RT-MD: Host-Centric Real-Time Detection of Multi-Domain DNS Data Exfiltration

Pengfei Ren
rpf@mail.ustc.edu.cn
University of Science and Technology of China
Hefei, Anhui, China

Lutong Chen
lutong98@ustc.edu.cn
University of Science and Technology of China
Hefei, Anhui, China

Xuanbo Huang
529550189@qq.com
Xi'an Jiaotong University
Xi'an, Shanxi, China

Jiankang Sun
jksun@mail.ustc.edu.cn
University of Science and Technology of China
Hefei, Anhui, China

Huanjie Zhang
james@ustc.edu.cn
University of Science and Technology of China
Hefei, Anhui, China

Jian Li
lijian9@ustc.edu.cn
University of Science and Technology of China
Hefei, Anhui, China

Kaiping Xue*
kpxue@ustc.edu.cn
University of Science and Technology of China
Hefei, Anhui, China

Abstract

DNS data exfiltration refers to abusing DNS as a covert channel to exfiltrate information, which has become a significant threat and leads to the development of numerous detectors. However, when deployed on DNS gateways, most existing detectors face three inherent challenges. First, long aggregation windows hinder online detection. Second, maintaining fine-grained state together with complex processing imposes scalability constraints. Third, adaptive multi-domain exfiltration enables attackers to evade detection. To address these challenges, we present RT-MD, a **R**eal-**T**ime, host-centric detector for multi-domain DNS data exfiltration. RT-MD uses space-efficient probabilistic data structures to estimate, on a per-host basis, the amount of potentially exfiltrated information, rather than tracking individual domains. This design enables robust detection of both conventional and **M**ulti-**D**omain exfiltration while simplifying threat response. RT-MD achieves $O(1)$ processing per query and is designed to operate within the state and throughput constraints of high-load DNS gateways. RT-MD is evaluated on large-scale datasets as well as through a real-world deployment on a large campus network. The results demonstrate that RT-MD detects extremely low-rate exfiltration down to 0.43 characters per second, while maintaining a false positive rate below 0.004, reducing the detectable exfiltration rate by approximately half compared to the prior state-of-the-art ibHH under comparable settings. RT-MD sustains a detection throughput exceeding 1.4 million queries per second, substantially outperforming representative prior detectors.

CCS Concepts

• **Networks → Web protocol security**; *Application layer protocols*;
• **Security and privacy → Firewalls**.

*Corresponding author

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Conference acronym 'XX, Woodstock, NY
© 2026 ACM.
ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

Keywords

DNS, Covert Channel, Data Exfiltration, Intrusion Detection System

ACM Reference Format:
Pengfei Ren, Lutong Chen, Xuanbo Huang, Jiankang Sun, Huanjie Zhang, Jian Li, and Kaiping Xue. 2026. RT-MD: Host-Centric Real-Time Detection of Multi-Domain DNS Data Exfiltration. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 16 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

The Domain Name System (DNS) [29] is a fundamental infrastructure of the Internet, mainly responsible for translating human-friendly domain names into corresponding IP addresses or other resource records. Its ubiquity makes it an attractive target and a convenient tool for malicious activity [18]. Among DNS-based abuses, DNS tunneling-based data exfiltration is widely used by attackers and has caused substantial losses [22], as it hides non-DNS traffic inside DNS queries and responses, providing a covert communication channel that is commonly used for Command-and-Control (C2) operations and data exfiltration. According to a DNS threat report [18] published by International Data Corporation (IDC), 32% of participating organizations experienced DNS tunneling attacks. A notable incident is that a variant of FrameworkPOS malware [31] stole more than 56 million payment card accounts from Home Depot in 2014 [22]. More recently, an Advanced Persistent Threat (APT) group used a malware called WellMess to exfiltrate confidential information over DNS from several organizations [7, 44].

Figure 1 illustrates the mechanism of DNS data exfiltration. An attacker can exfiltrate credentials by encoding them into the subdomain of a DNS query name, e.g., `secret.attacker1.com`. When the internal DNS resolver processes and forwards the query, the encoded secret is delivered to the attacker's authoritative DNS servers. Furthermore, an adaptive attacker may split a secret across multiple domains simultaneously to evade detection. For example, the string `secret` can be split across two independent DNS queries, `sec.attacker1.com` and `ret.attacker2.com`.

DNS data exfiltration remains effective even in strictly controlled environments as DNS serves as essential Internet infrastructure and is usually afforded permissive traversal, even though other

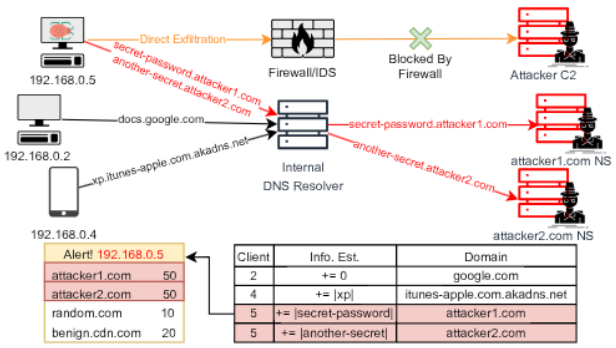


Figure 1: Multi-Domain DNS Exfiltration Detection with RT-MD.

protocols are applied with strict outbound filtering. In restricted networks, hosts may lack direct Internet connectivity but can still issue queries to internal DNS resolvers, which may bypass conventional egress controls. The indirection also complicates attribution: from the perspective of perimeter intrusion detection systems, the DNS resolver appears to be the source of exfiltration rather than the compromised internal host. Combined with the high volume and diversity of benign DNS traffic, these characteristics allow DNS data exfiltration to blend into normal operations and evade detection, which raises significant challenges for defending against such attacks.

To defend against DNS data exfiltration, a broad range of detection approaches has been proposed. Existing work spans rule- and model-based methods. These methods analyze DNS query names and related features to distinguish tunneling-based exfiltration traffic from benign activity.

However, when deployed on real-world DNS gateways, most existing DNS exfiltration detectors face three inherent challenges: online detection ability, scalability, and robustness to multi-domain exfiltration.

(1) **Online detection ability.** Many detection algorithms operate offline or rely on long detection windows for feature extraction [28, 34]. They group DNS queries over time and compute information-theoretic or statistical features on the resulting batches. This batch-oriented design introduces detection delay, so an attacker can exfiltrate a large amount of data before the system raises an alert.
(2) **Scalability under network constraints.** Another challenge is scalability in both state and throughput. Several detectors maintain detailed per-host or per-domain histories [21, 34]. The associated memory footprint grows with the number of hosts, domains, and window length, which limits practicality on high-load networks. Model-based methods [4, 19, 27, 39, 42, 43] require expensive feature extraction or deep architectures that are difficult to deploy on resource-constrained DNS gateways.
(3) **Robustness to multi-domain exfiltration.** Most detectors maintain a state per domain or per host-domain pair. When an attacker distributes exfiltrated data simultaneously across many domains, the detector must track a larger number of low-throughput channels and maintain more state, which increases the processing load and makes it harder to distinguish malicious domains from benign ones [19, 32, 34].

State-of-the-art work ibHH [32] proposes an efficient probabilistic algorithm and a constant-size data structure to improve throughput and scalability for real-time detection. However, ibHH still suffers from multi-domain attacks. Moreover, its throughput may still fall short of the query rates required by large-scale networks in practice. Therefore, there remains a need for a method that achieves high throughput and good scalability while efficiently defending against multi-domain attacks.

In this paper, we propose **RT-MD**, a **R**eal-**T**ime DNS exfiltration detection system designed to address these gaps. RT-MD is robust to **M**ulti-**D**omain exfiltration by estimating the volume of potentially exfiltrated data per host while tracking the most suspicious domains associated with each host. RT-MD performs online detection without a detection window by continuously aggregating per-host information estimates, which is crucial for reducing time-to-detection and enabling timely threat response. RT-MD achieves scalability under network constraints along two dimensions. **(1) Throughput scalability.** RT-MD is designed around constant-time processing per DNS query through a carefully structured algorithm that avoids expensive feature extraction and complex model inference. This algorithmic design enables RT-MD to sustain a detection throughput of 1.4 million queries per second. **(2) Resource efficiency on DNS servers.** To operate within the strict memory and computational constraints of production DNS resolvers, RT-MD leverages space-efficient probabilistic data structures, including Bloom filters [5] and Count-Min Sketches [10], to maintain summaries instead of detailed states for each host to achieve low and bounded memory overhead.

RT-MD achieves the best true positive rates on all of the datasets during evaluation on offline datasets while producing few false positives. The results show that RT-MD detects extremely low-rate DNS exfiltration down to 0.43 units/s, where a unit, as defined in Section 4.2.1, denotes sub-byte information encoded in a single DNS-name-legal, case-insensitive character, while maintaining a false positive rate below 0.004. Additionally, we collaborate with the university's network information center to conduct a real-world online evaluation. Our university network includes more than 202,445 unique active IP addresses and generates 721 million DNS log entries per day, providing a representative view of large-scale operational environments. After 12 hours of the initial deployment of RT-MD, security operators only need to perform 7 operations per day on average to protect against DNS exfiltration.

We summarize our contributions in this paper as follows:

- We propose the first host-centric online DNS data exfiltration detection approach that estimates the amount of potentially exfiltrated information for each host, addressing the threat of multi-domain DNS exfiltration.
- We implement our detection algorithm with efficient probabilistic data structures to handle every query in $O(1)$ time with minimal memory usage, achieving a high throughput of 1.4M queries per second and enabling deployment in high-throughput DNS resolvers and large-scale networks.
- We evaluate RT-MD on large-scale datasets and live DNS traffic from our campus network, achieving detection of

extremely low-rate exfiltration (down to 0.43 units/s) with a false positive rate below 0.004 and high throughput. We open-source [1] our implementation to support reproducibility and facilitate future research.

The remainder of this paper is organized as follows. Section 2 provides an overview of related work. Section 3 formally defines the system model. Section 4 presents our detector RT-MD and thoroughly analyzes it. Section 5 shows the experiments and results, and Appendix E discusses deployment considerations.

2 Related Work

DNS data exfiltration and tunneling have been widely investigated in prior work. Wang et al. [40] provide a comprehensive survey of detection approaches and categorize them into rule- and model-based methods. In this section, however, we will view related work from three perspectives: scalability, detection window, and detection granularity. Finally, we present a multi-dimensional comparison of our method with representative prior work in Table 1.

2.1 Detection Scalability

Real-time detection requires online detection and throughput scalability for production environments. Window-based methods that employ long detection windows (Section 2.2) [27, 28, 34] inevitably lead to detection latency and therefore cannot operate online nor scale to real-time detection. Compute-intensive methods that rely on complex machine-learning, deep-learning, or large language models [4, 19, 21, 38] may operate online but face significant scalability challenges in production environments. Few approaches scale to true real-time detection in production environments: Qi et al. [36] use per-query bi-gram classification, and ibHH [32] continuously estimates per-domain exfiltration using probabilistic data structures. Our method, RT-MD, is designed to operate in real-time while substantially exceeding the throughput of ibHH, enabling deployment in large-scale networks.

2.2 Detection Window

A detection window defines an amount of feature data that an algorithm must collect before it can issue a detection decision. Based on whether or not features are aggregated before reaching a decision and the presence of a detection window, we divide related work into three categories.

Single-Request Classification. Some detectors work at the level of individual DNS queries and responses without aggregation [4, 39]. Qi et al. [36] employed bi-gram frequency analysis for per-query classification. Altuncu et al. [4] used a deep neural network to classify every DNS query. Sharma et al. [39] constructed a three-layer classifier, where the first two layers use bi-gram analysis and the BERT model to filter legitimate domains, while the last layer actively probes the suspicious domain. However, Žiža et al. [43] show that adversarial attackers can deceive single-request classifiers. Signature-based detectors [1] could detect in real-time, but cannot generalize to unknown attacks. Recently, with the advent of large language models, Sayed et al. [38] fine-tuned LLMs for DGA and DNS exfiltration detection.

[1] Our open-source implementation of RT-MD is available at `https://anonymous.4open.science/r/rt-md/`

Window-based Detection. Most current DNS exfiltration detectors rely on aggregating features over DNS traffic observed within a time window [27, 28, 34] or a window containing enough number of packets [19, 43]. These features are typically derived from statistical analysis of the DNS packets, such as character frequency, inter-arrival times, and traffic volume. The use of long aggregation windows significantly delays detection, thereby enabling attackers to exfiltrate large volumes of information prior to detection. For example, Paxson et al. [34] aggregate DNS traffic over a one-day window at the host-domain pair level and estimate an upper bound on transmitted information using multiple lossless compression algorithms, which is then compared against a 4 kB threshold. Liu et al. [27] detect low-throughput DNS tunnels using time-frequency features aggregated over multi-granularity sliding windows and classified via SVM. Nevertheless, achieving optimal AUC requires a detection window as long as 1200 seconds.

Window-free Online Detection. To the best of our knowledge, ibHH [32] is the only existing DNS exfiltration detection method that performs continuous, window-free aggregation for online detection. It uses probabilistic data structures to continuously estimate the volume of potentially exfiltrated information per domain without relying on fixed aggregation windows. Although ibHH periodically resets its data structures (e.g., every 120 seconds), this reset interval is intended to measure the amount of information exfiltrated within each interval rather than to define a detection window. Its detection is performed continuously, and alerts may be raised at any point within a reset interval. Our proposed RT-MD adopts the same window-free, online detection paradigm and extends it to a host-centric aggregation model, enabling real-time detection of multi-domain exfiltration while preserving constant-time processing and low memory overhead.

2.3 Detection Granularity

Detection granularity refers to the level at which malicious activity is identified and attributed, such as at the packet, flow, domain, or host level. Based on detection granularity, we classify prior work into three categories.

Domain-centric Detection. Most existing detectors attribute malicious activity to domains and aim to block them once identified [4, 19, 32, 36, 38, 39]. While effective for identifying suspicious domains, these approaches fundamentally attribute malicious behavior to domains rather than compromised hosts. In practice, this complicates incident response, as defenders must retrospectively analyze DNS logs to map flagged domains back to infected machines. Moreover, many domain-centric detectors are inherently vulnerable to multi-domain exfiltration, in which attackers distribute exfiltration traffic across multiple domains to maintain low per-domain throughput while achieving substantial aggregate throughput.

Host-Domain Pair Detection. A small number of approaches attribute malicious behavior at the level of host-domain pairs [34, 43]. By jointly considering the host and the queried domain, these methods partially alleviate the attribution ambiguity of purely domain-centric detectors and facilitate the identification of compromised hosts. However, they still treat each host-domain pair independently and therefore remain vulnerable to multi-domain exfiltration.

Host-Centric Detection. Several studies aggregate DNS traffic features at the level of individual hosts to detect exfiltration activity [21, 27, 28]. In particular, Ishikura et al. [21] observe that DNS data exfiltration necessarily induces cache misses at DNS resolvers and derive key cache-related properties to design both a rule-based filter and an LSTM-based detector. While these approaches enable direct attribution of malicious activity to compromised hosts, they do not report the associated suspicious domains, which complicates root-cause analysis and makes incident response harder in practice. In contrast, our approach provides host-centric detection while simultaneously reporting the set of suspicious domains involved, facilitating timely and targeted response.

Table 1: Comparison with Related Works

Method	Cat.	OL	LL	LC	HT	MD	DR	HR	NL	G
Altuncu et al. [4]	Model	✓	✓	✗	✗	✗	✓	✓	✗	✓
Liu et al. [27]	Model	✗	✗	✗	✗	?	✗	✓	✗	✓
DLAZE [39]	Model	✓	✗	✗	✗	✗	✓	✗	✗	✓
DTW [28]	Model	✗	✗	✗	✗	?	✗	✓	✗	✓
Žiža et al. [43]	Model	✓	✓	✗	✗	✗	✓	✓	✗	✓
You et al. [42]	Model	✗	✗	✗	✗	✗	✓	✗	✗	✓
GraphTunnel [19]	Model	✓	✗	✗	✗	?	✓	✗	✗	✓
Sayed et al. [38]	Model	✗	✗	✗	✗	✗	✓	✗	✗	✓
Qi et al. [36]	Rule	✗	✗	✓	✓	✗	✓	✗	✗	✓
Ishikura et al. [21]	Rule	✗	✗	✓	✗	?	✗	✓	✓	✓
Paxson et al. [34]	Rule	✗	✗	✓	✗	✗	✓	✗	✓	✓
DID [1]	Rule	✓	✓	✓	✓	✗	✓	✓	✗	✗
ibHH [32]	Rule	✓	✓	✓	✓	✗	✓	✗	✓	✓
RT-MD	Rule	✓	✓	✓	✓	✓	✓	✓	✓	✓

OL: The method is designed to perform online detection.
LL: The method has low latency in online detection.
LC: Low cost, suitable for running in resource-constrained environment, e.g., w/o GPU.
HT: The method has high detection throughput.
MD: Robust against adversarial multi-domain exfiltration.
DR: The method can report suspicious domains.
HR: The method can report suspicious hosts.
NL: The method does not require labeled data for training/tuning.
G: The method could generalize to multiple kinds of attacks.

3 System Model

This section defines the system model of our method. We first describe the operational environment, followed by a characterization of the attacker's capabilities and exfiltration strategies. Finally, we specify the defender's vantage point and assumptions.

Environment: We consider a typical campus or enterprise network environment in which internal DNS resolvers are deployed, and security policies are configured to make the hosts to use these resolvers. The hosts may or may not have outbound Internet access, but could perform DNS resolution through the resolvers.

Attack Model: The attacker compromises one or more hosts inside the organization and attempts to exfiltrate data over DNS because of its ubiquity, covertness, and the fact that DNS traffic is often permitted even in restricted network environments. Prior incidents have shown that attackers may employ multiple exfiltration strategies and fall back to DNS exfiltration when others fail [15, 24].

The attacker controls authoritative name servers of one or more domains or reverse zones on the public Internet, which could be done by registering new domains at a registrar or owning a block of consecutive IP addresses. This enables the use of multiple domains for DNS data exfiltration. The attacker exfiltrates information by encoding it into DNS queries that are sent through internal DNS resolvers. We only consider DNS data exfiltration that encodes data into the subdomain portion of DNS query names [15, 22], which represents the dominant and highest-capacity information vector in practice. While it is theoretically possible to encode information into other DNS fields, such as query type or class, these fields offer extremely limited information capacity and therefore support only negligible exfiltration throughput. To the best of our knowledge, there are no documented real-world cases that rely on such fields for DNS data exfiltration. Similarly, timing-based covert channels are possible but are inherently noisy, difficult to scale, and unreliable for sustained data transfer.

Defense Model: The internal DNS resolver rejects invalid queries (e.g., queries that use invalid characters in query names) and then sends queries to our detection system in real time. DNS query names are usually case-insensitive, but if query name casing were preserved end-to-end, an attacker could exploit it to increase exfiltration throughput. We assume that the internal DNS resolver enables the widely deployed 0x20-bit encoding mechanism [11] that mitigates DNS forgery and cache poisoning attacks. As a result, the resolver randomizes the casing of query names before sending them to authoritative servers, preventing attackers from reliably encoding information in character casing. Therefore, the attacker can encode information using only case-insensitive characters.

4 RT-MD Detection System

RT-MD is designed to provide real-time, scalable, and robust detection of DNS data exfiltration under high-throughput constraints. As shown in Figure 2, our detector consists of two modules: **Query Pre-Processing Module** (Section 4.1) that splits the query name and rapidly filters out domains that are considered impossible for exfiltration, and **Information Tracking Module** (**ITM**) (Section 4.2) that tracks the amount of potentially exfiltrated information of each host and related domains with space-efficient probabilistic data structures. We also present this pipeline in pseudo code form in **Algorithm 1**. We analyze the time and space complexity of RT-MD in Section 4.3 and consider adversarial attacks against RT-MD in Section 4.4.

4.1 Query Pre-Processing Module

This module is responsible for extracting **subdomain** and **attribution domain** from an input DNS query and deciding whether the attribution domain could be used for exfiltration based on allowlists. The subdomain refers to the part of the domain name that an attacker could encode data into, while the attribution domain refers to the part of the domain name that is considered unavailable for exfiltration. The attribution domain is usually a domain name that an attacker can buy at a domain registrar, which is not

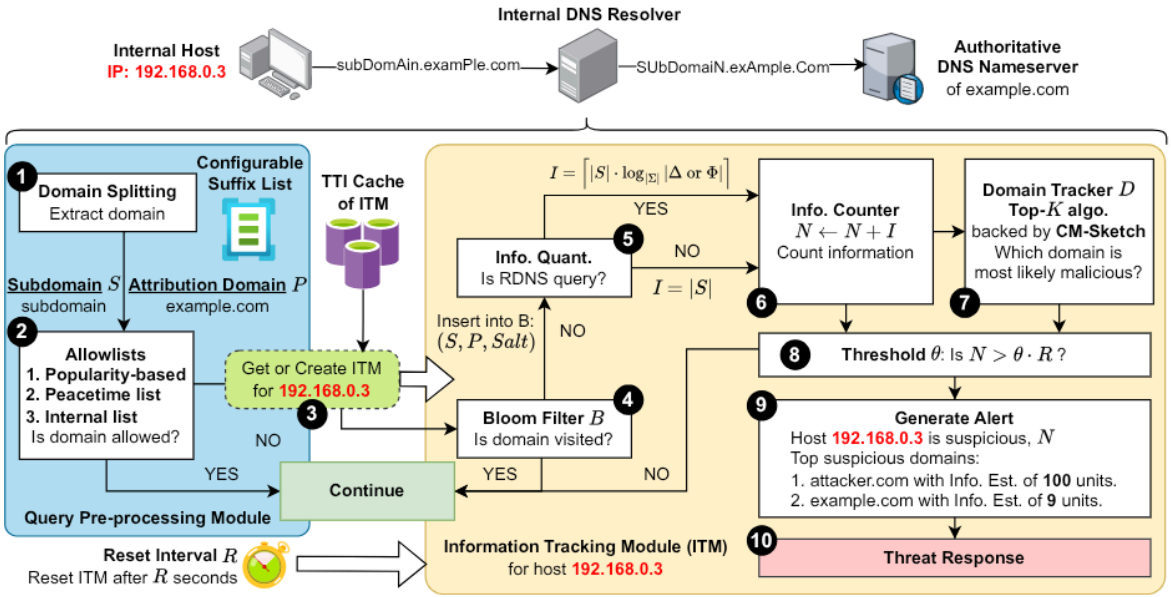


Figure 2: Overview of RT-MD. When a host sends a DNS query, (1) We extract the subdomain and attribution domain. (2) If the attribution domain is allowlisted, continue the query. (3) Else, we retrieve or create an ITM for this host. (4) The Bloom filter in ITM maintains whether or not the query name has already been queried by this host to avoid over-counting. (5) We quantify the amount of potentially exfiltrated information differently for conventional and RDNS queries. (6) The quantified amount of information is added to the information counter. (7) We update the domain tracker to track top-K suspicious domains. (8-9) We generate an alert containing the host, counter, and top-K suspicious domains if the average exfiltrating rate N/R exceeds θ.

necessarily a second-level domain. The allowlists are necessary as sometimes DNS is used to transmit data for benign purposes like anti-malware [30] and parental control [12]. Moreover, legitimate services employing high-cardinality subdomains can easily trigger false positives without allowlists because of the variety of subdomains involved [19].

4.1.1 Domain Splitting. For normal queries, we split the query name D into subdomain S and attribution domain P via a Configurable Suffix List, which is a superset of the Public Suffix List (PSL) [35]. However, an attacker owning an IP address block could use an authoritative nameserver of a reverse zone with reverse DNS lookups for data exfiltration. To facilitate better information attribution, for IPv4 reverse lookups, we set the attribution domain to its /16 network range, and for IPv6 reverse lookups, we set the attribution domain to its /48 network range. For example, for query name `1.1.64.100.in-addr.arpa`, the subdomain is `1.1` while the attribution domain is `64.100.in-addr.arpa`.

4.1.2 Popularity-based allowlist. Based on the assumption that it is hard for an attacker to compromise highly popular domains at scale, we allowlist global popular domains. This assumption is consistent with prior works in DNS exfiltration research [2, 32, 34]. Specifically, we use the TRANCO list[2] [25] generated on 07 April 2025 for a global popularity-based allowlist of the top one million domains.

[2]Available at https://tranco-list.eu/list/LJL44

4.1.3 Peacetime allowlist. The global popularity-based allowlist alone is insufficient to cover regionally or locally popular domains. Following the peacetime/wartime model in ibHH [32], we construct a peacetime allowlist from the operational environment itself. During the peacetime phase, we assume that no active DNS data exfiltration is present in the monitored network. RT-MD executes in monitoring mode, and any alerts raised during this period are treated as false positives. The domains appearing in these alerts are collected and added to the peacetime allowlist. Afterwards, RT-MD is switched to wartime (enforcing) mode. This model implies that RT-MD is designed to detect new DNS data exfiltration behaviors and does not retroactively identify exfiltration activity that was already ongoing before deployment.

4.2 Information Tracking Module

For a query that does not match the allowlists, we process it in the Information Tracking Module, which is dedicated to the host identified by its source IP address H (We discuss NAT-related limitations and mitigation in Section E). We maintain a cache C of ITM. To save memory resources, we use a TTI (Time to Idle) eviction policy to evict the ITMs for inactive hosts that do not perform non-allowlisted DNS queries in $2R$ seconds. By efficiently aggregating the amount of potentially exfiltrated information for each host, we can identify the compromised host and provide a ranked list of potentially malicious domains, which is useful for the security response team to identify the malware. This module is divided into six parts: the information quantification algorithm that quantifies the

amount of information in the query, the Bloom filter and information counter that count potentially exfiltrated unique information from this host, the domain tracker that tracks suspicious domains, a detection threshold θ, and a reset mechanism that resets the module at a configured reset interval R.

4.2.1 Information Quantification. Since the attacker could encode information using only case-insensitive characters in DNS labels, corresponding to the following alphabet

$$\Sigma = \{a, b, \ldots, z, 0, \ldots, 9, \text{“.”}, \text{“_”}, \text{“-”}\},$$

which contains $|\Sigma| = 39$ characters. Each character in Σ conveys a **unit of information** (denoted simply as **unit** in the rest of this paper), which is approximately $\log_2 |\Sigma| \approx 5.29$ bits. In practice, the achievable information capacity is further constrained by DNS protocol limitations [29], such as the maximum label length of 63 characters and the requirement for labels to be non-empty. Therefore, the per-character information estimate above represents an upper bound on the attacker's practical exfiltration capacity. The subdomain S is a string over the alphabet Σ, and we measure the units of information in it by taking its length: $I(S) = |S|$.

Since the allowed characters are more restricted in reverse DNS lookups, we refine our approach for such cases to reduce false positives [3]. For IPv4 reverse lookups, the allowed alphabet is

$$\Delta = \{0, 1, \ldots, 9, \text{“.”}\},$$

which only contains $|\Delta| = 11$ characters. Therefore, the quantified amount of information is

$$I(S) = \left\lceil |S| \cdot \log_{|\Sigma|} |\Delta| \right\rceil.$$

For IPv6 reverse lookups, the allowed alphabet is

$$\Phi = \{0, 1, \ldots, 9, a, b, c, d, e, f, \text{“.”}\},$$

which only contains $|\Phi| = 17$ characters. Thus, the quantified amount of information is

$$I(S) = \left\lceil |S| \cdot \log_{|\Sigma|} |\Phi| \right\rceil.$$

4.2.2 Bloom Filter and Information Counter. We maintain an integer counter N that represents the amount of potentially exfiltrated unique information. A straightforward approach would track all previously queried domain names exactly with a hash set to avoid counting any of them twice; however, such an approach would require memory proportional to the number of distinct queries issued by a host, which could cause unbounded memory usage and make the detection algorithm unsuitable for deployment in mission-critical Internet infrastructures like the DNS resolvers.

To achieve bounded memory usage while ensuring that we do not count duplicated queries, we maintain a Bloom filter $\mathbf{B}$ that probabilistically tracks whether the host has already queried the full domain name or not. Bloom filters provide constant-time membership tests with fixed memory cost, at the expense of a low false positive rate. A false positive Bloom filter query would treat a query as already queried and lead to an underestimate of information. A random salt s is used for every ITM to harden the probabilistic data structures against adversarial attackers (Section 4.4.2) that may attempt to exploit the low false positive rate of Bloom filters. For the incoming query name D, if $(D, s) \in \mathbf{B}$, the host already queried D, and we do not add the quantified information to the counter N. Instead, if $(D, s) \notin \mathbf{B}$, the subdomain could be used to exfiltrate new information, and we update the counter by

$$N \leftarrow N + I(S),$$

and add D to the Bloom filter:

$$\mathbf{B} \leftarrow \mathbf{B} \cup \{(D, s)\}.$$

4.2.3 Domain Tracker. For ranking and reporting potentially malicious domains that a host has contacted, we maintain the top-K suspicious domains ranked by the amount of potentially exfiltrated unique information attributed to each domain.

A naive implementation requires maintaining the exact number for all contacted attribution domains, which would require memory proportional to the number of attribution domains for each host. Such an implementation would waste precious memory resources, and it is vulnerable to DoS attacks because an adversarial attacker could query a large number of attribution domains to exhaust memory resources. Since our detector is deployed directly on DNS resolvers and designed to operate continuously on a per-host basis, we require bounded memory usage while still accurately identifying the most suspicious domains. We employ a probabilistic Top-K algorithm [20, 41] that is widely used in Big Data and network monitoring.

This data structure provides two operations. For a Top-K data structure T, procedure Update-TopK(T, e, f) mutates T to add f to the frequency of event e, and procedure Extract-TopK(T) extracts an ordered list of the top-K events with their estimated frequency. In our algorithm, we use a fixed-size linked list and a fixed-size doubly-linked ordered hash map for precisely tracking the top-K elements with their keys and a Count-Min Sketch [10] with conservative update [14] to track other elements. This hybrid design balances accuracy reporting for heavy hitters with bounded memory usage per host. When updating the information counter, we also update the domain tracker with its attribution domain as the event and $I(S)$ as the addition to its frequency:

$$\text{Update-TopK}(T, (P, s), I(S)).$$

4.2.4 Reset Mechanism. To make our detector remain vigilant over prolonged operation, it is essential to reset the ITM periodically. Without such a mechanism, the information counter would grow monotonically, losing its interpretability as a measurement of the amount of recently potentially exfiltrated information and increasing the risk of false alarms triggered by benign domains, and the Bloom filter would eventually become too saturated to report false positive presence of domains that could lead to an underestimate of exfiltrated information. Similarly, the domain tracker might accumulate stale heavy hitters. Thus, for every **reset interval** of R seconds, we reset the ITM so that it focuses on measuring the amount of potentially exfiltrated information inside that reset interval.

4.2.5 Threshold. We maintain a global detection threshold θ. The setting of θ is completely transparent and explainable. It is the upper bound of the average information exfiltrating rate measured inside

[3] We acknowledge that it is technically possible to use other characters in `in-addr.arpa` and `ip6.arpa` domains. But such usage is generally not expected, and CAs no longer issue certificates to domain names on reverse zones after 2026-03-15 [6]. Thus, we assume either the resolver is configured to reject such queries or IDS rules are present to catch such anomalies.

a reset interval. If N exceeds $\theta \cdot R$ after an update to N, we raise an alert for host H with its potentially exfiltrated information amount N and top-K suspicious domains with the amounts of potentially exfiltrated information to each of them collected from Domain Tracker.

Algorithm 1: RT-MD Algorithm.

Input: Configurable Suffix List CSL; Allowlists A; Threshold θ; Reset Interval R; DNS Query Stream Q; Max number of hosts M;
Output: Alert Stream O (may contain duplicate hosts).

Initialize ITM cache C with TTI eviction duration of $2R$;
foreach *incoming DNS query* q *in* Q **do**
  Extract host IP H, timestamp t and qname D from q;
  Split D into attribution domain P and subdomain S according to CSL;
  if $P \in A$ **then continue**;
  if $H \notin C \vee t - C[H].reset_ts > R$ **then**
    if $H \notin C \wedge |C| \geq M$ **then**
      output alert into O: too many hosts;
      continue
    else
      $C[H] \leftarrow$ **new** Information Tracking Module;
      $C[H].s \leftarrow$ **random** fixed-length bytes;
      $C[H].\mathbf{B} \leftarrow$ **new** Bloom Filter;
      $C[H].N \leftarrow 0$;
      $C[H].T \leftarrow$ **new** TopK data structure;
      $C[H].reset_ts \leftarrow t$;
    end
  end
  if $(D, C[H].s) \notin C[H].\mathbf{B}$ **then**
    $C[H].\mathbf{B} \leftarrow C[H].\mathbf{B} \cup \{(D, C[H].s)\}$;
    $C[H].N \leftarrow C[H].N + I(S)$;
    Update-TopK$(C[H].T, (P, C[H].s), I(S))$;
  end
  if $C[H].N > \theta \cdot R$ **then**
    output alert into O: t, H, $C[H].N$, Extract-TopK$(C[H].T)$;
  end
end

4.3 Complexity Analysis

In this section, we analyze the time and space complexity of RT-MD.

4.3.1 Time Complexity. In the Query Pre-Processing module, we extract the query name in $O(1)$ time, as its maximum length is 255 characters. Then we split it into the attribution domain and the subdomain in $O(1)$ time because of the length limitation of the query name, and the length of the configurable suffix list is constant. After that, allowlist matching happens in $O(1)$ time as a hash table lookup. For queries that do not match the allowlists, we perform the following steps: If there is no ITM for the source IP address in the cache, we initialize a new one in $O(1)$ time and insert it into the cache in $O(1)$ time as a hash table insertion. Then, we check the presence of the domain in the Bloom filter in $O(1)$ time. If it tests negative, we update the Bloom filter and counter in $O(1)$ time. As for the domain tracker, updating the linked list costs $O(K) = O(1)$ time, while updating the doubly-linked hash table and potentially the Count-Min Sketch costs $O(1)$ time. Thus Update-TopK overall costs $O(K) = O(1)$ time. If the threshold is exceeded, RT-MD collects top-K domains with Extract-TopK in $O(K) = O(1)$ time, as it is a simple iteration over the linked list. When a reset interval is reached, we reset the Information Tracking Module in $O(1)$ time. As a result, the detector processes each DNS query in $O(1)$ time.

4.3.2 Space Complexity. The sizes of allowlists and the public suffix list are constant at runtime, therefore costs $O(1)$ storage. For each ITM, the information counter and Bloom filter cost $O(1)$ storage. The domain tracker of the ITM consists of a linked list of at most K elements, which costs $O(K) = O(1)$ storage, a doubly-linked hash table whose size is bounded by K, which costs $O(K) = O(1)$ storage and a Count-Min Sketch that costs $O(1)$ storage. We maintain a cache, which is internally a hash table that maps host IP addresses to ITMs. This cache costs $O(M)$ storage, where M is the number of hosts to be monitored. So the space complexity of our detector is $O(M)$. In practice, we set a limit on the number of hosts and generate an alert if there are more hosts than we can handle, so the memory usage is bounded and thus suitable for deployment in resource-constrained environments. We evaluate the practical resource usage in Section 5.3.1.

4.4 Adversarial Attacks Against RT-MD

In this section, we consider and analyze adversarial attacks against RT-MD.

4.4.1 Evasion via False-Positive Cover Traffic. Within each reset interval, an attacker can send more false positive queries with higher information estimates to K benign domains while sending real exfiltration queries to the malicious domain. An intentional false positive alert will be generated with the K benign domains not including the attacker's domain that is used for exfiltration. Such an attack would only be possible if the K benign domains are not in the allowlists. If executed naively, this strategy would generate a large number of `NXDOMAIN` responses when intentional false positive queries with high information estimate contain non-existent subdomains on a benign domain, which could easily be detected by anomaly-based IDS [26]. Furthermore, after the benign domains are added to the allowlists, the attacker would lose its concealment. Thus, such an evasion requires attackers to regularly rotate the set of benign domains to stay under the radar, which is not practical since the attacker does not know all the allowlists.

4.4.2 Robustness of Probabilistic Data Structures. Probabilistic data structures like Bloom filter and Count-Min Sketch used in our work and HyperLogLog++ used in ibHH [32] may be vulnerable when used in adversarial environments [9, 33, 37]. An attacker with prior knowledge about them could try to circumvent detection by crafting a special encoding that could make the encoded domains collide with the hash functions of the Bloom filters with high probability. This would enable the attacker to exfiltrate more information than

our estimation. However, the length and charset restrictions of DNS label names [29] significantly increase the difficulty of constructing such encodings while efficiently exfiltrating information. Moreover, Clayton et al. [9] proved that using per-representation salts in the Bloom filter could guarantee provable security when the representation is kept private from the attacker. We implemented this mitigation in our detector (Section 4.2.2). Additionally, Clayton et al. [9] proved that the security of the Count-Min Sketch can be achieved by combining a per-representation salt, thresholding and keeping the representation private from the attacker. All probabilistic data structures used in RT-MD are hardened against adversarial attacks with very little performance overhead.

5 Evaluation

In this section, we introduce the datasets and the baseline in Section 5.1, study parameter sensitivity in Section 5.2, compare against ibHH [32] and baseline in Section 5.3, and perform an ablation study against the components that contributed to the reduction of false positives (Section 5.4). We deploy our method in a large-scale real-world campus network to further evaluate the effectiveness, scalability, and maintenance overhead (Section 5.5).

5.1 Preliminaries

5.1.1 Datasets. In this study, we primarily use public datasets like DNS queries collected by Žiža et al. [43] and DNS tunnel traffic collected by Gao et al. [19]. We also collect DNS queries from our campus for real-world evaluation and serving as benign background traffic to complement the GraphTunnel dataset. In addition, we create and publish a dataset of adversarial multi-domain DNS exfiltration. A summary of the datasets is available in Table 2.

Public dataset (Ziza). Žiža et al. [43] collected more than 35M DNS requests from a national ISP in Serbia over 24 hours, during which they registered `dnsresearch.ml` domain for experimenting with DNS exfiltration with adversarial attacks and modified exfiltrator behavior. We denote this dataset as $\mathcal{D}_Z$.

Mixed dataset (GraphTunnel). Gao et al. [19] published their GraphTunnel dataset, encompassing 962,859 DNS records generated from a wide array of DNS Tunneling tools. Additionally, this dataset includes a large volume of script-generated benign traffic from a single host, which is unrealistic from the perspective of a host detector like RT-MD. In order to replace the unrealistic benign traffic with realistic traffic, we collected 26 hours of DNS query logs from 5 main DNS resolvers deployed in our campus network. We manually labeled our collected data, where the results are available in Section 5.1.2. Furthermore, although the dataset covers a wide variety of DNS tunneling tools, the IP addresses of the hosts that generate the data are concentrated. To test the methods' effectiveness against individual tunneling tools, we apply pseudo-NAT to the GraphTunnel dataset to map different tunneling tools to distinct IP addresses. We partition this dataset into three parts, where the first 10 hours of data are used for threshold tuning, the next two hours of data are used for peacetime allowlist generation and the rest for evaluation. We denote this dataset as $\mathcal{D}_G$.

Multi-Domain Exfiltration dataset. To evaluate the effectiveness against adversarial multi-domain exfiltration, we generated an all-malicious dataset by simulating a naive DNS exfiltration that splits the payload and uses base64-like encoding to exfiltrate the data to authoritative nameservers of multiple attribution domains. We do not mix benign DNS queries into it, as it serves as a testbed and only contains malicious exfiltration queries. The first part of this dataset is generated by exfiltrating data at the rate of about 32 B/s using 2, 4, 8, 16, 32, and 64 different attribution domains. The second part of this dataset simulates low-throughput data exfiltration by exfiltrating data at the rate of about 1 B/s using 2, 4, and 8 different attribution domains. We denote this dataset as $\mathcal{D}_M$.

Table 2: A summary of datasets used in this paper

Dataset	#Hosts	#PSD*	#Queries	#TP Hosts+
$\mathcal{D}_Z$	35,989	74,677	35,074,151	2
$\mathcal{D}_G$	202,445	231,137	781,351,105	19
$\mathcal{D}_M$	9	140	23,361	9

*Public suffix domain. +True positive hosts.

5.1.2 Analysis of Background Traffic in $\mathcal{D}_G$. For $\mathcal{D}_G$, since the background traffic is unlabeled, we run RT-MD and ibHH with a low threshold to label it. We present an analysis of alerts generated by ibHH and RT-MD as follows.

True Positives. Two hosts actively exfiltrate UNIX timestamps in millisecond precision by prefixing the timestamp string in front of an identifier to form a third-level domain of `1bsone.com`. The exfiltrated timestamps are monotonically increasing and usually about 185ms behind the time we received that DNS query. Based on Open-Source Intelligence (OSINT), we couldn't associate any kind of malicious activity with this domain. So we deemed it some sort of benign heartbeat over DNS, but still labeled the two hosts as true positives.

Special hosts. RT-MD detects one host that is regularly triggering alerts with surprisingly high information estimates. It is querying popular and unpopular domains at an exceptionally fast pace. From its IP address, we could tell that it is the mail server of our campus. Thus, such behavior is expected as the mail server sends and receives a large volume of email, which requires resolving tons of unpopular domain names due to SPF, DKIM, and more.

Reverse DNS queries. RT-MD and ibHH both experience false positives from reverse DNS queries. Despite that, we try to combat this with more accurate information quantification for reverse DNS queries (Section 4.1.1) and a peacetime allowlist, there are still false positives caused by hosts executing lots of reverse DNS queries in a burst.

Misconfiguration. Several hosts are querying SVCB (Service Binding) and HTTPS records directly on IP addresses, e.g., `_50443._https.<IPADDR>`. Some hosts seem to be including the port of the website in the DNS query name, which creates a non-existent TLD. Aside from that, there are misconfigured hosts that query domains under non-existent TLDs like `dhcp` and `getcacheddhcpresults forcurrentconfig` on our DNS server. We are also quite surprised that some hosts query Android package names under several non-existent TLDs like `fileprovider`.

High-Cardinality Subdomains. Some services use DNS to encode long and machine-generated identifiers into the domain name. For example, Synology uses `*.direct.quickconnect.cn` for their NAS remote access service. Since they added it to the

Public Suffix List and the service is using a long machine-generated identifier in **fifth**-level domain, both ibHH and RT-MD report such domains as false positives. However, such kinds of false positives could be easily solved by allowlisting the public suffixes.

Confirmed NAT Gateways. RT-MD reports some NAT gateways as false positives because Network Address Translation could let multiple hosts share a single IP address, but RT-MD would treat it as a single host. Address sharing makes RT-MD more likely to generate false alerts, as it could dramatically increase the DNS query frequency and diversity for a single IP address. We discuss NAT-related problems in Appendix E.0.1.

5.1.3 Baseline. We introduce another host-centric method called UniqD (**Algorithm** 2) , which is designed from a different perspective. It tracks the number of unique FQDNs a host queries in a reset interval using HyperLogLog++. It shares the Query Pre-Processing Module from RT-MD and will generate an alert if the average query rate of unique FQDNs exceeds the threshold. This baseline method also utilizes the allowlists. However, since it could not report related suspicious domains, we do not perform peacetime allowlist generation for it. Instead, this method will use the peacetime allowlist generated for RT-MD.

5.2 Sensitivity Analysis

We perform a sensitivity analysis of the threshold and reset interval parameters to better understand their influence on detection capability.

5.2.1 Reset Interval. We perform the sensitivity analysis of the reset interval parameter for RT-MD and our UniqD baseline method on our largest dataset $\mathcal{D}_G$. During this analysis, we fix the detection threshold at 1 unit/s for RT-MD and at 0.5 FQDN/s for UniqD. We vary the value of the reset interval from 20 seconds to 30 minutes with a step of 20 seconds. As shown in Figure 3, increasing the reset interval causes a significant reduction of false positive alerts and a minor reduction of true positive alerts. As increasing the reset interval increases the amount of information that could be exfiltrated for that interval, such a reduction of alerts is expected. Thus, increasing the reset interval brings both a positive effect of reducing false alarms and a negative effect of detecting true positives. We repeat this analysis with allowlists disabled in order to isolate the effect of the reset interval parameter without the influence of allowlists.

5.2.2 Threshold. We perform the sensitivity analysis of the reset interval parameter for RT-MD and our baseline method UniqD on our largest dataset $\mathcal{D}_G$. We tune the detection threshold from 0.1 unit/s to 320 unit/s for RT-MD and 0.1 FQDN/s to 10 FQDN/s for

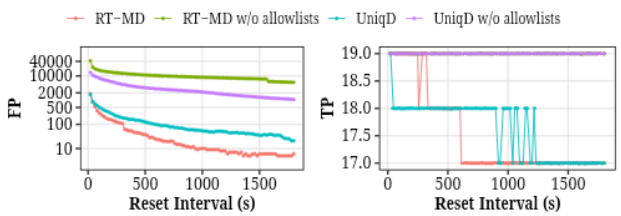


Figure 3: Sensitivity Analysis of Reset Interval.

Algorithm 2: UniqD Algorithm

Input: Allowlists A; Threshold θ; Reset Interval R; DNS Query Stream Q; Max number of hosts M;
Output: Alert Stream O (may contain duplicate hosts).

Initialize cache H with TTI eviction duration of $2R$;
foreach *incoming DNS query* q *in* Q **do**
  Extract host IP H, timestamp t and query name D from q;
  Split D into attribution domain P and subdomain S according to CSL;
  if $P \in A$ **then continue**;
  if $H \notin C \vee t - C[H].reset_ts > R$ **then**
    if $|C| \geq M$ **then**
      output alert into O: too many hosts;
      continue
    else
      $C[H] \leftarrow$ **new** UniqD Module;
      $C[H].salt \leftarrow$ **random** fixed-length bytes;
      $C[H].reset_ts \leftarrow$ t;
      $C[H].counter \leftarrow$ **new** HyperLogLog++ ;
    end
  end
  Insert $(D, C[H].salt)$ into $C[H].counter$;
  $v \leftarrow$ Calc. HyperLogLog++ estimate from $C[H].counter$;
  if $v > \theta \cdot R$ **then**
    output alert into O: t, H, v;
  end
end

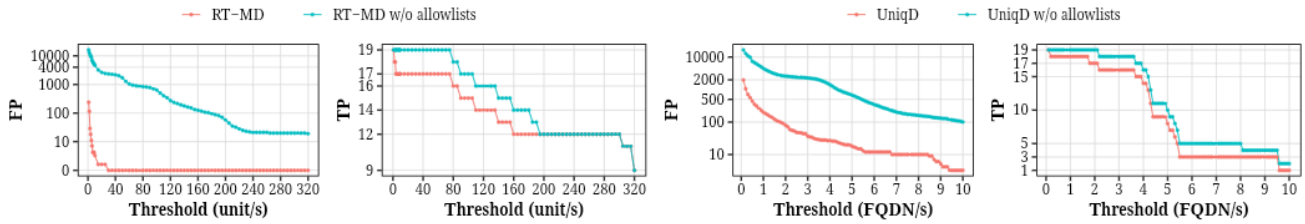
UniqD. During this experiment, we fix the reset interval to 120 seconds for both methods.

We present the results for RT-MD in Figure 4a. When the threshold is about 1 unit/s, RT-MD could detect all true positives while producing about 100 false positives. As the threshold increases, the number of false positives drops significantly. When the threshold is 40 unit/s, RT-MD does not produce any false positives while still detecting all malicious true positives in $\mathcal{D}_G$. The allowlists play a key role in reducing the false positives while only causing a minor reduction of true positives. On the right of Figure 4a, the two true positives that are not detected when allowlists are enabled, compared to the case without allowlists, correspond to two hosts in our campus network that send timestamps over DNS and are deemed benign in Section 5.1.2.

We present the results for UniqD in Figure 4b. When the threshold is about 2 FQDN/s, UniqD can detect all malicious true positives while producing about 100 false positives. However, its detection capability quickly decreases as the threshold increases. When the threshold is 10 FQDN/s, UniqD detects only a single true positive but produces 3 false positives.

5.3 Comparison with Existing Methods

Aside from the baseline method, we also compare our method with ibHH [32], which is the SOTA method for detecting DNS exfiltration with information estimates in real-time. Since RT-MD



(a) RT-MD (b) UniqD Baseline

Figure 4: Sensitivity Analysis of Threshold Parameter.

Table 3: Comparison of the Methods.

Method	Dataset	Acceptable FPR = 0.01				Acceptable FPR = 0.001				Acceptable FPR = 0.0001			
		TPR	FPR	TTD*	THS+	TPR	FPR	TTD*	THS+	TPR	FPR	TTD*	THS+
RT-MD	$\mathcal{D}_Z$	**1**	0.0067	230	0.1250	**1**	0.0008	30	1.1417	**1**	**0.00003**	3	10.950
	$\mathcal{D}_G$	**1**	0.0067	1137	0.2750	**0.9474**	**0.0007**	132	2.0083	**0.8947**	0.00008	31	4.3750
	$\mathcal{D}_M$	**1**	0	9	0.2750	**0.6667**	0	6	2.0083	**0.6667**	0	6	4.3750
UniqD	$\mathcal{D}_Z$	**1**	**0.0041**	144	0.0250	**1**	**0.0007**	25	0.1083	**1**	0.00006	4	0.3750
	$\mathcal{D}_G$	**1**	0.0103	1738	0.1167	**0.9474**	0.0009	170	1.2833	0.8421	0.00018	47	3.3750
	$\mathcal{D}_M$	0.6667	0	6	0.1167	0.5556	0	5	1.2833	0	0	0	3.3750
ibHH (hosts)	$\mathcal{D}_Z$	**1**	0.0118	404	0.6250	**1**	0.0024	84	1.0333	**1**	0.00006	4	5.3751
	$\mathcal{D}_G$	**1**	**0.0049**	847	0.8583	0.8947	0.0008	155	2.6334	**0.8947**	**0.00004**	24	6.1835
	$\mathcal{D}_M$	0.5556	0	5	0.8583	0.4444	0	4	2.6334	0.2222	0	2	6.1835

* The total number of detections.

+The detection threshold of the method. For RT-MD and ibHH, it is reported in units of information per second. For UniqD, the unit is FQDN/s.

detects suspicious hosts while ibHH detects suspicious domains, we could not compare them directly. To compare the effectiveness of detecting suspicious hosts, we map the domain detection results of ibHH to the hosts that queried them. By converting ibHH's domain detection results to host detection results, we get an offline variant of the ibHH method that can detect hosts, therefore comparable to RT-MD. Such a conversion is also operationally-realistic, since defenders routinely trace malicious domains back to the hosts that queried them by searching the DNS logs.

We generate the peacetime allowlists for RT-MD and ibHH on $\mathcal{D}_Z$ and $\mathcal{D}_G$. Since UniqD cannot generate a peacetime allowlist but shares the Query Pre-Processing Module with RT-MD, we use the peacetime allowlist generated by RT-MD for it in all the experiments. We employ the same popularity-based allowlist as described in Section 4.1.2 for all the methods in all experiments. We also employ the same configurable suffix list for all the methods. For $\mathcal{D}_Z$ and $\mathcal{D}_G$, we configured the threshold parameter of the methods by tuning them with different acceptable false positive rates: 0.01, 0.001, and 0.0001. For the adversarial multi-domain exfiltration testbed $\mathcal{D}_M$, we use the threshold tuned on $\mathcal{D}_G$.

We configured other parameters of the methods as follows:

1) ibHH: We set the reset interval to 120 seconds and cache size to 1000 entries according to [32].

2) RT-MD and UniqD: We set the reset interval to 120 seconds, which is the same as the value used for the ibHH method.

Table 3 shows the comparison results with different acceptable FPR values. We observe that the effective FPR for certain methods deviates slightly from the target acceptable FPR, which is expected due to the data distribution difference between the tuning and the evaluation set.

For the acceptable FPR of 1%, RT-MD is the only method that could detect all true positives on all datasets. The threshold for RT-MD is 0.275 unit/s on $\mathcal{D}_G$, which guarantees a very short time-to-detection that is crucial for rapid prevention to reduce depredation. For common credit card data in Track 2 format [17], exfiltrating a single credit card in plain text requires at most 40 units of information since no encoding is required for numeric data. Combined with a reset interval of 120s, a host could only exfiltrate 33 units of information before causing an alarm. Thus, our method could raise an alert before the second credit card gets exfiltrated, catching the exfiltration after only the first credit card gets exfiltrated. While for ibHH, the attacker could exfiltrate 102.996 units of information to a single domain before causing an alarm. The attacker would get caught when exfiltrating the third credit card. It is impossible to do such an information estimate analysis for UniqD since it focuses on the unique query count to domains that could be potentially used for exfiltration. But if we assume that the attacker encodes each credit card into a single DNS query, UniqD would catch it after 14 credit cards get exfiltrated. For acceptable FPRs of 0.1% and 0.01%, our method also achieves the highest TPR on all datasets.

On the multi-domain exfiltration testbed $\mathcal{D}_M$, our method achieves the best performance over all acceptable FPRs. It can be seen that domain-centric detection methods like ibHH inherently cannot counter such adversarial multi-domain DNS exfiltration. Even though

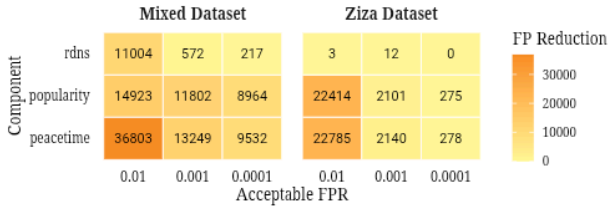


Figure 5: False Positive Reduction By Component of RT-MD

Table 4: Resource Usage and Throughput Measurement

Method	CPU Time (Second)	Max. Memory (MiB)	Throughput (Queries per sec)
RT-MD	262.460	1680.55	1,426,698
UniqD	228.596	342.519	1,642,800
ibHH	1061.93	339.007	355,474

our baseline method UniqD performs a little better than ibHH on such adversarial attacks, it fails to detect any true positives for an acceptable FPR of 0.01%, which shows the unreliability of surface-level features like unique FQDN count. Our method detects all of the conventional multi-domain exfiltration true positives across all three acceptable FPRs but misses the three low-throughput multi-domain exfiltration hosts for acceptable FPRs of 0.1% and 0.01%. ibHH could not even detect all conventional multi-domain exfiltration attempts for the highest acceptable FPR of 1%, which shows the vulnerability of domain-centric detection methods against multi-domain exfiltration.

On the largest dataset $\mathcal{D}_G$, all three methods could detect all true positives from the original GraphTunnel dataset [19]. However, for acceptable FPRs of 0.1% and 0.01%, most methods cannot detect the two true positive hosts from our campus network that send timestamps over DNS (Section 5.1.2).

5.3.1 Throughput and Resource Usage. We measure the throughput and resource usage of the methods on a regular desktop computer with CachyOS installed, which comes with 16GB of DDR4 RAM and AMD Ryzen 5 5600G CPU. This test environment represents a reasonable hardware specification that large enterprises need to deploy a high-performance DNS resolver.

We assess the resource usage and throughput by benchmarking the methods on $\mathcal{D}_G$ dataset five times and take the average of max memory usage (measured as the maximum resident set size of the process), CPU time spent in userspace and throughput. During this benchmark, we turn off alert reporting to avoid excessive IO operations and avoid updating evaluation-specific data structures to avoid incurring extra CPU and memory costs.

The results of this benchmark are available in Table 4. UniqD achieves the best balance between memory usage and throughput due to its simplicity, while ibHH achieves the lowest memory footprint with the lowest throughput, but spends much more CPU time compared with other methods. To the best of our knowledge, our method achieves the highest throughput of 1.4 million queries per second compared to previous work, exceeding the throughput of ibHH by more than a factor of four. This exceptional throughput indicates that our method could scale effectively from large campus networks to huge enterprise environments. Although our RT-MD method is the most memory-intensive of all, it does not hurt the utility of our method in production environments because the results show that 2GB of memory is enough for our large campus network of 202,445 hosts with unique IP addresses. Additionally, we measured the exact memory usage of our method for a single host. Monitoring a single host costs 32.2 kB of memory, illustrating that our method can simultaneously monitor 500,000 active IP addresses on the test machine, showing great potential for deployment in huge enterprise environments.

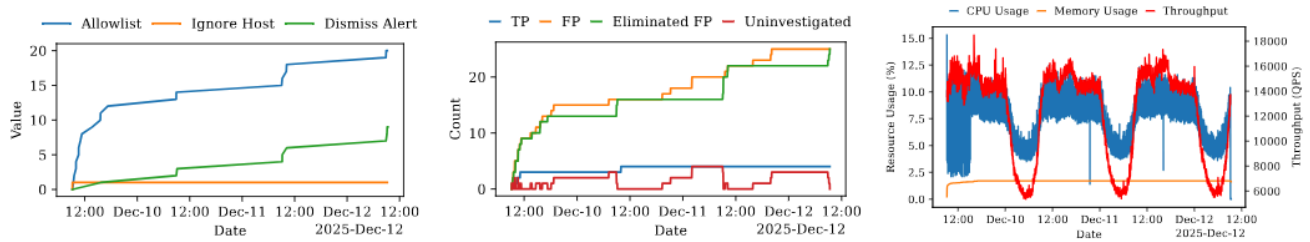
5.4 False Positive Reduction By Component

We perform an ablation study against the components responsible for reducing false positives of RT-MD on $\mathcal{D}_G$ and $\mathcal{D}_Z$ using the parameters tuned for different acceptable FPRs in Section 5.3. We remove all allowlists to form a basic version of our method. For our specialized case of RDNS handling (Section 4.1.1), we still attribute the information quantification to /16 and /48 network ranges separately for IPv4 and IPv6, but we use our method's default information quantification algorithm in this basic version. Then we add each component to it and measure the reduction of false positives.

We illustrate the results of this experiment in Figure 5. We note that the total reduction in false positives does not equal the reduction observed from the basic version to RT-MD, due to overlapping false positive reduction contributed by multiple components. When the acceptable FPR is 0.1% or 0.01%, the primary drivers for the reduction of false positives are the popularity-based allowlist and peacetime allowlist, where they contribute almost equally to it. On $\mathcal{D}_G$, the effectiveness of our specialized algorithm for reverse DNS queries increases as the acceptable FPR increases. It is expected as increasing the acceptable FPR also increases the threshold, making long IPv6 reverse DNS queries more prone to trigger false positives. On $\mathcal{D}_Z$, we do not observe a significant effect of our specialized algorithm for reverse DNS queries, which could be explained by the scarcity of IPv6 reverse DNS queries in the specific network environment where Žiža et al. [43] collected the dataset. There are only 8,053 reverse DNS queries in the evaluation set of $\mathcal{D}_Z$, compared to 2,512,110 in $\mathcal{D}_G$. On $\mathcal{D}_G$, it is observed that the peacetime allowlist plays a more important role than the popularity-based allowlist.

5.5 Real-World Evaluation

Previously, we focused our evaluation metrics on true and false positive hosts, which is an outcome after evaluation. However, for real-world usage, we must focus on the number of alerts since a host might have a firing alert in several non-continuous reset intervals. Unlike domain-based detectors like ibHH, where a false alert could be simply silenced by allowlisting the domain, usually, we cannot silence an RT-MD alert by allowlisting the host. Thus, we additionally evaluate the practicality of our method on a real-world live DNS query stream by evaluating the amount of required human labor. We create a control panel for RT-MD that displays the alerted hosts and their most significant alert. A security operator could view the alerts and perform one of the following three actions: 1) Add an attribution domain or a suffix to the allowlist, after which



(a) Manual Operations (b) Summary of Alerts (c) Throughput and Resource Usage

Figure 6: Experiment Results of Real-World Evaluation.

our system would try to deduct the information estimate from allowlisted attribution domains for existing alerts and automatically silence alerts that are below the threshold after deduction. 2) Dismiss the alert, in case the alert is a false alarm. 3) Ignore a host, which will cause the system to dismiss the current alert of the host and ignore future alerts about the host. The last action should be used with great care, as it excludes the host from our detection system.

To avoid downtime and accidental breakage of DNS services, we do not directly deploy our method on our internal DNS resolvers. Instead, we use a real-time DNS log stream forwarded from already deployed data collectors at the network perimeter in our campus network. Using a post-detection filter, we manually ignored some NAT gateways already known to us that are generating a high volume of DNS traffic. Such NAT gateways are prone to cause false positives, and we propose a cascade deployment model as a mitigation in Appendix E.0.1.

We organize this five-day experiment as follows. We run RT-MD in peacetime mode for 1 day to generate a peacetime allowlist. Then we tune the threshold for an acceptable FPR of 0.001 for another day, and finally evaluate for 3 days. During the evaluation phase, we set up the following four DNS data exfiltration cases in our campus network.

- **Iodine** [13] is a popular DNS tunneling VPN. We exfiltrated 1MB of random data from host H_1 through it.
- **dnstt** [16] is a DNS tunnel that works at transport layer. We exfiltrated 1MB of random data from host H_2 through it.
- **dnspot** [3] is an open-source C2 framework implemented through DNS tunneling. We deployed its C2 server on the authoritative nameserver H_3 and its agent on H_4, which reflects the scenario of DNS-based C2 malware. We perform data exfiltration by sending commands from H_3 to H_4 that list and steal text files.
- **Naive exfiltration**. Additionally, we emulate a scenario where the attackers steal credentials by putting them inside the subdomain part of the DNS query without any encoding or encryption. We exfiltrated the top 10,000 popular passwords by sending out DNS queries that each contains one password as the subdomain.

We depict the amount of human labor in Figure 6a. Most of human labor is concentrated in the first 12 hours after deployment, where we allowlist 12 domains, dismiss one false positive alert, and ignore one host. The ignored host is our mail server, which constantly produces false alerts due to high-volume queries to unpopular domains. In the remaining 60 hours of evaluation, we only allowlist 8 domains and dismiss 8 alerts. Thus, our method causes a negligible maintenance burden and is highly suitable for deployment in production environments.

Figure 6b shows the summary of alerts on the dashboard. Our detector detects all four DNS data exfiltration scenarios we set up. The increment of eliminated false alerts can be explained by manual operations in Figure 6a. There are never more than four alerts pending investigation on the dashboard. During evaluation, the commercial intrusion detection system in our campus network does not detect our DNS data exfiltration attempts and does not detect other DNS data exfiltration behavior.

We report DNS query throughput and resource usage of our detector in Figure 6c. This real-world evaluation is conducted on a virtualized server with 64GB of RAM and 32 cores running at 2.6GHz. The reported CPU usage is the utilization rate of a single core, since our detection algorithm is currently single-threaded. The CPU usage is about 7.5% to 11% when the DNS query throughput is about 14k to 16k queries per second during the day. During late night, the CPU usage drops as the DNS query throughput drops, as expected. The memory usage of our detector never exceeds 1.8%, or 1.152 GB. The low CPU overhead and reasonable memory usage allow us to implement our detector as a software upgrade to an existing DNS server, making our approach cost-effective.

6 Conclusion

In this paper, we proposed RT-MD, a cost-efficient, real-time, and high-throughput DNS data exfiltration detector designed for deployment at internal DNS resolvers and in large-scale networks. RT-MD addresses several fundamental challenges that limit existing detection approaches. First, by adopting a host-centric detection model, RT-MD overcomes the limitations of domain-centric detectors and enables effective detection of multi-domain DNS exfiltration, an adversarial evasion strategy. Second, RT-MD is built upon probabilistic data structures and operates in constant time, achieving high throughput and low memory overhead, which makes it practical for deployment under high-volume DNS traffic and resource constraints of DNS resolvers. Third, extensive evaluation demonstrates that RT-MD consistently outperforms prior state-of-the-art methods in detecting both conventional and adversarial DNS exfiltration attacks, while maintaining a low false positive rate. Finally, real-world deployment of RT-MD in our large-scale campus network shows the low maintenance overhead for security operators and operational practicality.

References

[1] Sanjay Adiwal, Balaji Rajendran, Pushparaj Shetty D., and Sithu D. Sudarsan. 2023. DNS Intrusion Detection (DID) — A SNORT-based Solution to Detect DNS Amplification and DNS Tunneling Attacks. *Franklin Open* 2 (March 2023), 100010.

[2] Jawad Ahmed, Hassan Habibi Gharakheili, Qasim Raza, Craig Russell, and Vijay Sivaraman. 2020. Monitoring Enterprise DNS Queries for Detecting Data Exfiltration From Internal Hosts. *IEEE Transactions on Network and Service Management* 17, 1 (March 2020), 265–279.

[3] Ali. 2023. *dnspot: End-to-end Encrypted DNS Tunnelling and C2 framework.* GitHub. https://github.com/mosajjal/dnspot/

[4] M. A. Altuncu, F. K. Gulagiz, H. Ozcan, O. F. Bayir, A. Gezgin, A. Niyazov, M. A. Cavuslu, and S. Sahin. 2021. Deep Learning Based DNS Tunneling Detection and Blocking System. *Advances in Electrical and Computer Engineering* 21, 3 (2021), 39–48.

[5] Andrei Broder and Michael Mitzenmacher. 2004. Network applications of bloom filters: A survey. *Internet mathematics* 1, 4 (2004), 485–509.

[6] CA/Browser Forum 2025. *CA/Browser Forum Latest Baseline Requirements.* CA/Browser Forum. https://cabforum.org/working-groups/server/baseline-requirements/requirements/#422-approval-or-rejection-of-certificate-applications

[7] National Cyber Security Centre. 2020. *Advisory: APT29 targets COVID-19 vaccine development.* Technical Report. National Cyber Security Centre. https://www.ncsc.gov.uk/files/Advisory-APT29-targets-COVID-19-vaccine-development.pdf

[8] S Cheshire and M Krochmal. 1987. RFC6762 Multicast DNS. Internet Requests for Comments, 67 pages. http://www.rfc-editor.org/rfc/rfc6762.txt

[9] David Clayton, Christopher Patton, and Thomas Shrimpton. 2019. Probabilistic Data Structures in Adversarial Environments. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security.* ACM, London United Kingdom, 1317–1334.

[10] Graham Cormode and S. Muthukrishnan. 2005. An Improved Data Stream Summary: The Count-Min Sketch and Its Applications. *Journal of Algorithms* 55, 1 (April 2005), 58–75.

[11] David Dagon, Manos Antonakakis, Paul Vixie, Tatuya Jinmei, and Wenke Lee. 2008. Increased DNS Forgery Resistance through 0x20-Bit Encoding: Security via Leet Queries. In *Proceedings of the 15th ACM Conference on Computer and Communications Security.* ACM, Alexandria Virginia USA, 211–222.

[12] DavidP. 2014. *Is This Malware?* ESET Security Forum. https://forum.eset.com/topic/2037-is-this-malware/

[13] Erik Ekman. 2023. *Iodine v0.8.0: Official git repo for iodine dns tunnel.* GitHub. https://github.com/yarrick/iodine/tree/v0.8.0

[14] Cristian Estan and George Varghese. 2002. New directions in traffic measurement and accounting. *SIGCOMM Comput. Commun. Rev.* 32, 4 (Aug. 2002), 323–336. https://doi.org/10.1145/964725.633056

[15] Matthieu Faou. 2023. MoustachedBouncer: Espionage against foreign diplomats in Belarus. https://www.welivesecurity.com/en/eset-research/moustachedbouncer-espionage-against-foreign-diplomats-in-belarus/

[16] David Fifield. 2024. *dnstt v1.20241021.0 – DoH- and DoT-capable DNS tunnel.* bamsoftware. https://www.bamsoftware.com/software/dnstt/

[17] International Organization for Standardization. 2006. *Information technology — Identification cards — Financial transaction cards.* Standard. International Organization for Standardization, Geneva, CH.

[18] Romain Fouchereau. 2023. *2023 Global DNS Threat Report – Augmenting Cyber Threat Intelligence.* Technical Report. International Data Corporation. https://efficientip.com/wp-content/uploads/2023/09/IDC-2023-DNS-Threat-Report.pdf

[19] Guangyuan Gao, Weina Niu, Jiacheng Gong, Dujuan Gu, Song Li, Mingxue Zhang, and Xiaosong Zhang. 2024. GraphTunnel: Robust DNS Tunnel Detection Based on DNS Recursive Resolution Graph. *IEEE Transactions on Information Forensics and Security* 19 (2024), 7705–7719.

[20] Nuno Homem and Joao Paulo Carvalho. 2010. Finding Top-k Elements in Data Streams. *Information Sciences* 180, 24 (Dec. 2010), 4958–4974.

[21] Naotake Ishikura, Daishi Kondo, Vassilis Vassiliades, Iordan Iordanov, and Hideki Tode. 2021. DNS Tunneling Detection by Cache-Property-Aware Features. *IEEE Transactions on Network and Service Management* 18, 2 (June 2021), 1203–1217.

[22] Vitali Kremez. 2019. FIN6 "FrameworkPOS": Point-of-Sale Malware Analysis & Internals - SentinelLabs. https://www.sentinelone.com/labs/fin6-frameworkpos-point-of-sale-malware-analysis-internals/

[23] Kristijan Ziza. 2023. DNS Exfiltration Dataset. https://data.mendeley.com/datasets/c4n7fckkz3/3

[24] Picus Labs. 2024. *APT29 Explained: Cozy Bear's Evolution, Techniques, and Notorious Cyber Attacks.* Picus Labs. https://www.picussecurity.com/resource/blog/apt29-cozy-bear-evolution-techniques

[25] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. 2019. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS 2019).* Internet Society, San Diego, California, 1–15.

[26] Guannan Liu, Lin Jin, Shuai Hao, Yubao Zhang, Daiping Liu, Angelos Stavrou, and Haining Wang. 2023. Dial "N" for NXDomain: The Scale, Origin, and Security Implications of DNS Queries to Non-Existent Domains. In *Proceedings of the 2023 ACM on Internet Measurement Conference.* ACM, Montreal QC Canada, 198–212.

[27] Xiaoyu Liu, Weixuan Mao, Anqi Wang, Zheng Li, Hui Xue, Yijing Zhang, Jianjun Lin, Xiaodu Yang, Ziqian Chen, and Bo Sun. 2023. DNS Tunnel Detection for Low Throughput Data Exfiltration via Time-Frequency Domain Analysis. In *ICC 2023 - IEEE International Conference on Communications.* IEEE, Rome, Italy, 2331–2337.

[28] Stefan Machmeier and Vincent Heuveline. 2024. Detecting DNS Tunnelling and Data Exfiltration Using Dynamic Time Warping. In *2024 8th Cyber Security in Networking Conference (CSNet).* IEEE, Paris, France, 83–91.

[29] P Mockapetris. 1987. Domain names - implementation and specification. RFC 1035. doi:10.17487/RFC1035

[30] Asaf Nadler. 2022. On the Vulnerability of Anti-Malware Solutions to DNS Attacks. *Computers & Security* 116 (2022), 102687.

[31] G DATA CyberDefense 2014. *New FrameworkPOS Variant Exfiltrates Data via DNS Requests.* G DATA CyberDefense. https://www.gdatasoftware.com/blog/2014/10/23942-new-frameworkpos-variant-exfiltrates-data-via-dns-requests

[32] Yarin Ozery, Asaf Nadler, and Asaf Shabtai. 2024. Information Based Heavy Hitters for Real-Time DNS Data Exfiltration Detection. In *Proceedings 2024 Network and Distributed System Security Symposium.* Internet Society, San Diego, CA, USA, 1–15.

[33] Kenneth G. Paterson and Mathilde Raynal. 2022. HyperLogLog: Exponentially Bad in Adversarial Settings. In *Proceedings of 2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P).* IEEE, Genoa, Italy, 154–170.

[34] Vern Paxson, Mihai Christodorescu, Mobin Javed, Josyula Rao, Reiner Sailer, Douglas Lee Schales, Mark Stoecklin, Kurt Thomas, Wietse Venema, and Nicholas Weaver. 2013. Practical Comprehensive Bounds on Surreptitious Communication over DNS. In *22nd USENIX Security Symposium (USENIX Security 13).* USENIX Association, Washington, D.C., 17–32.

[35] Mozilla Foundation 2007. *Public Suffix List.* Mozilla Foundation. https://publicsuffix.org/

[36] Cheng Qi, Xiaojun Chen, Cui Xu, Jinqiao Shi, and Peipeng Liu. 2013. A Bigram Based Real Time DNS Tunnel Detection Approach. *Procedia Computer Science* 17 (2013), 852–860.

[37] Pedro Reviriego and Daniel Ting. 2020. Security of HyperLogLog (HLL) Cardinality Estimation: Vulnerabilities and Protection. *IEEE Communications Letters* 24, 5 (May 2020), 976–980.

[38] Md Abu Sayed, Asif Rahman, Christopher Kiekintveld, and Sebastian Garcia. 2024. Fine-tuning Large Language Models for DGA and DNS Exfiltration Detection . In *2024 Annual Computer Security Applications Conference Workshops (ACSAC Workshops).* IEEE Computer Society, Los Alamitos, CA, USA, 240–247. https://doi.ieeecomputersociety.org/10.1109/ACSACW65225.2024.00034

[39] Neha Sharma, Mayank Swarnkar, and Divyanshu. 2025. DLAZE: Detecting DNS Tunnels Using Lightweight and Accurate Method for Zero-Day Exploits. *IEEE Transactions on Network and Service Management* 22, 3 (2025), 1–1.

[40] Yue Wang, Anmin Zhou, Shan Liao, Rongfeng Zheng, Rong Hu, and Lei Zhang. 2021. A Comprehensive Survey on DNS Tunnel Detection. *Computer Networks* 197 (Oct. 2021), 108322.

[41] Tong Yang, Haowei Zhang, Jinyang Li, Junzhi Gong, Steve Uhlig, Shigang Chen, and Xiaoming Li. 2019. HeavyKeeper: An Accurate Algorithm for Finding Top-k Elephant Flows. *IEEE/ACM Transactions on Networking* 27, 5 (Oct. 2019), 1845–1858.

[42] Wei You, Chunlei Zhao, Chao Shan, Guozhong Dong, Jie Yu, and Yong Guan. 2024. DNS Covert Channel Attack Detection Based on Spatio-Temporal Feature Fusion. In *2024 IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA).* IEEE, Kaifeng, China, 2082–2088.

[43] Kristijan Žiža, Predrag Tadić, and Pavle Vuletić. 2023. DNS Exfiltration Detection in the Presence of Adversarial Attacks and Modified Exfiltrator Behaviour. *International Journal of Information Security* 22, 6 (Dec. 2023), 1865–1880.

[44] Michael Zuckerman. 2020. *APT29 Threat Group Targets COVID-19 Vaccine Research.* Infoblox. https://blogs.infoblox.com/security/apt29-threat-group-targets-covid-19-vaccine-research/

A Ethical Considerations

In dataset $\mathcal{D}_G$ and real-world evaluation(Section 5.5), we use real-world DNS logs from our campus network, which contain sensitive information like the IP address within our internal network and DNS query names. The data is not anonymized, but all analyses are conducted solely for evaluating our detection method, and no effort

is made to identify or profile individual users or devices. Access to the logs was authorized by the campus network administration and restricted to the research team. All results are reported at an aggregated level to avoid revealing information about specific users or hosts.

B Generative AI Usage

This paper was edited for grammar and fluency using OpenAI ChatGPT. We used ChatGPT to generate some auxiliary code in our experiments. For example, we used ChatGPT to assist us when writing scripts that parse the program outputs and export metrics of interest to CSV or other text-based formats. We also used ChatGPT to generate an initial implementation of the alert management user interface for our real-world evaluation. We have manually reviewed all AI-assisted/-generated code and verified its intended functionality. Our algorithm implementation and core experiment code are never assisted or generated by AI.

C Additional Allowlists

Many organizations have internal domain systems that use unregistered top-level domains like `.local`, `.corp`, or `.internal` [8, Appendix G]. As a result, domains using such TLDs are not covered by the popularity-based allowlist. However, since the resolution of such domains happens entirely inside the internal network, it cannot be used to exfiltrate information. The datasets we used in the evaluation do not feature such TLDs. But we would expect an allowlist with internal TLDs to be useful in large enterprises.

D Additional Experiment Results

We present the experiment results of the comparison experiment (Section 5.3) at the acceptable FPR of 0.005 in Table 5. The results in this table are used in the abstract and the Introduction section.

Table 5: Comparison of the Methods at Acceptable FPR 0.005.

Method	Dataset	Acceptable FPR = 0.005			
		TPR	FPR	TTD*	THS+
RT-MD	$\mathcal{D}_Z$	**1**	0.0034	117	0.2833
	$\mathcal{D}_G$	**1**	0.0039	680	0.4333
	$\mathcal{D}_M$	**1**	0	9	0.4333
UniqD	$\mathcal{D}_Z$	**1**	**0.0025**	88	0.0416
	$\mathcal{D}_G$	0.9474	0.0058	998	0.2167
	$\mathcal{D}_M$	0.6667	0	6	0.2167
ibHH (hosts)	$\mathcal{D}_Z$	**1**	0.0027	95	0.8333
	$\mathcal{D}_G$	0.8947	**0.0026**	458	1.0917
	$\mathcal{D}_M$	0.5556	0	5	1.0917

E Discussion

In this section, we discuss the deployment concerns of RT-MD.

E.0.1 Network Address Translation. In RT-MD, we use the source IP address as the unique identifier of a host. However, due to the scarcity of IPv4 addresses, Source Network Address Translation (SNAT) is widely deployed, which makes multiple hosts share a common IPv4 address or several addresses in an address pool. As shown in the analysis of unlabeled data (Section 5.1.2), it leads to false positives because the NAT gateway inherently produces more DNS queries than a single host.

To mitigate it, we propose a cascade deployment model, which is suitable for enterprises that could control all layers of NAT. First, our detector is deployed directly on the internal DNS resolver. Then, for every layer of NAT networks, a DNS forwarder with our detector built in is deployed and configured as the DNS resolver. The DNS forwarders use the top-level DNS resolver as upstream and establish a secure connection to the top-level detector to report alerts in NAT networks.

E.0.2 Encrypted DNS Traffic. RT-MD relies on the inspection of unencrypted DNS queries and therefore cannot directly analyze encrypted DNS traffic, such as DNS over TLS (DoT), DNS over HTTPS (DoH), and DNS over QUIC (DoQ). However, DNS data exfiltration typically targets enterprise environments, where organizations can enforce the use of internal DNS resolvers, enabling RT-MD to operate on real-time DNS logs. Alternatively, enterprises may deploy TLS interception and decryption mechanisms, allowing RT-MD to analyze the resulting plaintext DNS traffic.

F Open Science

We honor the Open Science policy to strengthen the transparency and reproducibility of our work. Both the available artifacts are listed, and the missing artifacts are justified in Section F.1. We provide instructions for evaluating our artifacts on public datasets in the separate *Artifact Appendix*. Additionally, in Section F.2 we provide instructions for the real-world evaluation conducted in Section 5.5.

F.1 List of Artifacts

The following artifacts are available and necessary for evaluating our method and compare it to other methods:

- **Our Source code.** We provide the source code of our algorithm and experiments at this anonymized repository: `https://anonymous.4open.science/r/rt-md/`.
- **ibHH Source Code.** We re-implemented ibHH's algorithm in our source code, but still use the dataset split script from it. The code is publicly available at `https://github.com/akamai/Information-based-Heavy-Hitters-for-Real-Time-DNS-Exfiltration-Detection`.
- **Ziza Dataset.** This is the dataset originally collected in [43]. It is publicly available at [23].
- **GraphTunnel Dataset.** This is the dataset created by Gao et al. [19]. It is publicly available at `https://github.com/ggyggy666/DNS-Tunnel-Datasets`.

The following artifact is used in our evaluation, but we cannot share it.

- **DNS Logs from Our Campus Network.** This artifact is used in conjuction with **GraphTunnel Dataset** to form the **Mixed Dataset**. We also used real-time DNS logs from our campus network in real-world evaluation (Section 5.5).

The DNS logs from our campus network cannot be shared because of privacy and safety concerns. As the logs contain sensitive

information, including the queried domains and host IP addresses, they could be easily used to profile individual hosts and users or reveal enough information for launching a cyber attack against the hosts. We have considered publishing an anonymized version of it by replacing IP addresses and domain names with fake ones. However, since our research method relies heavily on the subdomain part of the query name, any anonymization scheme that changes the length of the subdomain could affect the experiment results. But employing a weak anonymization scheme that preserves the length of the subdomain could make de-anonymization attacks practicable. Furthermore, fingerprinting could be done even with anonymized query names by analyzing the statistical features, like request inter-arrival time. Thus, for the privacy and safety of the users in our campus network, we cannot share this artifact.

F.2 Real-World Evaluation

F.2.1 Additional Environment Setup. First, build the project and enter the directory for conducting real-world evaluation.

```
cd "$srcdir"
cargo build --release --bin rtmd
mkdir -p reports/real && cd $_
```

We use `pidstat` command from `sysstat` for exporting the CPU and memory usage of our detection system in JSON format. However, most Linux distributions don't package `sysstat` software with JSON output formatting capability. Thus, this software needs to be built from source.

```
git clone https://github.com/sysstat/sysstat
./configure
make -j$(nproc)
export PATH="$PWD:$PATH"
```

F.2.2 Real-time DNS Log Stream Format. We expect the real-time DNS log stream to be made available in syslog format. Our program accepts syslog input from a UDP socket. The syslog content should conform to a simple space-separated key-value format, demonstrated by the following example:

```
q_time=2025-11-12T10:59:36.205934
↪  a_time=2025-11-12T10:59:36.227074 src=<IPADDR>
↪  sport=12345 dst=8.8.8.8 tid=23456
↪  q_name=example.com q_type=AAAA a_ip=127.0.0.1
↪  a_cname= error=
```

F.2.3 Threshold Tuning. To tune the threshold on live data for 1 day, run the following command:

```
../../target/release/rtmd -t 0.5 -R 120000 \
   --syslog 127.0.0.1:5150 --duration 86400 tune \
   --acceptable-fpr 0.001
```

The tuned threshold will be available in the program output.

F.2.4 Peacetime Allowlist Generation. To generate the peacetime allowlist from live data for 1 day, run the following command:

```
../../target/release/rtmd -t 0.5 -R 120000 \
   --syslog 127.0.0.1:5150 --duration 86400 peacetime
```

F.2.5 Online Evaluation. To start the online evaluation for 3 days, run the following command:

```
../../target/release/rtmd -t 6.975 -R 120000 \
   --syslog 127.0.0.1:5151 --port 5000 \
   --duration $((60 * 60 * 24 * 3)) \
   eval 2>&1 | tee output
```

As shown in Figure 7, the management dashboard will be available at `http://127.0.0.1:5000` once the evaluation starts. A human operator is expected to take actions to eliminate false alerts on it and deal with true positive alerts.

F.2.6 Plots. After the experiment finishes, aggregate the results by running the following commands:

```
cd "$srcdir"
source .venv/bin/activate
# Collect resource usage
scripts/resource-usage-export.py \
  reports/real/pidstat.json
# Collect throughput
scripts/throughput-export.py reports/real/output
```

Then create the plots:

The figure that shows the amount of human labor (Figure 6a) can be created by running:

```
scripts/human-ops-export.py reports/real/output
# The output is plot/real-world-human-ops.pdf
```

The figure that shows the number of remaining alerts over time (Figure 6b) can be created by running:

```
scripts/alert-series-export.py \
   reports/real/alerts.jsonl
# The output is plot/real-world-alerts.pdf
```

The figure that shows the resource usage and throughput (Figure 6c) can be created by running:

```
plot/real-world-throughput-and-resource.py
# The output is
# plot/real-world-throughput-and-resource.pdf
```

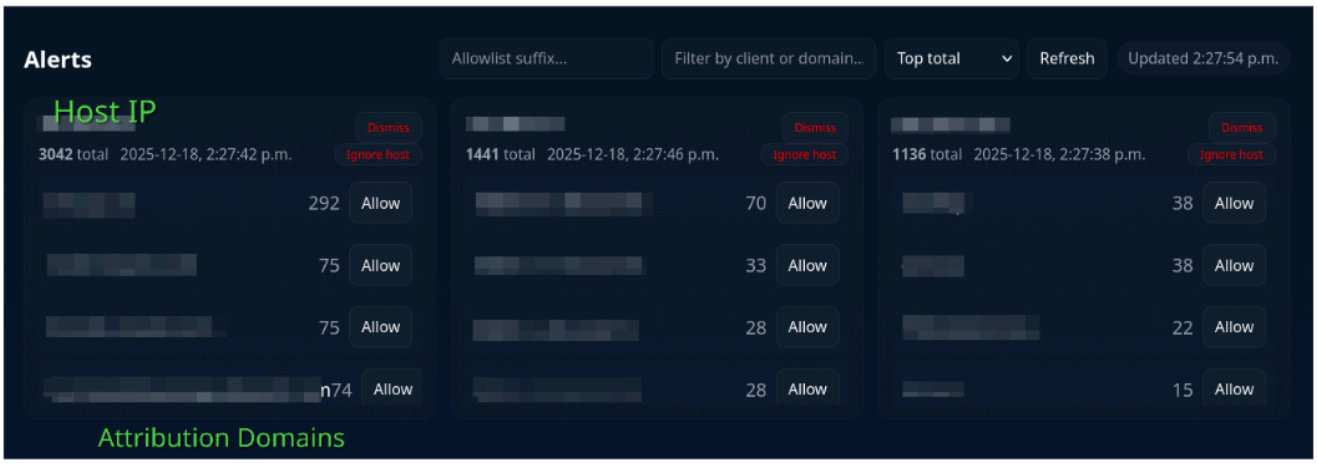


Figure 7: RT-MD Online Detection Dashboard