

namefill: Synthesizing DNS Naming Conventions

Joel Sommers[✉]
jsommers@colgate.edu
Colgate University
Hamilton, NY, USA

Mark Crovella
Boston University
Boston, MA, USA

Calvin Kranig
Wei-Shiang Wung
Paul Barford
University of Wisconsin-Madison
Madison, WI, USA

Eric Pauley
Virginia Tech & Terrace Networks
Blacksburg, VA, USA

Abstract

DNS naming conventions reveal information about infrastructure deployments, operational configurations, and network topology. In this paper, we present *namefill*, a tool that uses *programming by example* to synthesize executable functions that summarize DNS naming conventions. Unlike regular expressions, which match but cannot generate, a synthesized namefill function produces the predicted fully-qualified domain name (FQDN) for addresses within the prefix it was synthesized from, enabling active probing for as-yet undiscovered FQDNs and systematic auditing of existing records. Applied to 1.2 billion PTR records, namefill synthesizes 313K distinct naming conventions across 254K registered domains. Using a PTR record dataset from CAIDA's ITDK, we find that namefill covers 67.7% of IP/FQDN pairs and that its coverage is largely complementary to tools that focus on names that embed geotokens. We use the synthesized functions to systematically identify naming anomalies, including incorrect prefix bytes in PTR record names and inconsistencies in separator characters between IP address bytes. We identify 991 misconfigurations covering 236K PTR records and 1,639 inconsistencies covering 426K PTR records from 2026, and verify 43 misconfigurations with network operators. Lastly, we use the synthesized functions as a DNS record discovery tool to enumerate names in the DNS previously not captured in our source data. From 2.76M training names we find 194M previously unrecorded forward DNS names using 355M queries, a 70× increase.

CCS Concepts

• Networks → Naming and addressing.

Keywords

domain name system, naming conventions, program synthesis

ACM Reference Format:

Joel Sommers, Calvin Kranig, Wei-Shiang Wung, Paul Barford, Mark Crovella, and Eric Pauley. 2026. namefill: Synthesizing DNS Naming Conventions. In *Proceedings of the 2026 ACM Internet Measurement Conference (IMC '26)*, October 12–16, 2026, Karlsruhe, Germany. ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/3777912.3839791>



This work is licensed under a Creative Commons Attribution 4.0 International License. *IMC '26, Karlsruhe, Germany*

© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2327-8/2026/10
<https://doi.org/10.1145/3777912.3839791>

1 Introduction

Network operators use a variety of DNS naming conventions to structure the namespace of the zones they configure in order to identify and manage network resources. Naming conventions are used to identify router models, line cards, interfaces, and locations [8, 31], to signify access technologies [8, 21, 40], and for spam blocking and prevention [1, 22]. In addition, names employing alphanumeric sequences are used to manage resources such as web and mail servers, NATs, and virtual machines [8, 33]. DNS naming conventions are thus worthy of study since they play such an important role in network operations.

In prior work, host and interface naming conventions have been studied in the context of identifying and extracting geographic tokens for geolocation and Internet topology mapping [9, 16, 24, 25, 28, 31, 35], identifying routing and network access equipment [8, 21], and studying cloud infrastructure [33]. Commercial entities rely on maintaining large corpora of regular expressions to track patterns in IPv4 PTR records for spam blocklists and other reputational systems [1, 5]. Each of these prior works studies naming conventions in the DNS (particularly in PTR records) through the lens of a particular application. We are unaware of prior work to more broadly study characteristics of naming patterns and conventions in the DNS.

In this paper, we present *namefill*[30], which automatically synthesizes *executable functions* that both describe and generate FQDNs that follow learned naming conventions. Namefill programs are written in a domain-specific language (DSL) whose syntax defines the space of naming conventions we can represent, and whose programs are directly executable (§ 3). A namefill function parameterized with an IP address from the prefix it was generated from produces an FQDN, enabling active probing for new records and auditing of existing ones. Namefill complements prior work in studying various naming conventions, and extends techniques from programming by example (PBE) [19, 29] to synthesize one or more functions using a collection of IP/FQDN pairs. The output of our method is a set of synthesized functions, the IP/FQDN pairs each function matches, and the remaining set of IP/FQDN pairs that do not follow any observed pattern. We limit our focus to two specific types of naming patterns: those that follow an alphanumeric sequence, e.g., `smtp1.example.com`, `smtp2.example.com`, etc. and those that embed all or part of the associated IPv4 address in the name using some format and ordering, e.g., `002128000003.mbb.telenor.dk` (2.128.0.3). We also limit our focus to IPv4 (§ 10 discusses why).

We use namefill on a set of DNS PTR (reverse DNS) records provided by OpenINTEL[37] from March 2026. With the 1.2B records across 819K prefixes in this dataset, namefill synthesizes 313K distinct naming conventions across 254K registered domains. We find that there are more functions synthesized for sequential patterns than for functions with embedded IP addresses, but that the number of FQDNs covered by the embedded IP functions is $\sim 10\times$ more than those related to the sequential patterns. We also find that the structural vocabulary underlying these conventions is remarkably limited: across the entire dataset there are only 283 distinct cross-domain naming structures, and the top 5 account for $\sim 60\%$ of all FQDNs. Namefill discovers naming convention *instances* and their populations without supervision, though it does not discover convention *classes* beyond those its DSL can express.

We use namefill on a set of PTR records from CAIDA’s Internet Topology Data Kit (ITDK)[2] and compare the set of FQDNs covered by namefill’s synthesized functions with FQDNs that are handled by Hoiho[25] and Aleph[35] and find that these tools are largely complementary. Throughout this paper we say that a synthesized function *covers* an FQDN if evaluating that function with the associated IP address (or, for sequential functions, the preceding FQDN) reproduces the name exactly. Namefill covers 67.7% of the IP/FQDN pairs in the ITDK dataset, Hoiho matches 12.7% pairs, and Aleph matches 4.2%, but only 0.7% of FQDNs are covered by all three tools. For Namefill and Hoiho, the majority of names they cover are unique to each of them.

Embedded IP functions effectively invert BIND’s \$GENERATE macro [17], which operators use to automate creation of DNS records that embed IP address bytes in the FQDN. We identify functions that appear to indicate four bytes of the IP address in the name, but for which only three bytes match the prefix. We identify 991 such misconfigurations in the 2026 PTR record data affecting 236K PTR records, and another set of 989 misconfigurations in PTR record data collected 1 year earlier. We identify fixes to some of these year-over-year, and verify 43 prefix-level misconfigurations with network operators.

We also use namefill on a set of A records supplied by Rapid7 Project SONAR[20]. We use the generative nature of namefill functions as the basis for DNS record discovery. Establishing the existence of additional records in the DNS may be used for infrastructural census or analysis, *e.g.*, as in [33, 40], or for macroscopic analyses of the DNS. We sample 67K synthesized functions (after filtering out those related to wildcard records) and make DNS A record queries using FQDNs produced those functions. From 2.76M training names we find 194M previously unrecorded forward DNS names using 355M queries, a $70\times$ increase.

2 Related Work

There has been substantial prior work on studying naming patterns and FQDNs to assess common router naming conventions [8, 31, 39], extract geographic tokens [9, 16, 28, 35], and investigate broadband access characteristics [21, 40]. To dissect FQDNs, most prior work has relied on regexes, *e.g.*, [8, 31]. To automate construction of regexes, Luckie *et al.* developed methods for supervised learning of regexes based on observed router naming patterns [24, 25]. In a similar vein, Aleph [35] uses LLMs to generate regexes to extract

location-related tokens. Namefill, in contrast, is unsupervised and focuses on more general naming conventions.

Other work has investigated Markov models for understanding naming patterns and generating names that are statistically likely to be in the DNS in order to study network infrastructure [7, 33, 38]. These methods are often word-list based, and while they are effective for common subdomains, *e.g.*, mail, www, smtp, they cannot enumerate FQDNs with embedded IP addresses or more general sequential patterns.

Although we are not aware of prior work to investigate what kinds of tools network operators use to create DNS records with names that follow a desired pattern or convention, the widely-used BIND software provides the \$GENERATE macro for populating zone files according to a given pattern [17]. The \$GENERATE directive iterates over a range of values, substituting the iterator into a name template; each substitution may be modified as $\{\text{offset}, \text{width}, \text{radix}\}$, where *offset* is added to the iterator value, *width* specifies zero padding, and *radix* selects the output base: decimal, octal, hexadecimal in lower or upper case, or nibble format in lower or upper case for ip6. arpa zones [17]. Namefill’s transforms cover decimal and hexadecimal together with width padding; we omit octal, which we have not observed in practice, and the nibble formats, which apply to IPv6. \$GENERATE syntax maps directly onto namefill’s DSL transformation functions, although a key scope difference is that namefill can embed all four bytes of an IPv4 address whereas \$GENERATE can only vary host bits.

Lastly, our work is related to prior work in program synthesis and *programming by example* (PBE) [6, 13, 14, 26, 29]. A key difference is that prior PBE work assumes all input/output pairs relate to a *single* function to be synthesized, whereas in our setting there are possibly *many* functions synthesized from the provided IP/FQDN examples. Finally, in namefill’s program synthesis, it uses the *minimum description length* principle (MDL) [12, 27] as part of ranking and choosing among synthesized programs.

3 The Namefill DSL

In this section we describe namefill’s domain-specific language (DSL), including syntax and semantics of the language. We also discuss how patterns represented by synthesized functions are generalized into naming conventions. We note that the DSL and the executable functions namefill synthesizes are complementary notions: the DSL defines the space of naming conventions namefill can represent, and a program in the DSL is an executable function that generates FQDNs.

3.1 DSL overview

Figure 1 shows the Backus-Naur Form syntax for the namefill DSL. A program in namefill is either an IPProgram or a SeqProgram. Each of these programs produces a list of *segments*, each of which is evaluated and concatenated to form an FQDN string. An IPProgram accepts the 4 bytes of an IPv4 address and is composed of one or more *segments* (Seg+), where each segment is either a literal string (Lit) or an encoded byte of the IPv4 address (Enc). A SeqProgram accepts $fqdn_i$ and is composed of zero or more segments, followed by a sequence segment (SeqSeg), followed by one or more segments, the result of which is $fqdn_{i+1}$, the next FQDN in some sequence. For

an IPProgram, the specific Transforms for segments that encode IP bytes mirror standard formatters for BIND's \$GENERATE macro [17], except that the \$GENERATE macro only varies the host bits of an IP address, whereas an IPProgram can embed any combination of the four bytes of an IPv4 address. For a SeqProgram, a SeqSeg is based on a given charset C , with carry semantics R , allowing a wide range of character-based sequences. Although the syntax permits *mixed* programs and for a SeqProgram to have more than one SeqSeg, synthesis does not generate any such programs for reasons we discuss in § 4.4. Thus, an IPProgram has at least one embedded IP byte in the output FQDN and no SeqSeg, and a sequential program has exactly one SeqSeg and no IP byte encoding segments.

3.2 Semantics

A namefill program denotes a function from an input to an FQDN string. For an IPProgram P with segments $[s_1, \dots, s_n]$ and input $ip = (a, b, c, d)$:

$$\begin{aligned} \text{eval}(P, ip) &= \text{concat}(\text{eval_seg}(s_1, ip), \dots, \text{eval_seg}(s_n, ip)) \\ \text{eval_seg}(\text{Lit}(v), ip) &= v \\ \text{eval_seg}(\text{Enc}(i, T), ip) &= T(ip[i]) \end{aligned}$$

For example, for the IP/FQDN pair (41.14.153.0, vc-41-14-153-0.ums.vodacom.co.za), a corresponding IPProgram segment list is: $[\text{Lit}(\text{"vc-"}), \text{Enc}(0, \text{Ident}), \text{Lit}(\text{"-"}), \text{Enc}(1, \text{Ident}), \text{Lit}(\text{"-"}), \text{Enc}(2, \text{Ident}), \text{Lit}(\text{"-"}), \text{Enc}(3, \text{Ident}), \text{Lit}(\text{"ums.vodacom.co.za"})]$. Note that an IPProgram will only produce valid FQDNs if used with addresses drawn from a prefix associated with the function.

A SeqProgram P has exactly one SeqSeg(C, R); the surrounding Lit segments form a fixed prefix p and suffix s . For an input FQDN f :

$$\text{eval}(P, f) = p + \text{succ}(f[|p| : |f| - |s|], C, R) + s$$

where $\text{succ}(m, C, R)$ advances the variable token m by one step within charset C under carry rule R . Thus for the sequence of FQDNs beginning node-cg0.pool-1-10.dynamic.nt-isp.net, node-cg1.pool-1-10.dynamic.nt-isp.net, ..., a corresponding SeqProgram segment list is $[\text{Lit}(\text{"node-"}), \text{SeqSeg}(\text{eDIGITS_LOWER}, \text{CARRY_ROLLOVER}), \text{Lit}(\text{"pool-1-10.dynamic.nt-isp.net"})]$.

In subsequent examples using the namefill DSL, we opt for a simplified representation that eliminates most syntax while preserving key elements of IP byte transforms, sequences, and literals. In this simplified representation, the two examples above are:

- `vc-|ip:0:Ident|-|ip:1:Ident|-|ip:2:Ident|-|ip:3:Ident|.ums.vodacom.co.za`
- `node-|seq(DIGITS_LOWER, CARRY_ROLLOVER)|.pool-1-10.dynamic.nt-isp.net`

3.3 Why a DSL?

Prior work on analyzing FQDNs has largely relied on regular expressions [8, 25, 31, 35]. A regular expression (regex) is a *recognizer*: it decides whether a string matches a pattern. A namefill program, in contrast, is a *transducer*: a function that maps an IPv4 address (or an FQDN) to an FQDN. A regex matching `\d+-\d+-\d+-\d+\.example\.com` cannot report which digit group corresponds to which byte of the address, nor what encoding, nor can it produce the name

for a given IP address. It is this generative capability that enables the misconfiguration analysis of § 8 and the record discovery application of § 9. A namefill program can be viewed as a regex that extracts the digits from a name, together with a *decode plan* that says how to interpret the extraction, *i.e.*, the byte index and transform attached to each segment. The translation between the two forms is one-way: compiling a namefill program into a regex that recognizes the names it generates is straightforward, but doing so discards the decode plan, and recovering a decode plan from a regex is not possible in general. Moreover, an $\text{Enc}(i, T)$ segment expresses a cross-position arithmetic constraint that regular expressions cannot: backreferences can require that a string recur within a name, but they cannot compute base conversion or padding. Lastly, we take the widespread use of \$GENERATE as evidence that the hypothesis space defined by the DSL reflects deployed practice: by mirroring its formatters, the space of programs that namefill can synthesize is grounded in a mechanism operators actually use to construct these names, rather than chosen arbitrarily.

3.4 From Programs and Patterns to Naming Conventions

Namefill programs, in effect, express *patterns* used by network operators for naming hosts, interfaces, and other resources. In the rest of this paper, we interchangeably refer to namefill *programs* and *patterns*. A given pattern may constitute a naming convention if it is the only pattern we identify for a given registered domain, but patterns may be assembled into more abstract naming conventions that represent broader operational practices within a provider. We define three types of naming conventions:

- **L1 (exact) conventions** consist of the same pattern (including constants) used across ≥ 2 routable prefixes for the same registered domain (effective TLD+1, or eTLD+1). For example, US Cellular (T-Mobile) uses identical IPProgram segments across 2,587 prefixes of the form `ip:3:Ident|-|ip:2:Ident|-|ip:1:Ident|-|ip:0:Ident|.mobile.uscc.net`. To identify L1 exact conventions, we simply compare program segments for all programs for a given registered domain.
- **L2 (structural) conventions** employ the same *structure* across ≥ 2 prefixes for the same registered domain (eTLD+1) but constant values may differ. For example, wanadoo.fr uses the same 3-byte IPProgram embedding structure but uses different per-pool prefix constants across 51K prefixes. For a given registered domain, we censor literals then compare segments to identify L2 naming conventions. Note that by definition, $L1 \subseteq L2$.
- For **singleton conventions**, the pattern is distinct for a given prefix and registered domain (eTLD+1). Singleton patterns are not captured by either an L1 or L2 pattern.

Moreover, we also define *cross-eTLD+1 conventions* as an extension of L2 patterns in which all literals are censored — including the domain suffix — and only the segment structure is compared. Analyzing these general conventions offers a way to assess how broadly certain structural conventions are used across the DNS.

4 Synthesis Methods

In this section, we describe how namefill synthesizes programs from a collection of IP/FQDN pairs.

<i>Program</i>	::=	IPProgram SeqProgram	
<i>IPProgram</i>	::=	$\lambda(a, b, c, d). \text{Seg}^+$	
<i>SeqProgram</i>	::=	$\lambda(\text{fqdn}). \text{Seg}^* \text{SeqSeg} \text{Seg}^+$	
<i>Seg</i>	::=	Lit(v); $v \in \Sigma^*$ Enc(i, T); $i \in 0,1,2,3, T \in \text{Transforms}$ SeqSeg(C, R)	(* literal string *) (* IPv4 byte transforms *) (* sequence segment *)
<i>Transforms</i>	::=	Ident PadL3 HexL HexL2 HexU HexU2	(* decimal; e.g., 42 $\rightarrow$ "42" *) (* zero-pad to 3; e.g., 42 $\rightarrow$ "042" *) (* lowercase hex; e.g., 42 $\rightarrow$ "2a" *) (* lowercase hex; 2-char padded, e.g., 10 $\rightarrow$ "0a" *) (* uppercase hex; e.g., 42 $\rightarrow$ "2A" *) (* uppercase hex; 2-char padded, e.g., 10 $\rightarrow$ "0A" *)
<i>C(charset)</i>	::=	DIGITS LOWER UPPER DIGITS_LOWER DIGITS_UPPER	(* [0-9] *) (* [a-z] *) (* [A-Z] *) (* [0-9a-z] *) (* [0-9A-Z] *)
<i>R(carry)</i>	::=	CARRY_ROLLOVER CARRY_NUMERIC CARRY_EXPAND	(* fixed width; rightmost char cycles, carry propagates left *) (* numeric successor; width expands when rightmost digit is '9' *) (* fixed-charset expand; width grows when all chars at max *)

Figure 1: Namefill domain-specific language BNF syntax.

4.1 IPv4 Embedding Synthesis

For synthesizing embedded IP functions we take a deductive approach: for each IP byte and each of the 6 transforms, we find all positions in the FQDN where the transformed value occurs. We then censor matching characters and recurse on the remaining bytes in the address. Then, we generate Lit segments for uncensored substrings. For example, given the input pair (109.137.153.78, 01-078.b2c.rapirtelecom.ru) we observe matches for two byte transformations when iteration gets to the least significant byte: Enc(3, Ident) and Enc(3, PadL3)¹. Since the least significant byte of the IP address is the only one that occurs in the FQDN, the resulting programs are:

- 01-|ip:3:PadL3|.b2c.rapirtelecom.ru
- 01-0|ip:3:Ident|.b2c.rapirtelecom.ru

We then count for each program P and each (IP,FQDN) pair (ip, f), $\text{eval}(P, ip) == f$. That is, we identify the set of IP/FQDN pairs for which a program produces the FQDN, given the IP address as input. If the number of matching pairs is above a threshold value (which depends on the number of bytes in the IP address matched), we rank the generated programs, select the *best* one according to that ranking, remove all matching IP/FQDN pairs from further consideration, and continue synthesis. Note that because a set of IP/FQDN pairs is verified to work for a given function (i.e., applying the function with the IP address as input produces the expected FQDN), training set precision is 100% by construction. That does not mean, however, that a synthesized program correctly captures the *semantics* of a given naming convention.

The program ranking mechanism we use is based on the *minimum description length* (MDL) principle [27]. In this context,

¹Note that if we were to recurse, the censored FQDN would be 01-0**,b2c.rapirtelecom.ru when applying the transform Enc(3, Ident), or 01-***.b2c.rapirtelecom.ru when applying the transform Enc(3, PadL3). In other words, we simply replace the matching characters with asterisks.

the MDL principle implies that programs that minimize literal content (i.e., minimize characters not generated by IP byte transforms) should be preferred. In practice, we take a hybrid ranking approach because of some realities observed in naming conventions. A motivating example is the IP/FQDN pair 70.50.8.1, bras-base-mtrlpq0807w-grc-15-70-50-8-1.dsl.bell.ca. Note that the IP address is clearly embedded in the FQDN, but that the 08 in the CLI identifier coincidentally matches one of the IP bytes with hex (padded to two characters) encoding. Unfortunately, HexL2 or HexU2 encoding minimizes literal content but has the wrong semantics. Thus, we adopt the following ranking method:

- (1) Hybrid override: A program with consistent literal separators between IP bytes, at least as many IP bytes, and a smaller span (the bytes are packed more tightly in the FQDN) is preferred over the MDL winner, but only when the other program does not have consistent separators;
- (2) MDL principle: minimize literal content; more of FQDN explained by IP bytes is preferred;
- (3) Prefer segments with consistent literals (separators) between IP byte transforms;
- (4) Prefer segments with fewer distinct transform functions.

The hybrid override ranking step handles correct ranking of programs resulting from naming conventions like the bell.ca example above, as well as 16K other similar conventions, with the MDL criterion addressing the vast majority of other ranking problems. We do not claim that namefill programs perfectly capture all naming convention semantics, but as we discuss in § 10, we estimate that the fraction of such semantically incorrect programs is very small ($\sim 0.1\%$).

Lastly, programs are tagged as *high* confidence if the number of bytes matched is ≥ 3 and the number of training examples is

above a configurable adaptive threshold (two examples for four-byte matches up to four for single-byte matches). Other programs are tagged as *neutral* confidence.

4.2 Sequential Synthesis

For synthesizing sequential patterns, we first lexicographically sort FQDNs based on $(len(FQDN), FQDN)$, so that "9" comes before "10". We then take a two-pass approach:

- First, for $i \in 0..n - 1$, where n is the number of sorted IP/FQDN pairs, we identify the common prefix and suffix between f_i and f_{i+1} , such that each FQDN f can be represented as *prefix|token|suffix*. We then identify whether there is a common incremental pattern in *token* between ip_i and ip_{i+1} and f_i and f_{i+1} . If this incremental pattern continues for a configurable threshold number of IP/FQDN pairs, we synthesize a function to capture this sequence, marking it with *high* confidence.
- Second, we identify any sequential pattern in *token* for consecutive f_i and f_{i+1} , ignoring the related IP addresses. If such a pattern exists, we synthesize a program with either *neutral* or *low* confidence, as discussed next.

We do not strictly require that every token in a sequence be represented in a training set of FQDNs. It is sufficient to observe some *density* of tokens within a range. The required density is a configurable value and defaults to 0.2, meaning that 20% of tokens within a sequence must have been present in a set of FQDNs to infer a sequential pattern; § 4.5 shows synthesis is not sensitive to this choice. We tag sequences for which tokens increment but IP addresses do not as *neutral* if they are found to be present in a dense arrangement and as *low* confidence otherwise.

For each type of pattern, we infer the character set across all FQDNs identified to be part of a sequence, classifying the charset as DIGITS, LOWER, UPPER, DIGITS_LOWER, or DIGITS_UPPER. We then infer the carry rule: for DIGITS with variable-length tokens, we infer CARRY_NUMERIC and for fixed-length tokens we infer CARRY_ROLLOVER. For LOWER/UPPER with fixed-length tokens we infer CARRY_ROLLOVER, and otherwise we infer CARRY_EXPAND.

4.3 Synthesis Pipeline

Namefill (about 1.5K lines of Python) takes as input a series of IP/FQDN pairs derived from DNS PTR or A records. For processing large datasets, we take a divide-and-conquer approach by separating input into files based on routable prefix (we use CAIDA's Prefix to AS (pfx2as) mapping data [3]), then process each file separately, in parallel.

For a given input file, IP/FQDN pairs are grouped by registered domain (effective TLD+1). We then synthesize embedded IP functions, pruning IP-matched pairs from high-confidence patterns from input data, then synthesizing sequential FQDN programs. These functions are deduplicated if necessary and written to a SQLite database, along with the L1 (exact) and L2 (structural) naming conventions identified across all synthesized functions.

4.4 Scope of Synthesis

As noted in § 3, the DSL permits *mixed* programs and multiple SeqSegs, but synthesis does not currently generate either. These

restrictions follow from the staging of the pipeline rather than from the language. Embedded IP synthesis is *deductive*: the address supplies the exact strings to search for within the FQDN. Sequential synthesis is *inductive* over lexicographically adjacent names, with no such anchor. A mixed program thus requires a joint search in which the two hypotheses compete for the same characters, *e.g.*, given host-10-3 within a prefix containing the byte 10, it is ambiguous whether 10 is an embedded address byte or a sequence index; the MDL criterion alone does not resolve this ambiguity. Similarly, two independently varying tokens are not identifiable from a linear sort: the two-pass construction isolates a single varying window by construction, and a name such as a1-b2 is consistent both with two sequences and with one sequence over a composite alphabet; handling such cases requires a different formulation, *e.g.*, holding one token fixed while varying the other. Both are non-trivial extensions that we leave to future work.

4.5 Validation

Because each candidate function is verified against the IP/FQDN pairs it is claimed to explain before being emitted, training set precision is 100% by construction. As noted above, this does not guarantee that a program captures the intended *semantics* of a naming convention. To estimate how often it does not, we took two approaches: we scanned all synthesized functions using an approximate regular expression that flags hex byte transforms matching letters within domain-name words, and we manually examined a random sample of 10K functions. Together, these yield an estimate that ~0.1% of the functions synthesized for the 2026-03 rDNS data encode a byte transform that coincidentally matches characters unrelated to the address; we return to this issue in § 10. Synthesis is also insensitive to the sequence density parameter: varying it from 10–30% changes the number of low-confidence patterns by < 8% and FQDN coverage at higher confidence levels by < 9%. Lastly, we characterize the FQDNs for which namefill synthesizes no function at all in § 6, and show that the large majority are inherently opaque rather than missed by our methods.

5 Datasets and Data Processing

In this section we describe the datasets used in this paper, as well as synthesis processing time.

5.1 Datasets

We used two sets of reverse DNS (rDNS, *i.e.*, PTR record) datasets supplied by the OpenINTEL project [37], collected about a year apart (2025-03 and 2026-03). Each of these datasets is derived from a complete scan of PTR records from the globally routable IPv4 address space. We also used a set of forward DNS (fDNS, *i.e.*, A record) data supplied by Rapid7 Labs [20], collected in March 2025. Lastly, we used the IPv4 DNS names included with CAIDA's ITDK (2026-03) [2]; these are PTR-derived hostnames for interface addresses observed in TTL-expired responses to Ark traceroutes, and are thus themselves reverse DNS data, distinguished from the OpenINTEL rDNS datasets by covering router interfaces rather than the address space at large. Summary statistics for each of these datasets are shown in Table 1. For processing each dataset, we used a contemporaneously collected pfx2as dataset [3] to split datasets by prefix for

processing. For IP addresses with no explicit prefix in the pfx2as dataset, we used a /24 aggregate. Throughout, we group names by *registered domain* (eTLD+1), which we determine using the Mozilla Public Suffix List [10]. We note that a small number of providers embed one or more IP address bytes within the eTLD+1 itself.

5.2 Data processing

We ran namefill on each set of IP/FQDN pairs derived from PTR records or A records from datasets described above. Namefill completed synthesis for each dataset within a few hours; the largest dataset (fDNS 2025-03) took about 5 hours to complete. For processing each dataset, synthesis was parallelized with one worker process per available CPU core, and SQLite writes were serialized in the main process to avoid lock contention. The evaluation machine has 16 cores (AMD Ryzen AI Max+ 395 at 5.2 GHz) with 128 GiB DRAM and two SanDisk 4 TiB SN850X SSDs for storage. For the OpenINTEL 2026-03 PTR record dataset (1.2B records spanning 818,611 IP prefixes), total wall-clock time to run namefill over the entire dataset was 3h23min. Processing time is heavily right-skewed, with 70% of prefixes completed within the first hour at a throughput of approximately 9,500 prefixes per minute. Synthesis cost for a given prefix grows with the number of IP/FQDN pairs it contains, since each candidate program is evaluated against every pair in the group. As a result, many of the prefixes completed during the final hour were larger aggregates (/8–/20 prefixes), some with millions of IP/FQDN pairs.

6 PTR Naming Conventions

In this section, we describe naming conventions in the DNS, focusing on FQDNs present in the OpenINTEL 2026-03 PTR record dataset. We choose to frame our discussion around names in PTR records because much prior work on evaluating geotokens and other structure in FQDNs has focused explicitly or implicitly on PTR records (e.g., [9, 16, 21, 25, 31, 33, 35, 36, 39, 40]). We then discuss FQDNs in PTR records for which namefill does not identify a naming convention, followed by a comparison of two PTR record datasets collected one year apart. We finish this section by contrasting naming conventions observed in PTR and A records.

6.1 The PTR Naming Convention Landscape

We begin our description of PTR record naming conventions by examining the number of naming conventions of different types (cf. § 3.4) in the OpenINTEL 2026-03 PTR record dataset in Table 2. We observe in the table that a majority of patterns are aggregated into L1 or L2 naming conventions and that the number of L2-only naming conventions is roughly double that of L1. These findings generally hold for all the datasets we analyzed. Note that although the number of naming conventions in the singleton class is large (235K out of 313K), that number is dwarfed by the number of patterns aggregated into the L1 and L2 naming convention classes.

To ground further analysis of distributional characteristics of naming conventions, we next look at the *content* of the most common L1 and L2 naming conventions. We show in Table 3 the top 5 L1 exact conventions sorted by member pattern count. As an example of a generated name, the akamaitechnologies.com convention produces

a23-45-64-170.deploy.static.akamaitechnologies.com for the address 23.45.64.170. We first observe that the top L1 naming conventions are dominated by embedded IP patterns, and have a mix of both network service providers (bbtec.net, spectrum.com, prod-infinity.com.mx) and cloud/CDN providers (akamaitechnologies.com, hwclouds-dns.com). In fact, the top 100 L1 naming conventions are all embedded IP patterns and are dominated by residential/broadband ISPs (~70%), mobile providers (~10%), and hosting/datacenter and other cloud providers (~15%). (We classified providers manually by inspecting the registered domain and, where necessary, the operator’s website; these proportions are approximate.) The first sequential convention appears at rank 228 for entelchile.net (z|seq:DIGITS:CARRY_ROLLOVER|.entelchile.net) across 246 prefixes and 66.9K FQDNs.

For the top L2-only structural conventions shown in Table 4, we include counts of prefixes, patterns, and FQDNs since these conventions may span any number of prefixes and may be composed of any number of patterns that have a common structure. The top 5 L2-only conventions are entirely composed of cloud and network access providers, and, like L1, all of the top conventions embed IP addresses. Similar to the L1 conventions, the top 100 L2-only conventions are dominated by wireline and mobile access providers (~85%) along with cloud/CDN/hosting providers (~10%). Only one of the top 100 L2-only conventions uses a sequential pattern, which is Virgin Media (literal|seq:DIGITS:CARRY_NUMERIC|literal|.virginm.net). An example pattern that collapses into Virgin Media’s structural template is cpc89770-stok19-2-0-custN.1-4.cable.virginm.net for N from 1 to 16,381. The literal portions of this template vary within 9.9K patterns across 124 prefixes.

For the singleton class of naming conventions (examples not shown), they are dominated by regional wireline providers (~80%). Mobile access providers are much less prevalent, and global cloud, CDN, and hosting providers largely disappear because their multi-prefix footprints produce L1/exact or L2/structural conventions instead. Academic entities, governments, and business enterprises make up ~15% of singletons since they are commonly aligned on one prefix.

We now examine distributional characteristics of naming conventions. Figure 2 shows, for the OpenINTEL 2026-03 PTR record data, the cumulative fraction of FQDNs covered by the top- k registered domains sorted by total FQDNs descending (left), and the cumulative fraction of FQDNs covered by the top- k naming conventions (right), regardless of specific convention type (L1, L2, or sequential). In the left plot of Figure 2 we see that a small number of registered domains account for the bulk of FQDNs covered in the PTR record data. For example, 200 registered domains cover ~60% of all FQDNs. The right-hand plot illustrates the asymmetry between IP-embedding conventions which tend to have high coverage of FQDNs within and across prefixes, and sequential conventions, which have much lower coverage of FQDNs with median sequences of length 5 and average length of ~50. This asymmetry helps to explain why embedded IP patterns cover 90% of matched FQDNs despite being only 64% of the pattern count.

As a complement to the left-hand plot of Figure 2, we show in Table 5 the number of conventions per registered domain (eTLD+1), broken down into different classes of numbers of conventions. We

Table 1: Summary statistics for each dataset.

	rDNS 2025-03	rDNS 2026-03	fDNS 2025-03	ITDK (rDNS) 2026-03
Records	1,257,047,190	1,248,500,380	2,386,646,445	2,255,207
Prefixes	798,678	818,611	618,646	50,387
Registered domains	263,852	254,311	716,697	7,064

Table 2: Numbers of different types of naming conventions used in the OpenINTEL 2026-03 PTR record data.

	All	IP	Seq.
Total patterns	1,679,813	1,074,057	605,756
Total conventions	313,177	161,675	151,502
Exact (L1)	26,678	21,342	5,336
Structural (L2)	78,112	42,551	35,561
L2-only (not in L1)	51,434	21,209	30,225
Singleton (not in L2)	235,065	119,124	115,941
Patterns in L1 (%)	42.3	60.3	10.4
Patterns in L1 or L2 (%)	86.0	88.9	80.9
Patterns in singletons (%)	14.0	11.1	19.1

see that the vast majority of eTLD+1 use 5 or fewer naming conventions, with a single convention being the most common. We also observe $\sim 2.1\text{K}$ eTLD+1 using 6 or more. The largest number of conventions (112) is used by `ocn.ne.jp`, with `wanadoo.fr` (62), and `att.net` (32) rounding out the top 3.

Figure 3 shows the cumulative fraction of FQDNs covered by the top- k , separated by embedded IP programs (left) and sequential programs (right). Each curve is normalized by the total number of FQDNs for each program type. Note that there are separate curves in each plot for *all* L2 naming conventions and for L2-only conventions (*i.e.*, L2-only + L1 $\rightarrow$ L2). These plots show how much of the FQDN population is accounted for by the most widely used naming conventions. We see quite clearly that structural L2 conventions cover the majority of FQDNs for both embedded IP (77.9%) and sequential patterns (89.5%), that the overall coverage of singleton patterns is small (3.2% and 6.3%), and that exact L1 conventions are much more prevalent in the embedded IP patterns than for sequential patterns. The specific convention types among these convention-diverse domains are roughly split 80% embedded IP to 20% sequential, reflecting the fact that many of these domains have large and diverse network footprints.

Sequential conventions nearly always couple a literal service- or role-identifying token with a counter. By pattern count, mail and DNS infrastructure tokens dominate (*e.g.*, `mx`, `mail`, `ns`, `mta`, and `smtp`: 54K patterns); by FQDN count, large subscriber pools do (*e.g.*, `cust`: 5.8M names, `p`: 5.1M, `node`: 1.0M). Equipment identifiers also appear: 1,608 patterns with the token `vlan` cover 34.2K names. Sequences are typically dense: measuring occupancy as matched names divided by the span between the smallest and largest counters, the median pattern is fully populated (mean 0.88; 0.98 for high-confidence patterns, reflecting the density criterion of § 4), with equipment sequences the sparse exception — the `vlan` conventions occupy 10–19% of their observed VLAN ID spans. Dense

counters on service and subscriber-pool tokens are precisely the names that lend themselves to prediction (§ 9).

Lastly, we show in Table 6 the top 5 cross-registered domain conventions. For these conventions, we consider *only* the structure (like L2), but ignore the eTLD+1. We first observe that two of the top 5 are sequential patterns; the top convention uses a fixed-width numeric sequence, while the 5th-ranked convention uses a variable-width numeric sequence. We also observe that $\sim 60\%$ of all FQDNs are covered by the top 5 (out of 283) cross-domain conventions. The small number of distinct cross-domain structures (283) indicates that while naming conventions vary in literal values, the underlying structural vocabulary is remarkably limited.

6.2 FQDNs Uncovered by Namefill-Synthesized Conventions

Across the OpenINTEL 2026-03 PTR record dataset as a whole, namefill synthesizes functions covering 92.0% of FQDNs. In this section we briefly discuss the set of FQDNs that are *not covered* by namefill-synthesized conventions, *i.e.*, the set of FQDNs that do not exhibit the kinds of IP embeddings or sequential patterns addressed by namefill’s DSL. We classify these FQDNs into three categories: those that are *inherently opaque* with respect to a naming convention, those that are not covered by a namefill pattern because of too few training examples (*threshold-limited*), and those that use an IP encoding that goes beyond namefill’s DSL. We manually verified a sample of 1,000 names from each category. of Table 7 shows counts of FQDNs for two of the datasets we consider in this paper for each of these three categories.

For inherently opaque FQDNs, we observe names with no pattern (*e.g.*, `static.masmovil.com`), names with hash values (*e.g.*, `s0106889e6828fec3.vc.shawcable.net`), names with UUIDs (*e.g.*, `9b4509b2-1d6e-4b01-8a8c-9285c54a5f20.clouding.host`), and other names that are not amenable to namefill’s DSL. The only IP byte encodings that we have observed that are not covered by namefill’s DSL are those with Roman numeral encodings, *e.g.*, `pei-186-0-xcvi-ccxlv.une.net.co` which is mapped to 186.0.96.245. As shown in the table, these types of encodings are exceedingly rare.

The thresholds used in namefill to identify whether there are sufficient examples to constitute a pattern represent a tradeoff between covering more FQDNs ingested and synthesizing a broader set of functions, and not wanting to generate functions for phantom naming conventions. In particular, the thresholds for embedded IP patterns vary based on the number of bytes matched, with two FQDN examples sufficient to synthesize a function that matches all four IP bytes, and up to four examples required to synthesize a function that only matches a single IP byte. With sequential patterns, we require three examples in order to synthesize a function. In experiments using lower thresholds, we found through manual

Table 3: Top L1 uniform naming conventions by pattern/function count for the OpenINTEL 2026-03 PTR record data. Note that the number of prefixes is equivalent to the number of constituent patterns.

Registered domain	Type	Prefixes / FQDNs	Convention template
bbtec.net	IP	20K / 38M	softbank ip:0:PadL3 ip:1:PadL3 ip:2:PadL3 ip:3:PadL3 .bbtec.net
spectrum.com	IP	11K / 42M	syn- ip:0:PadL3 - ip:1:PadL3 - ip:2:PadL3 - ip:3:PadL3 .res.spectrum.com
akamaitechnologies.com	IP	8.6K / 13M	a ip:0:Ident - ip:1:Ident - ip:2:Ident - ip:3:Ident .deploy.static.akamaitechnologies.com
prod-infinitum.com.mx	IP	8.4K / 6.6M	dsl- ip:3:Ident - ip:2:Ident - ip:1:Ident - ip:0:Ident -dynamic.prod-infinitum.com.mx
hwclouds-dns.com	IP	7.0K / 4.0M	ecs- ip:0:Ident - ip:1:Ident - ip:2:Ident - ip:3:Ident .compute.hwclouds-dns.com

Table 4: Top L2-only structural naming conventions by FQDN count for the OpenINTEL 2026-03 PTR record data.

Registered Domain	Type	Prefixes / Patterns / FQDNs	Convention template
amazonaws.com	IP	901 / 906 / 71M	literal ip:0:Ident literal ip:1:Ident literal ip:2:Ident literal ip:3:Ident .amazonaws.com
spectrum.com	IP	20K / 21K / 55M	literal ip:0:PadL3 literal ip:1:PadL3 literal ip:2:PadL3 literal ip:3:PadL3 .spectrum.com
bbtec.net	IP	20K / 20K / 38M	literal ip:0:PadL3 ip:1:PadL3 ip:2:PadL3 ip:3:PadL3 .bbtec.net
comcast.net	IP	99 / 597 / 35M	literal ip:0:Ident literal ip:1:Ident literal ip:2:Ident literal ip:3:Ident .comcast.net
myvzw.com	IP	2.3K / 2.3K / 26M	ip:3:Ident literal ip:0:Ident literal ip:1:Ident literal ip:2:Ident .myvzw.com

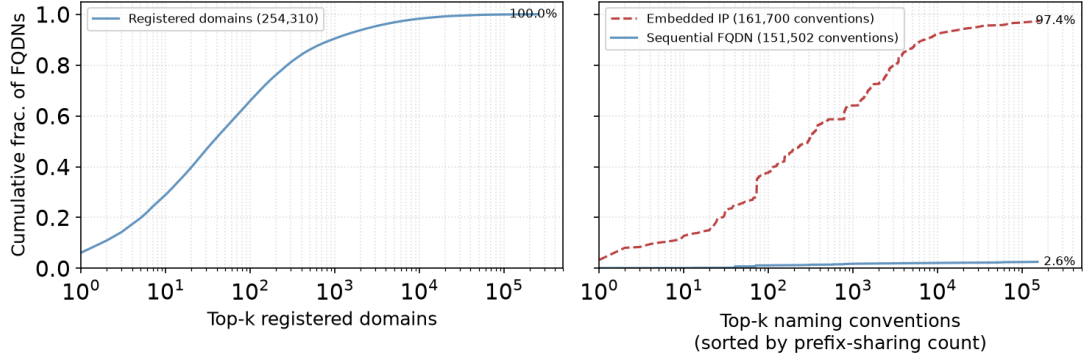
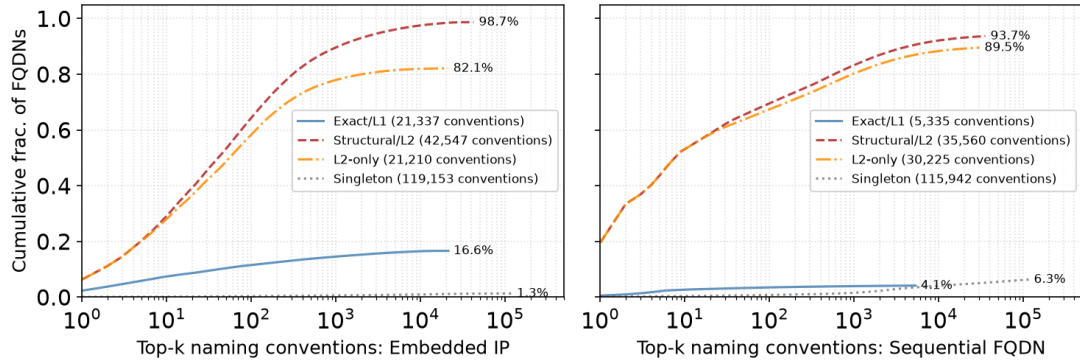
**Figure 2: Cumulative fraction of FQDNs in the OpenINTEL 2026-03 PTR record data for the top-k registered domains (left) and the cumulative fraction of FQDNs for the top-k naming conventions, sorted by prefix-sharing count (highest to lowest). Note the log scale on the x-axis.****Figure 3: Cumulative fraction of FQDNs in the OpenINTEL 2026-03 PTR record data by the top-k naming conventions, split into embedded IP conventions (left) and sequential conventions (right). Note the log scale on the x-axis.**

Table 5: The number of naming conventions used within a registered domain (eTLD+1) for the OpenINTEL 2026-03 PTR record data.

Conventions	eTLD+1	Conventions	eTLD+1
1	223,257	6–20	2,074
2–5	28,931	20+	49

checking that a majority of the additional functions synthesized did not appear to be semantically valid.

6.3 Convention Persistence and Churn

In this section, we briefly compare findings from synthesizing naming conventions from the OpenINTEL 2025-03 and 2026-03 PTR record datasets. Specifically, Table 8 shows the year-over-year changes in structural (L1+L2) and singleton naming conventions. We see that in aggregate, ~88% of naming conventions persist across these datasets (as weighted by FQDN coverage), with slightly more retired conventions than added ones. We also found that embedded IP conventions are slightly more stable than sequential conventions (91.5% vs 89.7%), but the differences are relatively small.

Upon digging into specific registered domains that see the biggest differences across the two datasets, we found some operators doing apparent consolidation of naming conventions, as well as some acquisitions, rebranding, and some entire withdrawals of naming conventions. For example, there are 130 embedded IP naming conventions across 130K FQDNs associated with `wifi4all.it` in the 2025-03 data which map directly to the registered domain `4all.it` in the 2026-03 data (same number of naming conventions, but the number of FQDNs grew to 133K). `wifi4all.it` was rebranded to `4all.it` in 2019[4], but the PTR records were apparently not updated until the time period between March 2025 and March 2026. Similarly, for `frontiernet.net`, we observe an increase in 286 L1 exact conventions (comprised of 1,404 embedded IP patterns) between these two datasets which is consistent with its growth in US rural fiber deployments [11]. Overall, examining changes in naming conventions over time could yield insights into how network infrastructure evolves; we plan to investigate this in future work.

6.4 Forward DNS Conventions: Similarities and Divergences

Lastly, we contrast the naming conventions observed through PTR (reverse DNS) records with those observed through A (forward DNS) records. For this analysis, we compare naming conventions synthesized by namefill in the OpenINTEL 2025-03 PTR record data with those synthesized in the Rapid7 2025-03 A record data. Specifically, we compare naming convention coverage and pattern types across prefixes. Overall, we find that there are 162K prefixes for which there are naming conventions synthesized in both rDNS and fDNS. There are an additional 188K prefixes in rDNS that have naming conventions associated with them (but none in the same prefixes for fDNS), and 339K prefixes in fDNS that have naming conventions associated with them (but none in the same prefixes for rDNS), and 102K prefixes for which there are no naming conventions synthesized at all.

We show in Table 9 how the 162K prefixes in common between rDNS and fDNS align or diverge depending on the majority of naming convention type (embedded IP or sequential). First, we see that ~90% of prefixes see consistency in the predominant naming convention strategies used in both PTR and A records.

Of the remaining fraction, 8.4% of prefixes have naming conventions in which PTR records are dominated by IP literals, but the forward records are dominated by sequential names. This is likely because rDNS naming conventions often use embedded IP conventions, *e.g.*, via BIND \$GENERATE, while forward zones use human-readable sequential names, and because the PTR zone may be controlled by a different organization (*e.g.*, the address space owner) than the one registering A records.

For the remaining portion, in which sequential conventions dominate PTR records while embedded IP conventions dominate A records, the likely causes mirror the above: different conventions for consumer PTR space versus business-tier A records, or resellers adding embedded IP names to the forward DNS.

7 Coverage Comparison with Related Tools

Prior work on PTR hostname analysis has largely focused on extracting geographic information. In this section, we show that the types of naming conventions that namefill captures are largely orthogonal to the kinds of structured geographic information exploited by Hoiho [25] and Aleph [35], each of which relies on regexes for matching geotokens.

7.1 Methodology

For this analysis, we use the IPv4 DNS names from a recent ITDK (2026-03) [2]. We run namefill on this data using a contemporaneously collected pfx2as dataset from CAIDA [3] to facilitate data processing. For Hoiho, we use the regular expressions included in the ITDK, and we additionally use the Hoiho API [15] to query names. We take the union of the names matching regexes in the ITDK and those for which we receive successful query responses to obtain the set of Hoiho-matched names. For Aleph, we use the v1 API and use the set of names for which we receive successful query responses as the set of Aleph-matched names [34]. We note that namefill synthesis on the ITDK 2026-03 dataset takes 61 seconds. From 2,255,207 input IP/FQDN pairs, namefill synthesizes 8,171 total naming conventions (1,917 L1/Exact, 1,409 L2/Structural-only, and 4,845 singletons).

7.2 Coverage and Complementarity

Table 10 compares namefill, Hoiho, and Aleph on coverage of the ITDK 2026-03 dataset as well as on other dimensions. We first observe in the table that namefill covers 67.7% of the IP/FQDN pairs in the data, compared with 12.7% of pairs for Hoiho and 4.2% for Aleph. We note, however, that these numbers are not directly comparable and are not intended as a ranking, since they measure different properties: for namefill, that a synthesized function reproduces a name exactly given the associated IP address; for Hoiho and Aleph, that a geographic token can be extracted. Names in the ITDK are most commonly router interfaces, which are the target of both Hoiho and Aleph. These tools are built for different purposes, and a more informative result is how little their coverage overlaps.

Table 6: Top 5 Cross-Registered Domain Conventions in the OpenINTEL 2026-03 PTR record data, sorted by number of registered domains sharing the convention.

Type	Patterns	eTLD+1's	FQDNs	Prefixes	Convention template
Seq.	394K	28K	15M	87K	literal seq:DIGITS:CARRY_ROLLOVER domain
IP	254K	11K	276M	208K	ip:0:Ident literal ip:1:Ident literal ip:2:Ident literal ip:3:Ident domain
IP	79K	8.5K	10M	24K	literal ip:3:Ident domain
IP	220K	7.2K	365M	158K	literal ip:0:Ident literal ip:1:Ident literal ip:2:Ident literal ip:3:Ident domain
Seq.	61K	5.0K	11M	15K	literal seq:DIGITS:CARRY_NUMERIC domain

Table 7: Taxonomy of names that are not covered by namefill synthesized functions.

Root Cause	Count (%)	
	rDNS 2026-03	ITDK 2026-03
Inherently opaque	89,975,502 (90.4%)	699,128 (96.0%)
Threshold limited	9,560,414 (9.6%)	29,301 (4.0%)
DSL gap	44,004 (0.0%)	121 (0.0%)

Table 8: Changes in structural (L1+L2) and singleton naming conventions in year-over-year PTR record data (2025-03 and 2026-03).

	Old	New	Persistent	Retired	Added
L1+L2	78,525	78,112	71,182	7,343	6,930
Singleton	245,645	235,065	212,892	32,753	22,173
Combined	324,170	313,177	284,074	40,096	29,103

Table 9: Comparison of naming convention types between fDNS and rDNS (2025-03).

Agreement	Count	Pct
Both IP-dominant (align)	120,516	74.3%
Both Seq.-dominant (align)	25,856	15.9%
rDNS=IP, fDNS=Seq. (diverge)	13,619	8.4%
rDNS=Seq., fDNS=IP (diverge)	2,061	1.3%

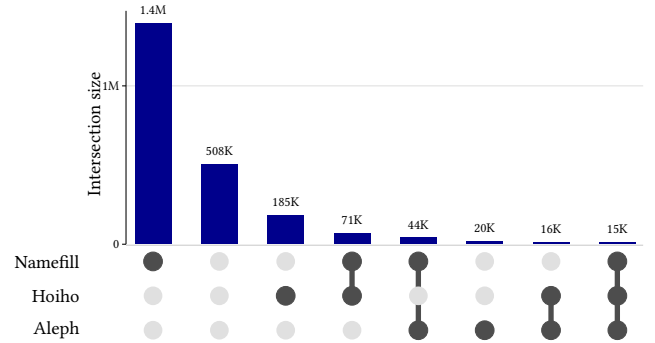
For those FQDNs that are not captured in functions synthesized by namefill, we see in Table 7 that the majority are classified as opaque, at least with respect to namefill’s DSL. We also see in Table 7 that 29K FQDNs are not covered because there are too few examples of either a sequential pattern or an embedded IP pattern. Lastly, we observe a small number of names for which namefill does not synthesize functions due to a DSL gap (all Roman numeral encodings).

The coverage figures also reflect differing input requirements and thresholds. Namefill requires only IP/FQDN pairs, and synthesizes a function once an adaptive threshold of examples within a prefix is met: two examples suffice for a function matching all four address bytes, up to four for a function matching a single byte. Hoiho learns geolocation regexes using router alias ground truth, dictionaries of geographic hints, and RTT measurements from vantage points at known locations, and accepts a naming convention

only when it extracts at least three unique RTT-consistent geohints from training data [25]; Aleph requires access to a commercial LLM API. Namefill’s comparatively permissive thresholds contribute to its coverage; § 6 discusses the tradeoff between lowering them further and inferring invalid naming conventions.

In Figure 4 we further break down differences in coverage of the FQDNs in the ITDK data. We see in the figure that, perhaps surprisingly, the *smallest* category is the set of names covered by all three tools (0.7% of the total), indicating that these tools have quite different coverage of the set of FQDNs and are thus complementary to one another. From Table 10 and Figure 4, we also see that the second largest category is the set of names uncovered by any tool (22.5%), suggesting that this set of names may not have any geographic or operational structure that can be reliably extracted.

Beyond comparing coverage, we observe that namefill’s functions can directly assist tools like Hoiho and Aleph. Because namefill establishes with high confidence that a particular substring of a name encodes a byte of the associated IP address, it can indicate which characters of a name to *exclude* when inferring geographic tokens, ASNs, or router names. Excluding such characters both avoids inference of phantom naming conventions and reduces the search space for the suffixes that dominate runtime in these tools.

**Figure 4: UpSet plot showing intersection coverage of namefill, Hoiho, and Aleph across 2,255,207 IP/FQDN pairs in ITDK 2026-03.**

8 Misconfiguration Detection

In this section, we investigate embedded IP functions synthesized by namefill as the basis for identifying misconfigurations and inconsistencies in PTR records. The underlying assumption in our

Table 10: Tool comparison on ITDK 2026-03 data (2,255,207 total IP/FQDN pairs).

Metric	namefill	Hoiho [25]	Aleph [35]
IP/FQDN pairs covered	1,526,657 (67.7%)	286,268 (12.7%)	95,265 (4.2%)
Type of structure	Naming convention functions (IP, Seq.)	Regexes to extract geotokens/location	Location
Supervision/input required (besides PTR rec.)	None	Ark alias ground truth; RTT from VPs at known locations	LLM API (\$\$)
Generative? (new names)	Yes	No	No
Unique coverage	1,396,397 (61.9%)	184,855 (8.2%)	20,145 (0.9%)

analysis is that a function synthesized from the majority of names within a prefix reflects the naming convention the operator intended to deploy. We do not claim to recover intent directly; rather, we treat the majority function as the convention for a prefix and examine names that deviate from it. The operator confirmations reported below serve as evidence to support this assumption.

8.1 Detection Heuristics

PTR records are often created through automated techniques such as BIND’s \$GENERATE macro or other scripts, which means that errors or inconsistencies introduced in using the macro or script are then propagated across a prefix or multiple prefixes. We investigate two classes of anomalies in PTR record names: (1) names for which a byte in the zone template was not updated when the template was copied to a new prefix; and (2) inconsistent conventions regarding separator characters (. vs -) between IP bytes.

For class (1), we explicitly separate cases in which we identify three dynamic bytes, and one wrong byte that differs by ± 2 from the expected value (which we term, for simplicity, “off-by-tiny”), and cases in which a wrong byte is off by any plausible value (“off-by-any”). We expect these situations to be the result of copy/pasting the template for an adjacent prefix and failing to update one of the octets, or as a result of a typo in template construction.

For class (2), we note that these are inconsistencies in naming conventions rather than misconfigurations, and may result from different engineers updating macros and templates with different preferences, habits, or understanding of conventions. We choose L2 naming conventions with at least 50 member functions, and where a minority ($\leq 25\%$) of functions differ by separator characters (- vs .) between embedded IP bytes with the majority.

8.2 Results and Validation

Table 11 summarizes our findings. We analyzed functions for the OpenINTEL rDNS 2026-03 data and identified 587 instances of off-by-tiny and 404 instances of off-by-any, manually inspecting all of these, and 1,639 instances of inconsistent byte separators. For each of these, we used the generative nature of the functions to perform PTR record lookups for each of the names in question and additionally performed A record lookups to determine whether the forward DNS zone showed differences. For off-by-* issues we also used CAIDA’s pfx2as data to compare the autonomous system of the (correct, configured) IP address with the implied incorrect address embedded in the name. We also reached out to operators

using the email address in the appropriate SOA record to verify (or refute) the misconfiguration. Examples of problems are as follows:

- For prefix 181.62.100.0/24, we observed names like `dynamic-ip-181611000.cable.net.co` (for address 181.62.100.0). Notice that 62 is the 2nd byte of the address but within the name it is 61. There were 155 prefixes affected for this provider, for a total of 65,688 PTR records.
- For prefix 190.205.56.0/21, we observed 1,000 FQDNs using hyphens between IP bytes, *i.e.*, `190-205-58-222.chc-00.rai.cantv.net`, but 42 using a dot between the 3rd and 4th bytes, *i.e.*, `190-205-57.99.chc-00.rai.cantv.net`.

Additionally, we performed the same misconfiguration/inconsistency detection on the OpenINTEL rDNS 2025-03 data and identified 572 instances of off-by-tiny, 417 instances of off-by-any, and 1,622 instances of inconsistent separators. We compared these findings year-over-year to identify which anomalies in 2025 changed in the 2026 data.

We also see in the table that there are similar numbers of off-by-tiny and off-by-any misconfigurations in the 2025 and 2026 data, with 161 conventions observed to be fixed year-over-year, but 143 new misconfiguration instances appearing in 2026. Depending on the prefix, the location or value of the mismatched byte can imply a different ASN from the name compared with the related IP address in the PTR record. The table shows that of the off-by-* misconfigurations in 2026, there are 345 that imply different ASNs, with the off-by-any errors responsible for most of these, since the value of the wrong byte is less constrained than off-by-tiny.

We sent email using the address in the appropriate SOA record to notify 465 operators of the potential misconfigurations we found in the 2026 data. Of the 465, 121 emails bounced, and 43 of the remaining 344 (12.5%) confirmed the errors (affecting $\sim 12K$ PTR records). This modest response rate is consistent with prior large-scale notification studies [23, 32]. No network operators refuted the misconfiguration we identified.

Table 11 shows that there are similar numbers of affected conventions, PTR records, and eTLD+1 for inconsistent separators, year-over-year. These inconsistencies may not be a problem for network operations, but they may have downstream effects for tools relying on consistent structure. For example, we identified one instance in which a Hoiho-generated regex correctly matches the majority of names for a given convention but misses a name with dots instead of hyphens between bytes of an IP address in the

Table 11: Summary of DNS misconfiguration candidates across the March 2025 and March 2026 rDNS datasets.

Type	2025			2026			pfx2as ASN (2026)			Year-over-year (Pats)			Confirmed
	Pats	eTLD+1	PTRs	Pats	eTLD+1	PTRs	Same	Diff.	Unk.	Fixed	New	Persist.	
Off-by-tiny	572	304	150589	587	315	154725	264	163	160	88	85	489	34
Off-by-any	417	266	84071	404	263	81096	138	182	84	73	58	345	9
Incons. sep.	1622	295	419862	1639	306	425809	–	–	–	289	306	1333	–
<i>Total</i>	2611	738	654522	2630	754	661630	402	345	244	450	449	2167	43

2026-03 PTR record data, and four such instances in the 2025-03 PTR record data².

9 Synthesized Naming Functions as a DNS Record Discovery Tool

In this section, we investigate using namefill’s synthesized functions as a DNS record discovery tool. Datasets of A records generally undersample structured IP spaces because their query inputs are themselves derived from prior observations, *e.g.*, names seeded from prior scans, certificate transparency logs, zone transfers where available, and rDNS data. For the portion of the fDNS namespace that is convention-following, we show that namefill’s synthesized functions can act as a structured enumerator. This technique thus can facilitate systematic identification of interfaces/devices for studying infrastructure, *e.g.*, as in [7, 40], for network topology studies, and/or for auditing address utilization.

9.1 Methodology

We randomly selected one concrete function from every distinct L2 structural convention, for each convention type. We filtered embedded IP functions related to prefixes of size /24–/16 and to only consider those prefixes with synthesized functions in the fDNS dataset but none in the corresponding rDNS dataset to target address space not already characterized through PTR records. We filtered sequential functions related to small (fewer than 10 names) sequences and additionally filtered out low-confidence patterns. Of the remaining functions, we used the methodology of [18] to filter out any wildcard-related functions. This filtering resulted in selecting 54,068 embedded IP functions and 12,571 sequential functions. For embedded IP functions, we generated names for all IP addresses within the prefix following the learned naming convention, and for sequential functions we generated sequential queries until observing three consecutive query failures, up to a maximum of 1M sequential queries.

We issued queries to Cloudflare’s public resolvers over a period of 7 days, rate-limited to 1K queries per second. Of the 355M queries issued, 155M failed or timed out (500 millisecond timeout, 3 retries). We observed no rate limiting or blocking. We used a single resolver rather than distributing queries across several because cross-resolver differences in caching and behavior would confound our results; we discuss load implications in Appendix A.

²Each of these would very likely resolve with additional examples of both types of separators to train Hoiho.

9.2 Results

Table 12 summarizes our results and Figure 5 shows the distribution of expansion factors by program type. For embedded IP functions, we used 857K names to synthesize functions and emitted 331M queries of which 176M were successful, yielding 175M more names than we started with, a 200× increase. For sequential functions, we started with 1.9M names and emitted 24M queries of which 21M were successful, with 19M new names, a 10× increase. These figures do not include sequences that reached the 1M limit. Combined across both pattern types, this represents 194M new names from 2.76M training names, a 70× overall increase.

Initially, we did not cap the length of the search to identify the span of a sequence of names and discovered some unexpectedly large sequences, causing us to place the cap of 1M. In particular, we identified sequential patterns appearing to be of length $2^{63} - 2$ employed by Akamai which are not part of wildcard records, but are likely to be automated responses³. Because sequential probing terminates after three consecutive query failures, the 1M cap is reached only where a zone answers essentially without bound. 20 other sequences reached the cap, several belonging to Chinese CDNs. Overall, most sequences identified through our probing were relatively short, with a median length of 30 and a mean of 1,681. Even sparse equipment-oriented sequences yielded new names: a vlan-numbered interface convention trained on 105 names resolved 13 more.

Table 12: DNS record discovery summary statistics.

	IP	Sequential
Patterns used for probing	54,068	12,571
IP/FQDN used for synthesis	856,799	1,900,067
Total queries	331,406,848	23,554,970
Successful queries	176,078,309	21,100,806
New FQDNs confirmed	175,221,510	19,200,739

10 Discussion and Limitations

In this section we comment on some limitations and additional observations. First, as stated above, namefill is currently limited to IPv4. Extending it to IPv6 is future work and will be non-trivial.

³Responses are only generated for queries that follow the structured pattern, thus they are not wildcards. We note that we estimated the sequence size through targeted experimentation, not exhaustive probing of the DNS.

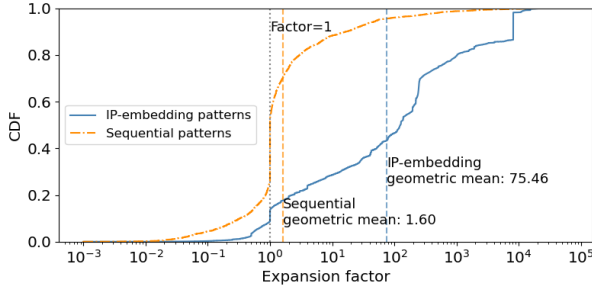


Figure 5: CDF of expansion factors.

The DSL requires new transforms: `ip6.arpa` names encode an address as 32 nibbles in reverse order, and forward names commonly embed hexets rather than byte-aligned values. Zero compression (`:`) means that a single address has multiple textual representations, and IPv6 encodings are frequently variable-width, lacking the zero padding that makes many IPv4 encodings self-delimiting. The deductive search, which enumerates positions and transforms for each of 4 bytes, grows substantially over 32 nibbles, and IPv6 rDNS corpora are considerably sparser than their IPv4 counterparts, posing challenges for both synthesis thresholds and evaluation. We note that `$GENERATE`’s nibble radixes exist precisely for populating `ip6.arpa` zones, suggesting that the class of conventions namefill targets is deployed in IPv6 as well. Another limitation, also noted above, is that namefill does not synthesize mixed programs or synthesize multiple sequences within a single function. We plan to address all of these issues in our ongoing and future work.

The distribution of naming conventions we report is necessarily a function of the DSL: additional segment types, e.g., city names, POP identifiers, or interface types, would split some structures we currently treat as identical and would move some singletons into the L2 class, since more of each name would be typed rather than literal. The counts we report should thus be understood as naming conventions *expressible in namefill’s DSL*. We note, however, that the small number of distinct cross-domain structures (283, cf. § 6) suggests that the structural vocabulary in use is quite limited, and thus that the scope of the DSL is not unduly narrow.

Partitioning input by routed prefix is primarily a parallelization and scoping device (§ 4), but it has consequences for the conventions we report. Because L1 and L2 conventions are defined across two or more prefixes, changes in prefix aggregation or deaggregation—driven by routing and traffic engineering rather than by naming practice—can move a pattern between the singleton and L1/L2 classes without any change in how an operator names hosts. Partitioning also affects recall: a convention used thinly across several prefixes, e.g., with a single name in each of three routed /24s, may fall below the synthesis threshold in each prefix and be missed entirely. Synthesis over coarser groupings would quantify this effect, and we plan to investigate it in future work.

While refining namefill, we found that in the fDNS data there were many FQDNs with “foreign” IP addresses embedded (*i.e.*, addresses that are not related to the FQDN through A records). These appear to be related to carrier-grade NAT deployments and CDN infrastructure. Namefill generally treats any adjacent foreign IP

addresses in related FQDNs as a sequence (instead of an embedded IP), and nearly all of these patterns were tagged as low-confidence due to data sparsity.

Lastly, we also observed in both the PTR and A record datasets that there were a small number of instances in which a valid hexadecimal character in part of the FQDN that is unrelated to an IP address byte could be included in the pattern. For example, for the IP/FQDN pair (192.168.10.200, `pool-200.example.com`), the `a` in `example` could be confused for the hex encoding of 10. Such a function still generates the correct FQDN, but does not reflect the correct semantics of the convention. We found this issue to affect ~0.1% of functions synthesized for the 2026-03 rDNS data.

11 Summary and Future Work

This paper introduces namefill, a tool that uses *programming by example* to synthesize executable functions that summarize DNS naming conventions and generate FQDNs that follow structured patterns. Applied to 1.2 billion PTR records, namefill synthesizes 313K distinct naming conventions across 254K registered domains.

We use the synthesized naming conventions to identify misconfigured and inconsistent PTR records: specifically, copy-paste errors in BIND `$GENERATE` zone templates that embed incorrect prefix bytes, and inconsistencies in separator characters between embedded IP bytes. We identify 991 misconfiguration candidates and 1,639 inconsistent conventions in the 2026-03 PTR data, and confirm 43 template errors with network operators. Lastly, we demonstrate namefill as a DNS record discovery tool. Using synthesized functions as structured enumerators, we issue 355M queries across fDNS data and identify 194M previously unrecorded forward DNS names, representing a 200× increase for IP-embedding conventions and 10× for sequential conventions.

In future work, we plan to extend namefill to synthesize *mixed* programs combining embedded IP bytes and sequential tokens in a single function, and programs for naming conventions that use multiple simultaneous sequences. We also plan to extend the DSL and synthesis algorithm to support IPv6. Finally, the year-over-year stability analysis suggests that longitudinal tracking of convention churn—rebranding, ISP consolidation, infrastructure withdrawal—could serve as a signal for studying how network infrastructure evolves over time; we plan to investigate this direction.

Acknowledgments

We thank Emmett Hintz (Colgate ’25) and Yabesi Witinya (Colgate ’26) for their help on preliminary work that led to this paper. We thank OpenINTEL and CAIDA for providing data for this project. We thank the network operators who graciously responded to our queries about possible naming misconfigurations. Lastly, we thank the anonymous reviewers for their feedback and suggestions. This material is based upon work supported by the National Science Foundation under Grant No. CNS-2312709, CNS-2312710, CNS-2312711, CNS-2319367, CNS-2319368, and CNS-2319369. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

- [1] Messaging Malware Mobile Anti-Abuse Working Group 2015. *M3AAWG Sender Best Common Practices*. Messaging Malware Mobile Anti-Abuse Working Group. Retrieved August 2026 from https://www.m3aawg.org/sites/default/files/doc_files/M3AAWG_Senders_BCP_Ver3-2015-02.pdf
- [2] Center for Applied Internet Data Analysis 2026. *The CAIDA UCSD Internet Topology Data Kit*. Center for Applied Internet Data Analysis. Retrieved August 2026 from <https://www.caida.org/catalog/datasets/internet-topology-data-kit/> March 2026.
- [3] Center for Applied Internet Data Analysis 2026. *The CAIDA UCSD Routeviews Prefix to AS mappings Dataset for IPv4*. Center for Applied Internet Data Analysis. Retrieved August 2026 from <https://www.caida.org/catalog/datasets/routeviews-prefix2as/> March 15, 2025; March 15, 2026.
- [4] 4All [n. d.]. *4All: About It, Offers, Coverage, Reviews and Contacts*. Retrieved August 2026 from <https://offerta-internet.it/4all>
- [5] Anonymous. 2025. Personal Communication.
- [6] Alberto Bartoli, Andrea De Lorenzo, Eric Medvet, and Fabiano Tarlao. 2016. Inference of regular expressions for text extraction from examples. *IEEE Transactions on Knowledge and Data Engineering* 28, 5 (2016), 1217–1230.
- [7] Timm Böttger, Felix Cuadrado, Gareth Tyson, Ignacio Castro, and Steve Uhlig. 2018. Open connect everywhere: A glimpse at the Internet ecosystem through the lens of the Netflix CDN. *ACM SIGCOMM Computer Communication Review* 48, 1 (2018), 28–34.
- [8] Joseph Chabarek and Paul Barford. 2013. What's in a name? Decoding router interface names. In *Proceedings of the 5th ACM Workshop on HotPlanet*. 3–8.
- [9] Ovidiu Dan, Vaibhav Parikh, and Brian D Davison. 2021. IP geolocation through reverse DNS. *ACM Transactions on Internet Technology (TOIT)* 22, 1 (2021), 1–29.
- [10] Mozilla Foundation. [n. d.]. *Public Suffix List*. Retrieved August 2026 from <https://publicsuffix.org>
- [11] Frontier Communications [n. d.]. Frontier nets near-record 103K residential fiber subs in Q1 as Verizon deal presses ahead. Retrieved August 2026 from <https://www.lightreading.com/broadband/frontier-nets-near-record-103k-residential-fiber-subs-in-q1-as-verizon-deal-presses-ahead>
- [12] Peter D Grünwald. 2007. *The minimum description length principle*. MIT press.
- [13] Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. *ACM Sigplan Notices* 46, 1 (2011), 317–330.
- [14] Sumit Gulwani, Aleksandr Polozov, Rishabh Singh, et al. 2017. Program synthesis. *Foundations and Trends® in Programming Languages* 4, 1-2 (2017), 1–119.
- [15] Center for Applied Internet Data Analysis [n. d.]. *Hoiho API*. Center for Applied Internet Data Analysis. Retrieved August 2026 from <https://api.hoiho.caida.org>
- [16] Bradley Huffaker, Marina Fomenkov, and KC Claffy. 2014. DRoP: DNS-based router positioning. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 5–13.
- [17] Internet Systems Consortium. [n. d.]. *BIND Primary File Extension: the \$GENERATE Directive (§3.5.7)*. Retrieved August 2026 from <https://bind9.readthedocs.io/en/latest/chapter3.html>
- [18] Andrew Kalafut, Minaxi Gupta, Pairoj Rattadilok, and Pragneshkumar Patel. 2010. Surveying DNS wildcard usage among the good, the bad, and the ugly. In *International Conference on Security and Privacy in Communication Systems*. Springer, 448–465.
- [19] Emanuel Kitzelmann. 2009. Inductive programming: A survey of program synthesis techniques. In *International workshop on approaches and applications of inductive programming*. Springer, 50–73.
- [20] Rapid7 Labs. [n. d.]. *Rapid7 Labs Open Data*. Retrieved August 2026 from <https://opendata.rapid7.com>
- [21] Youndo Lee and Neil Spring. 2017. Identifying and analyzing broadband Internet reverse DNS names. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. 35–40.
- [22] J. Levine. 2010. *DNS Blacklists and Whitelists*. Retrieved August 2026 from <https://datatracker.ietf.org/doc/html/rfc5782>
- [23] Frank Li, Zakir Durumeric, Jakub Czum, Mohammad Karami, Michael Bailey, Damon McCoy, Stefan Savage, and Vern Paxson. 2016. You've got vulnerability: Exploring effective vulnerability notifications. In *25th USENIX Security Symposium*.
- [24] Matthew Luckie, Bradley Huffaker, and K Claffy. 2019. Learning regexes to extract router names from hostnames. In *Proceedings of the Internet Measurement Conference*. 337–350.
- [25] Matthew Luckie, Bradley Huffaker, Alexander Marder, Zachary Bischof, Marianne Fletcher, and KC Claffy. 2021. Learning to extract geographic information from Internet router hostnames. In *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*. 440–453.
- [26] Aleksandr Polozov and Sumit Gulwani. 2015. Flashmeta: A framework for inductive program synthesis. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*. 107–126.
- [27] Jorma Rissanen. 1978. Modeling by shortest data description. *Automatica* 14, 5 (1978), 465–471.
- [28] Quirin Scheitle, Oliver Gasser, Patrick Sattler, and Georg Carle. 2017. HLOC: Hints-based geolocation leveraging multiple measurement frameworks. In *2017 Network Traffic Measurement and Analysis Conference (TMA)*. IEEE, 1–9.
- [29] A. Solar-Lezama. 2018. *Introduction to Program Synthesis*. Retrieved August 2026 from <https://people.csail.mit.edu/asolar/SynthesisCourse/TOC.htm>
- [30] J. Sommers. 2026. *namefill software*. Retrieved August 2026 from <https://github.com/jsommers/namefill>
- [31] Neil Spring, Ratul Mahajan, and David Wetherall. 2002. Measuring ISP topologies with Rocketfuel. *ACM SIGCOMM Computer Communication Review* 32, 4 (2002), 133–145.
- [32] Ben Stock, Giancarlo Pellegrino, Christian Rossow, Martin Johns, and Michael Backes. 2016. Hey, you have a problem: On the feasibility of large-scale web vulnerability notification. In *25th USENIX Security Symposium*.
- [33] Sousan Tarahomi, Raffaele Sommese, Pieter-Tjerk de Boer, Jeroen Linssen, Ralph Holz, and Anna Sperotto. 2024. Is a Name Enough? A First Look into Detecting Clouds Using DNS Pointer Records. In *2024 20th International Conference on Network and Service Management (CNSM)*. IEEE, 1–5.
- [34] TheAleph [n. d.]. *TheAleph API*. Retrieved August 2026 from <https://thealeph.ai/docs>
- [35] Kedar Thiagarajan, Esteban Carisimo, and Fabián E. Bustamante. 2025. The Aleph: Decoding Geographic Information from DNS PTR Records Using Large Language Models. *Proc. ACM Netw.* 3, CoNEXT1, Article 7 (March 2025), 20 pages. doi:10.1145/3709374
- [36] Olivier van der Toorn, Roland van Rijswijk-Deij, Raffaele Sommese, Anna Sperotto, and Mattijs Jonker. 2022. Saving Brian's privacy: the perils of privacy exposure through reverse DNS. In *Proceedings of the 22nd ACM Internet Measurement Conference*. 1–13.
- [37] Roland van Rijswijk-Deij, Mattijs Jonker, Anna Sperotto, and Aiko Pras. 2016. A high-performance, scalable infrastructure for large-scale active DNS measurements. *IEEE journal on selected areas in communications* 34, 6 (2016), 1877–1888.
- [38] Cynthia Wagner, Jérôme François, Radu State, Thomas Engel, Gerard Wagener, and Alexandre Dulaunoy. 2012. SDBF: Smart DNS brute-forcer. In *2012 IEEE Network Operations and Management Symposium*. IEEE, 1001–1007.
- [39] Ming Zhang, Yaoping Ruan, Vivek S Pai, and Jennifer Rexford. 2006. How DNS Misnaming Distorts Internet Topology Mapping. In *USENIX Annual Technical Conference, General Track*. 369–374.
- [40] Zesen Zhang, Alexander Marder, Ricky Mok, Bradley Huffaker, Matthew Luckie, Kimberly C Claffy, and Aaron Schulman. 2021. Inferring regional access network topologies: Methods and applications. In *Proceedings of the 21st ACM Internet Measurement Conference*. 720–738.

A Ethics

To synthesize functions that summarize naming patterns, `namefill` ingests IP/FQDN pairs from corpora of DNS A or PTR records. Although IP addresses may be considered sensitive data, they are not considered in the context of any traffic flow in this work which limits privacy implications. There has, however, been prior work to show that contents of PTR records contain sensitive data [36]. We have not explicitly observed any sensitive personal information to be included in a synthesized function, but it is possible for that to happen for sequences of names with related personal information, *e.g.*, `Rhodri-home-1`, `Rhodri-home-2`, etc. In such a case, the function could be used to identify other names that follow the same pattern. It is therefore possible that `namefill` gives an attacker a slight leg up in identifying and exploiting names that follow such patterns.

For operator notification of potential misconfigurations, we sent one message to the relevant organizational email address identified in the SOA record. The message included details of each of the prefixes (if more than one) relevant to the operator, along with examples of the potential misconfiguration and an inferred suggestion of the fix. We also asked each operator, if they were willing, to share information about their infrastructure, how records were generated, etc., in order to better understand processes that may have led to the misconfiguration, and asked if operators were willing for us to share our findings about their networks publicly. Any

examples of misconfigurations shared in the paper were done with the explicit permission of the operator.

Lastly, there was additional load placed on DNS infrastructure through our 355M-query probing campaign. We directed our queries to Cloudflare’s public resolvers, rate-limited to 1K queries per second, which is negligible relative to the query volume those resolvers handle. Querying a large caching resolver rather than authoritative servers directly also reduces the load reaching any individual authoritative operator, since responses for repeated or adjacent names within a zone are frequently served from cache. Distributing queries across multiple resolvers would spread this load further, but cross-resolver differences in caching and behavior would confound our results; at 1K queries per second we judged the load on a single large resolver to be acceptable.

B Generative AI Disclosure

Figure 4 was initially generated in TikZ using Claude Code 2.1.114 (with model Claude Sonnet 4.6) and subsequently modified by hand. In addition, Claude Code was used to translate code for doing DNS record discovery from hand-written code in Python to Go. The Go version was subsequently modified by hand. Experiments to evaluate different `namefill` program rankings were suggested by Claude Code, but ultimately done by hand. All code in `namefill` was written and tested initially (and extensively) by hand, but refactored and improved with assistance from Claude Code.

No generative AI tools were used to write text that appears in the paper.