger*, Marc Beret*, Raveen Wijewickrama†, Murtuza Jadliwala†, and Anindya Maiti*
*University of Oklahoma, Norman, OK, USA
†University of Texas at San Antonio, San Antonio, TX, USA
Email: seidenberger@ou.edu, marc.beret@ou.edu, raveen.wijewickrama@utsa.edu, murtuza.jadliwala@utsa.edu, am@ou.edu

***Abstract*—We introduce *NinjaDoH*, a novel DNS over HTTPS (DoH) protocol that leverages the InterPlanetary Name System (IPNS), along with public cloud infrastructure, to create a censorship-resistant moving target DoH service. *NinjaDoH* is specifically designed to evade traditional censorship methods that involve blocking DoH servers by IP addresses or domains by continually altering the server's network identifiers, significantly increasing the complexity of effectively censoring *NinjaDoH* traffic without disruption of other web traffic. We also present an analysis that quantifies the DNS query latency and financial costs of running our implementation of this protocol as a service. Further tests assess the ability of *NinjaDoH* to elude detection mechanisms, including both commercial firewall products and advanced machine learning-based detection systems. The results broadly support *NinjaDoH*'s efficacy as a robust, moving target DNS solution that can ensure continuous and secure internet access in environments with heavy DNS-based censorship.**

***Index Terms*—DoH, censorship resistance, DNS privacy, moving target defense.**

I. Introduction

The Domain Name System (DNS) is a crucial component of the Internet, responsible for translating human-readable domain names into machine-readable IP addresses. However, traditional DNS queries are sent in plaintext, making them vulnerable to various exploits by governments and organizations that enforce censorship through DNS-based firewalls. DNS-based firewalls have been used to enforce censorship against platforms such as Wikipedia, TikTok, and political websites in several regions [1]–[4]. Moreover, state-of-the-art web censorship circumvention proposals such as *NetShuffle* [5] require access to an uncensored DNS service, further necessitating a censorship-resistant DNS protocol.

DNS over TLS (DoT) [6], DNS over QUIC (DoQ) [7], and DNS over HTTPS (DoH) [8] encrypt DNS traffic, protecting it from eavesdropping, but DoT and DoQ operate on distinct ports (853/tcp and 853/udp, respectively), which make them easy to identify and block. In contrast, DoH integrates DNS requests with regular HTTPS traffic on port 443/tcp, the same port utilized for most encrypted web traffic. This makes it much harder for firewalls to block DoH without disrupting regular internet activities, as DoH is effectively masked within standard HTTPS traffic. As a result, DoH has the advantage of bypassing firewalls that block all outgoing DNS queries on port 53 or 853.

Efforts to counteract DoH's ability to evade DNS-based censorship have led to several research initiatives focusing on specifically detecting and blocking DoH traffic. Early approaches relied primarily on list-based methods, which are limited due to the increasing ease of self-hosting DoH servers [9], [10] on cloud infrastructure with vast IP address spaces that can make an infrequently updated blocklist ineffective [5], [11]. Entirely blocking outbound HTTPS traffic to hyperscalers such as AWS [12], Google Cloud [13], or Azure [14] is impractical for most censors as these hyperscalers also host numerous essential web services. More sophisticated, machine learning (ML)-based techniques [15]–[18] have been developed, emphasizing the necessity for more censorship-resistant DNS solutions.

In this paper, we introduce *NinjaDoH*, a novel DoH client-server protocol designed to be censorship-resistant. The *NinjaDoH* protocol employs a moving target defense by dynamically changing the server IP address through the use of public cloud infrastructure, and securely sharing the latest server IP address with the client(s). *NinjaDoH*'s client-side software continuously updates the operating system to use the most recent server IP address for DoH queries. To mitigate propagation delays in sharing of new IP addresses with the client, the server temporarily keeps older IP addresses active (alongside the new IP address), ensuring continuous availability for clients in-between IP updates. While there are various methods to securely share the latest server IP address, we leverage the InterPlanetary File System (IPFS) [19] for its decentralized nature, which makes it more difficult for adversaries to detect or block [20]. The *NinjaDoH* client integrates fully within the operating system, making it compatible with all browsers and applications without requiring any special configuration or additional plugin. This is in contrast to out-of-band DNS methods like DNS in Google Sheets [21] or DNS over Discord [22], which lack seamless integration with user environments.

Our key contributions in this paper are as follows:

- **Design and Implementation of *NinjaDoH* Protocol:** We design and then implement the *NinjaDoH* protocol, a moving

Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb) 2026
27 February 2026, San Diego, CA, USA
ISBN 978-1-970672-06-0
https://dx.doi.org/10.14722/madweb.2026.23006
www.ndss-symposium.org

target DoH server that leverages public cloud infrastructure and IPNS to dynamically rotate its IP addresses, making it resilient against list-based DNS blocking and detection methods. We provide our implementation on AWS via our code as an artifact.
- **Comprehensive Performance Evaluation:** We evaluate *NinjaDoH*'s performance in terms of DNS query latency and compare it against other DNS services, including well-known DoH providers and censorship-resistant alternatives like DoH over Tor. Our results demonstrate that *NinjaDoH* delivers low-latency performance, comparable to well-known public DoH services. We share our complete performance and ML-evasion datasets to serve as a foundational baseline for future research exploring moving-target defense architectures.
- **Evaluation of Censorship Resistance:** We empirically demonstrate *NinjaDoH*'s ability to evade both static, list-based blocking and more advanced ML-based detection systems that attempt to identify DoH traffic. This adaptability ensures the DoH service remains accessible in networks with heavy censorship while also being affordable.

II. Background and Related Work

Authenticated and Encrypted DNS. DNSSEC [23]–[25] was developed to authenticate query responses, but it lacked encryption. DoT [6] and DoQ [7], [26], [27] were later developed to encrypt DNS queries and thus protect DNS traffic from eavesdropping. Nonetheless, DoT, DoQ, and DNSSEC operate on distinct ports (853/tcp, 853/udp, and 53/tcp, respectively), making it easy for censors to detect and block [28]. When this traffic is blocked, users often have no choice but to use censoring DNS servers allowed in the censored network. DoH encrypts DNS traffic using standard HTTPS protocol, which is typically transmitted over port 443/tcp [8]. This allows DoH to blend in with regular web traffic, making it much harder for firewalls to detect and block without affecting normal HTTPS activity [28]. Proposals to enhance the privacy of DoH have been introduced, such as Oblivious DoH [29].

DNS Blocking and Filtering. Several previous studies have examined the extent of DNS-based censorship [3], [4], [30]–[41], wherein networks of various scales block access to external (unfiltered) DNS servers and mandate the use of DNS servers that provide manipulated or filtered responses. Jin et al. [42] found that using encrypted DNS resolvers allowed access to 37% of censored domains from vantage points in China, whereas none of the censored domains were accessible from Iran due to additional censorship methods such as SNI-based blocking of the websites [43], [44]. SNI-based blocking of websites can be bypassed using other censorship circumvention tools [45], [46].

DoH Blocking and Censorship Circumvention. Efforts to block DoH traffic range from static blocklists to ML-based classifiers [15]–[18], though many still exhibit high false-positive rates, underscoring the need for more resilient approaches like *NinjaDoH*. Other systems protect DNS queries by shifting them to alternative channels. DNS over Tor anonymizes query origin and destination via onion routing [47], while out-of-band methods such as DNS over Google Sheets [21] and DNS over Discord [22] provide unconventional but non user-friendly pathways. General censorship-circumvention tools including Tor [48] and VPNs [49]–[51] tunnel DNS and web traffic through overlay networks, but can themselves be blocked using deep packet inspection, IP filtering, or protocol fingerprinting [52]–[55]. Evasion techniques exist [56]–[58], yet the cat-and-mouse game between circumvention and censorship continues.

Moving Target Defense. Moving target defense strategies strengthen system resilience by continuously modifying system characteristics, adding complexity that disrupts adversaries' ability to predict and exploit vulnerabilities [59]–[61]. It is widely applied across different areas, including enhancing software security [62]–[68], establishing communication channels resistant to interference [69]–[71], protecting virtual machines hosted on shared infrastructure [72], [73], defending against DDoS attacks [74]–[78], securing critical infrastructure [79]–[83], and censorship-resistant web services [5], [11], [84], [85]. Dynamically changing (sub)domains have been explored as a moving target defense [5], [86]–[88], but they are unsuitable for *NinjaDoH* since they remain detectable without frequent root domain and IP changes [89], [90], and frequent root domain changes are economically constrained by ICANN's one-year registration minimum.

III. Adversary Model

Our adversary model for *NinjaDoH* assumes that censors implement DNS-based censorship by blocking access to external DNS resolvers and forcing users to rely on their own DNS servers with filtering rules (Figure 1). The censor controls the network, which may be managed by an ISP, an enterprise network administrator, or a government authority with regulatory oversight. We categorize the adversary's capabilities into the two following groups:

- **Blocking by IP or Domain Name**: Censors may attempt to block DoH traffic by maintaining blocklists of IP addresses or domains associated with DoH servers. *NinjaDoH* aims to counter this by employing dynamic IP address rotation using hyperscalers, making it difficult to maintain effective blocklists.
- **Machine Learning-Based Detection**: Advanced adversaries may use ML techniques to identify DoH traffic based on traffic flow characteristics. Although state-of-the-art ML models are capable of detecting regular DoH traffic, *NinjaDoH* aims to evade detection by minimizing flow durations by employing IP address rotation and using randomized DoH query paths to defeat active probing.

NinjaDoH is effective under the following assumptions about the censorship environment:

- **DNS-based Censorship**: The primary method of censorship is blocking external DNS resolvers and forcing users to use the censors' DNS servers that filter specific websites. *NinjaDoH* aims to provide a way to bypass these DNS restrictions.

- **Websites' IP Addresses Not Blocked**: It is assumed that the IP addresses of the websites themselves are not blocked at the network-level firewall. If the censor blocks entire IP ranges, clients must employ additional bypass techniques [5], [11], [45], [46], [56]–[58].
- **No Endpoint Control**: The adversary does not have control over the client endpoints. This implies that the adversary cannot force SSL/TLS interception or proxy use with SSL/TLS decryption, and users are free to install any software on their devices (such as the *NinjaDoH* client).
- **Neutral Cloud Provider**: *NinjaDoH* requires a neutral hyperscaler to host its server instance, ensuring that the cloud provider does not collude with the adversary to identify instances running *NinjaDoH* server code.
- **Use with Other Censorship Circumvention Tools**: *NinjaDoH* is not required on a VPN or the Tor network, as both of these tools can already enable access to external DNS resolvers. But if *NinjaDoH* is used in parallel with VPNs or Tor, it can provide low-latency DNS resolution, avoiding the higher latency caused by routing DNS queries through these networks. Moreover, in environments where Tor and VPN protocols (and as a result, censorship circumvention tools reliant on these protocols such as *SpotProxy* [11]) are blocked [52]–[55], *NinjaDoH* aims to provide an effective solution. *NinjaDoH will also improve the practicality and enable use of NetShuffle* [5], a web censorship circumvention protocol that can otherwise be blocked based on its static root domain usage.
- **Private Client-Server Communication**: *NinjaDoH* is designed to be hosted by individuals or small groups, ensuring that the secret shared between the server and the clients, which is used to communicate IP address changes, remains private. This prevents the adversary from knowing the latest IP addresses and subsequently blocking them.

IV. *NinjaDoH* SYSTEM MODEL

A. *System Requirements*

We define six requirements that address key dimensions of performance, resilience, scalability, and usability. **R1:** The system should provide low-latency DoH services, ensuring it performs efficiently under realistic workloads. **R2:** The system should outperform or match the performance of DoH over Tor, offering a practical and competitive alternative for censorship resistance. **R3:** It should bypass firewalls that rely on blocklists of known domains, IP addresses, or patterns, maintaining uninterrupted traffic flow in censored environments. **R4:** The system must evade detection by machine learning models, which includes three sub-requirements: **R4a:** It should resist detection by baseline ML detection models, ensuring resilience against existing state-of-the-art methods for identifying DoH traffic; **R4b:** It should remain resistant to adaptive adversaries, preventing detection even when adversaries specifically train their models on the system's traffic; and **R4c:** It must introduce scalability challenges for adversarial ML-based detection, making it impractical for adversaries to apply detection models at large scale. **R5:** The system should be cost-effective, enabling deployment and operation with reasonable resource consumption. **R6:** Finally, the system must afford a seamless and user-friendly experience, ensuring minimal disruption to regular internet usage and compatibility with common user environments. To meet these requirements, we present the *NinjaDoH* client-server architecture as shown in Figure 2.

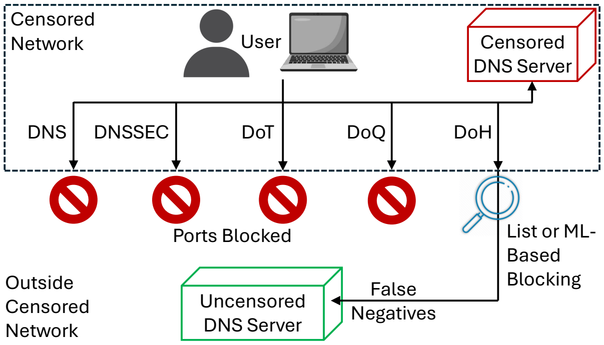


Fig. 1: Overview of the adversary model.

B. *The Server*

IP Address Space and Networking. The cornerstone of *NinjaDoH* is that it is a moving target DoH service. What makes it a moving target is that its IP address changes frequently, at a configurable frequency, a capability only made possible by having a large IP address allocation pool available to quickly swap in and swap out. Therefore, a public cloud provider that has access to such an IP allocation pool is a key component of this system design. We selected Amazon Web Services (AWS) as our public cloud provider, but we believe that any public cloud provider that allows for the dynamic allocation of IP addresses to a compute instance would be a suitable infrastructure layer for the system. At time of writing, the AWS IPv4 public address pool is estimated to be over 100 million addresses.

For the prototype system, three elastic network interfaces (ENI) were created inside a single region virtual private cloud (VPC). Each ENI functions as a virtual network interface card that can be dynamically assigned an IP address from AWS's IPv4 address pool. One of the ENIs serves to separate management plane traffic from the application traffic. The other two ENIs serve as the primary and alternate interfaces for application plane traffic. Both of these two application plane ENIs will route traffic to the same reverse proxy listening on all available interfaces. The *NinjaDoH* protocol can scale to an arbitrary number of ENIs in a deployment, given the compute instance can support the desired number of ENIs.

Compute. The system only requires modest compute resources, as the server runs a few lightweight services and does not require much persistent storage. The system was tested on a *t2.small* instance, which has 1vCPU and 2GB of RAM. Additionally, Ubuntu server's small operating system footprint and short interval persistent log keeping means that the block storage requirement is about 20GB.

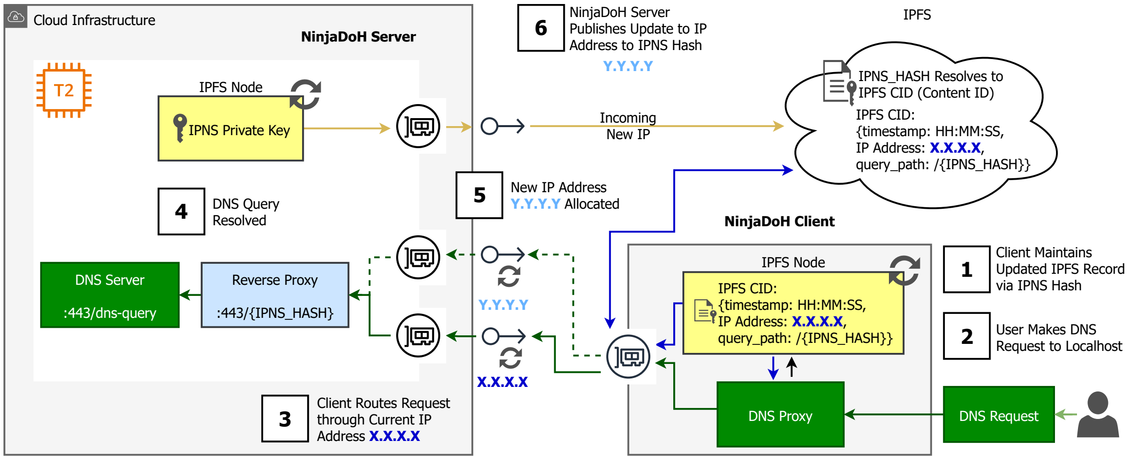

Cloud Infrastructure
NinjaDoH Server
T2
IPFS Node
IPNS Private Key
6
NinjaDoH Server Publishes Update to IP Address to IPNS Hash
Y.Y.Y.Y
IPFS
IPNS_HASH Resolves to IPFS CID (Content ID)
IPFS CID: {timestamp: HH:MM:SS, IP Address: X.X.X.X, query_path: /{IPNS_HASH}}
Incoming New IP
5
New IP Address Y.Y.Y.Y Allocated
4
DNS Query Resolved
NinjaDoH Client
DNS Server
:443/dns-query
Reverse Proxy
:443/{IPNS_HASH}
Y.Y.Y.Y
X.X.X.X
IPFS Node
IPFS CID: {timestamp: HH:MM:SS, IP Address: X.X.X.X, query_path: /{IPNS_HASH}}
1
Client Maintains Updated IPFS Record via IPNS Hash
2
User Makes DNS Request to Localhost
3
Client Routes Request through Current IP Address X.X.X.X
DNS Proxy
DNS Request

Fig. 2: Overview of the *NinjaDoH* protocol and architecture. The *NinjaDoH* client maintains an updated IPNS record with the latest server information via IPFS. When the user initiates a DNS request, it is first sent to the client's localhost DNS proxy, which then routes the request to the current IP address of the *NinjaDoH* server. At a configurable frequency, the server allocates a new IP address and publishes this update to IPNS. The client retrieves the updated IP address via IPNS and uses it for future DNS queries (until next IP update).

Software. Architecturally, the system requires a reverse proxy and a recursive DNS on the server. The reverse proxy has to be capable of forwarding HTTPS traffic from an arbitrary path from the application ENIs to the DNS service. The recursive DNS software has to be able to resolve DoH requests. Both have to be able to accept custom certificates and modify their certificates on-the-fly or with a short reload (so as not to affect the user experience). There are several software packages capable of meeting these requirements, but for our prototype we chose two widely deployed, free and open source software packages. The key external software dependencies of our implementation are *nginx* [91] and *AdGuard Home* [9]. A key, useful feature of *AdGuard Home* is that it can be configured to use multiple upstream DoH providers to resolve the client's query. When configured with several reputable upstream providers, there is a gained benefit of the user not creating an identifiable signature on any specific upstream provider.

Certificate Management. Certificate management ensures secure communication between clients and the DoH service. Given that *NinjaDoH* frequently rotates its public IP addresses, maintaining valid SSL/TLS certificates for these changing IP addresses presents unique challenges. Traditional certificate issuance processes are not designed for such dynamic environments, necessitating a custom solution. To address this, *NinjaDoH* employs an automated certificate generation and management process that dynamically creates and signs certificates whenever the set of public IP addresses changes. This process leverages a private Certificate Authority (CA) hosted on the server, allowing for rapid certificate issuance without relying on external CAs. Clients are configured to trust this private CA on setup, allowing them to trust new certificates on-the-fly despite the frequent IP changes. Additionally, the *NinjaDoH* CA can be installed solely for the local DNS proxy rather than system-wide, confining its trust scope to encrypted DNS traffic. While the ultimate safety of choosing which *NinjaDoH* server to trust lies with the user, the risk is mitigated by the user leveraging a secure, out-of-band channel that they trust for this key exchange.

The certificate management workflow is integrated into the system's orchestration code and operates as follows:

1) **Detection of IP Address Changes**: When the system allocates a new IP and associates it with one of the application ENIs, it triggers the certificate regeneration routine. This ensures that the certificates always reflect the current set of public IP addresses.
2) **Certificate Signing Request (CSR) Creation**: A CSR is created using the private key. The Common Name (CN) in the CSR is set to the newest IP address in the current list. An extension file is generated to include all current public IP addresses under the `subjectAltName` field. This ensures that the certificate is valid for any of the IP addresses clients may connect to.
3) **Certificate Signing & Reload**: The CSR is signed using the private CA's key and certificate to produce the server certificate. The certificate includes the SANs specified. After the new certificate is generated, the reverse proxy is reloaded to the updated certificate, ensuring that incoming connections are secured using the latest certificate without significant downtime.

Manipulating the Query Path. A DoH query requires a query path to reach the DoH service on the server. While not specifically required by the DoH RFC, Bottger et al. [28] show that the majority of DoH providers use the query path cited in the RFC examples, `/dns-query`. This pseudo-standard has been exploited as a technique to actively probe if an address is hosting a DoH service [15]. Therefore, the *NinjaDoH* protocol calls for using a non-standard query path. For our prototype, we set the query path as the IPNS hash. We propose that for a production environment, this path be randomized with varying lengths as to further obfuscate its packet structure signature.

By making this custom query path the only accessible path for accessing the DoH service, *NinjaDoH* prevents potential DoH downgrade attacks [92] and defeats identification via active probing on other common DNS server ports (such as 53 and 853). While query-path randomization does not fully obfuscate traffic, it hinders naive fingerprinting and reduces detection accuracy (see Section V-C).

IPFS Node. The InterPlanetary File System (IPFS) node distributes the updated public IP address to clients in a decentralized and censorship-resistant manner. Recent work has shown that IPFS maintains its functionality even under some of the most oppressive censorship regimes [20]. While certain IPFS gateways have been blocked in isolated events, the system does not rely on gateways as the user can connect directly to IPFS via a local node if a gateway is unavailable. Ongoing research has shown that sophisticated IPFS censorship strategies are actively being thwarted [93], and alternative mechanisms like domain-fronting remain viable if needed. By leveraging IPFS and the InterPlanetary Name System (IPNS), *NinjaDoH* disseminates routing information, without relying on traditional DNS infrastructure or via a censorable storage or communication medium. This process ensures that clients always have access to the current IP address, even as it changes frequently due to the moving target defense strategy.

The workflow for updating and publishing the IP address via IPFS and IPNS is integrated into the system's orchestration code and operates as follows:

1) **Adding Content to IPFS**: Upon allocating a new Elastic IP (EIP), the system creates a JSON object that includes the new IP address, a query path, and a timestamp. The JSON object is added to the local IPFS node using the IPFS HTTP API. The system receives a Content Identifier (CID) for the added content, which uniquely references the data in the IPFS network.
2) **Publishing to IPNS**: The system publishes the CID to IPNS using its private key. IPNS allows for mutable pointers to immutable content in IPFS, enabling clients to resolve the latest CID associated with a given IPNS address. The `PublishToIPFS` and `UpdateIPNSRecord` functions in Algorithm 1 make the new CID available to the client via IPNS using the specified key name.
3) **Verification and Propagation**: After publishing, the system verifies the new IPNS record by attempting to resolve it locally and may publish the new CID to the IPFS PubSub system to expedite propagation to connected peers. IPNS PubSub is experimental and we show in the evaluation that normal resolution via the DHT is sufficient.

The most critical step in this process is publishing the new CID to IPNS, which ensures that clients can always retrieve the latest IP address using the stable IPNS address. The parameters in the implementation code ensure that the publication is forced, resolves any conflicts, and sets appropriate lifetimes and time-to-live (TTL) values. After publishing, the function verifies the IPNS record by resolving it. By utilizing IPFS and IPNS, we avoid dependence on centralized services, which can be manipulated or blocked by adversaries.

Algorithm 1: NinjaDoHServerRoutine()

```
Data: Compute instance with set of network interfaces $l$;
      global address pool $p$; subset of IPs $p' \subset p$ where one
      IP maps to one interface $f(p' \rightarrow l)$
Result: IP address rotated and published via IPNS
while Service is running do
    Define: $l_x$ as the interface with the oldest IP
    // Add new IP from $p$ into $p'$ and assign it to $l_x$
    newIPAddress ← PullNewIP($p$);
    AssignIPToInterface(newIPAddress, $l_x$);
    // Release disassociated IP back to the pool $p$
    ReleaseDisassociatedIPs($p'$);
    // Use $p'$ to update certs
    UpdateSSLCertificates($p'$);
    // Generate a random query path
    queryPath ← GenerateRandomQueryPath();
    // Publish $p'$ via IPNS
    content ← CreateContent($p'$, queryPath, getTimestamp());
    cid ← PublishToIPFS(content);
    UpdateIPNSRecord(cid);
    // Wait for next scheduled rotation
    Sleep(rotationInterval);
end
```

Algorithm 2: NinjaDoHClientRoutine()

```
Data: IPNS key $k$; IPFS API endpoint $u$; DNS proxy
      configuration path $c$
Result: Continuous connectivity to the DoH server with
        dynamic IP updates
while Client is running do
    // Resolve the latest IPNS content
    cid ← ResolveIPNS($k$, $u$);
    if cid is valid then
        // Retrieve IP address and query path from
           IPFS
        (ip, queryPath) ← GetIPAndQueryPathFromIPFS(cid,
          $u$);
        if ip is new then
            // Update the DNS proxy config
            UpdateDNSConfig(ip, queryPath, $c$);
            ReloadDNSProxy($c$);
            // Log new IP address
            UpdateLastKnownIP(ip);
        end
        // Test DoH connectivity
        connectivity ← TestDoHConnectivity(ip);
        UpdateClientStatus(connectivity);
    end
    // Sleep before next update check
    Sleep(updateInterval);
end
```

This approach ensures continuous service availability by creating a "ladder" of IP addresses. As the system rotates IP addresses, clients connected via older IPs can still maintain their connections because each network interface always has a valid IP address assigned. By configuring a sufficient number of network interfaces and selecting an appropriate rotation interval, the system guarantees overlap between the old and new IP addresses. This overlap allows clients enough time to update to the new IP address before the one they are using is released back to the pool. Unlike a load balancer, which distributes traffic across multiple servers, *NinjaDoH* rotates IP addresses on a single server without redirecting client requests. This strategy ensures there is no downtime for valid IP addresses, preventing dropped requests and maintaining a seamless user experience during IP rotations.

C. The Client

The client component of *NinjaDoH* is designed to securely and reliably connect to the moving target DoH service provided by the server. Given the frequent rotation of the server's IP addresses and the utilization of a private CA, the client must dynamically update its configuration to maintain seamless connectivity. To achieve this, the client leverages its own access to IPFS to retrieve the latest server information.

IPNS Key Exchange & Root Certificate Installation. A foundational aspect of the client's operation is the secure exchange of a secret between the client and server during the initial setup. This secret is the IPNS name hash used by the server to publish its current IP address in IPFS. By possessing this IPNS key, the client can resolve the server's latest IP address. This mechanism ensures that only authorized clients, who have obtained the IPNS key through a secure channel, can access the service.

Securely exchanging this secret presents a challenge, to which we propose several mechanisms. Steganographic techniques can embed the IPNS key within images, audio files, or other media without arousing suspicion [94]. Another mechanism is to leverage end-to-end encrypted messaging applications such as Signal or WhatsApp who are designed to resist surveillance and interception [95]–[97]. While this out-of-band secret exchange could become a bottleneck to scaling a single *NinjaDoH* instance, our focus is on using *NinjaDoH* as an enabling mechanism for smaller, invite-only groups or as a bootstrap for other moving-target defenses such as *NetShuffle*.

Since the server uses a private CA to issue SSL/TLS certificates for its frequently changing IP addresses, clients must additionally install the server's root CA certificate on their machines. The transmission of the private CA certificate can be accomplished in a similar manner as the IPNS hash, or once the IPNS hash has been secretly received, the client can pull the private CA and the client software from IPFS.

Client Software and Operation. We implement a prototype *NinjaDoH* client in Python, as a terminal application that performs the key functions to ensure continuous access to a *NinjaDoH* server as seen in the Figure 9 screenshot. It interacts with a local IPFS node to resolve the IPNS key and retrieve the latest CID with the server's IP address information. Using the CID, the client fetches the JSON object containing the server's current IP address, query path, and timestamp. If the retrieved IP address differs from the one previously used, the client updates its local DNS resolver configuration accordingly. The client could use any other means to retrieve IPFS content, such as an IPFS gateway or access to a non-local IPFS node.

To update the DNS resolver configuration, the client utilizes `dnscrypt-proxy`[1], a flexible DNS proxy that supports encrypted DNS protocols, including DoH. The client generates a new DNS stamp based on the latest IP address and query path, then modifies the `dnscrypt-proxy` configuration file to include this updated stamp. Following the configuration update, the client reloads the proxy to apply the changes, ensuring that subsequent DNS queries are forwarded to the updated DoH server address.

Moreover, our implementation of the client continuously monitors connectivity to the server by performing DNS queries and measuring latency. Our interface offers users immediate feedback about the system's status and any connectivity problems. Additionally, *without a valid connection to the NinjaDoH server, DNS queries will fail, which will prevent unintentional DNS leaks. NinjaDoH* is resistant to DoH downgrade attacks [92] as long as browsers and applications are configured to use the operating system's default DNS settings, which are continuously updated by the *NinjaDoH* client.

To ensure timely updates, the client can listen for changes on the IPNS PubSub topic associated with the shared IPNS key or at some configurable frequency (for our implementation every 5 seconds), to resolve updates via IPNS. When the server publishes a new update, indicating a change in the service's IP address, the client retrieves the latest information and repeats the configuration update process. This mechanism allows the client to follow the server's moving target defense strategy without manual intervention of the end user. As long as the client is able to update the IP address of the server before that IP address is rotated out completely (a function of how many ENIs the server uses and its rotation interval), the client will retain service. One of the added benefits of a *NinjaDoH* client is that it maintains compatibility with existing operating systems, software, and the current DNS system, unlike out-of-band methods, as discussed previously.

Dependencies. The client relies on a few key dependencies and environmental requirements. An IPFS node or gateway must be accessible to resolve IPNS records and retrieve content from the IPFS network. A capable DNS proxy must be installed to handle encrypted DNS queries and support dynamic updates of the server's stamp. The client routine is illustrated in Algorithm 2.

V. Evaluation

We designed the system to meet the system requirements in Section IV-A, informed by the adversary model in Section III. Now, using our implementation of the system, we evaluate it against the system requirements.

A. Baselining Latency

To baseline the latency between the different DoH providers, we queried the top domains from the Cisco Umbrella Top 1M list[2], recording both ping and query response times. We used randomized subdomains to help prevent upstream caching.

[1] https://github.com/DNSCrypt/dnscrypt-proxy

[2] https://umbrella-static.s3-us-west-1.amazonaws.com/index.html

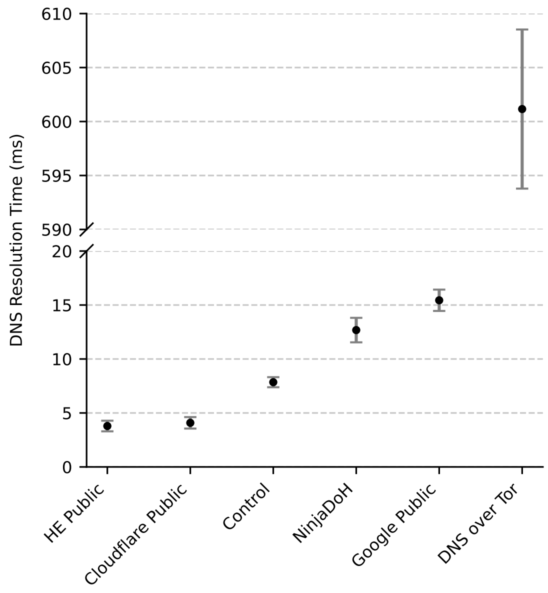


Fig. 3: Mean DNS resolution time with 95% confidence intervals for different DNS servers. The y-axis includes a break to show the large discrepancy between standard DNS servers and DNS over Tor.

Public DoH Providers. We evaluated the performance of different DNS resolvers, including the *NinjaDoH* server, a Control server that was hosted on the same AWS region with just *AdGuard Home* configured identically to the *NinjaDoH* implementation, and a group of Public DNS resolvers (Cloudflare, Google, and Hurricane Electric (HE)). The primary metric is the DNS resolution time, which is the ping adjusted time to return the DNS query measured in milliseconds (ms).

The *NinjaDoH* server demonstrated an average resolution time of **12.68** ms, comparable to the Control server's resolution time of **7.85** ms, indicating negligible performance differences. The Public DNS group demonstrated an average resolution time of **7.77** ms.

The confidence intervals indicate that the difference between the *NinjaDoH* and both the Control server and public DoH providers is, for an end user, practically negligible, where our system adds about 4–5 ms of latency to each request. This can be explained by both the added step of the client making a call to its local proxy instead of directly to the server and the slight delay during IP rotation. During this collection period, the *NinjaDoH* rotated IP addresses three times and had zero dropped client queries.

DNS over Tor. In this section, we compare the performance of *NinjaDoH* with DNS over Tor. While Tor itself may be blocked in the strictest censorship regimes, DoH over Tor is still a well-known, readily deployable solution for avoiding DNS-based blocking and represents a widely accepted reference point in the censorship-circumvention literature. Moreover, even in environments where Tor's default configuration is blocked, its pluggable transports often remain viable. We therefore use DNS over Tor as a natural, real-world baseline for censorship-resistant DNS resolution despite its performance overhead. The primary metric of comparison remains the DNS resolution time. DNS over Tor provides censorship-resistance by routing DNS requests through the Tor network, but it comes at the cost of increased latency. For the DNS over Tor experiments, we used Cloudflare's DoH service on Tor[3]. In total, 10 Tor circuits were used to collect data, with each circuit performing DoH lookups for 10 domains. The experiments were conducted by repeatedly generating new Tor circuits using the `NEWNYM` signal to obtain fresh exit nodes.

The *NinjaDoH* setup provided a significantly lower average resolution time, with a mean ping adjusted query time of **12.68** ms, compared to DNS over Tor, which exhibited a much higher mean resolution time of **601.16** ms. This large discrepancy in DNS resolution times highlights the latency overhead introduced by Tor's network routing. Figure 3 shows the combined, ping adjusted mean query resolution times for the public DoH providers, the control, NinjaDoH, and DNS over Tor.

> **Fast and Efficient DoH Service (R1 & R2).** Our evaluation confirms that *NinjaDoH* provides low-latency DoH services with an average resolution time of 12.68 ms. This is only about 4.91 ms slower than public DNS resolvers (average 7.77 ms). In stark contrast, DNS over Tor exhibits a significantly higher average resolution time of 601.16 ms, demonstrating the latency overhead of Tor's network routing. These results fulfill R1 by demonstrating efficient performance under realistic workloads and R2 by offering a practical alternative to existing censorship-resistant techniques with minimal additional latency.

B. *List-based Firewall Evasion*

List-based firewall blocking is the most prevalent method to restrict access to DoH services and enforce censorship policies. These firewalls maintain static blocklists of known DoH server domains and IP addresses, often sourced from public repositories. These can be curated lists specifically targeting DoH providers, or in the most extreme, lists to block all known public DNS providers.[4][5] Commercial providers advertise capabilities that they have the most up-to-date and relevant blocklists, and some governments and ISPs offer a Protective DNS (PDNS) service to their constituents and users which rely on such updated lists [98]. By blocking these known entities, censors aim to compel users to either fallback to unencrypted DNS queries, or use resolvers that they control, which can then be readily surveilled.

To evaluate the efficacy of *NinjaDoH* in evading such list-based blocking methods, we evaluated it against techniques across OSI layers. As summarized in Table I, *NinjaDoH* successfully bypasses domain and IP blocking by not utilizing a domain name and by dynamically changing its IP address. Additionally, it evades application-level identification by mimicking regular HTTPS traffic, avoiding typical DoH

[3] dns4torpnlfs2ifuz2s2yf3fc7rdmsbhm6rw75euj35pac6ap25zgqad.onion

[4] https://github.com/curl/curl/wiki/DNS-over-HTTPS

[5] https://public-dns.info/nameservers.txt

traffic patterns through mechanisms like changing the query path that could trigger deep packet inspection filters. While *NinjaDoH* cannot circumvent strict IP or domain allowlisting, where all unknown IPs and domains are blocked, this method is highly restrictive, costly to maintain, and impractical for most network environments.

> **Bypasses Firewall Blocklists (R3).** *NinjaDoH* successfully evades 4 out of 5 common DoH blocking methods employed by popular firewalls, including domain blocking, IP blocking, application identification, and SNI blocking. By not utilizing domain names, dynamically changing IP addresses, and mimicking regular HTTPS traffic, *NinjaDoH* maintains uninterrupted traffic flow in censored environments where blocklists are employed. This fulfills R3 by effectively bypassing current, prevalent censorship techniques.

C. Machine Learning Evasion

To evaluate the performance of *NinjaDoH* against machine learning (ML) detection techniques, various models from previous works have been selected. ML-based inspection of HTTPS traffic has become increasingly prevalent, and DoH is not inherently immune. Using the methodology of Jerabek et al. [15] with tooling from MontazeriShatoori et al. [16] (hereafter referred to as DoHlyzer[6]), five state-of-the-art model architectures were evaluated.

Models. Four of the models were Deep Neural Networks (DNNs). The *LSTM-Based Model* uses Long Short-Term Memory units to capture temporal dependencies in network traffic sequences, making it suitable for time-series data. *The Fully Dense Model* processes traffic flows independently, which allows for faster computation but may miss sequential patterns. The *CNN-Based Model* employs 1D Convolutional Neural Networks to detect local patterns within the data, identifying spatial or temporal correlations. Finally the *Hybrid LSTM-Dense Model* combines dense and LSTM layers, balancing feature extraction with sequential modeling for improved performance. Each DNN model was trained using different flow sequence lengths (from 4 to 10) to assess how traffic sequence length impacts detection performance. This allows for evaluation of both short-term and long-term traffic patterns. For the decision tree classifier, we reproduce the XGBoost model by Jerabek et al. [15] with original code and training data. XGBoost is a gradient boosting decision tree algorithm known for its effectiveness in structured data analysis.

Datasets. The training dataset used for the DNN models was sourced from the DoHLyzer repository, containing both DoH and non-DoH samples with time-series features. For the XGBoost model, the training dataset consists of PCAP files extracted from the `DoH-Gen-F-AABBC` database [99]. This dataset features encrypted traffic enriched with key TLS attributes such as `TLS_ALPN`, `TLS_JA3`, and `TLS_SNI`, which provide rich feature sets for training and evaluating ML models. Additionally, the dataset includes a list of known, public DoH providers' IP addresses.

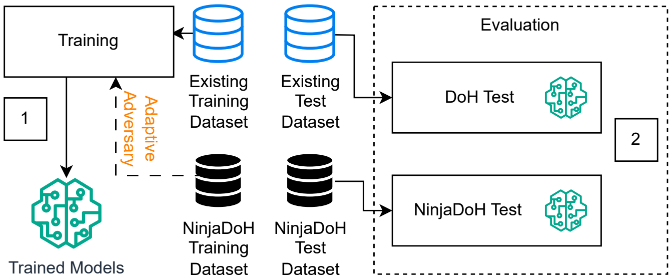


Fig. 4: Evaluation model to detect *NinjaDoH* traffic.

Flow Stitching. To prepare the training data, the traffic was consolidated into bidirectional flows, grouping packets based on common characteristics, such as source and destination IP addresses, ports, protocols, and timestamps, creating a comprehensive view of the entire traffic session. Some of the extracted features of the bidirectional flows from the PCAP files are *Mean Payload Size, Number of Packets, Client-to-Server Packet Ratio, and Mean Time Between Packets.* There are over a dozen features extracted from the flows, and the current approaches using these features have been shown to detect DoH flows. A flow is classified as DoH or non-DoH by the IP address list of known DoH IP addresses found in the dataset.

Training. The dataset was split it into a training set and a test set. The models were then trained on the training set (Step 1 in Figure 4). Following the training phase, the models were evaluated on the test set (Step 2 in Figure 4). The results from this evaluation on the baseline training data is consistent with the target from the prior work. F1 scores, precision, and recall metrics all ranging between 0.98 and 0.99, demonstrating the effectiveness of the models in detecting DoH traffic. Each DNN model was trained with varying flow sequence lengths, ranging from 4 to 10 timesteps. The sequence length, or the number of timesteps, plays a crucial role in capturing temporal dependencies within the data, especially for models like LSTMs that are designed for sequential information. By experimenting with different sequence lengths, we aimed to find the optimal configuration that maximized the model's performance. After training and evaluating the models across the various sequence lengths, the best sequence number for each model was determined based on the highest F1 score achieved during testing to balance precision and recall.

Results. We then evaluated the trained models against new traffic, this time captured from a *NinjaDoH* session. We captured three-minute's worth of typical browsing activity by an end-user who was using *NinjaDoH*, which became the evaluation set. The resulting PCAP file is processed to bidirectional flow format for compatibility with the models. Each trained model is subsequently given the evaluation set to see how well it can detect *NinjaDoH* traffic. In this scenario, the adversary faces trade-offs: false positives impact network usability, emphasizing the importance of precision, while missed detections (recall) reduce the adversary's effectiveness in censorship.

[6] https://github.com/ahlashkari/DoHLyzer

Undetectable by Current ML Models (R4a). The system effectively resists detection by baseline ML detection models. As shown in the Figure 5, the models achieve an average recall of 0.506, which is comparable to random guessing. Precision and F1-scores similarly reflect poor detection capability. This meets R4a by demonstrating resilience against existing state-of-the-art methods for identifying DoH traffic, rendering the detection models ineffective against *NinjaDoH* traffic.

D. Adaptive Adversary Evasion

In our evaluations so far, we have assumed a static adversary who employs their current censorship strategies and is either unaware of, or specifically not targeting *NinjaDoH* users. In this section, we evaluate our system against a notional, adaptive adversary that is both aware of, and specifically targeting *NinjaDoH*. This adversary can deploy their own *NinjaDoH* server outside their network, and then monitor the traffic from a client that they control within their network. There are several configurable parameters to a *NinjaDoH* deployment that an attacker wouldn't know, such as the IP rotation interval (the interval itself could be stochastic), the IP address blocks in use, or any additional traffic shaping techniques employed by the system deployer. However, they would know the general architecture and be able to deploy their own instance and experiment with their own parameters

Niktabe et al. [17] show that linear models specifically trained on known malicious DoH traffic data can have performance similar to deep neural networks in classification performance. This illustrates the potential power of an adversary who specifically trains a model against known *NinjaDoH* traffic. Linear models will have a faster inference time than deep neural networks in evaluating new traffic. However, the authors note that there still exists "Inadequate detection of obfuscated and disguised malicious traffic." Therefore, even if a model is specifically trained on *NinjaDoH* traffic, it will have to be able to conduct its inference fast enough in order to actually deny the service at a practical level.

Most previously known DoH providers are eventually censored by these approaches because once the censor has determined a domain or IP is classified as a DoH provider, that domain or IP is added to a blocklist. Since *NinjaDoH* is IP-agile and doesn't use domain names, *even if an adaptive adversary has a perfect model for detecting NinjaDoH traffic, they will only be able to temporarily deny service after the user's first query until the next IP rotation.*

To evaluate this, we assumed the role of a determined adversary that deployed their own system with perfect visibility into ground-truth network traffic. The system was configured with a pseudo-random IP rotation frequency between 1–3 minutes. Data was collected over a 15 minute period, which resulted in 10 distinct IP rotations and a >1 GB PCAP file to use for training. The traffic was captured and processed in the same manner as in Section V-C. This ground-truth *NinjaDoH* traffic is mixed with both "benign" DoH traffic and non-DoH traffic for training the models. The newly trained adversarial model was used on the same evaluation set as before, and Figure 6 shows that performance improves across all the model types that we evaluated with the *NinjaDoH* traffic present in the training set.

There are a number of reasons why this tailored model shows improvement. In the data processing step, flows are extracted from the `pcap` file, and then are processed into both statistical and time series features. Both of these feature sets that the models are trained on use flow sequences of packets between the DoH server and the client. We found that the *NinjaDoH* traffic produced a normally-distributed number of "clumps" of packets in each flow sequence, with $\mu = 10.30$, $\sigma = 5.20$. However, the benign DoH traffic produced much higher average clump sizes ($\mu = 135.64$) and a reflects a log-normal distribution with $\sigma = 1.57$, $\lambda = 26.68$. Therefore, it possible that the sequences of *NinjaDoH* traffic from one IP to the next are features that can be learned, leading to the model gaining predictive power from the information-sparse *NinjaDoH* flows. However, this improved performance does not currently approach an acceptable level for a would-be censor.

Resilience to Adaptive Adversaries (R4b). Even when facing adaptive adversaries who specifically train models on *NinjaDoH* traffic, detection performance improves but remains insufficient for effective censorship. The best model trained with *NinjaDoH* traffic achieves a precision of 0.764, recall of 0.635, and F1-score of 0.578. While these metrics are higher than those without targeted training, they do not reach levels that would allow a censor to reliably detect and block *NinjaDoH* traffic. This fulfills R4b by showing that the system remains resistant to adversaries who tailor their models to detect its traffic.

E. Scalability of Adversarial ML-based Detection

Detecting DoH traffic in large networks using ML models presents significant scalability challenges, especially against *NinjaDoH*. To block traffic effectively, an adversary must capture enough packets to reconstruct flows, extract features, and perform inference before the *NinjaDoH* client rotates to a new IP address. As network scale increases, the feasibility of this approach diminishes due to the growing data volume and computational demands.

To illustrate the computational burden on an adversary, we model the detection process and estimate the time required to process network traffic at different scales. Our model simulates the adversary's detection process, incorporating components that reflect real-world network conditions and constraints.

Flow arrivals are modeled as a Poisson process to represent the random nature of flow initiations in a network. The inter-arrival times (T_a) are exponentially distributed with rate λ, where $\lambda = \frac{\text{Flows per Minute}}{60}$. Flow durations ($D$) are modeled using a log-normal distribution to reflect the positive skew observed in real data, as evidenced by our collected flow duration statistics (Table II). This distribution accounts for the wide variability in flow lengths, with many short flows and a few long-lasting ones.

The adversary must process each flow within the IP rotation interval ($T_{\text{rotation}} = 60\,\text{s}$) , with a fixed processing time per flow (T_p), such as $0.1\,\text{ms}$, $1\,\text{ms}$, or $10\,\text{ms}$. Processing starts after the flow ends and a processor becomes available, calculated as $S_i = \max(E_i, A_j)$, where E_i is the flow end time and A_j is the processor's availability.

The performance metric of interest is the probability of detecting DoH flows ($P_{\text{detect\_DoH}}$). This is calculated by dividing the expected number of detected DoH flows by the total number of DoH flows. The expected detected DoH flows are computed as the product of the processed DoH flows and the true positive rate (TPR), i.e., $F_{\text{processed\_DoH}} \times \text{TPR}$.

$$P_{\text{detect\_DoH}} = \min\left(1,\ \frac{N_{\text{proc}}}{\lambda\, T_p}\right) \times \text{TPR}$$

Figure 8 presents the simulation results that show that unless the adversary can process all flows in near real-time, the probability of detecting DoH traffic declines significantly. Achieving this requires substantial computational resources, which is impractical and cost-prohibitive at scale. This scaling becomes even more impractical when considering the additional overhead of packet capture, flow reconstruction, and feature extraction required for the detection process.

Moreover, even if the adversary manages to detect and block an IP address, the benefit is short-lived due to frequent IP rotations. The adversary would need continuous detection process, incurring ongoing computational costs. This imposes a disproportionate burden on the adversary compared to potential minimal disruption caused to *NinjaDoH* users. This asymmetry underscores the inherent scalability challenges faced by adversaries attempting to detect and block DoH traffic in large-scale networks.

> **Scaling ML Detection is Impractical (R4c).** Our modeling suggests that adversaries face significant scalability challenges when attempting to detect *NinjaDoH* traffic using ML-based methods at large scale. Even with such resources, the detection probability remains limited by the ML model's true positive rate. This satisfies R4c by demonstrating that widespread ML-based detection is infeasible due to the disproportionate computational costs involved.

F. Costs to Deploy & Integrate

At time of writing, to implement our prototype of the server-side, the price is \$23.55 USD/month. Even cheaper infrastructure for the server may be available on different cloud providers, or by using more lightweight DNS resolver software. Monthly subscriptions for VPNs for a single user cost between \$10 and \$15, so this is a cost-efficient system that can deliver censorship-resistant DoH-as-a-service to many users for slightly more than a single VPN subscription. While this is not a VPN, it is useful for comparison in terms of cost to an end user. As seen in Figure 9, the screenshot of the prototype, the client implementation integrates into the end-user's existing operating system and workflow with no additional configuration or software to configure.

> **Affordable and User-Friendly (R5 & R6).** The system fulfills R5 by offering cost-efficiency for individual users, while scaling up to larger cloud instances remains affordable for privacy-focused groups and communities. It satisfies R6 through OS-level client integration, ensuring seamless compatibility with existing operating systems, software, and the current DNS infrastructure.

VI. Discussion and Future Work

Next, we discuss the practical applications of *NinjaDoH* and potential future work in censorship circumvention.

***NinjaDoH* Client as an Edge DNS Server.** While *NinjaDoH* was presented as a client-side DNS tool, its design naturally supports recursive use. A *NinjaDoH* client inside a censored network can operate as an uncensored edge DNS server (Figure 7), allowing others to query it directly. This removes the need for per-user *NinjaDoH* servers or shared IPNS hashes, simplifying deployment and improving scalability.

Performance and Usability. *NinjaDoH* achieves DNS latency roughly 50 times lower than the censorship-resistant DoH system built on Tor. It provides performance comparable to standard DoH, while Tor-based approaches incur heavy delays due to multi-hop routing. *NinjaDoH* also allows users to deploy their own infrastructure or rely on a trusted public instance, avoiding dependence on Tor's relays and exit nodes, which introduce overhead and additional vulnerabilities [100]. This reduces the attack surface and improves overall security.

Use of Hyperscalers. Similar to recent systems such as *NetShuffle* [5] and *SpotProxy* [11], *NinjaDoH* leverages cloud resources to rapidly rotate IP addresses, hindering effective blocklisting. At the same time, P2P overlays such as IPFS enable decentralized distribution of service information without relying on censorable DNS infrastructure. This novel combination of cloud agility and P2P networking shows how not only software solutions but also innovative use of on-demand, software-defined infrastructure can address complex security challenges.

Evaluation of Censorship Resistance. We demonstrate robust resistance against current methods of blocking DoH traffic, with additional security evaluations in Appendix B. It effectively evades list-based blocking techniques due to its dynamic IP rotation and lack of reliance on domain names. Even when an adaptive adversary trains models on *NinjaDoH* traffic, the cost to implement a real-time system with acceptable false positive rates is high, suggesting that it can withstand sophisticated censorship attempts.

VII. Conclusion

NinjaDoH is a moving target DNS over HTTPS protocol that rotates hyperscaler IP addresses and advertises them via IPNS instead of domains. Experiments with commercial firewalls and state-of-the-art ML models show that *NinjaDoH* can frustrate both list-based and learning-based blocking, significantly raising the real-time resources required for censorship. *NinjaDoH* helps secure DNS access and can bootstrap richer circumvention systems in censored networks.

References

[1] Wikimedia Foundation, "Wikimedia Foundation urges Pakistan Telecommunications Authority to restore access to Wikipedia in Pakistan," https://wikimediafoundation.org/news/2023/02/03/wikimedia-foundation-urges-pakistan-telecommunications-authority-to-restore-access-to-wikipedia-in-pakistan/, 2023.

[2] Haeluya Hadero, "TikTok compares itself to foreign-owned American news outlets as it fights forced sale or ban," https://apnews.com/article/tiktok-bytedance-ban-sell-china-e2d927825f1539c787c4b15b74b0919a, 2024.

[3] A. Master, "Modeling and characterization of internet censorship technologies," Ph.D. dissertation, Purdue University, 2023.

[4] N. P. Hoang, A. A. Niaki, J. Dalek, J. Knockel, P. Lin, B. Marczak, M. Crete-Nishihata, P. Gill, and M. Polychronakis, "How great is the great firewall? measuring china's {DNS} censorship," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 3381–3398.

[5] P. T. J. Kon, A. Gattani, D. Saharia, T. Cao, D. Barradas, A. Chen, M. Sherr, and B. E. Ujcich, "Netshuffle: Circumventing censorship with shuffle proxies at the edge," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 3497–3514.

[6] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. Hoffman, "Specification for DNS over transport layer security (TLS)," Tech. Rep., 2016.

[7] C. Huitema, S. Dickinson, and A. Mankin, "RFC 9250: DNS over Dedicated QUIC Connections," 2022.

[8] P. Hoffman and P. McManus, "DNS queries over HTTPS (DoH)," Tech. Rep., 2018.

[9] AdGuard, "AdGuard Home," https://github.com/AdguardTeam/AdGuardHome, 2024.

[10] Pi-hole, "Pi-hole," https://github.com/pi-hole, 2024.

[11] P. T. J. Kon, S. Kamali, J. Pei, D. Barradas, A. Chen, M. Sherr, and M. Yung, "{SpotProxy}: Rediscovering the cloud for censorship circumvention," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 2653–2670.

[12] Amazon, "Amazon Web Services (AWS)," https://aws.amazon.com, 2023.

[13] Google, "Google Cloud Platform," https://cloud.google.com, 2023.

[14] Microsoft, "Microsoft Azure," https://azure.microsoft.com, 2023.

[15] K. Jerabek, K. Hynek, O. Rysavy, and I. Burgetova, "Dns over https detection using standard flow telemetry," *IEEE Access*, vol. 11, pp. 50 000–50 012, 2023.

[16] M. MontazeriShatoori, L. Davidson, G. Kaur, and A. H. Lashkari, "Detection of doh tunnels using time-series classification of encrypted traffic," in *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*. IEEE, 2020, pp. 63–70.

[17] S. Niktabe, A. H. Lashkari, and D. P. Sharma, "Detection, characterization, and profiling DoH Malicious traffic using statistical pattern recognition," *International Journal of Information Security*, vol. 23, no. 2, pp. 1293–1316, 2024.

[18] R. Mitsuhashi, Y. Jin, K. Iida, T. Shinagawa, and Y. Takai, "Malicious dns tunnel tool recognition using persistent doh traffic analysis," *IEEE Transactions on Network and Service Management*, vol. 20, no. 2, pp. 2086–2095, 2022.

[19] Protocol Labs, "InterPlanetary File System (IPFS)," https://ipfs.tech/, 2024.

[20] L. Balduf, S. Rust, and B. Scheuermann, "I'm InterPlanetary, Get Me Out of Here! Accessing IPFS From Restrictive Environments," in *Proceedings of the 4th International Workshop on Distributed Infrastructure for the Common Good*, 2023, pp. 13–18.

[21] Cloudflare, "DNS in Google Sheets," https://developers.cloudflare.com/1.1.1.1/other-ways-to-use-1.1.1.1/dns-in-google-sheets/, 2024.

[22] ——, "DNS over Discord," https://developers.cloudflare.com/1.1.1.1/other-ways-to-use-1.1.1.1/dns-over-discord/, 2024.

[23] G. Ateniese and S. Mangard, "A new approach to dns security (dnssec)," in *Proceedings of the 8th ACM conference on Computer and Communications Security*, 2001, pp. 86–95.

[24] W. Lian, E. Rescorla, H. Shacham, and S. Savage, "Measuring the practical impact of {DNSSEC} deployment," in *22nd USENIX Security Symposium (USENIX Security 13)*, 2013, pp. 573–588.

[25] T. Chung, R. van Rijswijk-Deij, B. Chandrasekaran, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, "A longitudinal,{End-to-End} view of the {DNSSEC} ecosystem," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 1307–1322.

[26] M. Kosek, L. Schumann, R. Marx, T. V. Doan, and V. Bajpai, "Dns privacy with speed? evaluating dns over quic and its impact on web performance," in *Proceedings of the 22nd ACM Internet Measurement Conference*, 2022, pp. 44–50.

[27] M. Kosek, T. V. Doan, M. Granderath, and V. Bajpai, "One to rule them all? a first look at dns over quic," in *International Conference on Passive and Active Network Measurement*. Springer, 2022, pp. 537–551.

[28] T. Böttger, F. Cuadrado, G. Antichi, E. L. Fernandes, G. Tyson, I. Castro, and S. Uhlig, "An empirical study of the cost of dns-over-https," in *Proceedings of the Internet Measurement Conference*, 2019, pp. 15–21.

[29] S. Singanamalla, S. Chunhapanya, J. Hoyland, M. Vavruša, T. Verma, P. Wu, M. Fayed, K. Heimerl, N. Sullivan, and C. Wood, "Oblivious dns over https (odoh): A practical privacy enhancement to dns," *Proceedings on Privacy Enhancing Technologies*, 2021.

[30] P. Pearce, B. Jones, F. Li, R. Ensafi, N. Feamster, N. Weaver, and V. Paxson, "Global measurement of {DNS} manipulation," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 307–323.

[31] M. Wander, C. Boelmann, L. Schwittmann, and T. Weis, "Measurement of globally visible dns injection," *IEEE Access*, vol. 2, pp. 526–536, 2014.

[32] O. Farnan, J. Wright, and A. Darer, "Analysing censorship circumvention with vpns via dns cache snooping," in *2019 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2019, pp. 205–211.

[33] N. P. Hoang, S. Doreen, and M. Polychronakis, "Measuring {I2P} censorship at a global scale," in *9th USENIX Workshop on Free and Open Communications on the Internet (FOCI 19)*, 2019.

[34] A. A. Niaki, S. Cho, Z. Weinberg, N. P. Hoang, A. Razaghpanah, N. Christin, and P. Gill, "Iclab: A global, longitudinal internet censorship measurement platform," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 135–151.

[35] N. P. Hoang, J. Dalek, M. Crete-Nishihata, N. Christin, V. Yegneswaran, M. Polychronakis, and N. Feamster, "{GFWeb}: Measuring the great firewall's web censorship at scale," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 2617–2633.

[36] A. Bhaskar and P. Pearce, "Many roads lead to rome: How packet headers influence {DNS} censorship measurement," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 449–464.

[37] S. Burnett and N. Feamster, "Making sense of internet censorship: a new frontier for internet measurement," pp. 84–89, 2013.

[38] M. Kührer, T. Hupperich, J. Bushart, C. Rossow, and T. Holz, "Going wild: Large-scale classification of open dns resolvers," in *Proceedings of the 2015 Internet Measurement Conference*, 2015, pp. 355–368.

[39] P. Calle, L. Savitsky, A. N. Bhagoji, N. P. Hoang, and S. Cho, "Toward automated dns tampering detection using machine learning," *Free and Open Communications on the Internet*, 2024.

[40] K. Bock, Y. Fax, K. Reese, J. Singh, and D. Levin, "Detecting and evading {Censorship-in-Depth}: A case study of {Iran's} protocol whitelister," in *10th USENIX Workshop on Free and Open Communications on the Internet (FOCI 20)*, 2020.

[41] R. Sundara Raman, P. Shenoy, K. Kohls, and R. Ensafi, "Censored planet: An internet-wide, longitudinal censorship observatory," in *proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, 2020, pp. 49–66.

[42] L. Jin, S. Hao, H. Wang, and C. Cotton, "Understanding the impact of encrypted DNS on internet censorship," in *Proceedings of the Web Conference 2021*, 2021, pp. 484–495.

[43] W. M. Shbair, T. Cholez, J. François, and I. Chrisment, "Improving sni-based https security monitoring," in *2016 IEEE 36th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 2016, pp. 72–77.

[44] K. Bock, G. Naval, K. Reese, and D. Levin, "Even censors have a backup: Examining china's double https censorship middleboxes," in *Proceedings of the ACM SIGCOMM 2021 Workshop on Free and Open Communications on the Internet*, 2021, pp. 1–7.

[45] W. M. Shbair, T. Cholez, A. Goichot, and I. Chrisment, "Efficiently bypassing sni-based https filtering," in *2015 IFIP/IEEE International*

Symposium on Integrated Network Management (IM). IEEE, 2015, pp. 990–995.

[46] S. Satija and R. Chatterjee, "Blindtls: Circumventing tls-based https censorship," in *Proceedings of the ACM SIGCOMM 2021 Workshop on Free and Open Communications on the Internet*, 2021, pp. 43–49.

[47] Cloudflare, "DNS over Tor," https://developers.cloudflare.com/1.1.1.1/other-ways-to-use-1.1.1.1/dns-over-tor, 2024.

[48] R. Dingledine, N. Mathewson, P. F. Syverson *et al.*, "Tor: The second-generation onion router." in *USENIX security symposium*, vol. 4, 2004, pp. 303–320.

[49] C. Scott, P. Wolfe, and M. Erwin, *Virtual private networks*. "O'Reilly Media, Inc.", 1999.

[50] M. Feilner, *OpenVPN: Building and integrating virtual private networks*. Packt Publishing Ltd, 2006.

[51] J. A. Donenfeld, "Wireguard: Next generation kernel network tunnel." in *Network and Distributed System Security (NDSS) Symposium*, 2017.

[52] P. Winter and S. Lindskog, "How the great firewall of china is blocking tor," in *2nd USENIX Workshop on Free and Open Communications on the Internet, Bellevue, WA*. USENIX-The Advanced Computing Systems Association, 2012, p. 7.

[53] R. Singh, R. Nithyanand, S. Afroz, P. Pearce, M. C. Tschantz, P. Gill, and V. Paxson, "Characterizing the nature and dynamics of tor exit blocking," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 325–341.

[54] D. Xue, R. Ramesh, A. Jain, M. Kallitsis, J. A. Halderman, J. R. Crandall, and R. Ensafi, "Openvpn is open to vpn fingerprinting," *Communications of the ACM*, 2022.

[55] M. Zain ul Abideen, S. Saleem, and M. Ejaz, "Vpn traffic detection in ssl-protected channel," *Security and Communication Networks*, vol. 2019, no. 1, p. 7924690, 2019.

[56] Z. Wang, Y. Cao, Z. Qian, C. Song, and S. V. Krishnamurthy, "Your state is not mine: A closer look at evading stateful internet censorship," in *Proceedings of the 2017 Internet Measurement Conference*, 2017, pp. 114–127.

[57] L. Dixon, T. Ristenpart, and T. Shrimpton, "Network traffic obfuscation and automated internet censorship," *IEEE Security & Privacy*, vol. 14, no. 6, pp. 43–53, 2016.

[58] S. Hayeri, "GreenTunnel," https://github.com/SadeghHayeri/GreenTunnel, 2022.

[59] S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, *Moving target defense: creating asymmetric uncertainty for cyber threats*. Springer Science & Business Media, 2011, vol. 54.

[60] R. Zhuang, S. A. DeLoach, and X. Ou, "Towards a theory of moving target defense," in *Proceedings of the First ACM Workshop on Moving Target Defense*. ACM, 2014.

[61] J.-H. Cho, D. P. Sharma, H. Alavizadeh, S. Yoon, N. Ben-Asher, T. J. Moore, D. S. Kim, H. Lim, and F. F. Nelson, "Toward proactive, adaptive defense: A survey on moving target defense," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 709–745, 2020.

[62] D. Williams-King, G. Gobieski, K. Williams-King, J. P. Blake, X. Yuan, P. Colp, M. Zheng, V. P. Kemerlis, J. Yang, and W. Aiello, "Shuffler: fast and deployable continuous code {re-randomization}," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 367–382.

[63] P. Larsen, A. Homescu, S. Brunthaler, and M. Franz, "Sok: Automated software diversity," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 276–291.

[64] T. Jackson, B. Salamat, A. Homescu, K. Manivannan, G. Wagner, A. Gal, S. Brunthaler, C. Wimmer, and M. Franz, "Compiler-generated software diversity," *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, pp. 77–98, 2011.

[65] B. Baudry and M. Monperrus, "The multiple facets of software diversity: Recent developments in year 2000 and beyond," *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, pp. 1–26, 2015.

[66] K. Z. Snow, F. Monrose, L. Davi, A. Dmitrienko, C. Liebchen, and A.-R. Sadeghi, "Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization," in *2013 IEEE symposium on security and privacy*. IEEE, 2013, pp. 574–588.

[67] Y. Jang, S. Lee, and T. Kim, "Breaking kernel address space layout randomization with intel tsx," in *ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 380–392.

[68] J. Seo, B. Lee, S. M. Kim, M.-W. Shih, I. Shin, D. Han, and T. Kim, "Sgx-shield: Enabling address space layout randomization for sgx programs." in *Network and Distributed System Security (NDSS) Symposium*, 2017.

[69] Y. Gao, Y. Xiao, M. Wu, M. Xiao, and J. Shao, "Game theory-based anti-jamming strategies for frequency hopping wireless communications," *IEEE Transactions on Wireless Communications*, vol. 17, no. 8, pp. 5314–5326, 2018.

[70] H. Quan, H. Zhao, and P. Cui, "Anti-jamming frequency hopping system using multiple hopping patterns," *Wireless Personal Communications*, vol. 81, no. 3, pp. 1159–1176, 2015.

[71] J. Zhang and X. Wu, "Rl-based frequency hopping with block-shifted patterns: Balancing between anti-jamming performance and synchronization overhead," *IEEE Transactions on Vehicular Technology*, 2024.

[72] M. Torquato, P. Maciel, and M. Vieira, "Analysis of vm migration scheduling as moving target defense against insider attacks," in *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, 2021, pp. 194–202.

[73] ——, "Evaluation of time-based virtual machine migration as moving target defense against host-based attacks," *Journal of Systems and Software*, vol. 219, p. 112222, 2025.

[74] A. Aydeger, P. Zhou, S. Hoque, M. Carvalho, and E. Zeydan, "Mtdns: Moving target defense for resilient dns infrastructure," *arXiv preprint arXiv:2410.02254*, 2024.

[75] M. Wright, S. Venkatesan, M. Albanese, and M. P. Wellman, "Moving target defense against ddos attacks: An empirical game-theoretic analysis," in *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, 2016, pp. 93–104.

[76] Y. Zhou, G. Cheng, Y. Zhao, Z. Chen, and S. Jiang, "Toward proactive and efficient DDoS mitigation in IIoT systems: A moving target defense approach," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 4, pp. 2734–2744, 2021.

[77] T. Zhang, C. Xu, P. Zou, H. Tian, X. Kuang, S. Yang, L. Zhong, and D. Niyato, "How to mitigate ddos intelligently in sd-iov: A moving target defense approach," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 1, pp. 1097–1106, 2022.

[78] M. Nguyen and S. Debroy, "Moving target defense-based denial-of-service mitigation in cloud environments: A survey," *Security and Communication Networks*, vol. 2022, no. 1, p. 2223050, 2022.

[79] V. Heydari and S.-M. Yoo, "Securing critical infrastructure by moving target defense," in *Proc. 11th Int. Conf. Cyber Warfare Secur.(ICCWS)*, 2016, pp. 382–390.

[80] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Openflow random host mutation: transparent moving target defense using software defined networking," in *Proceedings of the first workshop on Hot topics in software defined networks*, 2012, pp. 127–132.

[81] V. Casola, A. De Benedictis, C. Mazzocca, and R. Montanari, "Designing secure and resilient cyber-physical systems: A model-based moving target defense approach," *IEEE Transactions on Emerging Topics in Computing*, 2022.

[82] Y.-B. Luo, B.-S. Wang, and G.-L. Cai, "Effectiveness of port hopping as a moving target defense," in *2014 7th International Conference on Security Technology*. IEEE, 2014, pp. 7–10.

[83] X. Xu, H. Hu, Y. Liu, H. Zhang, and D. Chang, "An adaptive ip hopping approach for moving target defense using a light-weight cnn detector," *Security and Communication Networks*, vol. 2021, no. 1, p. 8848473, 2021.

[84] V. Heydari, S.-i. Kim, and S.-M. Yoo, "Anti-censorship framework using mobile ipv6 based moving target defense," in *Proceedings of the 11th Annual Cyber and Information Security Research Conference*, 2016, pp. 1–8.

[85] ——, "Scalable anti-censorship framework using moving target defense for web servers," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 5, pp. 1113–1124, 2017.

[86] A. K. Sood and S. Zeadally, "A taxonomy of domain-generation algorithms," *IEEE Security & Privacy*, vol. 14, no. 4, pp. 46–53, 2016.

[87] Y. Fu, L. Yu, O. Hambolu, I. Ozcelik, B. Husain, J. Sun, K. Sapra, D. Du, C. T. Beasley, and R. R. Brooks, "Stealthy domain generation algorithms," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 6, pp. 1430–1443, 2017.

[88] L. Nie, X. Shan, L. Zhao, and K. Li, "Pkdga: A partial knowledge-based domain generation algorithm for botnets," *IEEE Transactions on Information Forensics and Security*, 2023.

[89] S. Li, T. Huang, Z. Qin, F. Zhang, and Y. Chang, "Domain generation algorithms detection through deep neural network and ensemble," in

Companion Proceedings of The 2019 World Wide Web Conference, 2019, pp. 189–196.

[90] H. S. Anderson, J. Woodbridge, and B. Filar, “Deepdga: Adversarially-tuned domain generation and detection,” in *Proceedings of the 2016 ACM workshop on artificial intelligence and security*, 2016, pp. 13–21.

[91] F5, Inc, “NGINX,” https://github.com/nginx/nginx, 2024.

[92] Q. Huang, D. Chang, and Z. Li, “A comprehensive study of {DNS-over-HTTPS} downgrade attack,” in *10th USENIX Workshop on Free and Open Communications on the Internet (FOCI 20)*, 2020.

[93] S. Sridhar, O. Ascigil, N. Keizer, F. Genon, S. Pierre, Y. Psaras, E. Riviere, and M. Król, “Content censorship in the interplanetary file system.” Network and Distributed System Security (NDSS) Symposium 2024, 2024.

[94] I. J. Kadhim, P. Premaratne, P. J. Vial, and B. Halloran, “Comprehensive survey of image steganography: Techniques, evaluations, and trends in future research,” *Neurocomputing*, vol. 335, pp. 299–326, 2019.

[95] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith, “Sok: secure messaging,” in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 232–249.

[96] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila, “A formal security analysis of the signal messaging protocol,” *Journal of Cryptology*, vol. 33, pp. 1914–1983, 2020.

[97] P. Rösler, C. Mainka, and J. Schwenk, “More is less: On the end-to-end security of group chats in signal, whatsapp, and threema,” in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018, pp. 415–429.

[98] E. Rodríguez, R. Anghel, S. Parkin, M. Van Eeten, and C. Gañán, “Two sides of the shield: Understanding protective {DNS} adoption factors,” in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 3135–3152.

[99] K. Jeřábek, K. Hynek, T. Čejka, and O. Ryšavỳ, “Collection of datasets with DNS over HTTPS traffic,” *Data in Brief*, vol. 42, p. 108310, 2022.

[100] D. Chao, D. Xu, F. Gao, C. Zhang, W. Zhang, and L. Zhu, “A Systematic Survey On Security in Anonymity Networks: Vulnerabilities, Attacks, Defenses, and Formalization,” *IEEE Communications Surveys & Tutorials*, 2024.

[101] “Fast flux: A national security threat,” National Security Agency (NSA), Cybersecurity and Infrastructure Security Agency (CISA), Federal Bureau of Investigation (FBI), Australian Signals Directorate’s Australian Cyber Security Centre (ASD’s ACSC), Canadian Centre for Cyber Security (CCCS), New Zealand National Cyber Security Centre (NCSC-NZ), Tech. Rep. U/OO/136180-25 — PP-25-1337, apr 2025, cybersecurity Advisory. [Online]. Available: https://media.defense.gov/2025/Apr/02/2003681172/-1/-1/0/CSA-FAST-FLUX.PDF

[102] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling, “Measuring and detecting fast-flux service networks.” in *Ndss*, 2008.

[103] J. Nazario and T. Holz, “As the net churns: Fast-flux botnet observations,” in *2008 3rd International Conference on Malicious and Unwanted Software (MALWARE)*. IEEE, 2008, pp. 24–31.

[104] A. Caglayan, M. Toothaker, D. Drapeau, D. Burke, and G. Eaton, “Real-time detection of fast flux service networks,” in *2009 Cybersecurity Applications & Technology Conference for Homeland Security*. IEEE, 2009, pp. 285–292.

[105] E. Pauley, P. Barford, and P. McDaniel, “DScope: A Cloud-Native Internet Telescope,” in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 5989–6006.

[106] E. Pauley, R. Sheatsley, B. Hoak, Q. Burke, Y. Beugin, and P. McDaniel, “Measuring and mitigating the risk of ip reuse on public clouds,” in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 558–575.

[107] O. Demigha and R. Larguet, “Hardware-based solutions for trusted cloud computing,” *Computers & Security*, vol. 103, p. 102117, 2021.

Appendix A
Intersection with Fast Flux and Implications

In April 2025, the NSA and other national cybersecurity agencies issued a bulletin highlighting “Fast Flux” as a national security threat [101]. Fast Flux, both its single-flux and double-flux variants, relies on rapidly changing DNS records to evade blacklists and takedowns [102], [103]. Malicious operators leverage extensive pools of compromised devices or bulletproof hosting to rotate the IP addresses (and sometimes the nameservers) of phishing sites, command-and-control (C2) infrastructure, and ransomware portals. By continuously shifting the network-level routing of malicious services, Fast Flux raises significant barriers to law enforcement and intelligence efforts.

Below, we examine where *NinjaDoH* intersects with these recent, national-level efforts, whether it could face collateral harm from new detection and blocking heuristics, and how *NinjaDoH*’s design mitigates such risks.

A. Background on Fast Flux Defense

The recent guidance from the NSA and allied agencies calls for more specific detection mechanisms to identify malicious Fast Flux. Some of the common recommended methods include, but are not limited to, the following:

- Short time-to-live (TTL) detection. Spotting extremely low DNS TTLs, often near zero, used by malicious Fast Flux domains.
- DNS or IP reputation filtering. Blocking newly registered domains or domains with suspicious rotation footprints.
- Network-level blocking of suspicious nameservers. Preventing rapid reassignments of authoritative DNS servers when they persistently resolve malicious domains.
- Protective DNS (PDNS) solutions. Encouraging ISPs and large organizations to adopt PDNS and reputation systems that automate the blocking of suspicious domains or address ranges.

Because *NinjaDoH* also uses frequent IP rotation (via public cloud elastic IPs) to bypass static blocklists, a natural question arises whether national-level efforts to crackdown on Fast Flux networks might inadvertently detect or block *NinjaDoH* endpoints.

B. Similarities and Key Differences

NinjaDoH’s moving-target design superficially resembles single-flux rotation: a DoH server with frequently changing IP addresses. However, unlike malicious botnets, *NinjaDoH* does not rely on bulletproof hosts or compromised machines. Instead, it leverages legitimate hyperscaler IP pools. Now, that does not mean that all the addresses are always perfectly “clean,” but the cloud providers do have know-your-customer (KYC) policies and dutifully cooperate with law enforcement, giving their addresses a certain degree of positive reputation. Furthermore, *NinjaDoH* does not broadcast short DNS TTLs into the global DNS system, as it does not rely on domain-based lookups from external resolvers. Rather, it privately informs clients of IP changes via IPNS content addressing. Below, we outline the core differences:

Infrastructure Source. Malicious fast flux actors typically operate large botnets or partner with bulletproof hosting services that ignore abuse requests. Such infrastructure is often spread across multiple ASes to thwart takedowns. By contrast, *NinjaDoH* runs on reputable hyperscalers (e.g., AWS or GCP). These platforms have finite IPv4 pools that they actively manage and monitor, thereby reducing the likelihood of hosting overtly malicious or legally noncompliant activities. Although an adversary could still attempt to block entire IP ranges from these providers, the significant collateral damage makes such actions impractical in most network applications.

Motivation and Enrollment. Traditional fast flux operations aim to prolong uptime for phishing sites, ransomware distribution, and other illicit content by evading IP-based blocks and takedowns. Attackers compromise machines or subvert hosting specifically to hide their true infrastructure. In *NinjaDoH*, the motivation is censorship-resilience: providing users with a rotating DoH endpoint for legitimate DNS resolution. Clients voluntarily enroll by obtaining the private IPNS key out-of-band; there are no "victim hosts." This user-driven enrollment model sharply contrasts with malicious botnets.

Scale and Behavior. Fast flux botnets commonly involve hundreds or thousands of rotating IPs, each with very short TTLs, often near zero, to ensure that addresses change with nearly every query. Because they frequently bulk-register disposable domains or use "throwaway" TLDs, their DNS patterns appear erratic at scale. *NinjaDoH*, on the other hand, allocates just a handful of cloud IPs at a configurable rotation interval. Rather than issuing DNS queries for new domains, it updates its clients via IPNS content addressing. As a result, an external observer does not see a large global flux event, but rather a moderate, private address shift, closer to normal cloud elasticity or load balancing. Consequently, *NinjaDoH*'s usage profile rarely appears suspicious to standard threat intelligence or DNS reputation systems, which predominantly look for mass domain churn or broad bulletproof hosting anomalies.

C. Potential for Collateral Blocking

Reputation-based blocking of rotating IPs. If security providers or national agencies adopt broad rules that flag any rapidly rotating IP endpoints, *NinjaDoH* could risk false positives. However, collateral damage concerns generally deter large-scale cloud IP blocks because hyperscalers host numerous legitimate services. Blocking entire address pools to catch ephemeral usage often proves unacceptable, especially in critical infrastructure or corporate settings where legitimate cloud-based applications would break.

Agencies have urged providers to expand DNS reputation analytics, but *NinjaDoH* avoids typical Fast Flux markers such as newly registered domains or unknown TLDs. Since clients do not publicly query a domain to reach *NinjaDoH* , rather they connect to ephemeral IP addresses discovered through IPFS, standard PDNS-based flux detection is largely circumvented.

Strict enterprise environments. Enterprises might still rely on policy-driven security solutions that block unrecognized outbound traffic. An organization with rigid outbound policies or endpoint device protection may block *NinjaDoH*. In extremely locked-down contexts (IP allowlists only), censorship-resistant tools face their toughest regime. Nonetheless, these environments are not the norm for national-level ISPs unless they are willing to accept heavy collateral disruptions.

D. Implications for Other Censorship-Resistant Tools

Unintended Consequences for Legitimate MTD Architectures. A central question is whether a robust push to block malicious fast flux might inadvertently compromise legitimate MTD solutions like *NinjaDoH*, *NetShuffle* [5], or rotating domain-fronting proxies. If an ISP or national regulator enforces broad "suspicious IP churn" detection, potentially guided by large-scale ML classifiers or fixed thresholds for what constitutes "excessive" rotation, legitimate ephemeral endpoints may be flagged.

However, such blanket approaches at the ISP level are rarely feasible. Hyperscalers host a vast array of critical services, so blocking these IP ranges outright can disrupt essential cloud-based applications. Consequently, censors or large enterprises typically adopt more granular strategies. In practice, the brunt of "Fast Flux" defenses tends to fall on suspicious or newly registered domains, identifying bulletproof hosting providers, or blocking known malicious IP clusters.

Collateral Damage vs. Granular Policy. While national agencies encourage precision detection of malicious flux domains and IP footprints [101], some PDNS solutions may overreach by automatically classifying any ephemeral address rotation as malicious. The result is that legitimate usage, from autoscaling backends in DevOps environments to censorship-resistant DNS, could face unwarranted blocks or false positives.

In practice, commercial PDNS and public resolvers still emphasize domain reputation rather than wholesale ephemeral IP blocking [104]. *NinjaDoH* largely sidesteps typical red flags by avoiding standard domain resolution altogether. Because its IP rotation stays within reputable cloud allocations and does not manifest as public DNS TTL churn, the resulting footprint is far less conspicuous than that of malicious flux networks.

Serving as an Anchor for Other Anti-Censorship Techniques. A positive corollary of *NinjaDoH*'s resilience is its utility as a bootstrap mechanism for additional censorship-resistance protocols. *NetShuffle* [5], certain pluggable transports, or domain-fronting methods often require an unblocked DNS channel to retrieve ephemeral entry points. *NinjaDoH* provides such a channel: once users have a stable, moving-target DoH resolver, they can securely learn or update addresses for advanced proxying or shuffle-based systems. By layering these techniques, the overall censorship-circumvention pipeline gains robustness.

Leveraging Fast Flux Guidance for Stealth. Interestingly, official recommendations for detecting criminal Fast Flux yield valuable lessons for *NinjaDoH* deployments:

- **Moderate Rotation Intervals.** Avoid overly aggressive churn (e.g., changing IPs every few seconds) that mimics large-scale botnets. A balanced, periodic rotation reduces detectability.
- **Legitimate IP Blocks.** Hosting on major cloud platforms dissuades broad IP blocking and leverages existing positive reputation pools.
- **Normal Handshake Patterns.** Matching common load-balancer behavior or autoscaling signals helps blend in with the "background" cloud environment.

By adhering to these recommendations originally aimed at uncovering malicious bulletproof hosting, *NinjaDoH* further reduces its risk of false positives and collateral blocking.

E. Summary of Expected Impact

While both *NinjaDoH* and malicious Fast Flux techniques utilize frequent IP rotation to evade static blocklists, their underlying motivations and architectures differ substantially, raising important questions about the governance of censorship-resistant technologies amid increasing scrutiny of evasive network behaviors. Whereas Fast Flux relies on suspiciously short TTLs, throwaway domains, and bulletproof hosting, *NinjaDoH* leverages reputable cloud infrastructure and private IPNS-based coordination. Although enterprises or authoritarian regimes might adopt broad heuristics that flag ephemeral IP usage, the collateral risk of blocking major cloud platforms makes such measures impractical at scale. *NinjaDoH*'s architecture can also serve as a foundation for other censorship-resistance tools (e.g., *NetShuffle* or domain-fronted proxies) by offering a stealthy DNS bootstrap path. Like many MTD systems, however, *NinjaDoH* illustrates the dual-use tension between circumventing censorship and complicating enterprise security controls. As ephemeral MTD strategies become more widespread, future research must continue to consider how to balance anti-censorship resilience with responsible network governance. For now, *NinjaDoH*'s operational profile positions it outside the primary scope of Fast Flux countermeasures.

Appendix B
Additional Security Analysis

We now analyze additional security considerations that are relevant for our proposed system.

IPv4 Reuse and Short-term Tenancy. Although the global IPv4 address pool of hyperscalers is incredibly large, by constantly rotating IPv4 addresses there exists the possibility that IP addresses are recycled and reused. There could be unintended consequences of using transitory IP addresses, and service could be denied if an in-use IP address was previously blocked by a network administrator because of the prior tenant of that address. IP reuse on public clouds is an area of active research, as Pauley et al. show [105], [106], cloud IPs receive significantly more targeting scanning than what is expected under normal conditions. This can be caused by misconfigurations of prior tenants, or that cloud IPs appear as more lucrative targets. This should have minimal effect on our system because of the transitory nature of our tenancy on any given IP address.

Let's assume an adversary gains access to the IPNS hash allowing them to know the IP addresses and follow along with the IP rotations of a particular *NinjaDoH* deployment. This would allow them to add those IP addresses to their blocklist. While this would deny use of that deployment, it would cause downstream negative effects on legitimate services that become tenants of those IP addresses. This negative effect on the censor would be compounded as the number of individual *NinjaDoH* deployments they block by using a compromised IPNS hash. This furthers the case against append-only, list-based approaches to blocking DoH. Additionally, to restore service, the *NinjaDoH* deployer would rotate their IPNS hash and securely communicate the hash rotation with their clients.

IPv6 Subnet Blocking. As IPv6 is orders of magnitude larger in address space than IPv4, with no real risk of exhaustion, list-based approaches for IP addresses blocking are not possible. However, that doesn't mean that censors can't take advantage of certain administrative realities present in how IPv6 subnets are allocated to users. The smallest allocation to an individual entity is $/64$. Therefore, if a censor detects *NinjaDoH* on IPv6, they could choose to block the entire $/64$ subnet associated with that IP. This would largely come down to a specific deployment consideration on how the public cloud provider will allocate IPv6 subnets to users.

Trusted Execution Environments (TEE). A sophisticated adversary who gains root access to the server instance providing the service would have the ability to either deny service by terminating the process, or most insidiously, conduct a DNS poisoning attack by hijacking the DNS resolver located on the instance. If the infrastructure serving the *NinjaDoH* service is compromised, we want to limit the adversary to, at most, a denial of service. We can maintain the integrity of the system and preserve the privacy of the DNS requests of the client by placing key cryptographic functions of the system inside a trusted execution environment. The large hyperscale public cloud providers allow for secure compute enclaves on their compute instances, which Demigha et al. [107] show can provide secure storage as well as protection from an OS-level compromise. The TEE in the cloud environment can hold the private key for the TLS session, as well as the private key for the IPNS name. Properly implementing this would ensure that even if the adversary gains root access on the host compute instance, they would not be able to manipulate the code, or access the private keys stored inside the TEE, maintaining the integrity of the service.

Appendix C
Additional Tables & Figures

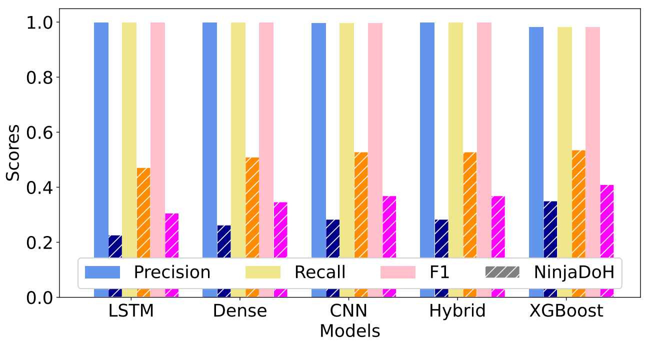


Fig. 5: Performance of ML models trained with DoH traffic in detecting test DoH vs. *NinjaDoH* traffic, showing that while regular DoH is accurately detected, *NinjaDoH*'s dynamic IP rotation and query path randomization make detection far more challenging.

TABLE I: Evasion of DoH blocking by popular firewalls.

DoH Blocking Method	Layer	Evaded	Adversarial Impact	Explanation
Domain Blocking	Network (3)	✓	Low-cost to implement and maintain as it relies on simple static blocklists. Minimal impact on the network and does not significantly degrade user experience.	Does not use a domain name so bypasses static domain blocklists.
IP Blocking	Network (3)	✓	Low-cost to implement and maintain as it relies on simple static blocklists. Minimal impact on the network and does not significantly degrade user experience.	The IP address agility of the system evades static IP blocklist.
Application Identification	Application (7)	✓	Moderate cost and complexity as it requires deep packet inspection. It can introduce slight latency but is more effective than simple IP or domain blocking.	Mimics regular HTTPS traffic to evade application-level filters. Does not follow the typical patterns of DoH traffic.
SNI Blocking	Session (5)	✓	Moderate cost with little impact on user experience. More complex to maintain with newer protocols like TLS 1.3, which makes SNI blocking harder.	Since the client connects via an IP address, the SNI field in the `ClientHello` is empty or omitted, preventing SNI-based blocking from matching a domain name.
IP Allowlisting	Network (3)	✗	High cost and impact, as it restricts user access significantly and requires substantial maintenance of trusted IP addresses. This method limits user capabilities and heavily controls the network environment.	Uses strict IP allowlisting; unknown IPs are blocked, so dynamic IPs are ineffective.

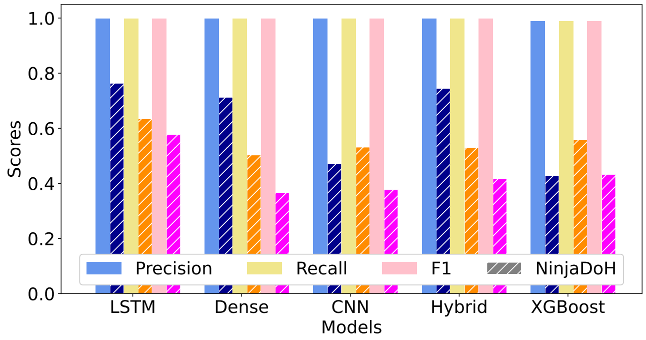


Fig. 6: Performance of ML models trained with DoH and *NinjaDoH* traffic in detecting test DoH vs. *NinjaDoH* traffic.

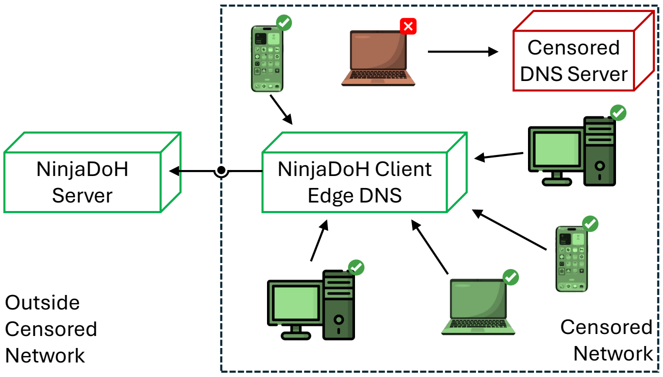


Fig. 7: *NinjaDoH* client deployed as a scalable edge DNS resolver inside a censored network.

TABLE II: Descriptive statistics for flow duration (in ms).

Flow Type	Mean	Median	Skew	N
Non-DoH	7430	313	3820	3395
NinjaDoH	742	246	4240	1606
Combined	5280	265	4740	5001

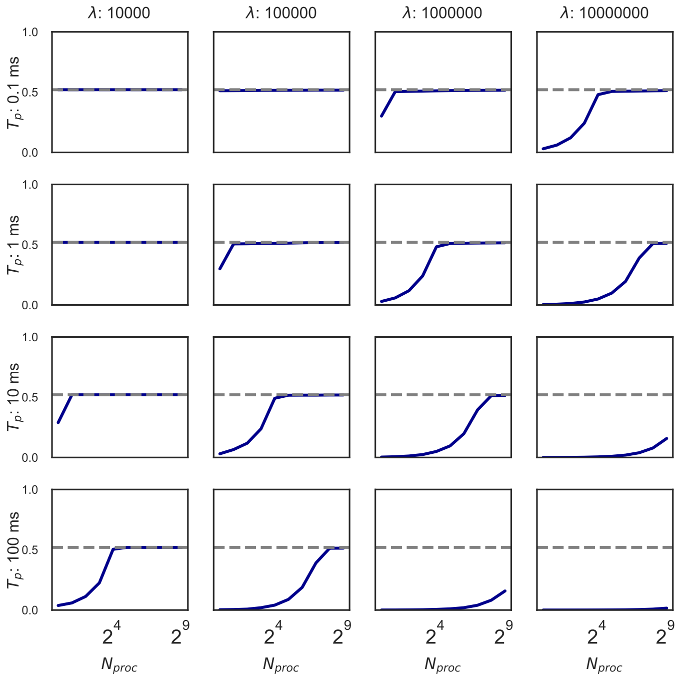
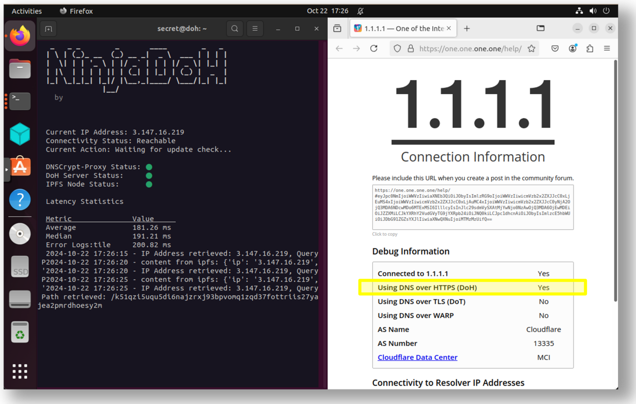


Fig. 8: Relationship between the number of cores (N_{proc}), flows/min (λ), and flow processing time (T_p) on the probability of detecting DoH. The dashed line represents the best ML model's true positive rate baseline (TPR=0.52).



```
 _   _ _       _       ____        _   _
| \ | (_)_ __ (_) __ _|  _ \  ___ | | | |
|  \| | | '_ \| |/ _` | | | |/ _ \| |_| |
| |\  | | | | | | (_| | |_| | (_) |  _  |
|_| \_|_|_| |_/ |\__,_|____/ \___/|_| |_|
            |__/
  by


 Current IP Address: 3.147.16.219
 Connectivity Status: Reachable
 Current Action: Waiting for update check...

 DNSCrypt-Proxy Status: ●
 DoH Server Status:     ●
 IPFS Node Status:      ●

 Latency Statistics

 Metric              Value
 Average             181.26 ms
 Median              191.21 ms
 Error Logs:tile     200.82 ms
 2024-10-22 17:26:15 - IP Address retrieved: 3.147.16.219, Query
P2024-10-22 17:26:20 - content from ipfs: {'ip': '3.147.16.219',
'2024-10-22 17:26:20 - IP Address retrieved: 3.147.16.219, Query
P2024-10-22 17:26:25 - content from ipfs: {'ip': '3.147.16.219',
'2024-10-22 17:26:25 - IP Address retrieved: 3.147.16.219, Query
Path retrieved: /k51qzi5uqu5di6najzrxj93bpvomq1zqd37fottriis27ya
jea2pmrdhoesy2m
```

1.1.1.1

Connection Information

Please include this URL when you create a post in the community forum.

```
https://one.one.one.one/help/
#eyJpc0NmIjoiWWVzIiwiaXNEb3QiOiJObyIsImlzRG9oIjoiWWVzIiwicmVzb2x2ZXJJcC0xLj
EuMS4xIjoiWWVzIiwicmVzb2x2ZXJJcC0xLjAuMC4xIjoiWWVzIiwicmVzb2x2ZXJJcC0yNjA2O
jQ3MDA6NDcwMDo6MTExMSI6IlllcyIsInJlc29sdmVySXAtMjYwNjo0NzAwOjQ3MDA6OjEwMDEi
OiJZZXMiLCJkYXRhY2VudGVyTG9jYXRpb24iOiJNQ0kiLCJpc1dhcnAiOiJObyIsImlzcE5hbWU
iOiJDbG91ZGZsYXJlIiwiaXNwQXNuIjoiMTMzMzUifQ==
```

Click to copy

Debug Information

Connected to 1.1.1.1	Yes
Using DNS over HTTPS (DoH)	Yes
Using DNS over TLS (DoT)	No
Using DNS over WARP	No
AS Name	Cloudflare
AS Number	13335
Cloudflare Data Center	MCI

Connectivity to Resolver IP Addresses

Fig. 9: Prototype *NinjaDoH* client (left window) running on Ubuntu Desktop, using a *NinjaDoH* server instance on AWS for DNS queries. DoH connectivity to *NinjaDoH* server is verified by https://one.one.one.one/help/ opened on a web browser (right window). Cloudflare is configured as the upstream DNS server on this instance of *NinjaDoH* server, as verified in the highlighted box.